Further developments of the AcBuilder tool for constructing geometrical models of aircraft

Pierre Saquet

07/03/2011 – 02/09/2011

KTH Royal Institute of Technology, Stockholm, Sweden

Supervisor KTH: Prof. Arthur Rizzi
Supervisor INSA: Prof. Luc Vervisch
Acknowledgments

In this page, I want to give a special thanks to people who help me in writing this master thesis by their everyday support in my work or in my personal life in Sweden.

First of all, I would like to thank my supervisor, Professor Arthur Rizzi, for his constant support and his trust in my work, and also for admitting me in the aerodynamic division and permitting me to come in KTH to write my master thesis. Thanks also to my co-supervisor, Jesper Oppelstrup, for his important and helpful remarks, comments and suggestions. From a personal point of view, it has been a pleasure to work in this fascinating project thanks to their enthusiasm and encouragements.

I would also like to express my gratitude to the whole team of our partner CFS engineering in Switzerland: Benoit Rey, Jan Vos, Alain Gheri, Dominique Charbonnier, Stéphane Sanchi, Elizabeth Mickaily – Huber. They hosted me with kindness during 2 weeks at the beginning of my project.

Here, I want to thanks my officemates who permit me to write my master thesis in a good ambiance and in a great work environment. Thanks to Mengmeng, Liana, Mercedes and Enrique. Thanks also to Benjamin and Laurent for their work and their help in my project.

I am deeply grateful to my corridor mates Maxime, Florian, Maeva, Julie, Benjamin, Marina, Maika, Christina, Jessica, Suzana, Valentino, Arman, Gauthier, Berkey, Steffen, Colin, Bettina and all the people from the Tyreso accommodation. I spent such a good time with them; I will never forget these few amazing months in Sweden.

I want to give a particular thanks to my roommate Simon for his support in all the situations and for all the great moments shared together. Particularly, thanks to him for his great sense of humour.

I have met a lot of people during these 6 months, thanks everyone for all the good times and experiences shared together in Stockholm. I wish to thanks especially Vivien, Diana, Marion, Mervi, Jonas without forgotten all the people playing football every week with me; it was so nice to find such a great group to play regularly.

Finally, I want to thank my friends from France, my brothers and my parents for their support despite the distance, I felt no so far away from home and I spent one of the best period of my life thanks to them.
Abstract

This report, along with Laurent Gourc’s and Ben Marchant’s reports, presents the work done on the development of the new AcBuilder, realized for CEASIOM. CEASIOM is a package of different modules, developed as part of the SimSAC project, which aims to Simulate Stability And Control Characteristics for Use in Conceptual Design.

First, the CEASIOM software is introduced in the context of the SimSAC project and, to know where the development of the aircraft builder tool (AcBuilder) is, an overview of the previous version is shown. Then, based on the issues noticed by the users and the programmers of CEASIOM, the goals of the project are presented in listing some modifications and improvements to bring to the software.

Secondly, the document treats about the requirements for the new AcBuilder development in order to reach the goals of the project. Those requirements come from both the programming languages used (Matlab and Java) and from the technical parts of the project (geometrical construction of aircraft).

Finally, this report presents the new AcBuilder tool, its new design interface, its new functionalities and the remaining improvements to implement in order to make the module compatible with the changes brought by the new requirements.
Contents
Acknowledgments .................................................................................................................. 2
Abstract .................................................................................................................................. 3
Contents ..................................................................................................................................... 4
List of figures ............................................................................................................................ 5
I. Introduction ............................................................................................................................ 6
   A. CEASIOM package ............................................................................................................. 6
   B. AcBuilder .......................................................................................................................... 10
      1. Geometric definition ...................................................................................................... 10
      2. Fuel tanks definitions ................................................................................................... 11
      3. Cabin and luggage definitions ...................................................................................... 12
      4. Centers of gravity computation .................................................................................... 13
      5. Technology definition .................................................................................................... 13
      6. XML file definition ....................................................................................................... 14
   C. Goals .................................................................................................................................. 14
II. Requirements for a new AcBuilder ......................................................................................... 16
   A. Programming language .................................................................................................... 16
   B. Communication Matlab ↔ Java ...................................................................................... 16
   C. Aircraft geometric definition ........................................................................................... 16
   D. Corrections and improvements ........................................................................................ 17
   E. Weights and balances, centers of gravity ...................................................................... 17
III. Functionalities of the new AcBuilder .................................................................................. 18
   A. Project menu .................................................................................................................... 19
   B. Number of elements and order ........................................................................................ 21
   C. Element or aircraft creation .............................................................................................. 22
   D. Geometric definition ........................................................................................................ 23
   E. Fuel tanks definition ......................................................................................................... 24
   F. Technology definition ....................................................................................................... 25
   G. Weights and balances ...................................................................................................... 27
IV. Conclusion ........................................................................................................................... 28
V. References ............................................................................................................................. 29
Appendix A: The AcBuilder: Geometry Definition for CEASIOM, I: Lifting Surfaces ............ 30
Appendix B: The AcBuilder: Geometry Definition for CEASIOM, II: Bodies .......................... 48
Appendix C: Program explanations .................................................................................. 70
Appendix D: AcBuilder N2 chart .................................................................................. 80

List of figures
Figure 1 - Virtual and classical aircraft design ................................................................. 6
Figure 2 - Schematic diagram showing the interactive nature of SimSAC software and its core modules for aircraft design ................................................................................. 7
Figure 3 - the conceptual design phase illustrated within SimSAC ...................................... 8
Figure 4 - Map of CEASIOM architecture ...................................................................... 9
Figure 5 - Geometric definition ...................................................................................... 10
Figure 6 - Fuselage cross section .................................................................................. 11
Figure 7 - Airfoils ............................................................................................................. 11
Figure 8 - Fuel tanks definition ..................................................................................... 12
Figure 9 - Cabin and luggage definitions .................................................................... 12
Figure 10 - Centers of gravity computation ................................................................... 13
Figure 11 - Technology definition ................................................................................ 13
Figure 12 - Tree structure of the aircraft geometrical data ............................................. 14
Figure 13 - Basic structure of the new AcBuilder ........................................................... 18
Figure 14 - Geometric definition (new AcBuilder) ....................................................... 19
Figure 15 - Project menu / New AcBuilder ................................................................... 19
Figure 16 – a) Elements box / b) Aircraft templates .................................................... 20
Figure 17 - Old AcBuilder (left) / New AcBuilder (right) ............................................. 20
Figure 18 - 2 aircrafts in the new AcBuilder ................................................................. 21
Figure 19 - Remove function ....................................................................................... 22
Figure 20 - Java class hierarchy ................................................................................... 23
Figure 21 - Cigar shape (fairing) .................................................................................. 23
Figure 22 – 2 aircrafts / fuel tanks definition ................................................................. 24
Figure 23 - Fuel tank cutout ........................................................................................ 24
Figure 24 - List box for technology definition .............................................................. 25
Figure 25 - 2 aircrafts / technology beam model .......................................................... 25
Figure 26 - 2 aircrafts / technology aero panel ............................................................... 26
Figure 27 - 2 aircrafts / technology spar location .......................................................... 26
I. Introduction

A. CEASIOM package

CEASIOM is developed in the scope of the SimSAC project. SimSAC stands for Simulating Stability And Control Characteristics for Use in Conceptual Design, and it is a project approved for funding by the European Commission 6th Framework Program on Research, Technological Development and Demonstration. The work began 1 November 2006 (more information can be found on the SimSAC project webpage [1]). The ultimate goal of the project is to improve the prediction of stability and control aircraft characteristics.

Present trends in aircraft design towards augmented-stability and expanded flight envelopes call for an accurate description of the non-linear flight-dynamic behavior of the aircraft in order to properly design the Flight Control System (FCS). Hence the need to increase the knowledge about stability and control (S&C) as early as possible in the aircraft development process in order to be "First-Time-Right" with the FCS design architecture.

FCS design usually starts near the end of the conceptual design phase when the configuration has been tentatively frozen and experimental data for predicted aerodynamic characteristics are available. Up to 80% of the life-cycle cost of an aircraft is incurred during the conceptual design phase so mistakes must be avoided. Today prediction errors related to S&C result in costly fly-and-try fixes, sometimes involving loss of proto-type aircraft and crew.

Figure 1 - Virtual and classical aircraft design
To meet these challenges SimSAC develops along two major axes:

- creation and implementation of a simulation environment, CEASIOM, for conceptual design sizing and optimization suitably knitted for low-to-high-fidelity S&C analysis; and
- an improved pragmatic mix of numerical tools benchmarked against experimental data.

More specifically, the SimSAC project contains 4 primary work elements:

- Development and subsequent linking of a series of disparate expert software modules into a computerized system (CEASIOM) that generates dependent parameter results for purposes of assessing S&C.
- Thorough benchmarking of the developed expert software tools.
- Application of CEASIOM to a select number of aircraft design problems.
- Wind-tunnel verification of a CEASIOM-designed aircraft.

CEASIOM (Computerized Environment for Aircraft Synthesis and Integrated Optimization Methods) [2], is a software originally based on Isikveren’s work [3], who created the MATLAB QCARD package for aircraft conceptual design with quasi-analytical shape definitions, aero-data correlations and performance predictions. It will initially focus on fast low fidelity analysis, and as appropriate, resort to higher fidelity numerical simulations. A remarkable characteristic of CEASIOM is that it will introduce stability and control optimization earlier in the design cycle than is standard practice nowadays. CEASIOM runs under either Windows or Linux, and it basic version requires a MATLAB license only. In executable form the code can be run without a license.

![Figure 2 - Schematic diagram showing the interactive nature of SimSAC software and its core modules for aircraft design](image-url)
CEASIOM is a tool used for the preliminary design and analysis of aircraft. It allows a user to, using a set of parameters, define the aircraft for various analysis tasks such as stability analysis and aeroelastics. CEASIOM is under constant development from universities in Europe within the SimSAC project. “The mission of the SimSAC project is to enhance the conceptual design and early preliminary design processes by developing an integrated digital design and decision making environment where for any given aircraft configuration the aerodynamic information for stability and control assessment can be computed at some user nominated fidelity in the conceptual design phase (Figure 2).”

![Diagram](image)

**Figure 3 - the conceptual design phase illustrated within SimSAC**

A brief description of the modules within the CEASIOM package is shown here (Figure 4):

- **AcBuilder** - Used to render the 3D model of the aircraft for the user to visualize it both externally and internally based on a number of basic parameters.

- **Geometry, Weights** - Integrated into the AcBuilder GUI, is used to calculate reference wing parameters, as well as weights and centers of gravity.

- **Propulsion** - This is used to calculate the thrust required and produced by the engines based on the altitude as well as the aircraft's parameters.

- **SUMO** - SUMO is used to generate a mesh for given aircraft geometry, it is required for some of the CFD solvers in CEASIOM, but is not required for others.
• **AMB – Aerodynamic Model Builder** is the GUI for the CFD solvers included in CEASIOM. There are 4 main solvers which can be used.

• **SDSA - Simulation and Dynamic Stability Analysis** is used to calculate both the dynamic and the static stability of the aircraft, as well as providing simulations of the aircraft in flight to analyze handling characteristics.

• **FCSDT - Flight Control System Design Toolkit** is used to design and analyze various aspects of the aircraft's flight control system.

• **NeoCASS – Next generation Conceptual Aero-Structural Sizing** is a completely MATLAB based tool which is primarily used to assess the aeroelastic qualities of the aircraft.

---

![Figure 4 - Map of CEASIOM architecture](image-url)
B. AcBuilder

AcBuilder is a CAD (Computerized Aided-Design) tool permitting to carry out the conceptual aircraft design. It is composed of different parts: the geometric definition, the fuel tanks definitions, the cabin and luggage definitions, the centers of gravity computation and the technology definition. In each part, the user can modify several parameters defining the aircraft geometry or properties (the total fuselage length for example). From those parameters, the program calculates the coordinates of the sections for creating each element (fuselage, wing, fuel tanks…). All the parameters used in AcBuilder are detailed in the XML file definition [4].

1. Geometric definition

This part concerns the aircraft geometric definition. In AcBuilder, an aircraft can be composed of 1 fuselage, 2 wings, 1 horizontal tail, 1 vertical tail, 2 pairs of engines, 1 pair of tail booms, 1 canard and 1 ventral fin. Each element is defined by some parameters. For example, the fuselage parameters are shown Figure 5.

![Figure 5 - Geometric definition](image)
The parameters values in green are calculated from others parameters and those in blue are chosen in a separate window. For example the user can choose the fuselage cross section (Figure 6) and the airfoil for the wing (Figure 7).

![Fuselage cross section](image1)

**Figure 6 - Fuselage cross section**

![Airfoils](image2)

**Figure 7 - Airfoils**

2. **Fuel tanks definitions**

After the external geometric part, the user can define the fuel tanks geometry (Figure 8) in the wing and in the fairing (made part of the wing in the AcBuilder definition) thanks to some parameters.
3. Cabin and luggage definitions

In this part, the user can define the number of passengers in the cabin, the cabin geometry, the space for the luggage… (Figure 9).
4. **Centers of gravity computation**

According with all the parameters defined previously, the software compute the centers of gravity of each element and the final aircraft center of gravity (Figure 10).

![Figure 10 - Centres of gravity computation](image1)

5. **Technology definition**

This last part defined the beam model for the structural and aeroelastic module (Figure 11).

![Figure 11 - Technology definition](image2)
6. **XML file definition**

All those parameters are stocked in an XML file, it is the database for all the others module, they will load the information from this file.

The geometrical data of the aircraft has a tree structure like the one shown in Figure 12. The aircraft has different child elements, which are its components (Fuselage, Wing 1, Wing 2, Horizontal tail, Vertical tail, Engines, etc.). Each of these child elements have child elements themselves, containing parameters which describe them. For example, the Wing can have child elements containing information about its span, area or control surfaces such as the ailerons. The fourth level of depth of the tree contains data that describes the control surfaces.

![Figure 12 - Tree structure of the aircraft geometrical data](image)

When the user saves an aircraft after changing some parameters and computing the weights and balances and the centers of gravity in AcBuilder, all the information are stocked in an XML file. In fact, AcBuilder is similar to an XML file editor with visualization.

**C. Goals**

The main goal of this project consists in continuing the development of the AcBuilder tool in CEASIOM software.
The need of pursuing the development of this module comes to several issues noticed by the users and the programmers.

First of all, AcBuilder is the only module in CEASIOM programmed with the JAVA language (mixed with Matlab). Most of the CEASIOM users and programmers are more familiar with Matlab than Java. That’s why a new version of AcBuilder programmed as much as possible in Matlab need to be developed.

Secondly, a lot of mistakes have been noticed by the users in the geometrical construction of the aircraft. The goal is to correct and improve the geometrical definition for all the elements constituting the aircraft.

Then, one of the goals consists in making AcBuilder more dynamic and flexible in order to create any shape of aircrafts and not only “conventional” ones (with 1 fuselage, 1 wing, 1 horizontal tail, 1 vertical tail…).

As a consequence, a new module AcBuilder need to be created with all those modifications. Thus, the last issue will be to integrate this new version in CEASIOM software that is to say to make compatible this new module with the others.
II. Requirements for a new AcBuilder

The following requirements will permit to reach the goals enunciated before.

A. Programming language

The AcBuilder module is essentially Java based (mixed with Matlab). Indeed, the coordinates calculation (which permit to create the elements surfaces) from the input parameters is made in java as well as the whole graphic interface. So, this is very difficult for someone who does not know about Java to understand the code and consequently modify it and improve it.

Thinking about this issue, the development team decided to rewrite the whole module in Matlab excepted for the visualization, which is better and faster in Java. By reducing the Java part merely as a viewer, the development of the module will be easier. Particularly, the computation of matrices corresponding to the elements surfaces must be programmed in Matlab to reduce the Java part.

B. Communication Matlab ↔ Java

In the current version of AcBuilder, both languages exchange some information and it could be a problem for the software stability. But now, according with the previous decision, which is to use the Java part as a simple viewer, all the computations can be made in Matlab and transferred to Java for the rendering. There will be no feedback from Java to Matlab. Thus the software stability will be improved.

C. Aircraft geometric definition

This section treats about the real function of AcBuilder, which is to create the aircraft geometry. In the current version, the user has no other choice but to respect the number of elements (1 fuselage, 2 wings, 1 horizontal tail, 1 vertical tail, 2 pairs of engines, 1 canard, 1 ventral fin and 1 pair of tail booms) and the order of creation of these elements (the fuselage, then the wings, the horizontal/vertical tails, the canard and the ventral fin, then the engines) because they are not independents. For example, an engine is built from the local coordinates of the wings and the wings from the fuselage.
Some users and researchers working on CEASIOM would like to create aircrafts without those restrictions. The elements dependency from each other has to be removed and the module need to be capable of creating as many elements as the user wants in order to build “unconventional” aircrafts.

D. Corrections and improvements

By using AcBuilder, the users found a lot of mistakes in the module. Indeed, some functions seem to be not working at all and some others do not work properly. Sometimes, there are also some mismatches between the current AcBuilder and the documentation supplied. AcBuilder needs an update to correct all the errors and to revise the documentation.

Two students, Laurent Gourc and Ben Marchant, were in charge of this part of the project as part of a course in KTH. Some sections of their report are presented in appendixes B and C.

Their reports [5], [6] show some issues about the geometrical construction of the aircraft and they proposed some improvements. Particularly, the surface generation is now written and computed in Matlab (and transferred later in java in coordinates shape) in a well-commented and documented Matlab code in order to be more accessible for engineers. If new parameters are needed to improve the aircraft geometry design, the solutions are just proposed and the next step consists in implementing them properly in AcBuilder. The revised documentation of the parameters used for the aircraft creation are listed in appendix A, section 4) for lifting surfaces and in Appendix B, section 3) b) for bodies elements.

E. Weights and balances, centers of gravity

A stand-alone module “Weights and balances” has been developed by Javier Muñoz Martín with different methods more detailed than the one used in AcBuilder. It could be really useful to integrate this new module in AcBuilder because the documentation is recent and the codes are well commented.

Those requirements imply new functionalities in AcBuilder as well as a new structure and a new graphic interface.
III. Functionalities of the new AcBuilder

A new AcBuilder have been created. In this new module, all the calculations are carried out in Matlab and the Java part is used merely as a renderer. As a consequence, the communications between those two languages are simplified because all the data go from Matlab to Java without feedback from Java to Matlab (Figure 13).

For the users’ convenience, the graphic interface of the new AcBuilder looks like the one used in the old version but now, it is separated in two windows: the one on the left is made in Matlab and the other one on the right is programmed in Java (Figure 14).

The Matlab window contains the different menus (project, geometry, weights and balance, technology, help), the list box and the parameters corresponding to the chosen menu and also the buttons permitting to choose a view.

The Java window serves to draw the aircraft element by element and the users can find some options in the menu permitting to change the color of the background and of the different elements constituting the aircraft.
This new module works in the same way than the old version, the users have to proceed step by step going through the different parts of the module: the geometric definition, the fuel tanks definition, the weights and balance, the centers of gravity and the technology definition.

A. Project menu

In the previous version, the user could start a new project by loading an aircraft template (Boeing 747, Ranger, Horizon…). Now the user has the choice to add a whole aircraft (“Load Aircraft” in Figure 15) or just an element (fuselage, wing, engine… by using “Load Element” in Figure 15). With this new function, an aircraft can be built element by element (Figure 16–a) by loading predefined templates (Figure 16-b) for each element.
The function permitting to add a whole aircraft is a little bit different than the one used in the old AcBuilder. Indeed, in the previous version, an aircraft is always defined by 10 elements (1 fuselage, 2 wings, 1 horizontal tail, 1 vertical tail, 2 pairs of engines, 1 canard, 1 ventral fin and 1 pair of tail booms. And the user has to choose if each element is present or not. Now when the user creates a new aircraft, only the elements really used to build it are loaded. For example, the “horizon” is made of 1 fuselage, 1 wing, 1 horizontal tail, 1 vertical tail and 1 pair of engines (Figure 17). There is no canard, ventral fin or tail booms.

In Figure 15, there are also 3 others functions in the project menu. The “Import XML” function is used to load an XML file from a project already created and the “Save XML” function permits to save the current project in an XML file. The last function “Export point grid” serves to export files in the right format for the SUMO module (this function is just partially working at this point of the development).
B. Number of elements and order

In the old AcBuilder, the definition of an aircraft is limited to the 10 elements mentioned previously and those elements are not independents: the coordinate system is not absolute but local. Indeed, the wings, the horizontal tail, the vertical tail, the canard and the ventral fin are related to the fuselage (the location of these elements are calculated from the fuselage length and high thanks to parameters indicating the percentage of length and high) and the engines are related to the wings.

The new AcBuilder avoids those two restrictions. The user can now create as many elements as he wants and in any order thanks to an absolute coordinate system: all the elements are independents (Figure 18).

![Figure 18 - 2 aircrafts in the new AcBuilder](image)

A new function permitting to remove an element has been implemented for the customers’ comfort (Figure 19). If the user wants to create a design close to a precise template but with a wing from another aircraft, this function is useful: he just has to remove the wing and add another one with another template. Of course if he wants to create a nonexistent wing in the database, he can change the parameters and invents his own wing. In Figure 19, the “Select / Unselect” button is just used to visualize or not the highlight element.
This new way to build the aircraft will imply a lot of modifications in the others part of AcBuilder (fuel, weights and balances, centers of gravity and technology).

![Figure 19 - Remove function](image)

### C. Element or aircraft creation

As it has been enunciated before, all the parameters and the information are stoked in an XML file. When the user creates an element or an aircraft, he has to choose a predefined template (Boeing 747, Ranger, Horizon…). At this point, the data stored in this XML file are copied in variables and tables in Matlab. After that, when the user changes some parameters or adds some elements, the calculation is made on those variables and the right function corresponding to this change is activated. This function will calculate all the coordinates permitting to create the surface from the parameters stocked in the Matlab variables (so if the user changes a parameter from the fuselage, the function corresponding to the fuselage is launched in order to compute the coordinates matrices). The way to calculate those coordinates is detailed in Gourc’s and Marchant’s reports [5], [6] (appendixes B and C). The next step consists in transferring the coordinates matrices from Matlab to Java in order to draw the aircraft surface.

The Java part is composed of a main class (AcBuilder), a class corresponding to the drawing window (GLViewer), a common class dealing with all the components (AcComponent) and a class for each element (Figure 20). The technique for drawing each element used in Java consists in creating small quads with the coordinates matrices coming from Matlab and in iterating them in the length and around the element. Finally, those quads form the element surface. For more efficiency, the program call only the function corresponding to the element the user wants to change or create.
Some details about how the program works for drawing elements are presented in Appendix C.

D. Geometric definition

The geometric definition of all the elements has been revised by L. Gourc and B. Marchant [5], [6]. The more important change is the separation between the wing and the fairing. Indeed, a fairing is now an independent element (Appendix B, 2), b). This new definition can be used not only for fairing element but also for all element shape like a cigar shape (canopy, tail booms…)}
E. Fuel tanks definition

As the geometric part can now define as many wings as the user wants, the fuel part has to accept any number of wings (Figure 22). The fuel tanks display is now in 3D for a better visualization. Two new parameters (cutout_length and cutout_position) permit to define properly the cutout (Figure 23).

![Figure 22 – 2 aircrafts / fuel tanks definition](image1)

![Figure 23 - Fuel tank cutout](image2)
F. Technology definition

The technology part has also been revised to be able to work well with more than 1 wing, 1 horizontal tail, 1 vertical tail and 1 canard. For the moment, the beam model (Figure 25), the aero panel (Figure 26) and the spar location (Figure 27) have been modified. The list box in the left window (to access the different tables) is now ordered by element for each part (beam model, aero panel, spar location)(Figure 24).

![List box for technology definition](image1)

**Figure 24 - List box for technology definition**

![2 aircrafts / technology beam model](image2)

**Figure 25 - 2 aircrafts / technology beam model**
Figure 26 - 2 aircrafts / technology aero panel

Figure 27 - 2 aircrafts / technology spar location
G. Weights and balances

At this point of the development, the weights and balances part is the same than the previous version.

This new AcBuilder is still in development phase; particularly some changes and improvements are needed in the weights and balances part. At this point, the module operates like the old version with a limited number of elements because of the weights and balances part.
IV. Conclusion

CEASIOM’s goals are to implement earlier in the design stage of an aircraft analyses in fields such as aerodynamics, propulsion or stability and control for instance. However, the CEASIOM package, as it was defined, would only permit the study of a “conventional aircraft”. As more and more projects aim to create “unconventional aircrafts”, the need for CEASIOM to evolve to a more general definition arises.

The geometric definition of each element constituting the aircraft has been improved. A lot of mistakes noticed by the customers have been corrected. All the elements are now independent and the order of creation has no longer influences on the software. The users can now create “unconventional aircrafts”.

In addition, the structure of AcBuilder has been completely updated: all the computations are made in Matlab and Java is only used as the renderer. It is now easier for the programmers to understand the module and to improve it. Moreover, this new structure brings more stability to the software.

The new module AcBuilder is not finished; the next step consists in improve the weights and balances part by integrating and making compatible the stand-alone “weights and balances” created by Javier Muñoz Martín.

CEASIOM software is always in development in some universities around the world; one of the goals for the future is to make a link between CEASIOM and CPACS (Common Parametric Aircraft Configuration Scheme). CPACS is used by DLR, Hamburg, Germany. It is a more flexible and advanced file format than the current XML file used by CEASIOM. For example, CPACS permits to add as many wings with as many sections with complex geometries as desired where CEASIOM can only create 2 wings with 4 sections (unlimited wings this new AcBuilder but still 4 sections per wing).

The implementation of CPACS into CEASIOM would bring more flexibility and fidelity in the aircraft design process.
V. References

[1] SimSAC project webpage. www.simsacdesign.org


Appendix A: The AcBuilder: Geometry Definition for CEASIOM, I: Lifting Surfaces

1) Geometry issues

   a) Winglet

There are a number of problems in the definition of the winglet. These problems are present in both the documentation as well as in the actual programming. This is possible since the documentation incorrectly defines the variables, whilst the variables actually used do not allow for certain geometries. In order to see the issues with creating the geometries, one must first look at the definition of the winglet. In the existing winglet documentation, the winglet is defined as shown in Figure A1a, this does not correspond with how CEASIOM treats the winglet, and this is shown in Figure A1b.

![Figure A1](image)

Figure A1 – Winglet definitions in (a) Documentation and (b) CEASIOM.

Now that the current geometry of the winglet has been defined correctly, it is possible to discuss the advantages and disadvantages of using this. The wing matrix is composed of airfoil cross sections at a given span with their defined sizes and positions. By defining the winglet as shown, it makes it very simple to add a winglet to the wing since the winglet span can be added to the main wing span to give the y position and the cant angle is geometrically identical to the main wing’s dihedral angle. This allows the winglet to be added to the main wing with minimal additional coding.

Although simple, there is a problem with defining a winglet in this way: it does not allow the creation of a vertical winglet since if the cant angle is set to 90° and the winglet span is finite, the winglet will extend to infinity since it will never reach the desired span. This problem is illustrated in Figure A2.
An additional problem with the current definition of a winglet is the presence of the variable “winglet root incidence”. It allows a user to choose their own value of the incidence at the root of the winglet. This should, however, be constrained to be equal to the wing tip incidence because otherwise, holes are opened and impossible geometries are created as shown in Figure A3. Although the configuration in Figure A3 is an extreme case, and it is unlikely that anyone would want to create a winglet with a 30° incidence, it does illustrate the issue clearly.

Figure A 2 - Attempted vertical winglet

Figure A 3 - Attempted extreme winglet incidence
b) Flags

Within the current AcBuilder, there are a number of somewhat redundant “Flags”. The issue with these is, although providing somewhat of a shortcut to a given configuration, they can be overwritten. For example: with the “empennage layout”, one can check a box to set the aircraft to a T-tail configuration, but then, upon unchecking the box, the horizontal tail does not revert to its previous position. Although not a problem for the program, it is not particularly beneficial, especially within the new AcBuilder, where elements are intended to be placed independently from one another.

c) Control surfaces

Although minor, there is an error in AcBuilder regarding the aileron span. It is defined correctly in the documentation, as a fraction of the wing section between kink2 and the wingtip. In AcBuilder, it gives the unit as meters, which could cause some confusion for a user.

Another issue within the aileron definition is the position, in the documentation, it is defined incorrectly. A revised, correct definition is shown in Figure A4.

![Figure A4](image-url)

Figure A 4 - (a) aileron position=0, (b) aileron position=1, (c) aileron position=anything else

2) Changes implemented

The surfaces within AcBuilder are currently calculated and rendered within Java. An alternative has been offered, whereby all of the user-defined AcBuilder variables are read by the program. And the surfaces are calculated as a series of matrices. These matrices can be used to directly plot the surfaces within Matlab, using the “surfl” command, however since this is relatively slow, the final rendering is completed using Java instead.

a) Wing surface generation

The surface of the wing is created by generating airfoil cross sections at a number of span stations: Root, Kink1, Kink2 and Tip. These cross-sections are created using an existing function
within the CEASIOM package named “wgcomp.m” with one minor change; The existing *wgcomp* returned a number of points for the airfoil cross section based on the number of data points in the data file. The revised version always generates the same number of points around the surface, regardless of the airfoil data file.

“wgcomp” was selected to be used almost unchanged since it is comprehensive. It deals with the airfoil generation effectively by accounting for factors such as dihedral and incidence. It reads from a library of airfoil cross sections, and then, using the variables input, sizes it accordingly. It is used slightly differently from its original use within the GEO package, since within GEO, it was used to create a wing with one airfoil profile across the whole span; this is not what is desired in the new AcBuilder, so its usage had to be modified slightly. Using it like this is slightly less efficient than it could be, however it is still accurate and provides reasonably fast results.

Once these airfoil cross sections/kinks have been created, they are then positioned according to the positioning variables such as: spanwise location, dihedral and sweep. Their positioning within the x-z plane is based on the position of the previous kink. This is shown in Figure A5, where the MATLAB code used for positioning the splines is shown. It uses the previous spline and the next span section along with the required angle.

![Figure A5 - Matlab code used to position the splines Where lswm is the leading edge sweep of each section and dihm is the dihedral of each section](image)

The processes behind the positioning of the airfoils must be explained carefully to clarify any future issues.

1. When the airfoil profiles are generated, the point at the leading edge is given the coordinates: \((x, z) = (0, 0)\).

2. Within the wgcomp MATLAB function, the airfoil is then rotated about its leading edge point \((0, 0)\) to obtain the required incidence.

3. The points to place these \((0,0)\) points are then generated using a y coordinate based on the span and x and z coordinates based on the calculations shown in Figure A5.

Shown in Figure A6 is the positioning of the splines. The points on the red line are the positions calculated based on the dihedral and sweep, and the blue lines are the airfoil splines. Only 2 sections are shown here for clarity.
Once the cross sections and the positions of these cross sections have been defined, the surface generation within MATLAB is trivial, it just requires the organization into the correct matrix format and the surface generation command. Within the Java renderer it may differ however with the points it should still be a relatively straightforward task.

One shortcoming of this method is the fact that by creating the airfoil splines in this way, in the $x$-$y$ plane, if high dihedral angles or sweep angles are attempted, it will cause strange connections between the two splines. This issue is dealt with in the revised AcBuilder wing generation discussed in Section 3) c).

![Figure A 6 - Positioning of the splines](image)

Due to the problem with the incompatibility between “wing tip incidence” and “winglet root incidence”, the variable “winglet root incidence” is constrained to always be equal to the wingtip incidence. This prevents an AcBuilder user from creating these impossible geometries. This does however remove the freedom of the user to change the “winglet root incidence” which may be required further downstream. If the variable is re- moved from the AcBuilder GUI, it is important that it is copied to the .xml file along with the “wing tip incidence”.

b) Other lifting surfaces

It was possible to create the surfaces for the: horizontal tail, vertical tail, and canard using the same code as for the main wing with only one minor change. Since these other surfaces are only designated 1 kink in AcBuilder, this had to be dealt with. The solution utilized within this code was to set one dummy kink, which overlapped with the wing root, this allowed the calculations to be executed in an identical manner to the main wing. Once the wing surface matrix was generated, this kink was removed to leave just the required cross sections in the matrix. This would prevent any issues, which may be caused by having overlapping sections in the wing.
c) Control surfaces

Once the wing surfaces were generated, it was necessary to generate the control surfaces for visualization purposes. The renderer used in the GUI of the new AcBuilder only required points of the control surface edges, and would draw the lines connecting these points. Since the points where these control surfaces begun were not always on the already generated kinks, interpolation was required to plot these points.

Although the final rendering is not done using the MATLAB surfl and plot commands, it was possible to use this to preview the control surface positions to confirm that within Java they would be plotted in the correct place. Illustrated in Figure A7 is a main wing with: flaps, slats and an aileron. Figure A8 shows the control surfaces plotted in MATLAB for the tail.

The deflection limits of the control surfaces are not taken into account when creating the surface geometry due to the fact that it is only the lines that are plotted in the rendering of the aircraft so are unused in this stage of CEASIOM.

![Figure A7 - Wing plotted in Matlab which included control surfaces](image1)

![Figure A8 - (a) horizontal tail / canard with elevator, (b) vertical tail with rudder](image2)
d) Position

As mentioned in I.C) and II.C), one of the goals of creating a new version of AcBuilder is to create more general aircraft geometries, one of the aspects of doing this is being able to place certain components independently of one another. For this reason, the positioning of lifting surfaces relative to fuselage parameters (apex locale, vertical locale and placement) have been omitted at this stage. This creates the surface of the wing with its own local coordinate system. The positioning of the wing is taken into account in the core of AcBuilder rather than in the surface generation.

Within the independent positioning, it can also be noted that flags are unused in the MATLAB code used to generate the wings. The reason for this being noted in Section 1) b). Additionally, the use of flags prevents the desired independence between different components on the aircraft.

3) Improvements

In order to address some of the problems mentioned in Section 1), solutions are offered in forms of both completed code, as well as a description of the processes within the code.

As already mentioned in Section 2), some of the issues have already been dealt with in the code for the new AcBuilder, because these changes will not harm the compatibility of the program further downstream. Others would require the redefinition of some variables, which are used later on in the CEASIOM package, and this would invalidate some of these later calculations. The changes offered in this section have not been included in the new AcBuilder, but have been written as academic exercises or potential future improvements.

a) Redesign of the aileron

As mentioned in Section 1) c), the aileron is incorrectly defined in the documentation, and the correct definition within AcBuilder is not sensible. Offered here is an alternative description of the aileron, which is more logical and offers a higher degree of accuracy.

The proposed new definition of the aileron position is to have: 0 putting the inboard edge of the aileron against kink2, 1 putting the outside edge of the aileron against the wingtip, and any number between 0 and 1 positioning it fractionally in between kink2 and the wingtip.

This solution is offered in a supplementary MATLAB code named “aileronsrevised”, which, although not implemented in the redesigned AcBuilder, it would be beneficial to add at some point.

It would be very beneficial to the flexibility of the aircraft design if all control surfaces could be plotted by just selecting points on the wing to connect, however, the tornado vortex lattice method within AMB, another module in CEASIOM, loses accuracy in its current version. Currently a new interface for the CFD solver is being developed which would allow for this kind of freedom.
b) Redesign of the winglet

The problems with the winglet are significantly more difficult to fix than the aileron. To understand the reason for this, one has to look in depth at how the surface of the wing is created. As mentioned in Section 2) a), currently (for a horizontal wing), a profile is created and mapped in 2D on the xz-plane at the y coordinate given by the information related to the kink positions. This is acceptable for traditional aircraft, where “horizontal” surfaces (main wing, horizontal tail, canard) are close to horizontal, and the vertical tail is vertical. This approach, however, is lacking when it comes to creating less traditional aircraft, such as v-tails.

For aforementioned reasons, improving the definition of the winglet is a significantly more challenging task than one may first think and would best be incorporated into an entire wing re-definition within CEASIM, which will be discussed.

c) Wing redesign

As mentioned in Section 3) b), the redefinition of the wing is a complicated task and would require the format of the whole of CEASIM to change. Offered in this section is a proposed redefinition of the wing, allowing the user great flexibility. Although currently somewhat crude, a MATLAB program has been written implementing this method.

The basis behind this newly designed wing is to give the user almost complete freedom with regards to the positioning of the wing. The user is allowed to fully specify the angles of the wing, leading to the generalized “lifting surface” which would allow a whole range of new aircraft configurations to be created. It would be particularly beneficial within the new version of AcBuilder being created since it should allow the user to add as many lifting surfaces as they would like. This creation would also give all of the lifting surfaces the same definition, making it easier to generate these lifting surfaces.

As can be seen when comparing Figures A9 and A2, this new wing definition allows vertical winglets to be created without failing and extending to infinity. The two Figures have identical values; the revised definition can cope, whilst the current definition within AcBuilder cannot.
The actual coding method behind creating the wing has not changed significantly. There are some minor changes to the code but it is mostly a redefinition of the variables, which allows the program to generate certain aspects without failing.

Since the existing code has been reading from the .xml file, some the actual inputted variables have remained unchanged in the new definition of the new wing, whereas their definitions have changed.

Figure A 10 - Proposed definition of the wing

Where:
Di dihedral inboard
Dm dihedral midboard
Do dihedral outboard
L1 = (span/2) * Spanwise_kink1
L2 = (span/2) * (Spanwise_kink2 – Spanwise_kink1)
L3 = (span/2) * (Spanwise_tip – Spanwise_kink2)
Figure A11 in particular shows that, in the current AcBuilder, the variables span and area depend on the lifting surfaces’ projection onto the principle planes. The proposed definition depends on the actual geometry of the wing rather than the projection. The definition of the dihedral has not changed, it is still the angle between the leading edge line and the x-y plane.

Although more flexible when it comes to angle sizes, there are some issues that would have to be carefully considered before implementing such a program, for example, the revised definition of the span is not intuitive like it was in the previous version of AcBuilder.

This edited version of the wing definition, not only allows more flexibility and accuracy when it comes to winglet definitions, it would also allow the user to generate very unorthodox wing configurations without failing, as illustrated in Figure A12. Figure A12 was created with the dihedral variables as follows:
- Dihedral inboard: 7°
- Dihedral midboard: 85°
- Dihedral outboard: 7°
- Winglet cant angle: -85°

85° has been chosen so that Figure A12a actually represents something, if 90° had been chosen, the wing would have extended to infinity and the picture would have been wholly meaningless. This problem is completely avoided in the updated version of AcBuilder, it can even create “Z” shaped wings without failing.

![Figure A 12 - (a) Attempted wing with old code, (b) Attempted wing with modified code](image-url)
One of the initial problems would be the definition of the span and the area; for the conventional wings created within AcBuilder, it is simply the projected planform into the xy-plane, which is used. The implication of this is that the area and span are unaffected by the dihedral area. This is illustrated in Figure A13. Despite having the same projected area, there is clearly a much larger surface area in Figure A13b. This is not an error, but more of an inconvenience for an aircraft designer who is trying to create a more complicated configuration.

As can be seen from Figure A10, there is a difference in the definition of the span from the old definition. In the old definition, it was the projected planform length upon the xy-plane. Although more useful for a main wing and a horizontal tail, this would not be beneficial for a more unconventional configuration such as V-tail. The new definition of the span, and also the area is shown in Figure A10.

This new definition of the span allows more complicated geometries to be modelled with relative ease. The variables could be changed somewhat in the .xml file, so that L1, L2 and L3 can be set independently and span can be calculated. Although, it is not beneficial to change the contents of the .xml file unless absolutely necessary, or in a major change of the CEASIOM package.

In a similar way to span, with the new definition, the area becomes the wing area rather than the projected area. This leads to additional accuracy, however care must be taken when it comes to defining the area of the wing, since it differs from the previous definition.

Another issue which has been dealt with is the connection issue between extreme kinks, in the current AcBuilder, as mentioned in Section 2) a), the wing surface is generated by connecting a number of cross sections in different x-z planes. This is problematic in the case of extreme dihedral angles since it creates strange, almost 2D wing thicknesses. There is a reasonably simple solution to this which has been implemented in the proposed program; the airfoil profiles at the kinks are taken to be the average of the dihedral angles on either side of them. For example: the rotation about the x axis of kink1 is given by (dihedral inboard + dihedral outboard)/2. There are however, some subtleties, which cannot be overlooked in this kink generation. One has to analyze the method in which the kinks are joined.
Since the airfoil profile used for the kink in the new wing code is geometrically identical to the unrotated profile in the old wing code, there are geometrical inaccuracies. These would be allowable for lower fidelity analysis, however, for more complicated analyses such as Edge, this issue may have to be addressed more thoroughly.

The issue in question is illustrated in Figure A14, there is a visible amount of thinning of the airfoil leading up to this joint. Before implementing this new wing definition, this would have to be changed in order to increase the accuracy.

![Figure A14 - Issue with new airfoil spline definition](image)

Despite this issue, it is still a large improvement over the method previously used where the geometry shown in Figure A14 is not even possible.

**Summary of advantages of new wing definition:**
- More flexibility allows a greater variety of wing shapes, this is beneficial for creating more unconventional aircraft.
- More robust when it comes to creating vertical sections on a horizontal wing.
- More suitable for any future developments which could involve unlimited kinks in an aircraft wing.

**Summary of issues and disadvantages of new wing definition:**
- Current variables in .xml file are not suited to these definitions since the names are misleading; careful consideration and redefinition would have to be undertaken in order to create simple, understandable variables for the user.
- The new program was written mainly as an academic exercise and should be analyzed and modified prior to actual incorporation into the AcBuilder package.

4) **Inputs used**

   a) **Wing area**

     Projected wing area onto x-y plane.
Span
Length of wing in y direction. (b in Figure A15)

spanwise kink1
Kink1 position as a fraction of the wingspan (s1 in Figure A15).

spanwise kink2
Kink2 position as a fraction of the wingspan (s2 in Figure A15).

LE sweep inboard
Leading edge sweep between root and kink1 (Λi in Figure A15).

LE sweep midboard
Leading edge sweep between kink1 and kink2 (Λm in Figure A15).

LE sweep outboard
Leading edge sweep between kink2 and tip (Λo in Figure A15).
taper kink1
Taper ratio of kink1 (t k1 in Figure A15).

taper kink2
Taper ratio of kink2 (t k2 in Figure A15).

taper tip
Taper ratio of tip (t t in Figure A15).

root incidence
Incidence at root.

kink1 incidence
Incidence at kink1.

kink2 incidence
Incidence at kink2.

tip incidence
Incidence at tip.

Figure A 16 - Dihedral definition

dihedral inboard
Dihedral angle between root and kink1 (see Figure A16).

dihedral midboard
Dihedral angle between kink1 and kink2 (see Figure A16).

dihedral outboard
Dihedral angle between kink2 and tip (see Figure A16).

airfoilRoot
Airfoil at root.

airfoilKink1
Airfoil at kink1.
airfoilKink2
   Airfoil at kink2.

airfoilTip
   Airfoil at tip.

Winglet

Present
   1 if winglet exists, 0 if not.

Span
   Length of winglet in y direction.

taper ratio
   Taper ratio between wing tip and winglet tip.

LE sweep
   Leading edge sweep of the winglet

Cant angle Angle
   between the winglet and the x-y plane.

root incidence
   omitted for reasons outlined in Section 1) a)

tip incidence
   Incidence of the winglet tip.

Flap

Present
   1 if flap exists, 0 if not.

root chord
   Fraction of the chord the flap extends to (from trailing edge) at the root.

kink1 chord
   Fraction of the chord the flap extends to (from trailing edge) at kink1.

kink2 chord
   Fraction of the chord the flap extends to (from trailing edge) at kink2.

Aileron

Present
   1 if aileron exists, 0 if not.
Chord
Fraction of the chord the aileron extends to (from trailing edge).

Span
Fraction of the spanwise difference between kink2 and tip.

Position
Flag representing the position of the aileron.

Limit deflection
omitted for reasons discussed in Section 2) c)

Slat

Present
1 if slat exists, 0 if not.

Chord
Fraction of the chord the slat extends to (from leading edge).

Root position
Spanwise position of the slat’s inner end as a fraction of the distance between root and kink1.

Tip position
Spanwise position of the slat’s outer end as a fraction of the distance between kink2 and tip.

Placement and apex locale
omitted for reasons discussed in Section 2) d)

b) Horizontal tail, canard and vertical tail

Area
Projected planform area on: x-y plane for horizontal tail and canard, x-z plane for vertical tail.

Span
Span in y direction for horizontal tail and canard, z direction for vertical tail.

Spanwise kink
Position of the kink as a fraction of the span.

taper kink
Taper ratio of the kink.
**taper tip**
Taper ratio of the tip.

**Horizontal tail and canard only -root incidence**
Wing incidence at root.

**Horizontal tail and canard only -kink1 incidence**
Wing incidence at kink.

**Horizontal tail and canard only -tip incidence**
Wing incidence at tip.

**LE sweep inboard**
Leading edge sweep between root and kink.

**LE sweep outboard**
Leading edge sweep between kink and tip.

**Horizontal tail and canard only -dihedral inboard**
Dihedral angle between root and kink.

**Horizontal tail and canard only -dihedral outboard**
Dihedral angle between kink and tip.

**airfoilRoot**
Airfoil at root.

**airfoilKink**
Airfoil at kink.

**airfoilTip**
Airfoil at tip.

**Horizontal Tail only-empennage layout**
Ommitted for reasons discussed in Section 1) b)

**Horizontal Tail and Canard only-elevator**

**Present**
1 if elevator exists, 0 if not.

**Chord**
Fraction of the chord the elevator extends to (from trailing edge).

**Span**
Fraction of the span that the elevator extends.
limit deflection
omitted for reasons discussed in Section 2) c)

Vertical Tail only-rudder

Present
1 if rudder exists, 0 if not.

Chord
Fraction of the chord the rudder extends to (from trailing edge).

Span
Fraction of the span that the rudder extends.

limit deflection
omitted for reasons discussed in Section 2) c)

limit tailplane deflection
omitted for reasons discussed in Section 2) c)

vertical locale and apex locale
omitted for reasons discussed in Section 2) d)
Appendix B: The AcBuilder: Geometry Definition for CEASIOM, II: Bodies

1) Issues with the previous AcBuilder

a) Fuselage

A mistake related to the distortion factors could be found in the previous AcBuilder. Instead of solely reshaping the section, it was also shifted upwards or downwards, depending on the new distortion coefficient, as shown in the following figures.

In B1, left, the aft distortion coefficient is 0.42. In B1, right, the new shape of the after centre part is presented along with the new cross section of the fuselage, and only the distortion coefficient is changed to 0.89. As it can be seen, the cross section is changing only slightly, but the after centre and tail parts of the fuselage are completely shifted upward in this case.

Furthermore, the documentation on the coefficients \textit{a1 and b1} was switched around.

b) Fairing

In the previous AcBuilder, the fairing was defined as part of the wing, and its position and definition were entirely \textit{dependent on some of the wing’s parameters}, only the length and height could be changed, hence its definition was \textit{extremely limited} (Figure B2).

However, as part of the new AcBuilder, each element has to be independent and be able to be placed in any order, hence the need for a new definition of the fairing arises.
c) Engine

In AcBuilder, the engines are only considered as geometrical entities; however, in the previous AcBuilder, a lot of parameters unrelated to the geometry could be set. These parameters, such as the max thrust or bypass ratio for examples, are not relevant to the geometry, hence they should not be present and set in AcBuilder, and they should be introduced in other relevant softwares of CEASIOM but not in AcBuilder. However, to avoid forcing other softwares of the CEASIOM package to update, a separate section in which these parameters are set can be thought of.

For the geometrical representation, only the maximum diameter, the fineness ratio and the toe and pitch angles are necessary, along with its absolute XYZ coordinates.

In the previous AcBuilder, the XYZ position of the engine was defined by 5 different parameters, which all have been abandoned for the new AcBuilder. The “nacelle body type” would move the engine forward or backward compared to the wing, the “layout and config” parameter was supposedly set to decide the engine’s position relatively to the wing, but as noticed, it was possible to override this parameter by setting it to „below the wing“ and finally place the engine above the wing, hence this type of inconsistency needs to be erased.

![Figure B 3 - Engine inconsistency, location of Engines 1 set as "on wing"](image)

d) Fuel

It was noticed that the fuel parameters were organized in 2 categories: Fuel tanks definition and Wingbox definition. In Wingbox definition, only the fuel section positions and length could be defined at the root, kink1, kink2 and tip. In Fuel tanks definition, a mixture of parameters for the fuel was presented. Some parameters were defining the geometry whilst some parameters were defining fuel options and weight.
To simplify the use of AcBuilder, it seems useful to reorganize these parameters into different categories: for example, one category for the geometry definition of the wings fuel tanks, and another one for the fuel options and weight.

Furthermore, several parameters did not work. *Centre_tank_portion_used* was supposed to define the percentage of the span-wise distance from the plane of symmetry ($y=0$) to the fuselage wing juncture, which is used as centre fuel tank width in the wing. However, any value could be entered but no change could be observed. *Wing_fuel_tank_cutout_pot* was not functioning either. As shown in Figure B4 and Figure B5, it is not apparent in both cases. Concerning the *cutout*, it can be added that no location or length could be defined.

![Figure B 4 - First example of malfunction of fuel parameters](image1)

![Figure B 5 - Second example of malfunction of fuel parameters](image2)
Other parameters concerning the fuel options and weight may have been working or not, they were not investigated as only the geometry is considered here.

A last point that can be mentioned is the too *simplistic 2D visualization* of the fuel tanks, not allowing to get an estimate of the volume, or to visualize correctly the place taken by the tanks in the wing.

2) Changes implemented

   a) Fuselage

   ![Illustration of some fuselage parameters](image)

   **Figure B 6 - Illustration of some fuselage parameters**

   The new code was based on a code developed by Politecnico di Milano present in NeoCASS under the name of „Geo_Fus_Shape.m“. However, a few mistakes could be found as it can be shown on the figure below.
The values to represent a B747 were used as input. Figure B7 shows the output from the original code. It can be seen that the after centre part and the tail are shifted downward compared to the front of the fuselage. The front of the nose, in this case, is placed at the origin of the coordinates system.

Because the after centre part is constant along the x axis, it is natural to have the set of coordinate in AcBuilder defined as:

The origin of the plane x0y is placed at the front of the nose, hence, for the nose tip, x and y are always equal to 0. The plane Ox axis is coincident with the aft-fuselage waterline. This coordinate system is shown on Figure B6.

The distortion coefficient problem from the existing AcBuilder and the code developed by Politecnico di Milano has been corrected. The 2 following figures show the output obtained with the new code for AcBuilder from the B747 inputs (dist coef at 0.42 and 0.89 respectively).

b) Fairing

It is now possible to create different fairings with the available parameters.

In this new version of AcBuilder, the fairing is defined as an element on its own, hence it can be placed even if no wing has been defined, it can be placed anywhere without any dependency on other element.

The fairing is defined as segments of 3 different super ellipsoids. The most general equation for a super ellipsoid is:
\[ \left(\frac{x - x_0}{a^{nx}}\right)^n + \left(\frac{y - y_0}{b^n}\right)^n + \left(\frac{z - z_0}{c^n}\right)^n = 1 \]

**Equation 1**

Figure is an example of the new possibilities offered by this fairing. It is now possible to create a canopy for instance.

![Figure B 9 - Example of canopy using the new fairing definition](image)

As examples, the following figures show different new shapes with the fairing. In all cases, the fore, mid and aft lengths are respectively equal to 1, 0.5 and 2 m, the width and height are respectively 1 and 0.4 m. Only the parameters \( n \) and \( nx \) are changed to obtain the different examples here.

![Figure B 10 - Fairing, \( n=3, nx=3 \)](image) ![Figure B 11 - Fairing, \( n=1, nx=1 \), sharp polyedron](image)

![Figure B 12 – Fairing, \( n=3, nx=1 \)](image) ![Figure B 13 – Fairing, \( n=2, nx=2 \), ellipsoid](image)
Because of the many different shapes that the fairing can have, one of the challenges was to be able to define it properly without having to generate too many points. To do so, a way of dispatching the points depending on the coefficients used needed to be implemented in order to make this feature automatic. 2 set of formulas were tested:

\[ r(\theta) = \left( \frac{\cos(\theta)^n + \sin(\theta)^n}{\frac{w}{2} + \frac{h}{2}} \right)^{-\frac{1}{n}} \]

\[ y(\theta) = r(\theta) \cdot \cos(\theta) \]

\[ z(\theta) = r(\theta) \cdot \sin(\theta) \]

**Set of Equations 1: distribution independent of the n coefficient**

\[ y(\theta) = |\cos(\theta)|^{\frac{2}{n}} \cdot \frac{w}{2} \cdot sgn(\cos(\theta)) \]

\[ z(\theta) = |\sin(\theta)|^{\frac{2}{n}} \cdot \frac{h}{2} \cdot sgn(\sin(\theta)) \]

**Set of Equations 2: distribution dependent of the n coefficient**

The set of equations used in the code is the 2nd set. It is important to notice that these examples of equations are correct for the middle section of the fairing; for the fore and aft sections, the effect of the first term of Equation 1 needs to be taken into account. For a more complete description of the fairing’s calculations, please refer to section 3) a) ii).
As it can be seen on Figure B6, the points defining the sections depend on the inputs used since the point distribution dependent on n was preferred. It was possible to define the fairing with its points separated by the same angle; however, as shown in the following figures, the fairing’s corners are better defined in the case of “n” dependent distribution, while the walls remain relatively constant. The fairing is generally better defined with a point distribution dependent on n.

Figure B 16 - Reference figure for points distribution comparison, n=6, nx=2

Figure B 17 - Points distribution dependent on n

Figure B 18 - Points distribution with constant angle

Figure B 19 - reference figure for points distribution comparison, n=2, nx=2

Figure B 20 - points distribution dependent on n

Figure B 21 - Points distribution with constant angle
c) **Engine**

As presented in section 1) c), the XYZ coordinates were defined by a total of 5 parameters and inconsistencies could be found. Furthermore, it was defined from a local point of view (X and Z were related to the fuselage whilst Y to the wing span). This is contradictory to the new library aspect and independence between each entity of the new AcBuilder. To place the engine, only the XYZ coordinates are necessary.

Only the parameters related to the geometry were kept, hence only the diameter, fineness ratio and the toe and pitch angles are necessary.

Some further modifications on the documentation were realized such as the calculation of the exact values of the coefficients necessary or the generation of more values of the parameter “t” (see following section), in order to define and represent the engine geometry correctly.

![Engine example](image)

**Figure B 22 - Engine example**

---

d) **Fuel**

Firstly, the issues found on the previous AcBuilder were corrected. The centre tank portion parameter is now taken into account. The cutout option has been corrected as well, adding 2 parameters to define it, its position relative to the wing and its length. Thanks to this correction, the user can now define the cutout entirely.
For a better visualization, the fuel was changed from a 2D representation to a 3D representation thus allowing to give the user the exact volume represented by the tanks.
Further suggested improvements to the fuel tanks can be made easily: The fuel tanks cannot be tilted, hence it cannot follow the different incidences of the wing sections, and they remain horizontal. It would be interesting to be able to set an incidence angle for the tanks to allow it to fit better in the wings. This could be implemented by adding a fuel incidence. Another improvement can be to allow the user to visualize cross sections of the wing and fuel tanks along the chord. Because in AcBuilder, no thickness is considered, this would allow to see how much space is left between the wing and fuel tanks, also, since the wing is defined by its outside surface and the tanks are defined by their inside surface, it is necessary to help the user take into account the thicknesses of the different elements into play. In the figure below, an example of the possible output is presented.

![Cross section of wing and fuel tank along chord](image)

**Figure B 25 - Example of possible cross section visualization along the chordwise direction**

3) **Documentation**

a) **Elements**

i) **Fuselage**

The fuselage is taken to be a three-segment body: the nose, the centre and the tail.

**Centre segment**

The centre segment can be separated into 2 sub segments. The aft centre segment is a constant segment along x for which the radius will be defined in the next paragraphs. The forward centre segment is defined by its first section (common with the nose) and its last section (common with the aft centre segment), it is then completed by a ratio between the 2 defining sections.
The total length of the centre segment can easily be obtained from the total length of the fuselage and the nose and tail lengths.

To delimit the forward segment from the aft segment, the fraction parameter (Fraction_fore) is used. It is the forward centre-fuselage length divided by centre-fuselage length.

Furthermore, the shift parameter allows the nose (hence changing the definition of the forward segment) to be shifted upward or downward compared to the aft segment waterline, line of reference. It is the vertical shift (in meters) of the forward centre fuselage water line compared to the aft fuselage water line (see Figure B6). It can be negative, which means that the forward fuselage would be lower than the after fuselage.

The cross-sections of the fuselage centre-segment consist of upper and lower lobes with a stipulation of symmetry about the x-z plane imposed, as shown in Figure B26. One can assume a basic circular geometry can be distorted into an ovoid shape by displacing the original origin by some proportion (henceforth designated as the distortion coefficient, or, ) of the maximum cross-section height (or ..._X_sect Vertical diameter), i.e. a circular geometry distortion coefficient would be , and all others would fall between, 0< ξ<1

Since an association can be established between the radius and angular sweep, the Fourier Series Expansion can satisfactorily represent this functional relationship as a sum of sine and cosine terms. Due to stipulation of symmetry about the x-z plane and placing an emphasis on compactness, a sufficiently accurate model of any continuous fuselage cross-section geometry can be obtained using an abridged form (Equation 2).
The coefficients can be evaluated using a numerical integration procedure like Simpson’s rule, or alternatively evaluated with non-linear regression techniques such as the Levenberg-Marquardt method.

**Equation 2**

\[ r(\phi) = a_0 + a_1 \sin(\phi) + b_1 \cos(2\phi) \]

Nose and Tail

Figure B27 shows the nose of an aircraft. Each section is partitioned into two segments delineated by a sweep line (down-sweep denoted by \( \phi \)) originating from the body apex or extremity to the fuselage centre-section vertical midpoint (at Fuselage Reference Plane or FRP). A supplementary parameter designated as the shield-sweep, \( \omega \), is also introduced and is essentially a measure of the angle of the body frontal face in the XZ plane. The convention discussed for the forward fuselage example (nose) is equally applicable for the aft fuselage body (tail) as well. Instead of down-sweep, generally an up-sweep would be considered, and the shield-sweep would be replaced by tail-sweep of the body lower portion, both measured anti-clockwise with respect to the FRP (Fuselage Reference Plane).

To give a more practical explanation of the calculations: the nose common section with the centre segment of the fuselage is used as reference, while the values calculated from Equation 6 will be used to re-evaluate the nose radius for all x positions.

*Figure B 27 - Parameters defining the nose of an aircraft*
Recalling the objective is to describe upper and lower segments using very simple analytical models whilst retaining a suitable level of accuracy, a review of plane curves revealed an adequate representation of geometry in side view could be obtained by using the algebraic model.

\[ z = \alpha \cdot x^\beta \]

**Equation 3**

where \( z \) denotes the vertical axis or water-line and \( x \) represents the longitudinal or fuselage station. The \( \beta \) coefficient is derived empirically and based on correlation to known aircraft fore and aft bodies, investigations have shown that the potential coefficient is a trigonometric function of \( \varphi \) and \( \omega \).

\[ \beta = 0.54 + 0.1 \cdot \tan (\omega - \varphi) \]

**Equation 4**

The \( \alpha \) coefficient is simply found by equating Equation 3 with the fuselage vertical radius corresponding to fuselage station \([ d_v \] from the body apex. The parameter \( \varepsilon \) is used to define the body length to diameter ratio.

\[ \alpha = \frac{d_v}{2} \cdot \frac{1}{(\varepsilon \cdot d_v)^\beta} = \frac{d_v^{(1-\beta)}}{2\varepsilon^\beta} \]

**Equation 5**

Invoking a geometric constraint of \( z(x = \varepsilon d_v) = \frac{d_v}{2} \pm \varepsilon \cdot d_v \cdot \tan (\varphi) \), where the axis convention dictates to denote observance of the upper body and \(+[ d_v]\tan(\varphi)\) the lower. A geometric description of the each upper and lower body segment as a function of body station is therefore given by

\[
\begin{align*}
  z(x) &= \begin{cases} 
    + \frac{d_v^{(1-\beta)}}{2} \left( \frac{x}{\varepsilon} \right)^\beta - (\varepsilon d_v - x) \cdot \tan(\varphi) & \text{for all } x \text{ above body apex} \\
    - \frac{d_v^{(1-\beta)}}{2} \left( \frac{x}{\varepsilon} \right)^\beta - (\varepsilon d_v - x) \cdot \tan(\varphi) & \text{for all } x \text{ above body apex}
  \end{cases}
\end{align*}
\]

**Equation 6**

To re-iterate, the ordinate values generated by Equation 6 adhere to the axis convention defined by the body down-sweep or up-sweep depending on whether the forward or aft fuselage are being considered respectively.
ii) Fairing

There is a propensity in the aircraft industry to incorporate conformal fuel tanks, i.e. more amorphous-looking tanks defined by tracing the wing-fuselage fairing geometry rather than installation of a series of box-like cells. For this reason, a relatively accurate geometric description of the fairing volume is required.

The wing-fuselage fairing is defined as a three-segment body: a fore super-ellipsoid, a super ellipse midsection and an aft super-ellipsoid. The most general equation for a super ellipsoid is:

\[
\frac{(x-x_0)^{nx}}{a^{nx}} + \frac{(y-y_0)^n}{b^n} + \frac{(z-z_0)^n}{c^n} = 1
\]

**Equation 7**

The super-ellipsoids generated in the fairing geometry have a common y and z exponent, henceforth simply designated as “n”. Besides, both the fore and aft super ellipsoids share the “nx” and “n” exponents.

To create the super ellipse midsection, the following equation is used:

\[
\left(\frac{y}{w/2}\right)^n + \left(\frac{z}{h/2}\right)^n = 1
\]

**Equation 8**

To create the fore and aft super ellipsoid, the contribution of the x component is evaluated for the necessary fore and aft length, the y and z coordinates are then calculated using the following formula:

\[
\left(\frac{y}{w/2}\right)^n + \left(\frac{z}{h/2}\right)^n = 1 - \left(\frac{x}{l}\right)^{nx}
\]

**Equation 9**

iii) Engine

To generate the geometry of the engine, the following equations were used:
where \( t = \frac{i}{n} \) for \( i = 0, 1, 2, \ldots, n \), \( l_{eng} \) and \( d_{eng} \) are the engine length and maximum diameter respectively. The scaling factors, \( \zeta_{lgt} \) and \( \zeta_{dia} \), are derived based on the properties inherent to each trigonometric function; their respective values are:

\[
\zeta_{lgt} = e^{-\frac{\pi}{2}}
\]

\[
\zeta_{dia} = \frac{1}{2} \left( 1 + \frac{e^{(\pi/4)\sqrt{2}}}{2} \right)
\]

The parameter, \( t \), can be thought of as a spacing variable and its purpose is to generate each \( x \) and \( z \) coordinates according to the number of segments deemed satisfactory. The entire engine is then generated by revolution of the curve created about the \( x \) axis (see Figure B28).

Figure B 28 - General representation of swept surface resulting from revolution of curve AB about the \( x \)-axis
The toe-in and pitch rotations are then considered individually as 2D rotations. The point around which rotation occurs is on the centreline in the front plane of the nacelle (centre symmetry of the front section).

iv) Fuel

The fuel tanks generated are considered as a series of square frustra attached (or not always if cutout is present) to one another. Because no twist is considered between each defining section, it will be seen further in this section that calculating the exact volume is possible.

The fuel tanks generated are placed in the wings, hence a lot of parameters from the wing are used to define them. The semi wingspan, spanwise position of the kink1, kink2, the chord of each section, the LE sweep angles and the dihedral angles; all these parameters are necessary in order to define the fuel tanks in the wing (note: it does not take into account the incidence of each sections, the fuel tanks are always horizontal). Additionally, more parameters are necessary to define them, such as the centre limit, the outboard limit, the presence or not of a cutout with its length and position, but also the section location and length chordwise as well as height at each defining section of the wing (root, kink1, kink2 and tip).

The fuel tanks are firstly calculated at the 4 sections of the wing (root, kink1, kink2, tip).

Following this step, the parameters „centre limit” and „outboard limit” are taken into account. They are used to calculate the minimum and maximum y position of the fuel tanks respectively. Once they are set and that they have been located between 2 sections (root, kink1, kink2 or tip), a linear interpolation is applied to calculate the new X and Z values of the minimum and maximum sections.

At this point, the “full” fuel tank is defined (see Figure B29). The next step is to take into account the cutout if present.

Figure B 29 - Top view of the wing fuel tank without cutout (left) and with cutout (right)
If the cutout option is set to 0, then no cutout needs to be calculated, hence the output can be sent out as the “full” fuel tank. If the cutout is set to 1, it is necessary to create 2 sets of XYZ matrices because the output is now 2 distinctive fuel tanks.

With the cutout position and length, its minimum and maximum y positions can be computed, then, the same calculation as in the step above is made. The minimum Y value of tank2 will be given by the maximum cutout position whilst the maximum Y value of tank1 will be given by the minimum cutout position. Once these 2 Y values are located between 2 sections (root, kink1, kink2 or tip), a linear interpolation is applied to calculate the new X and Z values of the minimum and maximum sections for tank2 and tank1 respectively.

Because the tanks are a series of square frustra, it is possible to calculate the exact volume. It is needed to notice that this is possible because the incidences of the different sections are not considered; hence no twist can be present on the fuel tanks.

To calculate the exact volume, the following formula is used:

\[
V = \frac{l}{3}(A_1 + A_2 + \sqrt{A_1A_2})
\]

Equation 12

\(A_1\) and \(A_2\) are the areas of the cross sections along the chord, \(l\) is the length of the fuel tank in the y direction. Equation 12 may be applied as many times as needed to calculate the total volume as there is in general more than one frustra to consider.

b) Inputs

This is the exhaustive list of the parameters used to generate the geometry of the elements mentioned (Fuselage, Fairing, Engine, Fuel) to create an aircraft in AcBuilder. A quick description for each of them is also given.

i) Fuselage

**Omega_nose**

Angle (in degrees) of the body frontal face in the x-z plane. It can be negative.

**Phi_nose**

Down-sweep angle (in degrees) originating from the body apex or extremity to the fuselage centre-section vertical midpoint. It can be negative. Notice that negative values indicate up-sweep, and “omega_nose” should be negative too.
**Epsilon_nose**
Nose length to diameter ratio.

**Nose_length**
Nose length, in metres.

**Omega_tail**
Angle (in degrees) of the body rear face in the x-z plane. It can be negative.

**Phi_tail**
Up-sweep angle (in degrees) originating from the body rear extremity to the fuselage centre-section vertical midpoint. It can be negative. Notice that negative values indicate down-sweep, and „omega_tail” should be negative too.

**Epsilon_tail**
Non-dimensional parameter used to define the aft fuselage length to diameter ratio.

**Tail_length**
Tail length, in metres.

**Forefuse_X_sect_vertical_diameter**
Vertical diameter (in metres) of the first cross-section of the centre fuselage.

**Forefuse_Xs_distortion_coefficient**
Distortion coefficient of the first cross-section of the centre fuselage.

**A0_fore**
Coefficient evaluated to calculate the radius of the forward centre section.

**A1_fore**
Coefficient evaluated to calculate the radius of the forward centre section.

**B1_fore**
Coefficient evaluated to calculate the radius of the forward centre section.

**Aftfuse_X_sect_vertical_diameter**
Vertical diameter (in metres) of the last cross-section of the centre fuselage.

**Aftfuse_Xs_distortion_coefficient**
Distortion coefficient of the last cross-section of the centre fuselage.

**A0_aft**
Coefficient evaluated to calculate the radius of the aft centre section.

**A1_aft**
Coefficient evaluated to calculate the radius of the aft centre section.
B1_aft
Coefficient evaluated to calculate the radius of the aft centre section.

Fraction_fore
Forward centre-fuselage length divided by centre-fuselage length.

Shift_fore
Vertical shift (in metres) of the forward centre fuselage water line compared to the aft fuselage water line. It can be negative, which means that the forward fuselage is lower than the after fuselage.

Total_fuselage_length
Total fuselage length, in metres.

ii) Fairing

Fore_length
Fore super-ellipsoid, x-semi axis.

Aft_length
Aft super-ellipsoid, x-semi axis.

Mid_length
Super ellipse midsection longitude.

Width
Fore and aft super-ellipsoid, y-axis.

Height
Fore and aft super-ellipsoid, z-axis.

n_exp
y and z exponent in the super-ellipsoid equation.

nx_exp
x exponent in the super-ellipsoid equation.

iii) Engine

d_max
Nacelle maximum diameter, in metres. It will be automatically estimated if set to zero.

fineness_ratio
Nacelle length divided by nacelle maximum diameter.
pitch
Nacelle pitch angle, in degrees. If positive, the nacelle is pointing upwards. The rotation point is the same as in the “toe_in” parameter

toe_in
Nacelle tow-in angle, in degrees. A negative value can be provided for tow-out angle (nacelle pointing outwards on starboard side). The point around which rotation occurs is on the centreline in the front plane of the nacelle (centre symmetry of the front section).

iv) Fuel

box_ea_loc_root
Wing box elastic axis chord-wise position at the root divided by local wing chord.

box_ea_loc_kink1
Wing box elastic axis chord-wise position at the first kink divided by local wing chord.

box_ea_loc_kink2
Wing box elastic axis chord-wise position at the second kink divided by local wing chord.

box_ea_loc_tip
Wing box elastic axis chord-wise position at the tip divided by local wing chord.

box_semispan_root
Wing box semi-span at the root, divided by local wing chord.

box_semispan_kink1
Wing box semi-span at the first kink, divided by local wing chord.

box_semispan_kink2
Wing box semi-span at the second kink, divided by local wing chord.

box_height_root
Wing box height at the root section (in metres).

box_height_kink1
Wing box height at the kink1 section (in metres).

box_height_kink2
Wing box height at the kink2 section (in metres).

box_height_tip
Wing box height at the tip section (in metres).

cutout_pos_span
Cutout centre semi-span position, divided by the halfspan
**Wing_fuel_tank_cutout_opt**  
This parameter specifies the presence or absence of a structural cut-out in the wing tank volume due to the presence of a power-plant. 0 = continuous wing fuel tank 1 = discontinuous wing fuel tank

**cutout_length**  
Length of the cutout (in metres).

**Outboard_fuel_tank2_span**  
Wing tank maximum span divided by wing semi span.

**Centre_tank1_portion_used**  
Percentage of the span-wise distance from the plane of symmetry (y=0) to the fuselage-wing juncture which is used as centre fuel tank width in Wing 1. If equal to 100, the whole width of the fuselage-wing juncture is used as a centre tank.

**halfspan**  
Halfspan is equal to half of the total wing span.

**chord_root**  
Chord of the root section.

**chord_kink1**  
Chord of the kink1 section.

**chord_kink2**  
Chord of the kink2 section.

**chord_tip**  
Chord of the tip section.

**LE_sweep_inboard**  
Sweep angle (in degrees) of the leading edge line in inboard.

**LE_sweep_midboard**  
Sweep angle (in degrees) of the leading edge line in midboard.

**LE_sweep_outboard**  
Sweep angle (in degrees) of the leading edge line in outboard.

**dihedral_inboard**  
Dihedral angle (in degrees) of inboard.

**dihedral_midboard**  
Dihedral angle (in degrees) of midboard.

**dihedral_outboard**  
Dihedral angle (in degrees) of outboard.
Appendix C: Program explanations

1) Elements creation

The program being too long, the element fuselage will be presented for example. The same structure is used for all the elements.

```matlab
function [] = load_fuselage(structure)

The function used to create the fuselage is called load_fuselage. Its parameter structure contains the XML file information thanks to the function xml_load.

```matlab
fuselage_element_structure = structure;
```

The XML structure is loaded in a local variable fuselage_element_structure.

```matlab
element_type = 0;
counter(1) = counter(1) + 1;
counter(2) = counter(2) + 1;
index(counter(1)) = counter(2);
selected(counter(1)) = 1;
```

A number is attributed to each element type (0 for the fuselage). A counter is used to know how many elements exist in the aircraft design and a specific counter for fuselage element is also used. The function selected permits to display or not the element.

```matlab
% give a name to this component
if counter(1) == 1
    component_name = 'Fuselage';
    char_name = char(component_name);
    elements_gui.(component_name) = fuselage_element_structure.Fuselage;
```

At the beginning, two cases are distinguished: if the element is the first to be created or not. In the following explanations, just the first case will be presented because it is really similar, this differentiation is only used for avoiding a problem concerning the names attribution.

The structure elements_gui will contain all the information like the XML file but in Matlab. The program creates dynamically a field for this structure with the name of the component and fills in this field with the corresponding part (here the fuselage part) of the structure loaded (fuselage_element_structure) from the XML file.

```matlab
% for weights and balance
elements_gui.cabin = fuselage_element_structure.cabin;
elements_gui.Baggage = fuselage_element_structure.Baggage;
elements_gui.misellaneous = fuselage_element_structure.misellaneous;
```

Some fields are added in the same way than previously, they will be used later for the weights and balances computation.
The design information from the fuselage is stocked in a table (`fuselage_element_data`).

The program adds the name of the new component in a list box (this list box corresponds to the display of the elements existing in the aircraft design (on the left window Figure 14)).

Those two commands add the list names in the list box and select the line corresponding to the last element created.

A table name is created dynamically in order to access this table easily later in the program. Here the name will be `element_1` (because it’s the first element created)
This code represents the creation of this table with all the specifications needed (the position, the title, the number of columns, the data, the functions used when the user selects the table (select_fuselage_parameter) or when he changes the value of a parameter (table_fuselage_element)).

```matlab
% create handle for this table
h = findobj('tag', table name);
handles.(table_name) = h;
```

This variable named handles lists all the tables name in order to have access to these tables later.

```matlab
% select the component
listbox();
```

This function permits to select the right table corresponding to the name selected in the list.

```matlab
% Matlab calculation - return matrix coordonates
if counter(1) == 1
    name = component_name;
else
    name = component_name{counter(1)};
end
```

At this point, the Matlab calculation begins. First the program loads the name of the component (always a difference if it’s the first element or not)

```matlab
% translations
if counter(1) == 1
    x_translation = elements_gui.(component_name).x;
y_translation = elements_gui.(component_name).y;
z_translation = elements_gui.(component_name).z;
else
    x_translation = elements_gui.(component_name{counter(1)}).x;
y_translation = elements_gui.(component_name{counter(1)}).y;
z_translation = elements_gui.(component_name{counter(1)}).z;
end
```

The coordinates system is managed separately from the other parameters.

```matlab
% call function matrix_fuselage
[X_fuselage, Y_fuselage, Z_fuselage, N_fuselage, X_fus_tech, Z_fus_tech] = ...
    matrix_fuselage(elements_gui.name, x_translation, y_translation, z_translation);
```

The function matrix_fuselage is called. It returns the coordinates matrices $X_{fuselage}$, $Y_{fuselage}$, $Z_{fuselage}$, the normal matrices $N_{fuselage}$ (useful for the calculation of the brightness in Java) and $X_{fus\ tech}$ / $Z_{fus\ tech}$ (only used for a fuselage element for the technology module). This function uses the equations described in the appendixes B or C depending on what type of element is created.
The coordinates matrices for the fuselage are now stocked in some big matrices \((X, Y, Z\) and \(N)\) where all the coordinates from all the elements created will be present at the end.

This function \(\text{Draw}\) will transfer the matrices from Matlab into Java and activate the Java part for drawing the elements in the Java window.

Those last commands adapt the width of the table columns with the size of the window.

\(\textbf{2)}\quad \textbf{Function \textit{Draw}}\)

This function is used to draw the elements.

The program loads some information like the number of the index in the list box element, the name of the element and if the element type is a wing, a horizontal tail or a vertical tail.

If the element is the first one, the Java part is called by pushing a virtual button at the end of the function \(\text{Draw}\). The variables from Java are stocked in a Matlab variable \((ACB\_test)\).
The function `loadData()` transfers the data loaded previously into Java (elements number, element name, element type, element index)

```matlab
% applicable or not the symmetry to the engine
if counter(8) ~= 0
    for i=1:counter(8)
        engine_string = strcat('Engines', num2str(counter(8)));
        list_index_engine = get(handles.listbox, 'String');
        list_index_engine = cellstr(list_index_engine);
        for j=1:counter(1)
            if strcmp(char(list_index_engine(j)), engine_string) == 1
                table_name = strcat('element', num2str(j));
                engine_element_data = get(handles.(table_name), 'Data');
            end
        end
        symmetry = str2double(mat2str(cell2mat(engine_element_data(4,3))));
        setSymmetry(symmetry);
    end
end
```

The parameter symmetry for the engine is managed separately from the others.

```matlab
% Fill in the matrix in Java for the geometric elements
size_X = size(X);
for i=1:size_X(2)
    for j=1:size_X(1)
        ACB_test.setValue(list_index-1, X(list_index,i,j), Y(list_index,i,j), Z(list_index,i,j),...) 
        X(list_index, i, j), Y(list_index, i, j), Z(list_index, i, j), X(list_index, i-1, j-1); 
    end
end

% Fill in the matrix in Java for the fuel boxes
if (counter(1) ~= 0 ||isempty(wing_test)) || counter(2) ~= 0
    size_X_mod = size(X_mod);
    for i=1:size_X_mod(2)
        for j=1:size_X_mod(1)
            ACB_test.setValueMod(list_index-1, X_mod(list_index,i,j), Y_mod(list_index,i,j), Z_mod(list_index,i,j), ...) 
            X_mod(list_index, i, j), Y_mod(list_index, i, j), Z_mod(list_index, i, j), X_mod(list_index, i-1, j-1); 
        end
    end
end
```

The coordinates matrices are transferred to Java thanks to the Java functions `setValue` (elements geometry) or `setValueMod` (used for the fuel tanks geometry).

```matlab
% Fill in the matrix in Java for the control surfaces
if {counter(3) ~= 0 || counter(4) ~= 0 || counter(5) ~= 0} && ... 
    ~isempty(wing_test) || ~isempty(horizontal_tail_test) || ~isempty(vertical_tail_test))
    for k=1:2
        for j=1:2
            ACB_test.setValueControlSurface(list_index-1, k-1, X_Control_surface(list_index,k,j), ...) 
            X_Control_surface(list_index, k, j), Z_Control_surface(list_index, k, j), j-1); 
        end
        elseif k == 3 || k == 4
            for j=1:4
                ACB_test.setValueControlSurface(list_index-1, k-1, X_Control_surface(list_index,k,j), ...) 
                X_Control_surface(list_index, k, j), Z_Control_surface(list_index, k, j), j-1); 
            end
            else
                for j=1:3
                    ACB_test.setValueControlSurface(list_index-1, k-1, X_Control_surface(list_index,k,j), ...) 
                    X_Control_surface(list_index, k, j), Z_Control_surface(list_index, k, j), j-1); 
                end
        end
    end
end
```
The coordinates matrices for the control surfaces are separated from the others. The function `setValueControlSurface` is used in Java.

```java
ACB_test.addElement();
```

The function `addElement()` will call the right class corresponding to the element type and the drawing process will begin.

3) Java drawing process

At this point, the Java drawing process will be presented in focusing on one element: the fuselage.

```java
public class AcBuilder_test
    implements Runnable, WindowListener

public void run()
{
    // openGL window
    glv = new GLViewer(this);
    // initialize GUI
    this.frame.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
    this.frame.setSize(900, 750);
    this.frame.setLocation(500, 50);
    this.frame.setMinimumSize(new Dimension(650, 500));
    this.frame.addWindowListener(this);

    JPopupMenu.setDefaultLightWeightPopupEnabled(false);
    fc.setCurrentDirectory(projPath);
    fc.setFileFilter(ff);
    menuBar.add(menu);
    menuItem = new JMenuItem("Background color");
    menuItem.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            selColor();
        }
    });
    menu.add(menuItem);
}
```

The class `AcBuilder_test` is the main class of the Java part.
The function `run()` operates directly when the class AcBuilder_test is called. All those commands initialize the GUI interface of the Java window.

```java
menuItem = new JMenuItem("Element color");
menuItem.addActionListener(new ActionListener()
{
    public void actionPerformed(ActionEvent e)
    {
        ChangeColor(AcBuilder_test.element_index - 1);
    }
});
menu.add(menuItem);
Container pane = this.frame.getContentPane();
pane.setLayout(new BoxLayout(pane, BoxLayout.X_AXIS));
JPanel acvis = new JPanel(new BorderLayout()); // panel with OpenGL window
acvis.setBorder(BorderFactory.createEmptyBorder(5, 5, 5, 5));
acvis.setMinimumSize(new Dimension(500, 500));
acvis.add(glv.getCanvas(), BorderLayout.CENTER);
pane.add(acvis);
this.frame.setJMenuBar(menuBar);
this.frame.setVisible(true);
}
```

// set the name of the component
public void getName(String name)
{compNames[element_index - 1] = name;}

// set the current number of elements
public void setNumber(int number)
{element_number = number;}

// set the current selected element
public void setIndex(int index)
{element_index = index;}

// set the type of the element (Fuselage, Wing, ...)
public void setType(int type)
{element_type = type;}

// use for the symmetry of engines
public void setSymmetry(int symmetry_matlab)
{symmetry[element_index - 1] = symmetry_matlab;}

// set the mode currently used (geometry, fuel, weights and balance, technology)
public void setCurMod(int mode)
{curMod = mode;}

// set the index for an element type
public void setElementTypeIndex(int type_index)
{element_type_index = type_index - 1;}

These functions permit to transfer some information from Matlab to Java (type, name, index and number of elements, symmetry for engines and the menu currently used in AcBuilder).

```java
// transfer the matrices from Matlab to Java for the geometrical components
public void setValue(int number, double MatlabValueX, double MatlabValueY, double MatlabValueZ,
                     double NormalValueX, double NormalValueY, double NormalValueZ, int i, int j)
{
    Matrix_data_Z[number][i][j] = (float)MatlabValueX;
    Matrix_data_X[number][i][j] = (float)MatlabValueY;
    Matrix_data_Y[number][i][j] = (float)MatlabValueZ;
    Matrix_normal_Z[number][i][j] = (float)NormalValueX;
    Matrix_normal_X[number][i][j] = (float)NormalValueY;
    Matrix_normal_Y[number][i][j] = (float)NormalValueZ;
}

// transfer the matrices from Matlab to Java for the fuel tanks
public void setValueMod(int number, double MatlabValueX, double MatlabValueY, double MatlabValueZ,
                        double NormalValueX, double NormalValueY, double NormalValueZ, int i, int j)
{
    Matrix_data_Z_mod[number][i][j] = (float)MatlabValueX;
    Matrix_data_X_mod[number][i][j] = (float)MatlabValueY;
    Matrix_data_Y_mod[number][i][j] = (float)MatlabValueZ;
    Matrix_normal_Z_mod[number][i][j] = (float)NormalValueX;
    Matrix_normal_X_mod[number][i][j] = (float)NormalValueY;
    Matrix_normal_Y_mod[number][i][j] = (float)NormalValueZ;
}

// transfer the matrices from Matlab to Java for the control surfaces
public void setValueControlSurface(int number, int cont_surf_number, double MatlabValueX,
                                   double MatlabValueY, double MatlabValueZ, int i)
{
    Matrix_Control_Surface_Z[number][cont_surf_number][i] = (float)MatlabValueX;
    Matrix_Control_Surface_X[number][cont_surf_number][i] = (float)MatlabValueY;
    Matrix_Control_Surface_Y[number][cont_surf_number][i] = (float)MatlabValueZ;
}

// transfer the matrices from Matlab to Java for the centers of gravity
public void setValueCGs(double MatlabValueCGs[][]) {
    for(int i = 0; i < 19; i++)
    {
        for(int j = 0; j < 3; j++)
            Matrix_centers_of_gravity[i][j] = (float)MatlabValueCGs[i][j];
    }
}

// transfer the matrices from Matlab to Java for technology module
public void setValueTechnology(double MatlabValueTechnology[][]) {
    for(int k = 0; k < 5; k++)
    {
        for(int i = 0; i < 3; i++)
        {
            for(int j = 0; j < 36; j++)
                Matrix_technology[k][i][j] = (float)MatlabValueTechnology[k][i][j];
        }
    }
}
```
Those functions transfer the different matrices from Matlab to Java.

```java
public void setValueCStech(double MatlabValueCStech[])
{
    for(int k = 0; k < element_number; k++)
    {
        for(int i = 0; i < 4; i++)
        {
            for(int j = 0; j < 3; j++)
            { Matrix_CStech[k][i][j] = (float)MatlabValueCStech[k][i][j];
        }
    }
}
```

The function `addElement()` chooses the right class corresponding to the element type.

Finally, the drawing process begins by calling the class `AcFuselage` (for the fuselage example).
For the geometric mode (mode 0), two lists are called \texttt{(gl.glCallList)} with the specifications of the material color, brightness...

One list corresponds to some graphics objects building thanks to the coordinates matrices.

\begin{verbatim}
    gl.glNewList(index + 2, GL.GL_COMPILE);
    gl.glEndList();
\end{verbatim}

A new list is created by \texttt{gl.glNewList} and ended by \texttt{gl.glEndList}.

\begin{verbatim}
    glBegin(GL.GL_QUAD_STRIP);
\end{verbatim}

This function permits to create quads with coordinates.

\begin{verbatim}
    gl.glNormal3f(Matrix_normal_X[index][i][j], Matrix_normal_Y[index][i][j], Matrix_normal_Z[index][i][j]);
    gl.glVertex3f(Matrix_data_X[index][i][j], Matrix_data_Y[index][i][j], Matrix_data_Z[index][i][j]);
\end{verbatim}

The function \texttt{glVertex3f} creates a point (vertex) and the function \texttt{glNormal3f} creates also a point but only used for the brightness algorithm in Java.

With four points a quad is created and by iterating on the length and around the element, the surface of the element is created.

The same way of programming is used with the other elements (wings, horizontal tail, vertical tail, canard, engines, fairing, and ventral fin). The other parts of the module like the fuel tanks or the centers of gravity are also based on this type of programming.
## Appendix D: AcBuilder N2 chart

The N2 Chart is a visual matrix representing functional or physical interfaces between system elements. The diagram is applied to identify, define, tabulate, design, and analyze functional and physical interfaces. The variable N denotes the number of elements in the $N \times N$ matrix whose relationships are illustrated. The user arranges the functional entities on the diagonal axis whereas the interface inputs are placed in the column and the outputs in the row of the function. A blank square constitutes no interface between the respective entities. Figure D1 depicts a directional flow of the interfaces between entities within the N2 Chart. Data flows in a clockwise direction between the functions. The symbol $F_1 \rightarrow F_2$ indicates the data flow from function $F_1$ to function $F_2$ and the symbol $F_1 \leftarrow F_2$ denotes the feedback.

![Figure D1 - Composition of N2 chart](image-url)
<table>
<thead>
<tr>
<th>Components</th>
<th>Wing parameters</th>
<th>Geometric parameters (all the components)</th>
<th>Geometric parameters (all the components)</th>
<th>Geometric parameters (all the components)</th>
<th>Geometric parameters (all the components)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fuel</td>
<td>Fuel parameters</td>
<td>Fuel parameters</td>
<td>Reference wing parameters for wings, horizontal tails, vertical tails and canards</td>
<td>Weights and balances parameters (cabin, baggage, fuel weights...)</td>
<td>Weights and balances parameters (cabin, baggage, fuel weights...)</td>
</tr>
<tr>
<td>Geometry output</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Centers of gravity</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Technology</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>