Master of Science Thesis

Synthesis of Novel Aircraft Concepts for Future Air Travel

Development of a Conceptual Design Environment for Conventional and Unconventional Aircraft Configurations

10-01-2014

R.J.M. Elmendorp B.Eng.

Faculty of Aerospace Engineering · Delft University of Technology

TU Delft Delft University of Technology

Synthesis of Novel Aircraft Concepts for Future Air Travel

Development of a Conceptual Design Environment for Conventional and Unconventional Aircraft Configurations

Master of Science Thesis

For obtaining the degree of Master of Science in Aerospace Engineering at Delft University of Technology

R.J.M. Elmendorp B.Eng.

10-01-2014

Faculty of Aerospace Engineering · Delft University of Technology

Delft University Of Technology
Department Of
Flight Performance and Propulsion

The undersigned hereby certify that they have read and recommend to the Faculty of Aerospace Engineering for acceptance a thesis entitled **"Synthesis of Novel Aircraft Concepts for Future Air Travel"** by **R.J.M. Elmendorp B.Eng.** in partial fulfillment of the requirements for the degree of **Master of Science**.

Dated: 10-01-2014

Head of department: _____
prof.dr.ir. L.L.M. Veldhuis

Supervisor: _____
dr.ir. R. Vos

Reader: _____
dr. ir. R. de Breuker

Abstract

In the last 60 years many new technologies have entered the aerospace industry, but the overall aircraft design remained virtually unchanged. If we compare an aircraft built in the 1960's with the latest generation, they look strikingly similar. Only small evolutionary changes entered the commercial aircraft market. A lot of these changes are driven by the ever-lasting quest to reduce the amount of burned fuel. However, the reduction in fuel usage which can be gained with these small evolutionary changes decreases every aircraft generation.

A revolutionary change in the aircraft design is needed to make step-change in aircraft fuel efficiency. Changing the configuration of the aircraft could create opportunities for aerodynamic and structural improvements, which result in higher fuel efficiency.

Current tools used in the field of aircraft design use a lot of empirical data obtained from the analysis of existing aircraft. These tools are not capable of correctly analysing unconventional aircraft configurations.

A design tool, called the Initiator, is created which is able to synthesise a conceptual aircraft design based on a given set of top level requirements. The Initiator is able to design and analyse the following aircraft configurations:

- Conventional aircraft
- Canard (fore-plane) aircraft
- Three-surface aircraft
- Prandtl (box-wing) aircraft
- Blended-Wing-Body aircraft

From top level requirements a first estimation of the aircraft characteristics is performed. The aircraft geometry is sized based on the estimated aircraft weight, wing loading and thrust-to-weight ratio. This geometry is used in a chain of analysis modules which estimate the aircraft weight and aerodynamic performance. The design process incorporates two convergence loops: The first loop repeats the aircraft sizing and analysis until the maximum take-off weight converges. The second loop re-estimates the fuel weight until the harmonic range matches the requirements.

Physics based analysis methods are preferred over empirical methods since they are easier to adapt to unconventional aircraft. The weight estimation methods for the main wing and fuselage are Class

II.V methods, which means that the main structure is sized using physical calculations. Secondary structures are still estimated using empirical relations, which is sufficient unless the analysed parts differ greatly from the data used to create the empirical methods. All other parts (everything except the main wing and fuselage) are estimated used Class II methods, which are still highly empirical.

The aerodynamics are calculated using a vortex lattice method. since this method is not capable of calculating the profile drag, an empirical method is used for the zero-lift drag estimation.

The Initiator is verified by comparing the output of the Initiator with existing aircraft. A selection of thirteen reference aircraft varying from small regional jets to wide-body long-range jet-powered aircraft is made to verify the tool. Top level requirements are defined which match the payload, harmonic range and runway performance specifications of the reference aircraft. The aircraft generated from these top level requirements are compared to the existing aircraft.

The design process is proven to work, since it converges to a feasible aircraft design which complies with the top level requirements. Since process contains design loops, inaccuracies in analyses propagate easily through the whole design. By comparing the maximum take-off weights and operational empty weights, it can be shown that the generated aircraft are similar to the reference aircraft. Nine out of the thirteen generated aircraft are estimated to within 10% of the reference aircraft weights.

Visual inspection and comparison of external aircraft dimensions show that the implemented design rules are capable of generating an aircraft which is similar to the reference aircraft.

The design tool was used to compare the different aircraft configurations. It is shown that a design process which iterates on the aircraft maximum take-off weight and adjusts the fuel mass to match range requirements works for conventional, canard, three-surface and Prandtl aircraft. Testing the design process for the Blended-Wing-Body was unfortunately not possible with the current state of the sizing methods.

It is shown that the canard aircraft provides a 12% reduction in fuel mass and a 28% reduction in operational empty mass in comparison with a conventional aircraft designed for the same payload and harmonic range. However, since none of the analysis methods have been validated the confidence in the results gained from the configuration comparison in low.

Overall can be concluded that it is possible to model a wide range of different aircraft configurations using the implemented sizing rules and analysis methods.

Acknowledgements

This thesis is the conclusion of the great time I had studying as an Aerospace Engineering student at the Delft University of Technology. Since no project can be completed without the support of other people I would like to use the next few lines to thank the people who supported my during the last part of my Master.

First of all I would like to thank my supervisor Roelof Vos who guided me through the whole project and was always available for feedback on my work or just a good talk about aircraft design. Second I would like to thank the members of my committee: Leo Veldhuis and Roeland de Breuker who took the time to asses my work and share their expert insights.

I would also like to thank my friends for their support and my fellow students who made working at the faculty a pleasure.

Finally, A special thanks to my parents and sister who have always supported me in everything I did and encouraged my to pursuit my dreams.

Contents

List of Figures

List of Tables

Nomenclature

Latin Symbols

A	Wing aspect ratio	[-]
b	Wing span	[m]
C_D	Drag coefficient	[-]
C_{D_0}	Zero-lift drag coefficient	[-]
C_L	Lift coefficient	[-]
C_{L_α}	Lift curve slope	[-]
$C_{L_{\text{max}}}$	Maximum lift coefficient	[-]
C_{m_α}	Pitching moment coefficient	[-]
c_r	Root chord length	[m]
c_t	Tip chord length	[m]
e	Span efficiency factor	[-]
h_{cr}	Cruise altitude	[m]
c_T	Specific fuel consumption	$[]1/s]$
L/D	Lift-to-Drag ratio	[-]
M	Mach number	[-]
N_{pax}	Number of passengers	[-]
R_h	Harmonic Range	[km]
S	Wing planform area	[-]
T_{static}	Static thrust	[N]
W_p	Payload mass	[kg]

X	Range parameter	[km]

Greek Symbols

λ	Wing taper ratio	[-]
$\Lambda_{0.25}$	Quarter-chord sweep angle	[-]

Abbreviations

AVL	Athena Vortex Lattice
BPR	By-pass ratio
BWB	Blended-Wing-Body
DEE	Design and Engineering Engine
FM	Aircraft Fuel Mass
KPI	Key Performance Indicator
MDO	Multi-disciplinary Design Optimisation
MTOM	Aircraft Maximum Take-off Mass
OEM	Aircraft Operational Empty Mass
PLM	Aircraft Payload Mass
PRE	Payload-Range Efficiency
TLRs	Top-Level Requirements
TSA	Three-Surface Aircraft
ULD	Unit Load Device
UML	Unified Modeling Language
XML	Extensible Markup Language

Part I

Thesis

Chapter 1

Introduction

In the last 60 years many new technologies have entered the aerospace industry, but the overall aircraft design remained virtually unchanged. If we compare an aircraft built in the 1960's with the latest generation, they look strikingly similar. One can say that the changes are more of an evolutionary nature, real revolutionary changes have not entered the commercial aircraft market since the introduction of the jet engine in the 50's[1].

A lot of these changes are driven by the ever-lasting quest to reduce the amount of burned fuel. The design changes result in of course environmental and, with the ever-rising oil prices, economical benefits. Figure 1.1 presents the total fuel consumption and the fuel consumption per seat of aircraft introduced in the last half century.

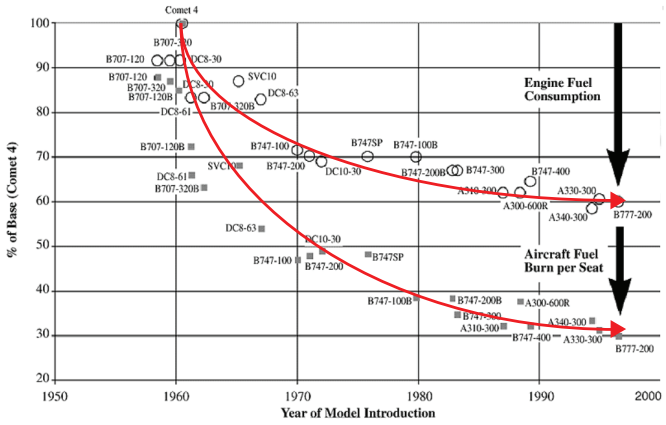


Figure 1.1: History of the aircraft fuel consumption (source: [1])

The introduction of the De Havilland DH.106 Comet brought the jet engine to the civil transport aircraft market. From the Comet's introduction onwards, the fuel efficiency of the engines and the total aircraft started reducing every year and is levelling out at around 30% of the fuel burned per seat in comparison to the Comet. As can be seen in Figure 1.1 the curve seems to almost reached

[1] A notable exception is the short popularity of the supersonic transport aircraft (Concorde & Tupolev Tu-144)

its asymptote; to introduce another step-change in aircraft fuel efficiency the evolutionary change pursued in the last half-century will probably not be sufficient. A revolutionary change in the aircraft design is needed.

One big change would be the overall aircraft configuration. As can be seen in Figure 1.2 a lot of aerodynamic advantages can be gained from unconventional wing shapes. Also the general layout of the aircraft could create opportunities for aerodynamic and structural improvements. Figure 1.3 shows a wide variety of different configurations for subsonic transport aircraft.

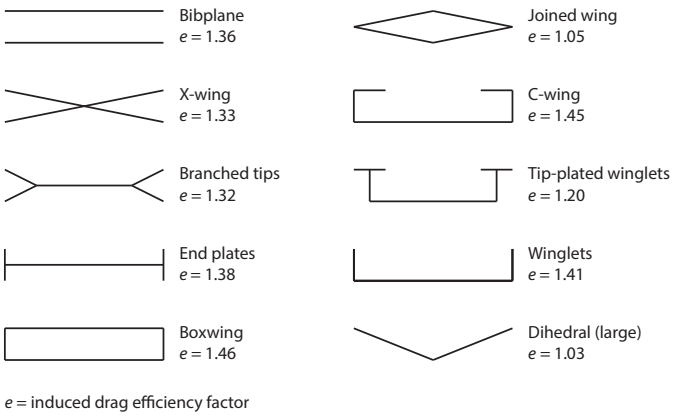


Figure 1.2: Span efficiency of different non-planar wing configurations (based on [2])

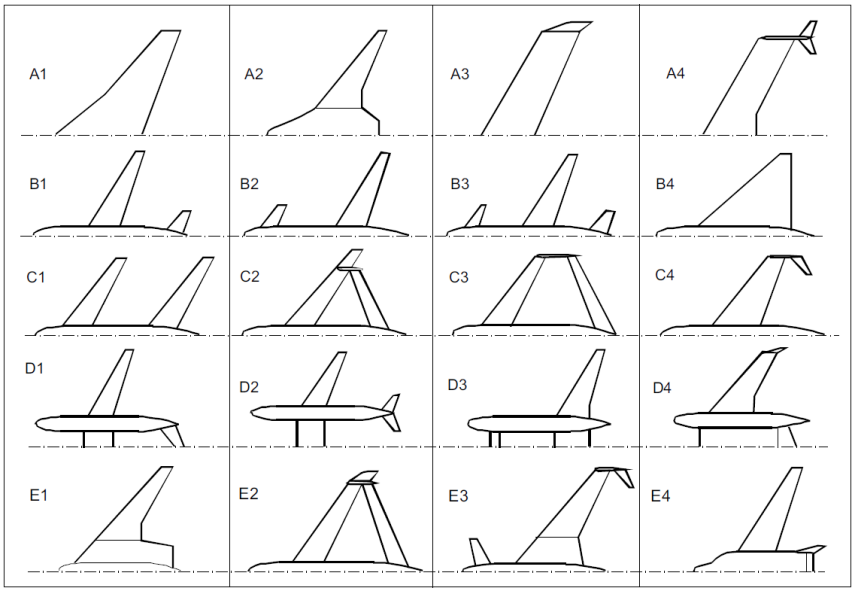


Figure 1.3: Aircraft configuration matrix (source: [3])
A: Flying wings
B: Planar monoplane, single body
C: Non(co)planar wings, single body
D: Planar monoplane, multi-bodies
E: Hybrid configurations

1.1 Research Question and Thesis Goal

The possible improvements which could be gained by using different aircraft configurations poses the following research question:

> ***Which aircraft configuration has the potential to introduce a significant increase in fuel efficiency?***

Current tools used in the field of aircraft design leverage the knowledge gained from aircraft developed in the last half-century. The aircraft metrics and performance characteristics are captured in databases and empirical design methods are derived from this information. Since these methods are based on knowledge of existing aircraft, they can only be used to design aircraft with design features that are similar to the aircraft present in the database.

The aforementioned method is inadequate in the design of unconventional aircraft. Since there is no real performance information available of unconventional configurations, the design of such an aircraft requires more effort than designing a more conservative and traditional design. In order to be able to get insight into the performance of unconventional aircraft concepts a tool needs to be conceived which enables the synthesis of such aircraft in an efficient and fast manner very early in the design process.

Since the tool should be able to design and analyse unconventional aircraft, favour should be given to first principle physics-based methods. This will make the tool sensitive to design changes which are not captured by empirical methods. An aircraft design process should be implemented which enables the synthesis of conventional and unconventional aircraft based on a set of top-level requirements. In order to be able to make a fair comparison of the different aircraft all designs should be analysed with the same methods, regardless of aircraft configuration. In other words: all methods should be able to analyse a wide range of aircraft configurations.

At Delft University of Technology previous research has been done on conceptual design methods for conventional and box-wing aircraft [17], three-surface aircraft [18] and Blended-Wing-Body aircraft [19] and [20]. The effort of these projects should be combined in a single design tool to be able to compare the different aircraft concepts. Besides the conceptual design tools which create a first aircraft design based on top-level requirements more sophisticated design programs are developed as the Design and Engineering Engine. The design tool should be able to fulfil the role of the Initiator, the program which creates the first estimate of an aircraft design and provides the input for higher-fidelity analysis methods.

Because of work done by the previously mentioned conceptual design tools the initial implementation is limited to the design of:

- Conventional aircraft
- Canard (fore-plane) aircraft
- Three-surface aircraft
- Prandtl (box-wing) aircraft
- Blended-Wing-Body aircraft

The design tool should be developed with modularity in mind. Adding more aircraft design configurations in the future should be supported and extending the analysis capabilities should be possible without the need of re-writing the whole program.

All the above results in the following thesis goal:

> ***The development of a flexible automated conceptual design environment for the synthesis and analysis of conventional and unconventional aircraft designs.***

1.2 Report Structure

This report consists of two parts. Part I describes the work done in the context of the thesis, Part II contains the implementation details of the design tool.

First, background information about the different aircraft configurations, the aircraft design process and an introduction to the Design and Engineering Engine developed at Delft University of Technology is given in Chapter 2.

The architecture of the design tool is an important choice since it eventually influences the flexibility of the design tool. This is presented in Chapter 3. Also the implemented design process and the used design and analysis methods are elaborated in this chapter.

To be able to use the design tool it needs to verified. This is done by comparing the design tool output to data from existing aircraft. The method and results of the tool verification can be found in Chapter 4.

Chapter 5 will present an application of the design tool by using it to compare a set of conventional and unconventional aircraft designed for a wide set of payload and range requirements.

The thesis is concluded in Chapter 6 and recommendations for future research are presented.

Chapter 7 gives an introduction to the second part of this report: the code documentation.

Chapter 8 draws an outline of the structure of the program. Here the composition of the different components and their implementation is shown.

Chapter 9 presents the aircraft parts also known as the *High Level Primitives*. The different parameters which are required to define the parts and the methods of generating the geometry are elaborated.

In Chapter 10 the installation and ways of operating the program are explained. Also the structure of the files which are needed to operate the application is presented.

Chapter 2

Background

2.1 Aircraft Configurations

2.1.1 Conventional aircraft

(a) De Havilland DH.106 [21]

(b) Boeing 367-80 [22]

(c) Airbus A350XWB [23]

(d) Sud Aviation SE 210 Caravelle [24]

(e) Bombardier CRJ900 [25]

Figure 2.1: (Modern) jet transport aircraft

The majority of all currenly operated aircraft can be called "Conventional aircraft". As mentioned previously, the De Havilland DH.106 Comet (Figure 2.1a) brought the jet engine to the commercial transport aircraft market. An unique design feature were the in the wing root integrated engines.

The Boeing Dash 80 (later developed into the Boeing 707 in 1958) placed the engines in nacelles underneath the wing, as can be seen in Figure 2.1b. This solved a lot of structural problems, with

the added advantage that bigger engines could be installed without a complete redesign of the aircraft. This tube-and-wing configuration with podded engines has remained unchanged during the last six decades, with the latest example the Airbus A350 (Figure 2.1c).

A slight variation on this configuration comes in the form of fuselage mounted engines. This configuration is particularly popular with regional jet aircraft. An 1959 example of this configuration is the Sud Aviation Caravelle (figure 2.1d) and a modern example is the Bombardier CRJ900 in Figure 2.1e.

2.1.2 Canard aircraft

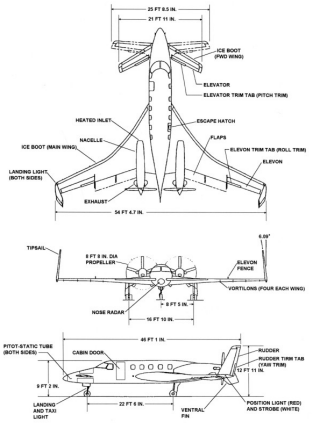


Figure 2.2: Three-view of the Beechcraft Starship [4]

The canard aircraft is a variation on the conventional aircraft. Instead maintaining longitudinal stability and control with the horizontal stabiliser attached to the tail, the canard aircraft sports a horizontal stabiliser in front of the main wing. A flying example of a canard aircraft is the Beechcraft Starship (Figure 2.2).

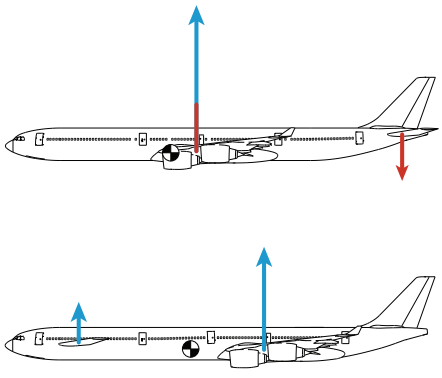


Figure 2.3: Top: trimmed conventional aircraft; Bottom: Trimmed canard aircraft (arrows not true to scale)

Assuming the centre of gravity positioned between the canard and main wing, the aircraft can be trimmed with both surfaces providing lift. This decreases the induced drag of the aircraft in comparison to the conventional aircraft, since the later always has a down force on horizontal stabiliser, which the main wing needs to compensate for with extra lift. This is illustrated in Figure 2.3.

2.1.3 Three-surface aircraft

The three-surface aircraft is based on the same principle as the canard aircraft with the added flexibility of an additional control surface. The basic configuration is a fuselage, main wing, canard and a horizontal tail. The three surfaces allows the designers to have more design freedom while maintaining the aircraft stability and the lack of negative lift on the tail surface. An example of a three surface aircraft is the Piaggio P180 Avanti, which can be seen in Figure 2.4.

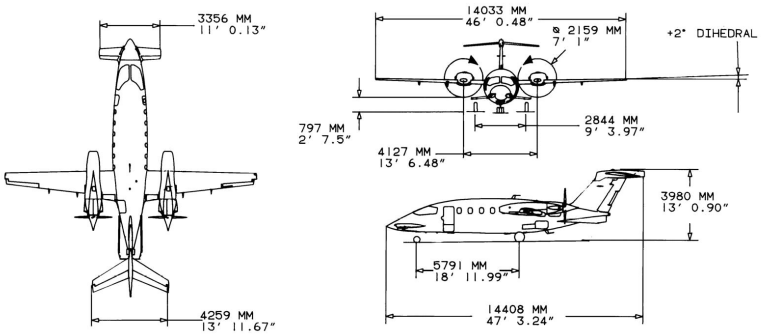


Figure 2.4: Three-view of the Piaggio P180 Avanti [5]

2.1.4 Prandtl aircraft

Figure 2.5: Impression of the Lockheed Greener Aircraft [6]

The Prandtl-plane design is based on research by Ludwig Prandtl in 1924 [26]. The Box-wing design is a derivative of Prandtl's "Best wing system" and a great induced drag reduction is expected from this design.

Currently there are no commercial available aircraft which feature the Prandtl-plane configuration. The Lockheed submission on the NASA Greener Aircraft project in Figure 2.5 is an example of a conceptual Prandtl-plane design.

2.1.5 Blended-wing-body aircraft

Figure 2.6: Boeing X-48B [6]

The Blended-wing-body aircraft concept is an aircraft where the body and wing are integrated into one blended shape. The fuselage has an airfoil shaped cross-section and is designed to contribute significantly to the lift. Since the aircraft weight is better distributed span-wise over the aircraft, less bending moments are introduced into the aircraft structures. This should result into a more fuel efficient aircraft.

Currently there are no commercially flying bended-wing-body aircraft. Boeing and NASA are conducting experiments with the remote-controlled X-48 aircraft (Figure 2.6) which should provide insight into Blended-wing-body performance in the near future.

2.2 Aircraft Design Process

The aircraft design process is a complex task involving a lot of different disciplines. Because there are is a lot of mutual influence between the different disciplines the design synthesis is an iterative process.

The section discusses two different approaches to conceptual aircraft design. First the traditional conventional aircraft design process is discussed. Secondly the Design and Engineering Engine as under development at Delft University of Technology is presented.

2.2.1 Conventional Conceptual Design Process

The design methodologies currently in widespread use in the aerospace industry are based on methods developed in the 1960's and 1970's. The conceptual design phase of these methods are characterised by the extensive usage of empirical methods to create a first estimate of the aircraft. These empirical relations are based on previously designed aircraft, which are sometimes tuned to

account for increases in technological capabilities. Only later in the detailed design computational methods and wind-tunnel tests are used to analyse the aircraft.

This approach enables designers to create aircraft concepts and estimate their performance with good accuracy, but restricts the design to aircraft which are similar to the aircraft on which the empirical methods are based.

2.2.2 Design and Engineering Engine

The development of unconventional aircraft concepts needs a different approach to the design process. Instead of relying on empirical relations to design and analyse the aircraft components a semi-empirical or fully physics based methods are preferred. The interactions between disciplines are modelled directly, which moves the aircraft design process in the realm of multi-disciplinary design optimisation (MDO).

The design starts by creating a first estimation of the aircraft based on top-level requirements. This design is analysed and the result is used to create a higher-fidelity design of the aircraft in question. Every step of increasing fidelity needs a starting point, this task is preformed by the so-called *Initiator*. In fact, every design loop can be regarded as the Initiator of the next design loop; as is illustrated in Figure 2.7. The first Initiator which generates a first preliminary design of the aircraft is the program developed in the context of this Master's thesis project. Every step in the design process is performed by a Design and Engineering Engine (DEE).

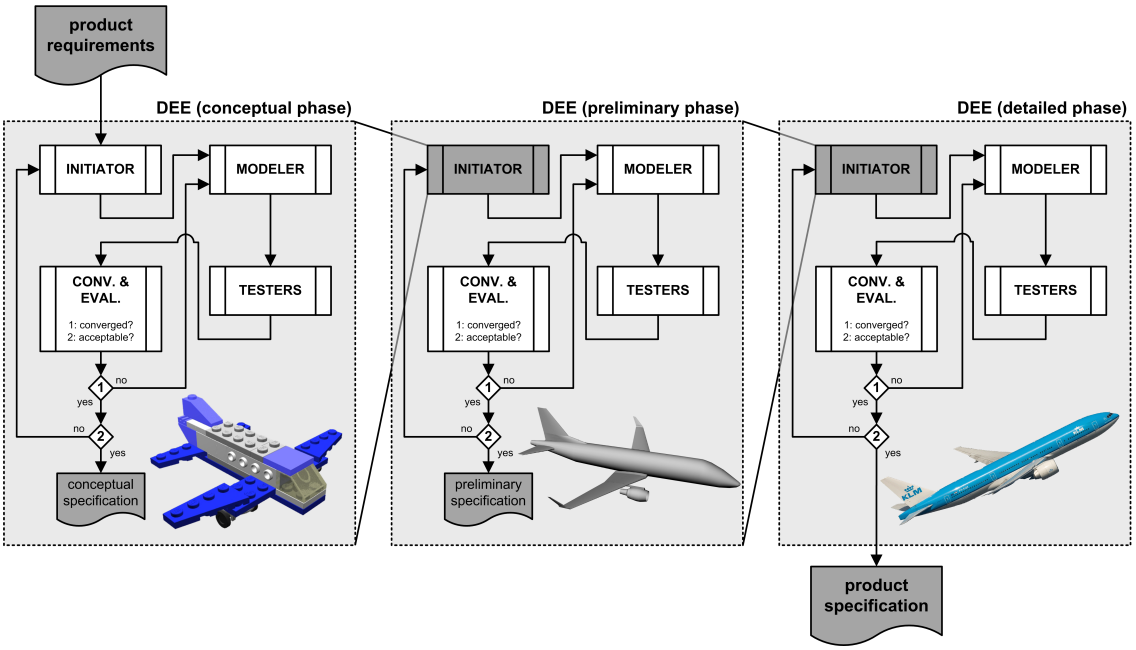


Figure 2.7: Flowchart an aircraft design process using Design and Engineering Engines (source: [7])

In Figure 2.8 the layout of a Design and Engineering Engine is shown in a more detailed flow chart. Here can be seen that for the different disciplines the same aircraft geometry generated by the multi model generator is used to create their input. The output is combined to calculate the performance

of the analysed aircraft. This information is used to determine the feasibility of the design and can be used as an objective function of the optimiser.

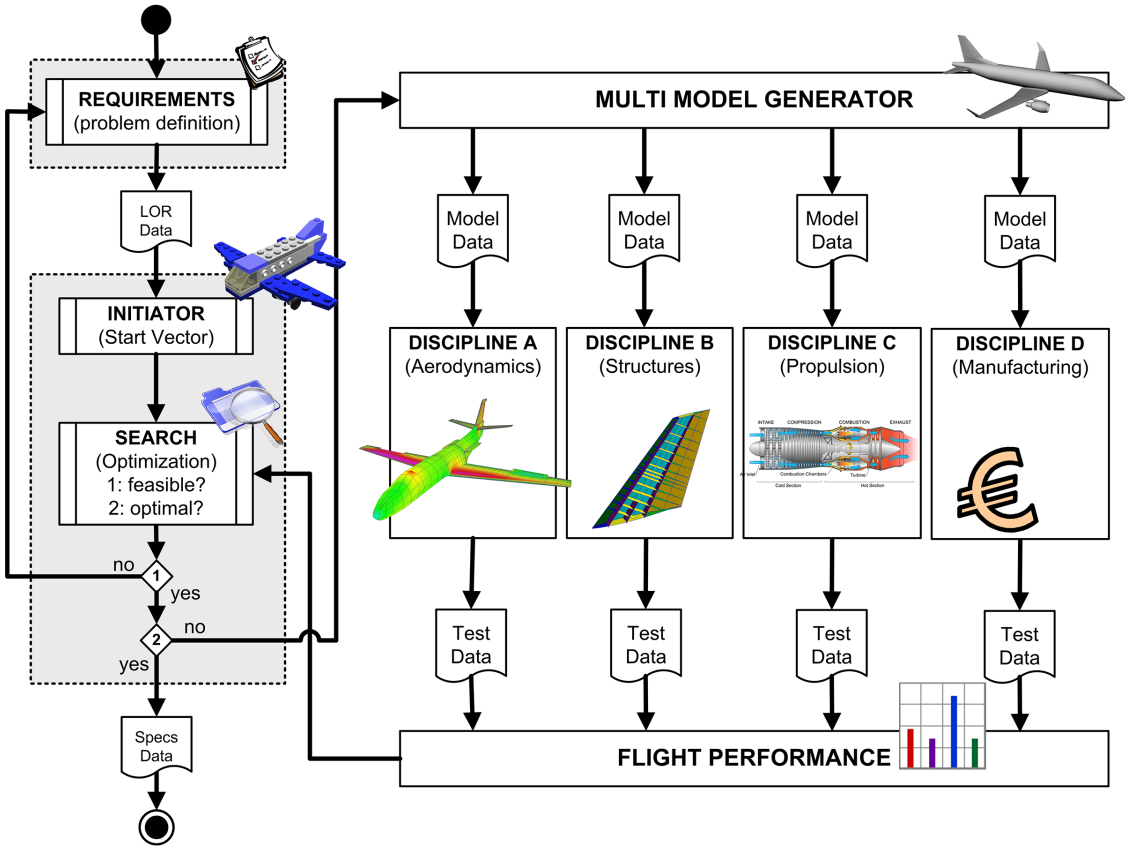


Figure 2.8: Flowchart a Design and Engineering Engine (source: [8])

Chapter 3

Design Tool Description

This Chapter describes the design tool: *the Initiator* developed in this thesis project. First the global architecture of the tool is presented. In Section 3.2 the aircraft design process implemented in the tool is elaborated. The different modules of the Initiator are presented in Section 3.3.

3.1 Software architecture

The Initiator is developed in a modular manner. To enable this modularity object-oriented programming is used. All the different program features are defined as classes. Once a class is instantiated it will be represented as an object in the program. Basic classes are created which can be extended to represent more sophisticated object.

The Initiator consists of module and geometry objects which are controlled by a controller object. All program flow is directed by the controller object. The top-level layout of the Initiator can be found in an UML in Figure 3.1. Modules can specify modules they depend on for which the controller will make sure they are completed before the module is started. There are four different types of modules defined:

- Sizing modules
- Analysis modules
- Design modules
- Workflow modules

Sizing modules perform the preliminary sizing of the aircraft based on the top-level requirements and configuration settings that are given as an input. The resulting design can be used by analysis and design modules to respectively analyse the aircraft (aerodynamics, weights, etc.) or design a specific part (cabin, control surfaces, etc.). The workflow modules facilitate the Initiator by providing XML read/write methods and implement design workflows (to converge to a consistent design).

The aircraft is built up of separate parts called high-level primitives. Each primitive is able to generate its own geometry. Besides this, the primitives implement methods which are able to calculate for example surface areas, mean aerodynamic chords, etc, whatever is applicable to the primitive in question. The main primitives are:

- **Wing**: Used for wings, tail surfaces, etc.
- **Fuselage**: Used for conventional as well as oval fuselages
- **Engine**: Used to model engine pods

Besides the main primitives, some simple primitives are defined to model cargo containers, landing gear, spars and combine wings into a box-wing. For more information about the implementation of the primitives, please read Chapter 9. Geometry is generated on-the-fly when it is needed and is automatically flagged for re-generation when parameters are changed. This means that at all times an up-to-date model of the aircraft is used, without the need to re-generate all geometry after every change.

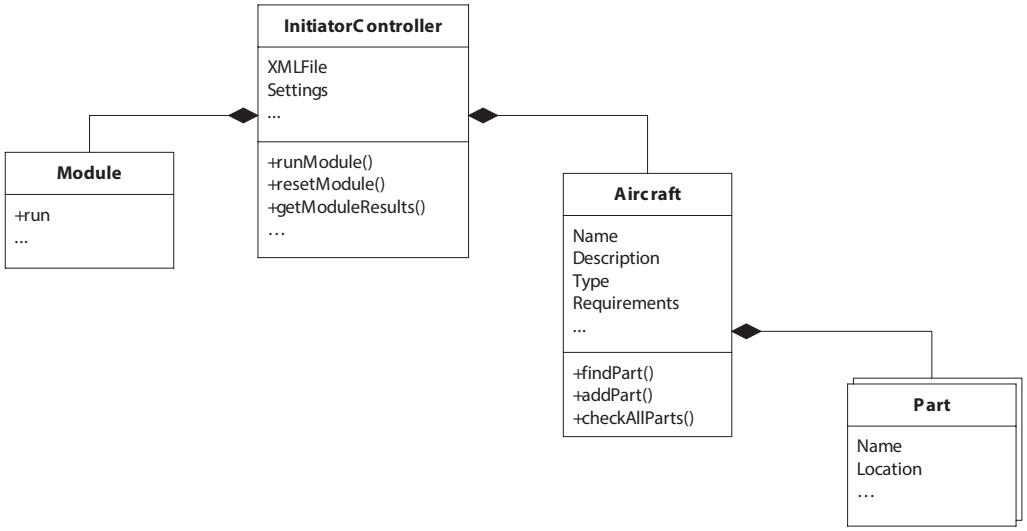


Figure 3.1: Top-level UML of the Initiator

3.2 Design process

This section describes the design process. The process flow leverages two different mechanisms in the Initiator. First, the sequence of modules is controlled by module dependencies. This provides the basic forward-feeding module flow. The feedback loops are generated by the *DesignConvergence* module, which resets a set of modules, changes input parameters and calls the controller to re-run the modules.

A high-level view of the design process can be found in Figure 3.2. The whole process starts with the definition of the top-level requirements (mission requirements), configuration parameters and a set of performance parameters (initial guess of the drag polars, specific fuel consumption and $C_{L_{\max}}$ values). This is represented in the first row in Figure 3.2.

From the specified top-level requirements a Class I weight estimation is performed. The wing loading and thrust-to-weight ratio of the aircraft is determined by evaluating performance and regulatory constraints. A Class II weight estimation (Raymer [27]) is performed to get a more design-sensitive weight and centre-of-gravity estimation.

The Class II weight results are used to perform an aerodynamic analysis by using a vortex-lattice method (AVL [28]). The aerodynamic forces and moments on the aircraft surfaces are used in the Class II.V methods for a more refined and physics based estimation of the wing and fuselage weight.

Since the vortex-lattice method is an inviscid analysis method, the profile drag is calculated separately. This profile drag is calculated empirically using Torenbeek's method [29]. All aerodynamic and weight information is gathered and the overall aircraft performance is calculated.

After every design loop the calculated (Class II.V) weight is compared to the weight previously calculated. If the deviation is larger than 1% of the maximum take-off mass, the aircraft weight and performance data is fed back to the Class I methods, the wing loading and thrust-to-weight ratio are recalculated and the whole design process is repeated until the weight converges.

When the aircraft weight is converged, the range is tested against the payload-range requirements. To meet the required range, the sensitivity of the fuel mass on the aircraft range is determined using a linear regression and the fuel mass for the next iteration is calculated. Because this changed fuel weight also alters the gross aircraft weight, the whole weight convergence loop is repeated. This process is repeated until the calculated range is within 1% of the required range. One aircraft takes around 300 seconds on an Intel i7-3610QM CPU to converge for both the range and weight.

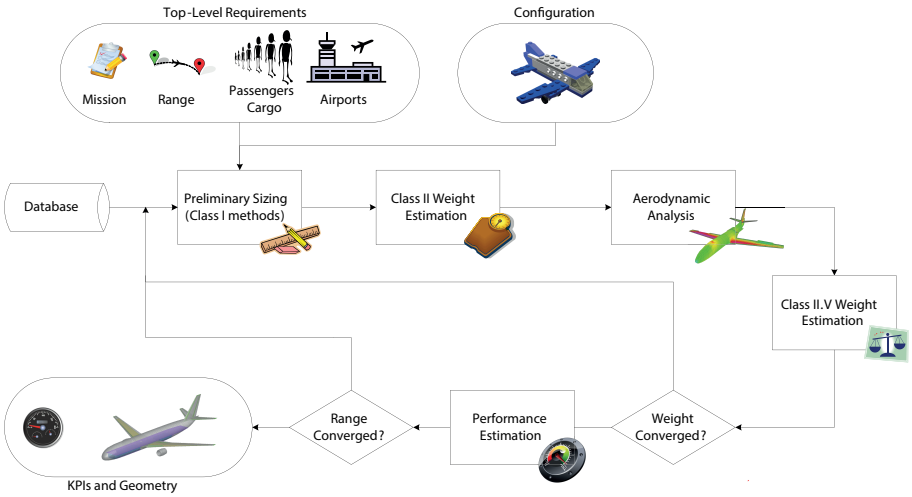


Figure 3.2: An abstract overview of the design process

3.3 The Modules

This section gives an overview of all modules currently implemented in the Initiator. In Figure 3.3 the N^2 chart of the module can be found. Inside the first black square all sizing modules can be found. These modules generate a first estimation of the aircraft which is subsequently used by the design and analysis modules, which can be seen in the second big square in the N^2 chart. The

Initiator contains two loops, which are marked with red squares in the N^2 chart. The inner loop makes sure the weight and aerodynamic loads converge. To converge the wing loading and weights the outer loop is used.

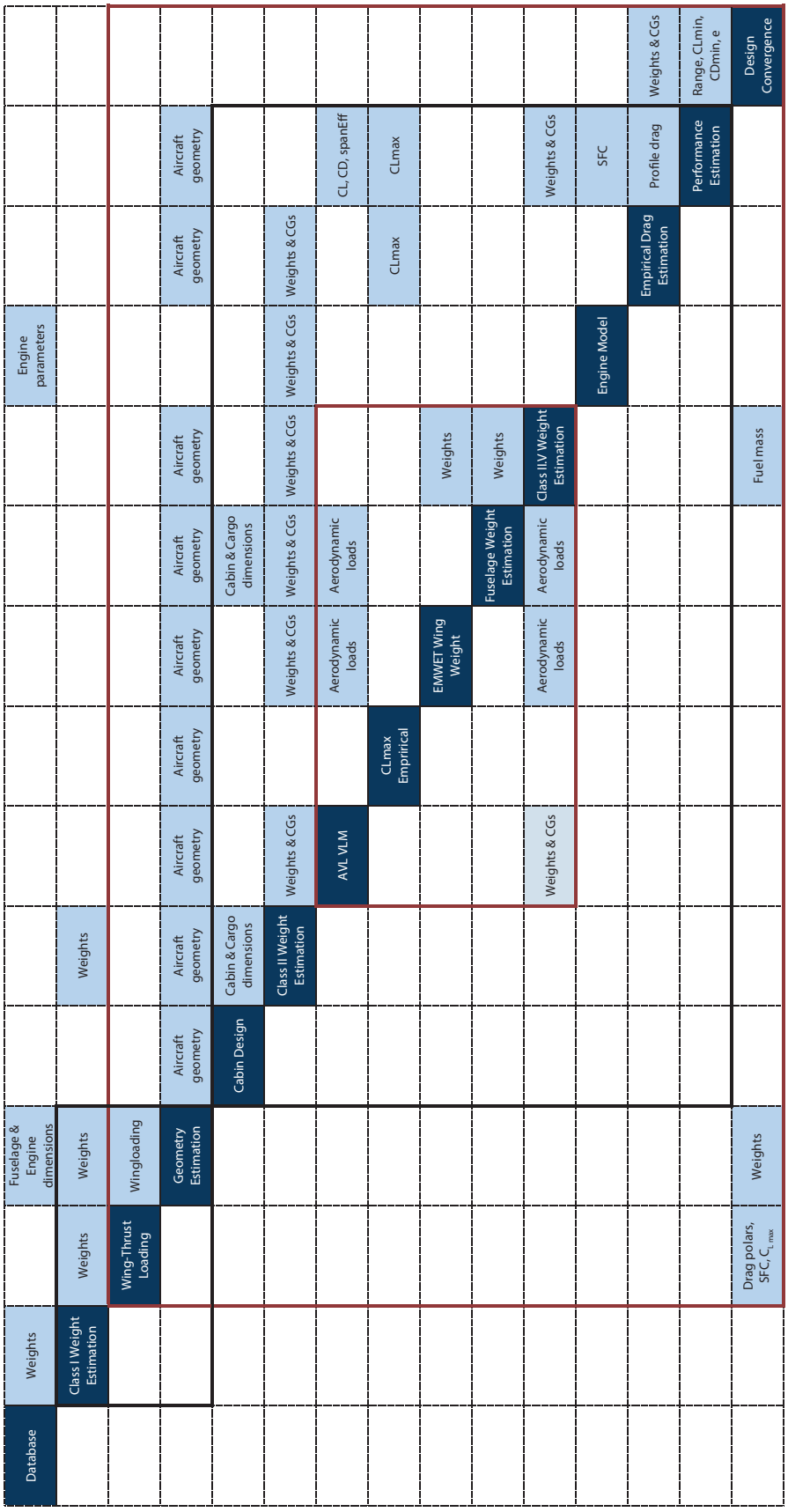


Figure 3.3: N^2 chart of the modules

3.3.1 Sizing Modules

The sizing modules are used to create an initial preliminary sizing of an aircraft based on a set of top-level requirements. They are skipped when a pre-defined aircraft is detected in the XML-file. The end result is a first estimation of the aircraft geometry, weights, propulsion and performance.

Module: *Class1WeightEstimation*

This module developed by Slingerland as part of the MSc. thesis performs a Class I weight estimation [30]. The module is able to size an aircraft based on multiple payload-range combinations and calculates an harmonic range for which all missions are feasible. With this information the required fuel mass is calculated with the 'lost range' method [3]. The relation between the OEM and MTOM is estimated using a linear regression. By using this relation in combination with the fuel mass and payload mass the MTOM and OEM of the aircraft in question can be calculated [27].

Module: *WingThustLoading*

This module chooses the wing loading and thrust-to-weight ratio of the aircraft. A large number of constraints are used to limit the design space:

- Take-off distance
- Landing distance
- Cruise speed
- Climb gradients
- Climb rate
- Stall speed
- Turn rates

The evaluation of the constraints are based on Roskam [31]. For a full description of the module please read Slingerland's Master thesis [30].

Module: *GeometryEstimation*

The *GeometryEstimation* module uses the wing loading and weight information to create a first estimate of the aircraft geometry. The wings are sized to meet the wing loading estimated by the *WingThrustLoading* module where canard planform is included to get the required loading. Since the canard is estimated with volume coefficient methods [27], the canard panform area is defined as a function of the wing planform area and mean aerodynamic chord. Therefore the wing and canard sizing is solved simultaneously and iterated until the desired planform area is achieved. All other tail surfaces are also estimated using the volume coefficient method. The sweep angle and taper ratio are calculated using empirical relations used by Raymer [27]. The fuselage is sized to meet the required cabin floor to hold the payload by using a pre-set passenger density. The slenderness is estimated using database values of aircraft with similar payload-range requirements. The engines length and diamter is calculated by using the thrust to interpolate database information.

The main part (which is the fuselage for the configurations currently implemented in the Initiator) is place at (0,0,0). All other parts are positioned with respect to this part by using length fractions defined in settings. These settings for the different configurations are fractions of the main part dimensions.

Design rules for the Blended-Wing-Body aircraft are created, but only create feasible aircraft geometries for very high payload requirements.

3.3.2 Analysis Modules

The Analysis modules are used to analyse the aircraft generated by the sizing modules or a predefined aircraft. All analysis modules require a fully defined aircraft; in other words: the sizing modules need to be completed, or an aircraft needs to be present in the XML file.

Module: *Class2WeightEstimation*

The *Class2WeightEstimation* module performs a Class II weight estimation for all the different parts of the aircraft. It uses the method as written by Raymer [27]. Note that the connectors of the prandtl-wing and winglets are currently not included in the weight estimation.

Module: *AVLVLM*

AVL is a vortex-lattice method developed by Mark Drela of MIT [28]. The *AVLVLM* module creates an AVL model of the aircraft and performs several AVL runs on a trimmed aircraft to get:

- Stability and Control derivatives
- C_L and C_D at several angles of attack to calculate the drag polar
- Forces and moments at $n = 1$ and $n = n_{\max}$

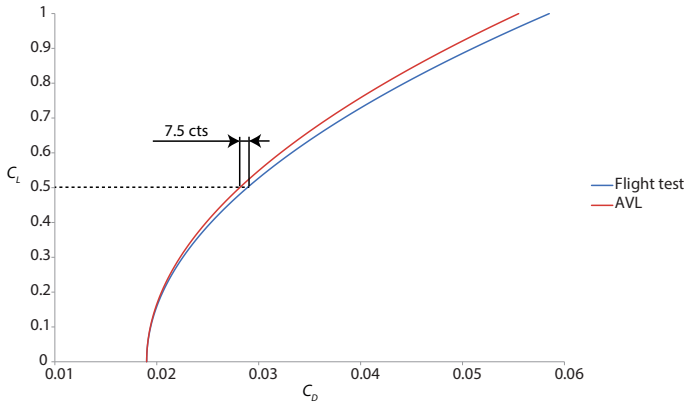


Figure 3.4: Drag polar comparison of AVL with flight test data from the Airbus A320-100 [9]

Figure 3.4 shows the comparison of an AVL model of the Airbus A320-100 and flight test data. Because vortex-lattice is an inviscid method, the C_{D_0} from the flight test data is also used to construct the AVL drag polar. Note that the AVL model has symmetric airfoils (which is not the case

on the real A320) and the twist angles could have a measurement inaccuracy since these were not clearly visible in the technical drawings. As can be seen the drag polar is estimated quite well, with an error of 7.5 drag counts at $C_L = 0.5$, providing the estimation of the zero-lift-drag is accurate. Since the vortex-lattice method is incapable of modelling thickness effects, inaccuracies may occur in the case of very thick lifting surface (as is the case with the Blended-Wing-Body).

Module: *TornadoVLM*

This module is capable of modelling the aerodynamics of the aircraft lifting surfaces with the *Tornado* vortex-lattice method [32] which is extended by Vaessen [10] with a relaxed wake model and an improved compressibility correction. An example of the output of the Tornado module can be seen in Figure 3.5. Currently *AVL* is used in favour of *Tornado* since the Trefftz-plane analysis of *Tornado* is very inaccurate. Another fact in favour of *AVL* is the runtime; *AVL* is multiple times faster than *Tornado*. Tornado is validated by Vaessen during his Master's thesis work. The results of a few validation cases can be found in Figure 3.6.

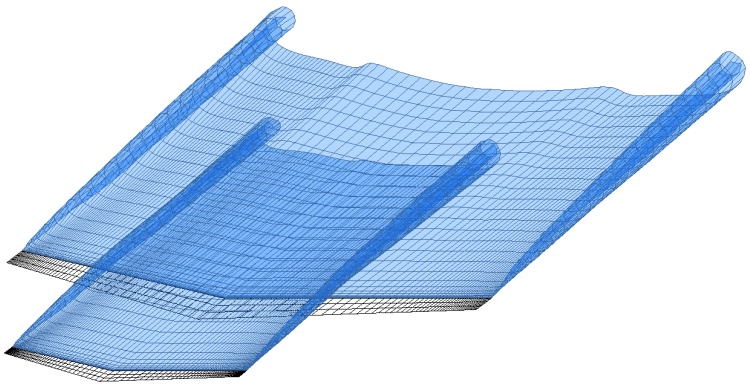


Figure 3.5: Wake visualisation plot from Tornado

Module: *CLmaxEmpirical*

This module, developed by Jan Mariens, calculates the $C_{L_{\max}}$ of a clean wing and is based on the ESDU 89034 method [33]. Since it does not give reliable results, the method is currently disabled and $C_{L_{\max}}$ values provided by the user and used. This needs to be fixed in a future version.

Module: *EmpiricalDragEstimation*

The *EmpiricalDragEstimation* is developed by M. Warmenhoven [34] and is a software implementation of the method described in Appendices F and G of prof.dr.ir Torenbeek's *Synthesis of Subsonic Aircraft Design* [29]. The method can estimate the drag of an aircraft in cruise condition and with different low-speed configurations (including flap deflections). Currently only the cruise drag is calculated, from which only the profile drag is used in other modules of the Initiator. The induced drag is already acquired from the *AVLVLM* module. Note that the method is an empirical method based on data from conventional aircraft configurations. Please be suspicious to results

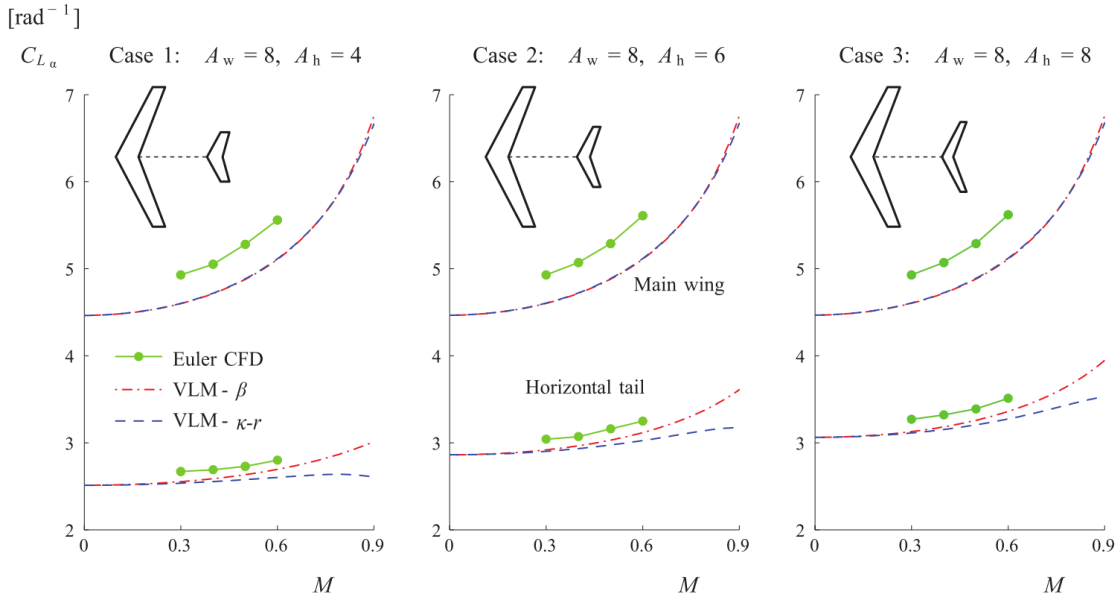


Figure 3.6: Validation cases of Tornado with varying aspect ratios, source: [10]

calculated with this module in the case of unconventional configurations. In Warmenhoven's report the tool is compared with the Fokker 100 aircraft. The tool gives a C_{D_0} of 0.0215, where the real aircraft has a C_{D_0} of 0.0188, an overestimation of 14%. This is compensated in the tool by underestimating the C_L in cruise condition. Since the C_{D_0} is the only result used by the Initiator, this could result in inaccurate drag estimation results.

Module: *EMWETWeight*

This module is a wrapper around the student version of the *EMWET* Class II.V wing weight estimation method developed by A. Elham as part of his PhD thesis [35]. *EMWET* is a quasi-analytical wing weight estimation method which uses load data from an aerodynamic analysis to calculate the material distribution in the wing box which is needed to withstand the loads. [11] Semi-empirical relations are used to estimate the secondary weights. Figure 3.7 shows the correlation between the calculated wing box weight and the actual wing weight. This fit is used inside the program to derive the wing weight after the wing box sizing.

Module: *FuselageWeightEstimation*

This module performs a Class II.V weight estimation of the fuselage and is developed by K. Schmidt as part of his Master's thesis work [12]. Is is able to perform a weight estimation on conventional as well as oval-cross-subsection fuselages. It uses the moment and forces calculated by *AVL* as well as component weights calculated by the Class II or Class II.V weight estimation methods. In Figure 3.8 the verification of the fuselage weight estimation can be found. The reference aircraft used are all quite small, therefore it would be advisable to validate the method with a more varied set of reference aircraft.

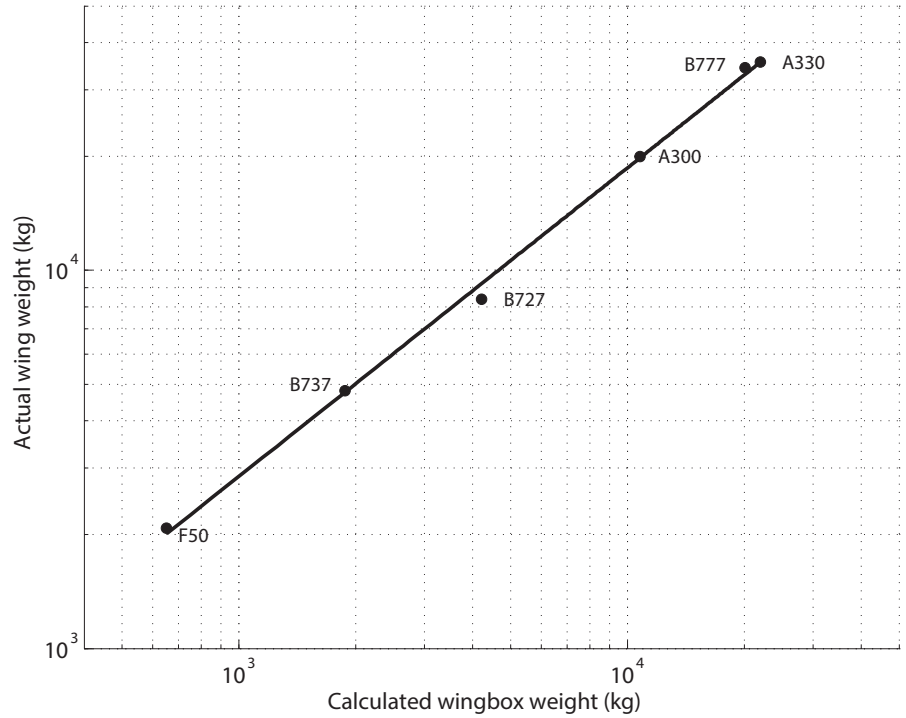


Figure 3.7: Relation between wing box weight and total wing weight (source: [11])

Module: ***Class25WeightEstimation***

The *Class25WeightEstimation* extends the Class II weight estimation module by iterating *AVL*, *EMWET* and the Class II.V fuselage weight estimation method until the maximum take-off mass, wing mass and fuselage mass stabilise within a pre-set margin. All other parts (eveything except the main wing and fuselage) are calculated with the same methods as the Class II weight estimation.

Module: ***HighLiftDevices***

This module, developed by Jan Mariens, uses the ESDU 99031 method [36] to estimate the lift-curves of the wing-fuselage combination with high-lift devices. It evaluates all possible flap/slat combinations and chooses the most simple solution which meets the runway requirements. Since this module had a very long runtime (around 1 minute on an Intel i7-3610QM CPU) it is not included as a dependency for other modules. Also the accuracy of the method regarding unconventional aircraft configurations is unknown.

Module: ***PerformanceEstimation***

The *PerformanceEstimation* module combines all data calculated by the analysis modules to create the aircraft drag polar, manoeuvre loading diagram and payload-range diagram. The drag polar is constructed by combining the induced drag from the Trefftz plane analysis of *AVL* and the profile drag calculated with empirical methods. The drag polar is commonly approximated with a polar

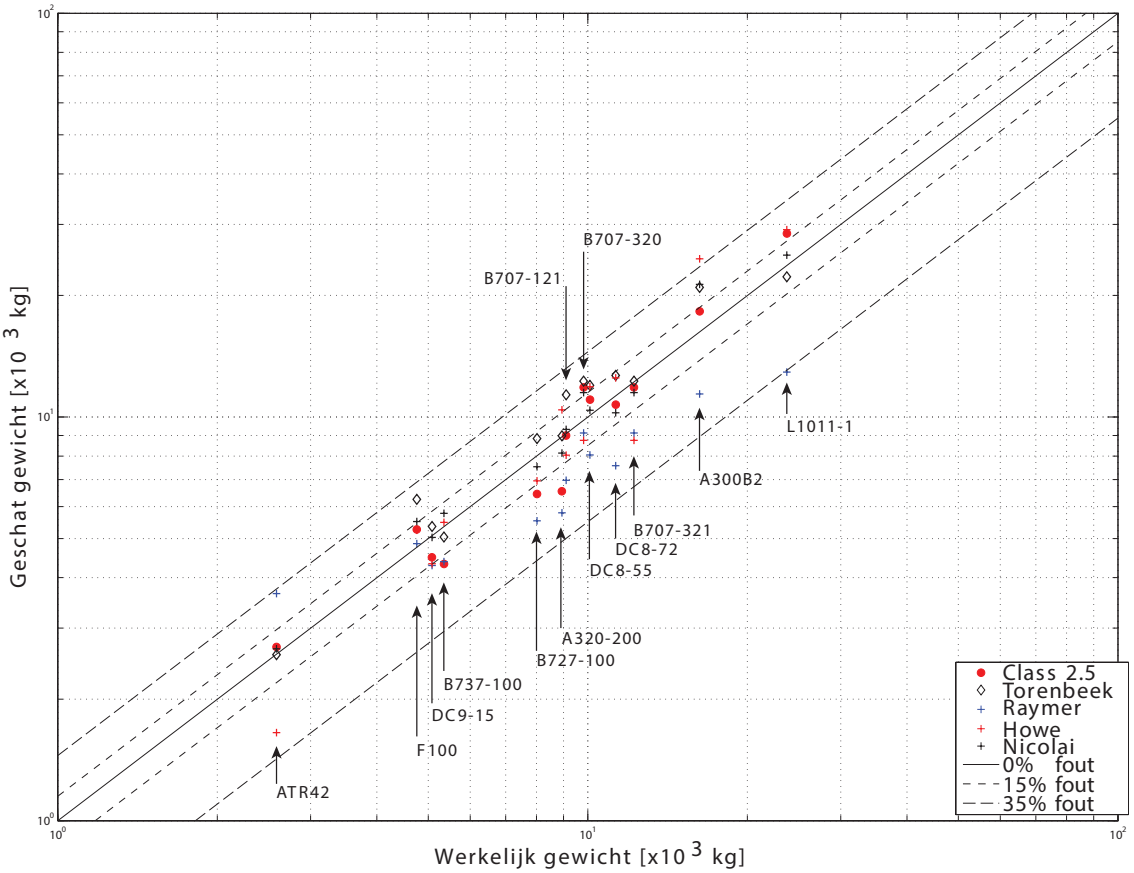


Figure 3.8: Verification of the fuselage weight estimation method (source: [12])

which has the form:

$$C_D = C_{D_0} + \frac{C_L^2}{\pi \cdot A \cdot e} \tag{3.1}$$

To calculate the values of C_{D_0} and e the method presented in Section 5.3 of [29] is used, which is illustrated in Figure 3.9. A linear fit is made around the cruise lift coefficient for $C_D = f(C_L^2)$. The intersection of this line with the $C_L^2 = 0$ is the value of C_{D_0} of the polar approximation.

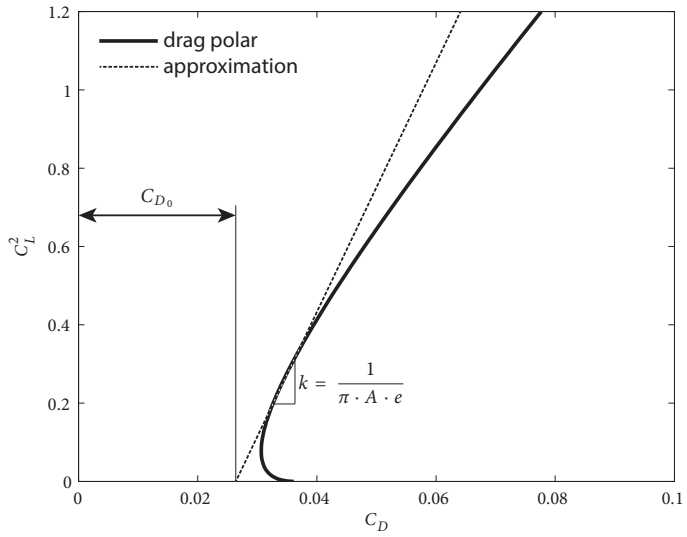


Figure 3.9: C_{D_0} and e estimation

3.3.3 Design Modules

Design modules add or change the aircraft design. In implementation they are not different from the analysis modules. However, to separate analysis from design, a different module type is created.

Note that including design modules in an optimiser workflow need to be done with caution. One has to be sure that parameters which are present in the design vector are not changed by the modules.

Module: *CabinDesign*

This module creates the fuselage interior by creating the cabin floor (and wall and roof in an oval fuselage) and the cargo bay. It creates a *Cargo* part into the fuselage and fits ULDs container into the bay. The module is created as part of Schmidt's Master thesis [12].

Module: *ControlAllocation*

The *ControlAllocation* module allocates control surfaces to various parts of the aircraft by using some simple design rules and writes the results to the wing parts. This module will be extended by [37].

Module: *PositionLandingGear*

The *PositionLandingGear* module estimates the placement of the landing gear by using a large set of constraints which result in the best placement of the landing gear. The module is developed by Heerens [38].

3.3.4 Work-flow Modules

The work-flow modules don't analyse or change the aircraft directly. They implement design work-flows and in- and output modules.

Module: *XMLReader*

The *XMLReader* reads the aircraft definition file and stores the data into the MATLAB objects for use by the modules. Existing data previously loaded are overwritten by the values in the XML file.

Module: *XMLWriter*

The *XMLWriter* module writes all data from the Initiator to the aircraft definition file. Existing data in the file is overwritten.

Module: *DesignConvergence*

This module implements the design loop which feeds back the Class II.V weight estimation to re-evaluate the wing loading and thrust-to-weight ratio. The weight converge when the change between two iterations fall within a pre-set tolerance. The module also has a loop which changes the fuel mass until the desired range is met. The work-flow can be found in the activity diagram in Figure 3.10. This is the implementation of the design process outlined in Figure 3.2.

Module: *InteractiveMode*

The interactive mode implements a text interface for the user. It has the possibility to run and reset modules and print results in the console.

Module: *BatchMode*

The *BatchMode* module reads the `<runList>` tag in the aircraft definition file and runs the modules in the order they are specified in the file.

Module: *PlotTool*

The *PlotTool* module has two different run cases. When invoked from the *InteractiveMode* it presents the user an input prompt from where the user can plot the aircraft geometry and figures created by the modules. When called from *BatchMode*, it reads its own module input and plots the specified figures.

Module: *ReportWriter*

The *ReportWriter* module collects results from the analysis modules and writes this information to a nicely formatted PDF file. This module required a working LaTeX installation. An example of a report generated with the module can be found in Appendix A.

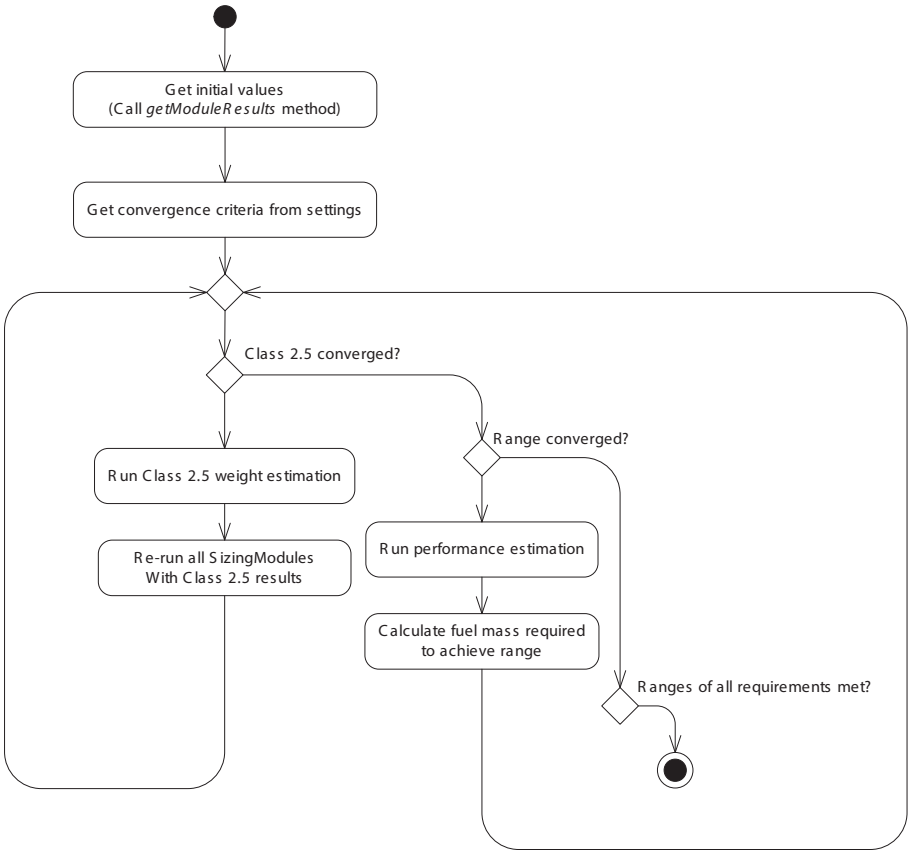


Figure 3.10: Activity diagram of the DesignConvergence module

Chapter 4

Design Tool Verification

In this chapter the Initiator design tool developed in this thesis project is compared against existing aircraft. The Initiator is a tool designed to synthesise a preliminary aircraft design from a set of top-level requirements. An aircraft design is in essence a non-unique solution to a problem posed by a set of TLRs. Since the Initiator is a design tool there is, in contrast to an analysis tool, no "right" answer which can be used to validate the tool. However, assuming commercially designed aircraft represent an optimally designed aircraft, the design tool output can be verified against existing aircraft designed for similar top-level requirements. Each analysis tool should be validated separately, this is however outside the scope of this thesis. The current state of the different modules can be read in Section 3.3.

A selection of aircraft varying from small regional jets to wide-body long-range, jet-powered aircraft is made with different wing and engine configurations. Aircraft are chosen for which sufficient reference data could be acquired. The aircraft are generated by the Initiator utilising a design routine which designs an aircraft for a specified payload versus harmonic range combination. The requirements are set to match the performance characteristics of the reference aircraft. The generated aircraft are compared with the reference aircraft designed for the same payload-range requirements.

4.1 Reference aircraft

The full list of reference aircraft requirements can be found in Table 4.1. Of all aircraft the performance gross weights and empty weights are available. Six aircraft (A300 B2-100, A320-200, B737-200, DC-10-30, MD-80 and F-100) can be compared using a more detailed weight breakdown acquired from [14] and [15]. Drag polars from flight-test data is available for the A320-200, B737-200, B737-800, DC-10-30 and the MD-80. Figure 4.1 shows the harmonic range and maximum payload mass of different aircraft used for the tool verification.

From the aircraft performance specifications top level requirements are created and Initiator input files are created. An example of an input file (the Airbus A320-200) can be found in Appendix C. Besides the requirements, the configuration (low or high wing, standard or T-tail, number of

Table 4.1: Reference aircraft requirements, source: [13]

Aircraft	N_{pax} [-]	W_p [kg]	M_{cruise} [-]	h_{cr} [m]	R_h [km]	$L_{takeoff}$ [m]	$L_{landing}$ [m]
A300-B2-100	269	31750	0.78	10058	1720	1850	1768
A320-200	150	20536	0.76	11278	2870	2180	1440
A340-300	295	50800	0.82	11887	9167	3000	1964
A340-600	380	67400	0.82	11887	10556	3100	2240
A380-800	555	83900	0.85	11887	12149	2990	2160
B737-200	97	11385	0.73	10668	1774	1829	1350
B737-800	162	21319	0.79	11887	1363	2101	1440
B777-300	394	64000	0.84	10668	3142	2574	1860
BAe 146-200	88	10250	0.65	9144	2000	1646	1192
F70	70	9302	0.77	10668	1085	1296	1210
F100	107	11300	0.72	10668	2556	1856	1321
DC-10-30	285	46180	0.82	9449	6995	2996	1820
MD80	155	18236	0.76	10668	1453	2195	1481

engines) and the wing aspect ratio is defined. Also the $C_{L_{max}}$ of the aircraft in clean, take-off and landing conditions need to be specified. The choice of the landing $C_{L_{max}}$ has great influence on the aircraft performance since this is currently the only constraint for the maximum wing loading.[1] The $C_{L_{max}}$ values are chosen such that the wing loading of the generated aircraft lies within the loading of its reference aircraft. The used parameters can be found in Table 4.2, general setting used for all configurations can be found in Table 4.3. All other aircraft parameters are estimated by the Initiator. All aircraft are designed by running the *DesignConvergence* module. The global working of this design convergence module can be found in Chapter 3.

Table 4.2: Initiator input parameters

Aircraft	A	$C_{L_{max,landing}}$	$C_{L_{max,take\text{-}off}}$	$C_{L_{max,clean}}$
A300-B2-100	7.73	2.6	2.2	1.2
A320-200	9.39	3.2	2.2	1.2
A340-300	10.02	2.5	2.2	1.2
A340-600	9.16	2.5	2.2	1.2
A380-800	7.52	2.6	2.2	1.2
B737-200	8.83	3.2	2.2	1.2
B737-800	9.45	3.4	2.2	1.2
B777-300	8.68	2.8	2.2	1.2
BAE146-200	8.98	2.8	2.2	1.2
F70	8.43	3.0	2.2	1.2
F100	8.43	3.0	2.2	1.2
DC-10-30	6.91	2.7	2.2	1.2
MD80	9.62	3.2	2.2	1.2

[1] In future Initiator versions the wing loading will also be constrained by buffet onset in cruise conditions

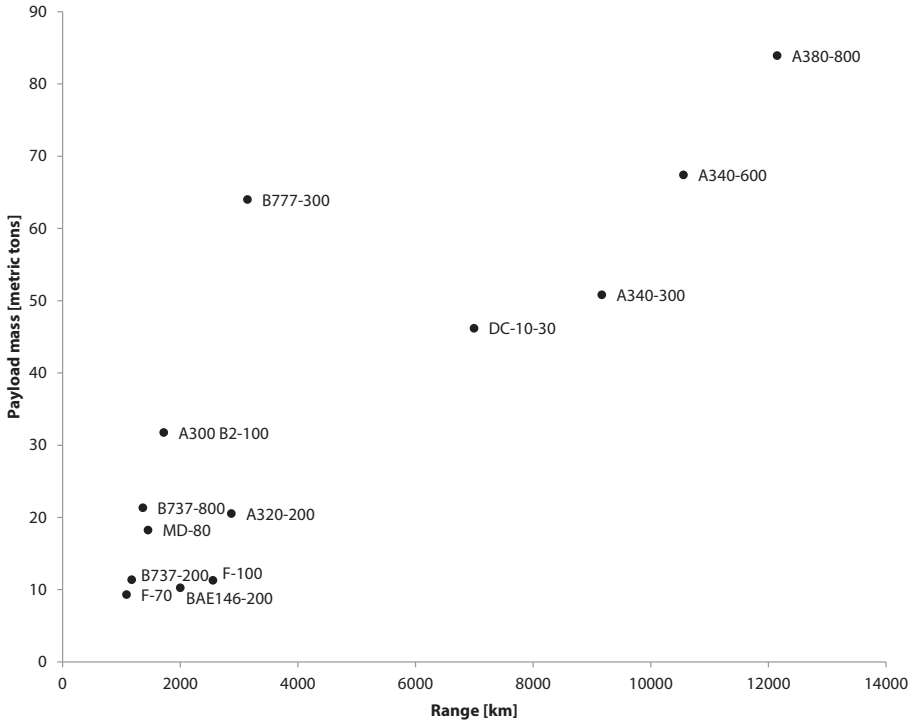


Figure 4.1: Payload - harmonic range combinations of the reference aircraft

Table 4.3: Initiator performance settings

Parameter	Value	Description
$\Delta C_{D_{0,\text{take-off}}}$	0.0350	Zero-lift drag increment, take-off configuration, flaps & landing gear
$\Delta C_{D_{0,\text{landing}}}$	0.0850	Zero-lift drag increment, landing configuration, flaps & landing gear
$\Delta e_{\text{take-off}}$	0.05	Span efficiency increment, take-off configuration
$\Delta e_{\text{landing}}$	0.10	Span efficiency increment, landing configuration
BPR	6.0	Engine bypass ratio

4.2 Comparison based on Design point

The design point is the wing loading - thrust-to-weight-ratio combination chosen for the designed aircraft. The design space is the space set up by all possible wing loading versus thrust-to-weight ratio combinations. The design space is constrained by a multitude of requirements. The choice of design point has a big influences on the aircraft design. But because limited information is available at the conceptual design phase a lot of assumptions need to be made. A majority of constraints on the wing loading are a result of the high-lift performance of the aircraft. Because there is no high-lift or clean $C_{L_{\max}}$ prediction available in the tool at the time of writing, all $C_{L_{\max}}$ values are set by the user. The design point which results in the highest wing loading and still meet all the constraints is selected by the design tool. For more information please read Slingerland's Master's thesis [30].

As can be seen in Figure 4.2, the wing loading of the aircraft with the A320-200 requirements matches the wing loading of the aircraft present in the Initiator database. Note that the "reference aircraft" presented in the Figure are not the reference aircraft used in this Chapter. The refer-

ence aircraft used in the sizing are gathered from a database inside the Initiator. Based on the payload-range requirements a selection of similar aircraft is made. For more information please read Slingerland's Master's thesis [30].

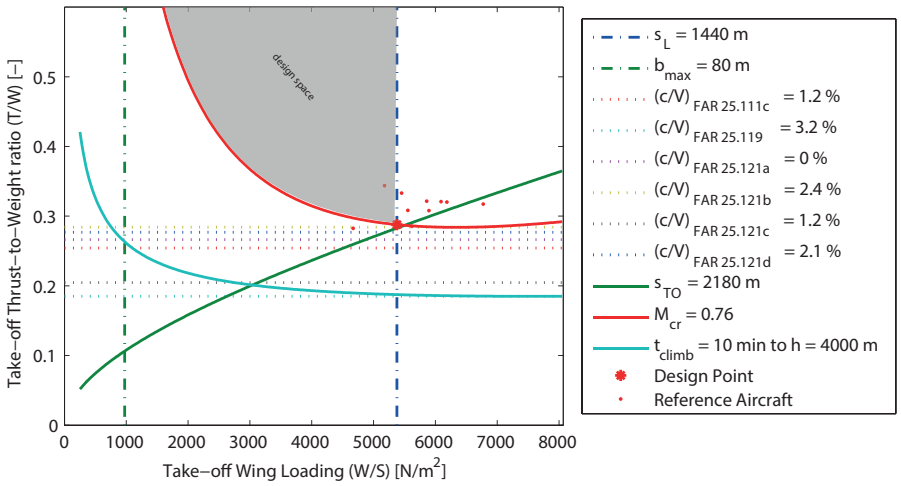


Figure 4.2: Design point of the Initiator generated aircraft according to A320-200 specifications

The active constraints are the landing distance, cruise speed and take-off length. In other words, the design space is bound by these constraints. Note that the landing and take-off distances are highly dependent on the $C_{L_{\max}}$ values supplied by the user, therefore it cannot be stressed more that a good $C_{L_{\max}}$ estimation method is required to improve the design tool and make it less dependent on values given by the user. The cruise speed constraint (and also the climb requirements) are highly sensitive to changes in the aircraft drag polar. Since the profile drag is currently estimated with an empirical method, its validity cannot be ensured for unconventional aircraft configurations.

All design points of the generated aircraft are compared to the reference aircraft in Figure 4.3. The wing loading estimation is quite good, although this is highly influenced by the selected $C_{L_{\max}}$ values. The thrust-to-weight ratio is not estimated correctly for all aircraft. Further inspection reveals that the thrust-to-weight ratio is driven up by the cruise speed requirement in the cases where the thrust-to-weight ratios have a large difference.

For example the generated aircraft with Airbus A380-800 requirements has a thrust-to-weight ratio of 0.34, where the "real" A380-800 has a thrust-to-weight ratio of 0.24. The cause of this high value of the cruise speed constraint is the high drag coefficient, which is a direct result of the high cruise C_L of 0.75. The C_{D_0} is 203 drag counts, where the total C_D is 491 cts; from this can be concluded that the lift-induced drag is very high, which is a result of the high cruise lift coefficient. In reality this high C_L could result in buffet onset in cruise conditions. Since the aircraft is always designed such that the lift equals the weight, this means that the wing loading is too high for the aircraft. Therefore a buffet onset constraint is needed to limit the wing loading, which would result in a lower cruise C_L. This lower C_L will result in a lower C_D during cruise, which decreases the required thrust (or trust-to-weight-ratio). This results into smaller engines, but larger wings. The other aircraft which have a large thrust-to-weight ratio show the same high cruise C_L and corresponding high C_D values.

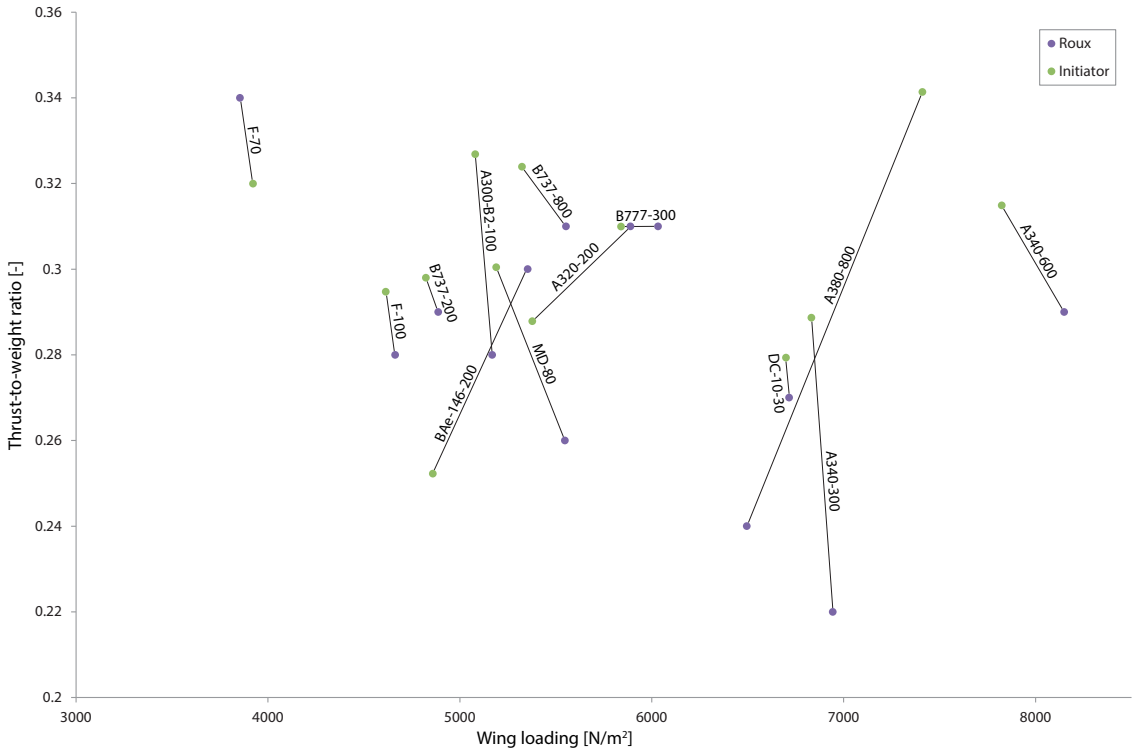


Figure 4.3: Comparison of the design point of the Initiator generated aircraft and data from Roux [13]

4.3 Comparison based Geometry

The Initiator design tool generates 3-dimensional geometry of the designed aircraft. The configuration (high or low wing, tail type) and the aspect ratio are defined by the input. All other variables are calculated by the design tool. The Initiator uses a lot of settings to generate the aircraft. The most relevant settings are listed in Table 4.4. This section will compare the geometry generated by the Initiator with the actual aircraft.

Figure 4.4 compares the A320-200 with an aircraft generated by the Initiator with the same requirements. As can be seen the geometry is estimated quite well, note that since the aspect ratio is a user-input, this is an exact match. The vertical tail size (and to a lesser degree, the horizontal tail too) is a little underestimated. However, the vertical tail is positioned further aft than the reference aircraft, which results in a longer moment arm.

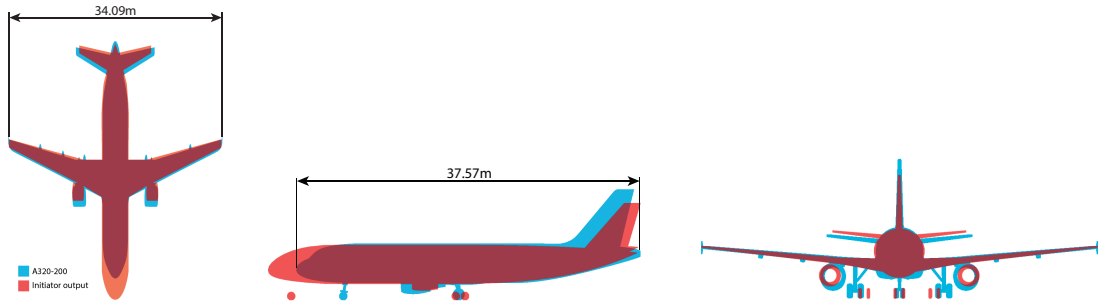
The fuselage length is clearly overestimated. Its planform area is sized by using a pre-set passenger density. This setting is currently an average density found in literature for a three-class configuration. (For more information please see [12]). This could be improved by making this parameter sensitive to range, since longer hauls require more room per passenger than short flights. Also there is no distinction made between single and twin-aisle fuselages.

Other little differences can be seen between the compared aircraft geometries. The nacelle diameter is underestimated, this is caused partly by the underestimated thrust-to-weight ratio, but since the scaling factor used by the design tool are probably based on older reference aircraft (Since they come from Raymer [27]) they are not suited to estimate modern high by-pass ratio engines. The

Table 4.4: Initiator geometry settings

Parameter	Value	Description
Wing		
Position	0.48	Fraction of fuselage length
Spar positions	0.15 0.75	Chordwise position of spars
Kink location	0.3	Spanwise location of the kink
Airfoils	B737	
Fuselage		
Nose fineness ratio	0.18	Fraction of fuselage length
Aft fineness ratio	0.55	Fraction of fuselage length
Pax density	1.29	Pax per m^2
Horizontal tail		
Aspect ratio	5.0	Fixed HT aspect ratio
Taper	0.35	Fixed HT taper ratio
Position	0.05	Fraction of fuselage length measured from end
Airfoil	NACA0012	
Vertical tail		
Aspect ratio	1.6	Fixed HT aspect ratio
Taper	0.35	Fixed HT taper ratio
Aspect ratio (T-tail)	1.0	Fixed HT aspect ratio
Taper (T-tail)	0.7	Fixed HT taper ratio
Position	0.05	Fraction of fuselage length measured from end
Airfoil	NACA0012	
Engines		
Span location (single)	0.35	Fraction of half-span
Span location (dual)	0.3 0.7	Fraction of half-span
Fuselage location	0.7	Fraction of fuselage length

wing kink (and with it, the landing gear) is placed too far inboard, but this can be changed by altering settings listed in Table 4.4. Also the nose-gear is placed too far to the front of the nose. The fairing between the fuselage and wing, which houses the landing gear, is not modelled by the Initiator. This could have a small impact on the wetted area of the aircraft, which in turn affects the C_{D_0}.



(a) Top **(b)** Side **(c)** Front

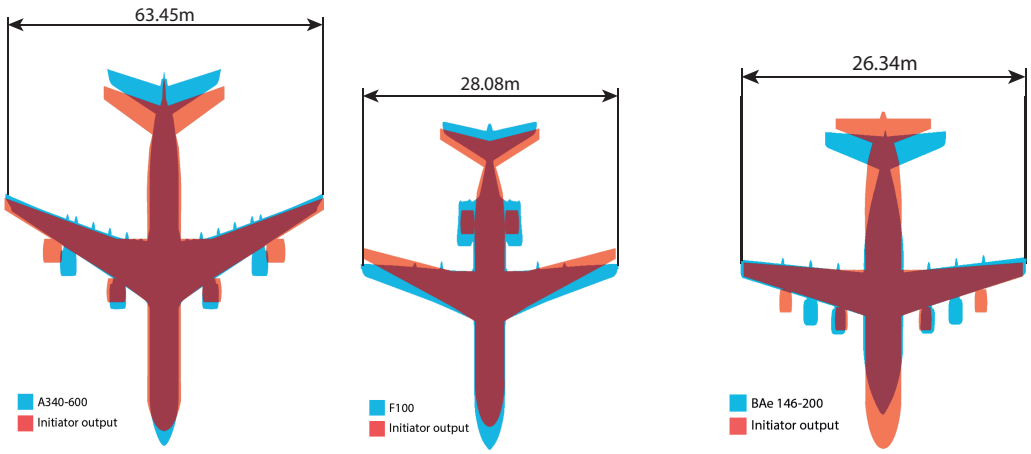
Figure 4.4: A320-200 geometry comparison

Figure 4.5: A320-200 comparison

Figure 4.6a shows the A340-600 top-view compared with the tool output. In contrast to the A320-200, is the fuselage length underestimated but it is slightly wider. The wing span is a bit larger when the winglets of the reference aircraft are not taken into consideration. Also the placement of the outboard engines is too far outboard, but this is easily fixed by changing a setting. Interesting to see is that the tail size is now overestimated, though the same design rule (Raymer's volume coefficient method [27]) is used for all aircraft.

In Figure 4.6b the Fokker 100 aircraft is compared, which is an aircraft with fuselage-mounted engines and a T-tail. As can be seen the wing sweep is overestimated. The wing sweep is calculated by using a simple relation between the cruise Mach number and the sweep angle. This is a very crude method which could be the cause of the difference. The fuselage length is underestimated. However, when inspecting the figure, it can be seen that the "real" F100 fuselage width is decreased between the engines. To account for this lost floor area, the fuselage is made longer. The engine are under-sized. This is simply because the same design rule is used for both wing mounted and fuselage mounted nacelles.

The British Aerospace 146-200 is a high wing aircraft with four engines. As can be seen the estimate of the wing is quite good, however the tail is again undersized. The fuselage is clearly overestimated (again probably due to the passenger density effect as discussed with the A320-200) and the outboard engines are again placed too far outboard.



(a) A340-600 **(b)** Fokker 100 **(c)** BAe 146-200

Figure 4.6: Comparison of top views

Table 4.5 lists the geometry parameters of the aforementioned aircraft. One reference aircraft of every aircraft configuration used in the tool verification was selected. The other aircraft show similar differences in geometry and are therefore not listed in the table.

Table 4.5: Comparison of geometry parameters

			A320-200		A340-600		BAE146-200		F100	
Dimension	**Description**	**Unit**	**Reference**	**Initiator**	**Reference**	**Initiator**	**Reference**	**Initiator**	**Reference**	**Initiator**
Wing										
S	planform area	m^2	122.4	122.8	439.4	440.4	77.3	73.4	93.5	88.3
b	span	m	33.9	34.1	63.5	63.7	26.3	25.8	28.1	27.4
λ	taper ratio	-	0.25	0.21	0.22	0.18	0.36	0.27	0.24	0.22
$\Lambda_{0.25}$	$1/4c$ sweep	$^\circ$	25	24.5	31.1	29.1	15.0	14.5	17.5	20.9
Fuselage										
L_f	length	m	37.6	40.7	73.5	66.6	26.5	31.0	32.5	34.2
D_f	diameter	m	4.1	4.2	5.6	6.5	3.56	3.2	3.3	3.6
Horizontal tail										
S	planform area	m^2	31.0	26.0	98.1	113.1	15.6	15.0	21.7	19.3
b	span	m	12.5	11.5	22.6	23.9	11.1	8.7	10.0	9.9
λ	taper ratio	-	0.33	0.35	0.39	0.35	0.41	0.35	0.39	0.35
$\Lambda_{0.25}$	$1/4c$ sweep	$^\circ$	28.0	27.4	29.9	32.6	20.0	16.2	26.0	23.4
Vertical tail										
S	planform area	m^2	21.5	18.3	51.4	75.1	11.6	10.9	12.3	12.6
b	span	m	5.9	5.4	8.8	11.0	4.9	3.3	3.3	3.6
λ	taper ratio	-	0.35	0.35	0.36	0.35	0.67	0.70	0.74	0.70
$\Lambda_{0.25}$	$1/4c$ sweep	$^\circ$	35.0	36.7	39.5	43.6	36.0	21.7	0.9	31.4
Engine pods										
L_e	length	m	4.4	2.9	4.7	5.7	2.6	2.1	5.1	2.5
D_e	diameter	m	2.4	1.8	3.1	3.6	1.4	1.1	1.7	1.5
Y_m	spanwise position	%	34.0	35.0	29.5 61.8	30.0 70.0	31.5 50	30.0 70.0	-	-

4.4 Drag polar comparison

The aerodynamic analysis in the Initiator consist of two different modules. First, the induced drag is estimated with AVL, a vortex-lattice method [28]. The profile drag is estimated with the method presented in Appendices F & G of Dr. Torenbeek's Synthesis of Subsonic Airplane Design [29]. The output of these two analyses are combined to create an estimate of the drag polar. Since there is currently no reliable high-lift estimation available in the Initiator, the changes of the drag polar for take-off and landing conditions are defined as a fixed increase in the span efficiency e due to flap deflections and a fixed increase in C_{D_0} due to flap and landing gear deflections. These values can be found in Table 4.3. An example of the resulting drag polars can be found in Figure 4.7.

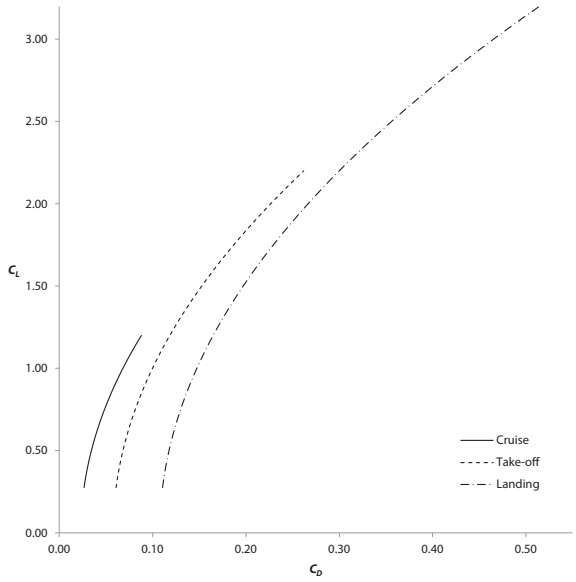
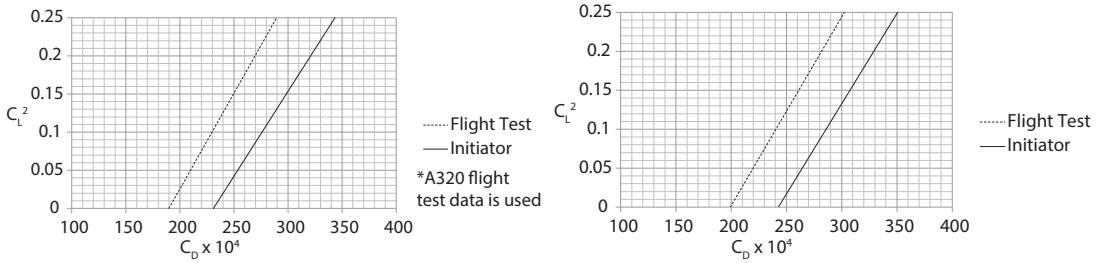


Figure 4.7: Drag polar output from the Initiator (A320-200)

In Figure 4.8 C_D - C_L^2 is plotted of aircraft for which flight test data is available. Overall can be seen that the zero-lift drag (C_{D_0}) is overestimates by an average of 25% with respect to the flight data. The slope of the C_D - C_L^2 lines is, with exception of the DC-10-30 estimated quite nicely. This means that the estimation of parameter k, ($k = \frac{1}{\pi A e}$) is accurate. The empirical drag estimation needs to be further investigated to solve this difference between the Initiator results and flight test data.



(a) C_D - C_L^2 Airbus A320-200*

(b) C_D - C_L^2 Boeing 737-200

(c) C_D - C_L^2 Boeing 737-800

(d) C_D - C_L^2 McDonnell Douglas DC-10-30

(e) C_D - C_L^2 McDonnell Douglas MD-80

Figure 4.8: Cruise drag polar comparison

4.5 Comparison based on Weight

<table>
<tr><td rowspan="2">Aircraft structure</td><td rowspan="2">Powerplant</td><td colspan="2">Airframe equipment and services</td><td rowspan="2">Operator items</td><td rowspan="2">Payload</td><td rowspan="2">Fuel</td><td rowspan="3">Main groups</td></tr>
<tr><td>Fixed equipment</td><td>Removeable equipment</td></tr>
<tr><td colspan="3">Basic Empty Weight</td><td colspan="4" rowspan="5">Typical Weights</td></tr>
<tr><td colspan="4">Manufacturers Empty Weight</td><td colspan="3">Load</td></tr>
<tr><td colspan="5">Basic Operational Empty Weight</td><td colspan="2">Useful load</td></tr>
<tr><td colspan="6">Zero Fuel Weight</td><td>Fuel Weight</td></tr>
<tr><td colspan="7">Maximum Take-Off Weight</td></tr>
</table>

Figure 4.9: Weight definitions (modified from: [14])

The conventional method of breaking up the aircraft weight is by identifying the operational empty mass (OEM), the fuel mass (FM) and the (maximum) payload mass (PLM) which together add up to the maximum take-off mass (MTOM). All the different weight definitions can be found in Figure 4.9.

As can be seen in Figure 4.9 the MTOM and OEM of nine of the thirteen reference aircraft are estimated with an error less than 10%. The maximum take-off mass and the operational empty mass are generally underestimated. These two weights are linked; a higher operational empty mass will also result in a higher take-off mass due to the snowball effect.

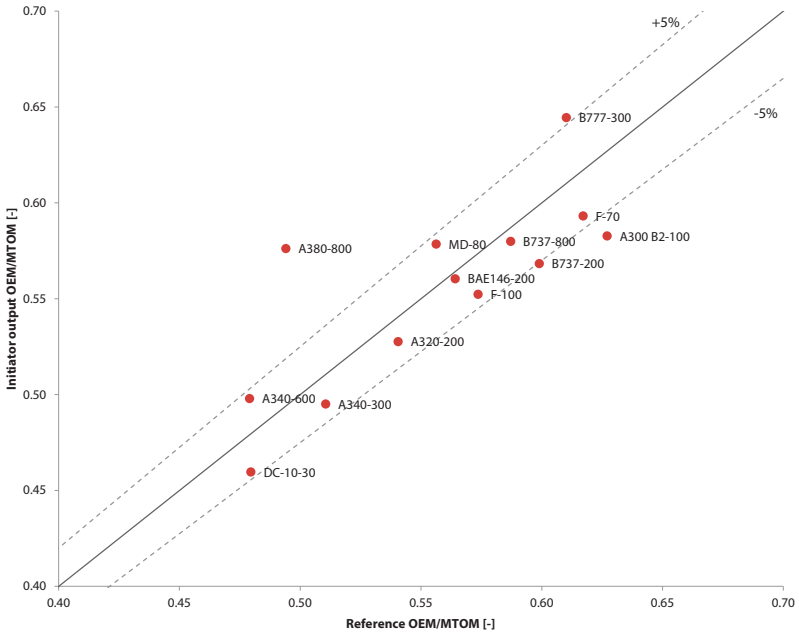


Figure 4.10: Comparison of the OEM/MTOM fraction of the generated aircraft with reference data from [13]

An interesting measure of how well the weight estimation performs is by comparing the $\frac{\text{OEM}}{\text{MTOM}}$ fraction of the reference aircraft with the Initiator output. In Figure 4.10 can be seen that the

$\frac{\text{OEM}}{\text{MTOM}}$ fraction is estimated quite accurately. Since the payload mass is a top-level requirement and therefore equal for both the reference aircraft and the Initiator generated aircraft, it can be concluded that the estimation of the fuel fraction is also accurate. (Since MTOM = OEM + PLM + FM) Of course the absolute value of the fuel mass will be underestimated due to the underestimation of the maximum take-off mass. However, this can not be tested directly, since the fuel masses for the harmonic range points are not known for the reference aircraft.

The operational empty weight is the sum of all aircraft structures, components and operational items. In Figure In Figure 4.13 can be seen that the fuselage is consequently over-estimated (average: 5% of the MTOM, 11% to 16%; overestimated by 43% w.r.t. reference mass), where the wing weight is underestimated (average: -4% of MTOM, 12% to 8%; underestimated by 32% w.r.t. reference mass). These two effects cancel each other out, which results in an operational empty mass estimation which is quite accurate, as is seen before in Figure 4.10. Note that, because of the cancelling effect improvement of the fuselage or wing weight estimation could result in a worse operational empty weight prediction. However, this should not be a reason to refrain from improving the estimation methods.

Overall can be concluded that the weight estimation implemented in the Initiator performs quite well; the aircraft synthesised by the design tool show comparable weights as the reference aircraft designed for the same top-level requirements.

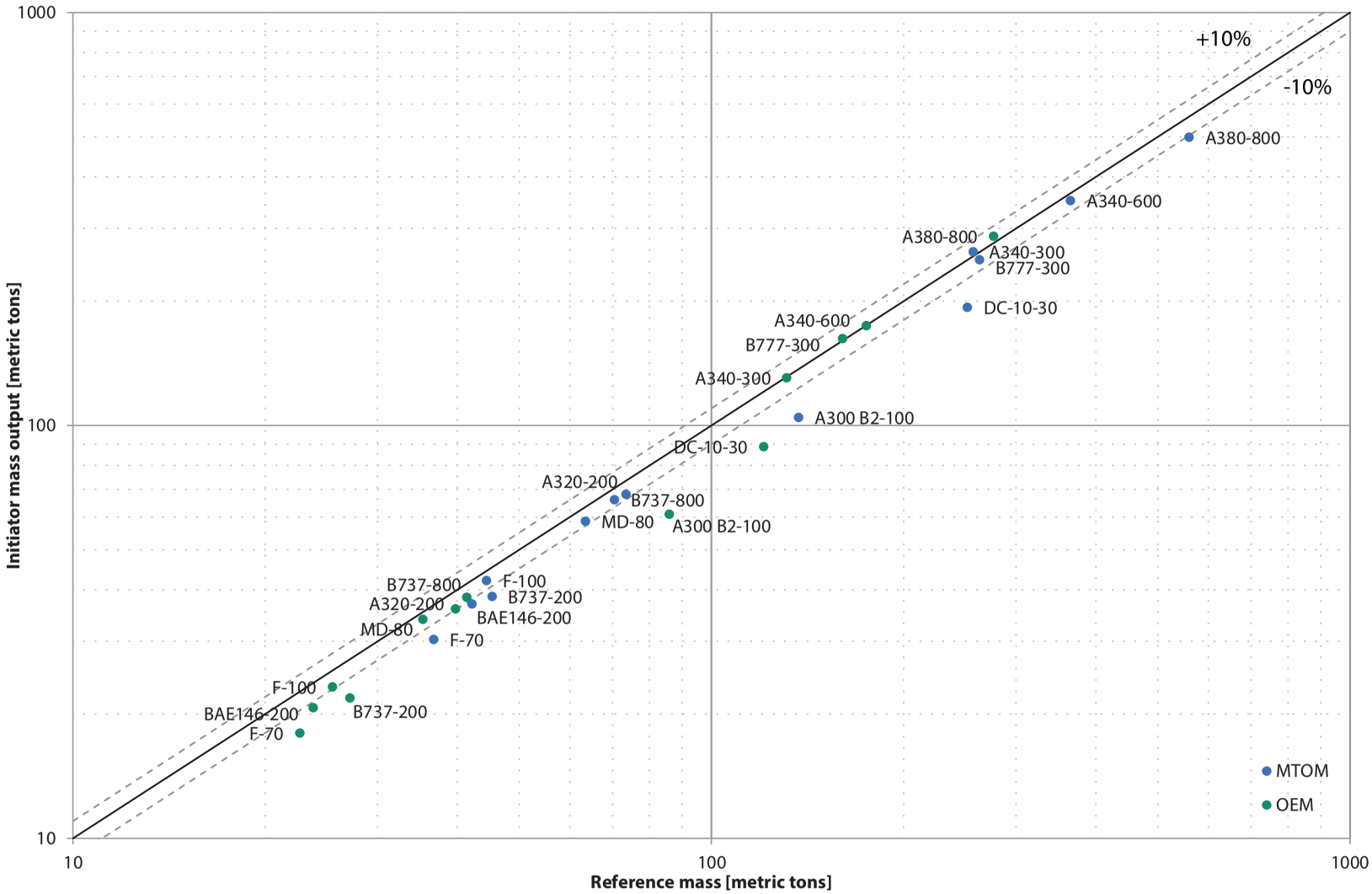
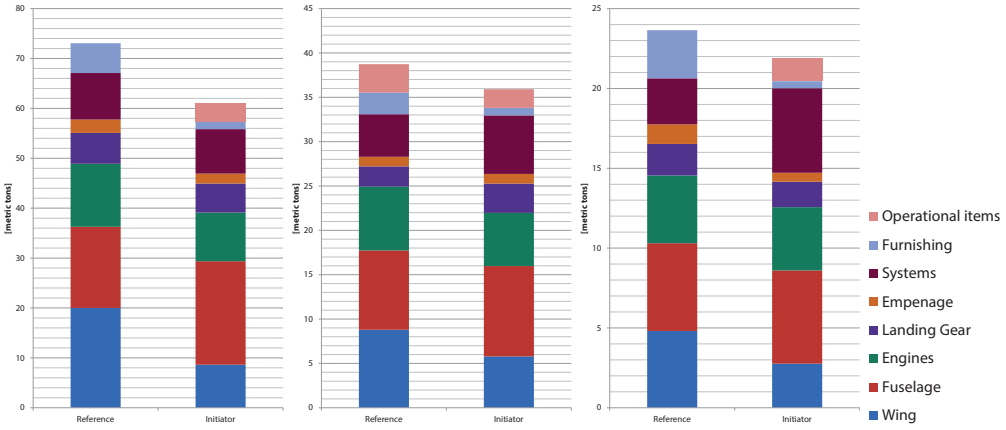


Figure 4.11: Comparison of the maximum take-off mass (MTOM) and operational empty mass (OEM) calculated from the Initiator with reference aircraft weight data from Élodie Roux [13]



(a) A300 B2-100 **(b)** A320-200 **(c)** B737-200

Figure 4.12: Comparison of the weight breakdown

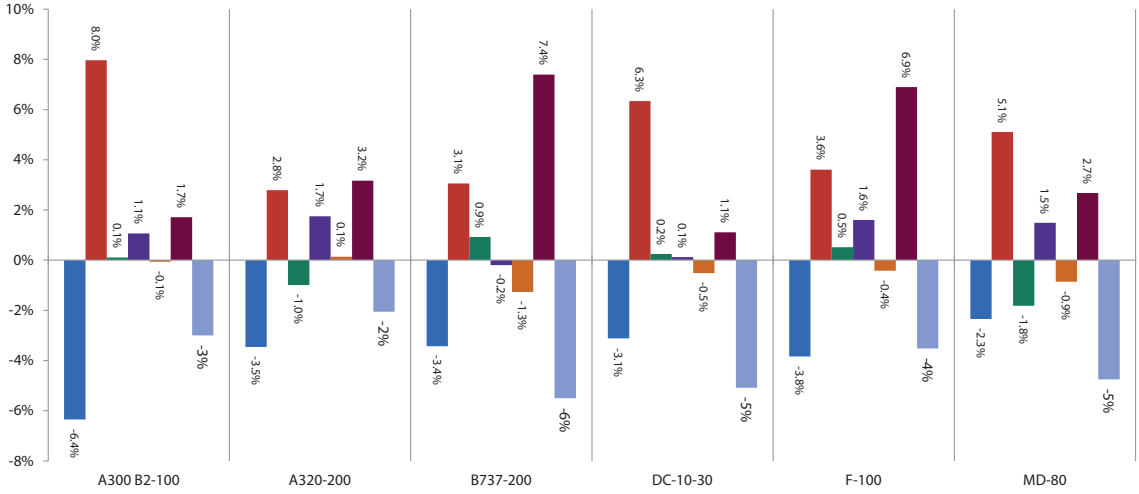
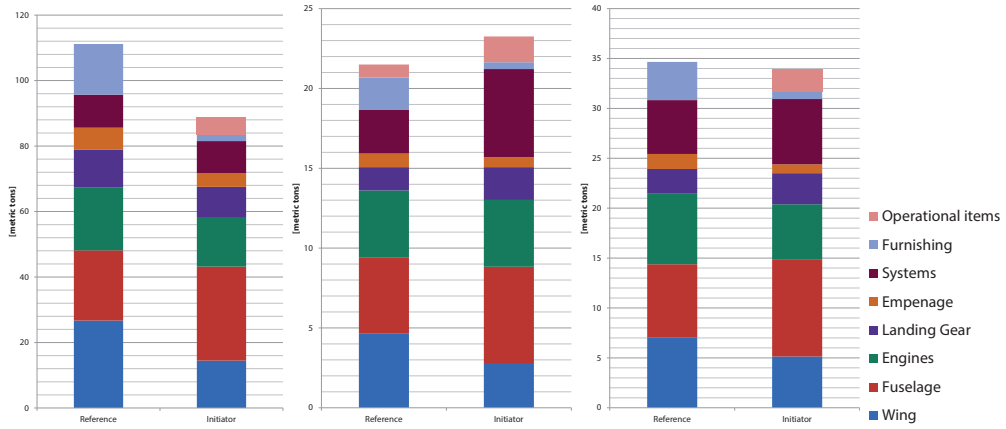


Figure 4.13: Difference in weight fraction of the maximum take-off mass of the Initiator generated aircraft compared with reference data from [14] and [15]



(a) DC-10-30 **(b)** F-100 **(c)** MD-80

Figure 4.14: Comparison of the weight breakdown (continued)

4.6 Conclusions

From this chapter can be concluded that the aircraft generated by the Initiator match the reference aircraft quite well.

From the difference in design point between the Initiator output and the reference aircraft can seen that a good estimation of the maximum lift coefficients is crucial for the design tool to find the right design point. However, this estimation functionality is currently not available in the Initiator.

A few generated aircraft have a high cruise lift coefficient, which resulted in a high vortex drag due to the small wing planform. Real aircraft could encounter buffet onset at these high lift coefficients during cruise therefore a wing-loading constraint needs to be added which takes these effects into account.

The geometry estimation performed good, because they are based on rather simple design rules variations from the reference aircraft can be expected. The horizontal and vertical tail prediction methods currently use volume coefficient methods, which defines the tail area as a function of the main wing area and the distance between the aerodynamic centres of the surfaces. These methods could be improved by taking stability and controllability into account. This would also benefit the estimation of control surfaces on unconventional aircraft, since this is less reliant on empirical data.

The fuselage sizing could be improved by creating a more accurate representation of the cabin and fitting a fuselage around it. Something similar is done by P. van der Linden in this Master's thesis [39]. Here T. Langen's Initiator [17] is used where the cabin design is offloaded to DARFuse, which sizes the cabin by taking a lot of design choices into account.[2]

The aerodynamics modules need to be improved. Since quite low-fidelity aerodynamic tools are used and there is a geometric difference between the Initiator generated and the reference aircraft an exact match cannot be expected. However, the on average 25% overestimation of the zero-lift-drag should be investigated in future versions. A more sophisticated profile drag estimation may be needed, also because the current tool is only applicable to conventional aircraft configurations.

The weight estimation methods perform good, nine out of the thirteen aircraft's operational empty weight and maximum take-off weight are estimated to within 10% of the reference aircraft weights. Currently the main wing and fuselage are estimated with Class II.V methods, which are less reliant on empirical data and use load cases to size the main structures. This approach could be extended to also size the tail surfaces. The wing weight is currently underestimated by (on average) 32% (4% of MTOM); the fuselage is overestimated by (on average) 43% (5% of MTOM). This needs to be investigated to improve the accuracy of the weight prediction.

The different disciplines (aerodynamics, weight, etc.) in the design process are very closely coupled. For example a slight change in the aerodynamics could have great influences on the structures. To verify and, if possible, validate the modules they need to be tested separately. In other words, testing a module which output could influence its own input in the next iteration (practically anything in a design loop) cannot be validated in-place. The analysis modules need to be validated separately for known cases and the bounds for which the tools are valid need to be identified. This information can then be used to increase the confidence in the design tool output.

[2] Unfortunately this could not be implemented into the design tool discussed in this Thesis, because the version of DARFuse was built on an old version of GDL, which did not run on the author's computer. The new versions of DARFuse (built on a newer version of GDL) lacked the XML read/write capability needed to implement the connection between MATLAB and GDL

Chapter 5

Configuration Comparison

In this Chapter the Initiator is used to compare different aircraft configurations by designing them for varying payload-range requirements and deriving key performance indicators from the designed aircraft.

First the top level requirements are determined in Section 5.1. With these requirements the aircraft are generated by the Initiator. This results in 176 converged aircraft configurations, which can be seen in Section 5.2. The output is analysed and key performance indicators (KPIs) are calculated for every aircraft. The different KPIs used are presented in Section 5.3. The results are presented in Section 5.4.

5.1 Top level requirements

The top level requirements specify the mission and performance characteristics for which the aircraft is designed. Two main top level requirements are the payload mass and range. An other design variable which can be chosen freely in the Initiator is the wing aspect ratio. The upper and lower bounds on the aspect ratio are set to 7 and 11, since these are roughly the minimum and maximum values found in the reference aircraft used in Chapter 4. The payload and range bounds cannot be set by simply setting minimum and maximum values, since designing an aircraft for a large payload on a short range or a long-range aircraft with a small payload would create infeasible designs with the currently implemented methods. The top-level requirements are specified for the following aircraft configurations:

- Conventional aircraft
- Canard aircraft
- Three-surface aircraft
- Prandtl aircraft

Unfortunately the sizing methods of the Blended-Wing-Body aircraft are not robust enough to be used in the configuration comparison. Currently the BWB sizing methods only work for aircraft

with a very big payload, which does not result in a converged and feasible design with the current methods.

The design space on the range versus payload mass plane is chosen by selecting the minimum and maximum payloads and ranges of the reference aircraft selected in Chapter 4. Two lines are drawn between the origin and reference aircraft which result in the largest angle between the lines. The intersection point between the two slanted lines and the payload and range bounds are used to create a convex quadrilateral which enables indexing of the design space with two variables. The process can be seen in Figure 5.1.

Inside this prism-shaped design space design points are defined using latin hypercube sampling, which makes sure that the largest variety of variable combinations is chosen for the given amount of design points. For more information about these sampling methods, please read Slingerland's Master's thesis [30]. The design space and the design points can be seen in Figure 5.2. To limit the computation time (around 12 hours on an Intel i7-3610QM CPU) 50 design points per configuration are chosen, which results in a total of 200 aircraft.

Table 5.1 list the remaining requirements used in the configuration runs. The coefficients a_{to} and a_l used to calculate the landing and take-off distances are determined using a quadratic fit between the harmonic range and the distance of the reference aircraft from Chapter 4.

Table 5.1: Requirements used for the configuration comparison runs

Parameter	Value
$C_{L_{\text{max,clean}}}$	1.2
$C_{L_{\text{max,takeoff}}}$	2.2
$C_{L_{\text{max,landing}}}$	2.8
Landing distance	$a_{to} \cdot R^{0.5}$
Takeoff distance	$a_l \cdot R^{0.5}$

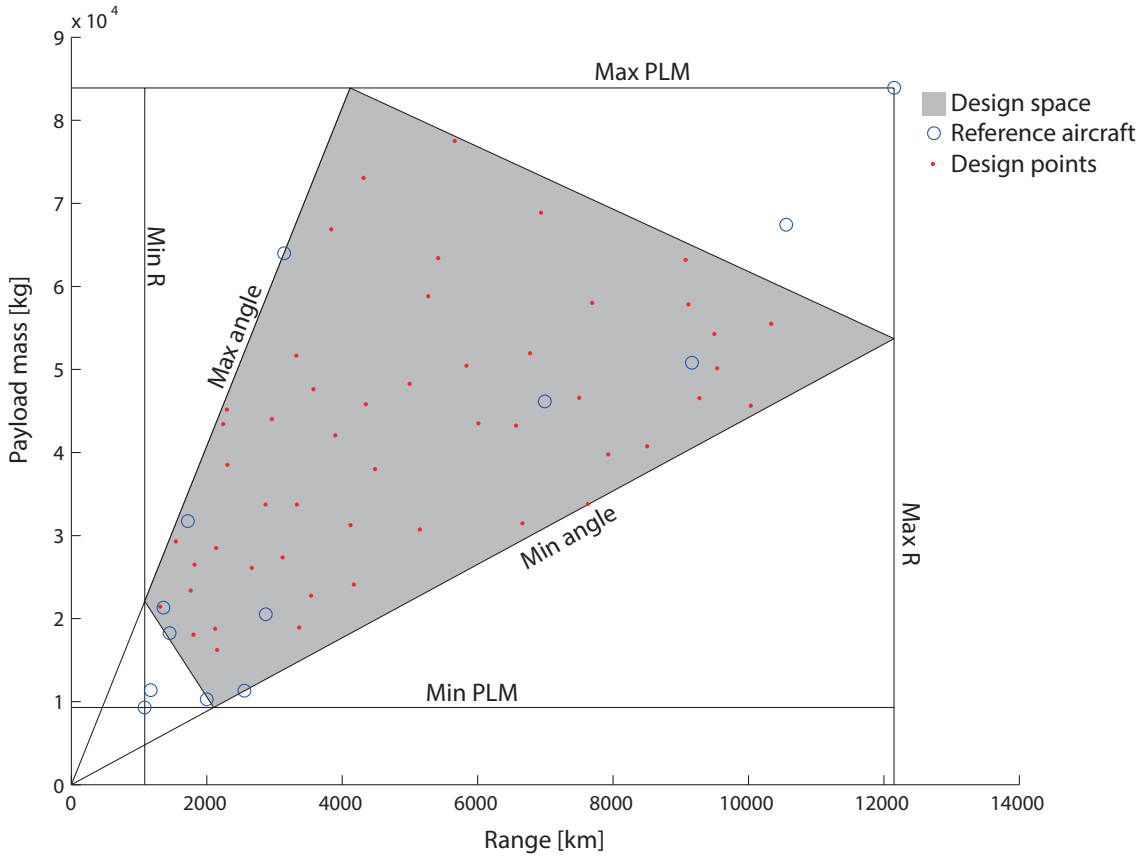


Figure 5.1: Definition of the design space and design points

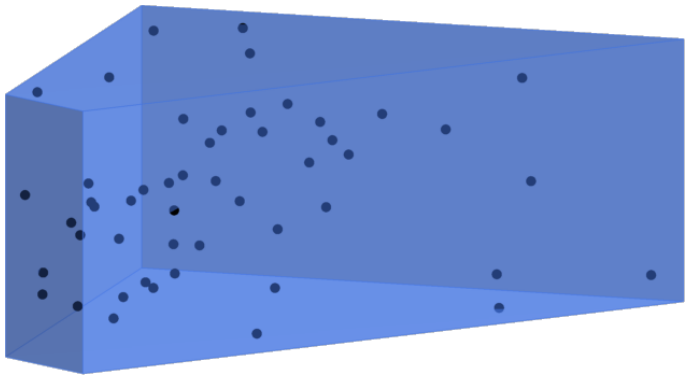


Figure 5.2: 3D-view of the design space

5.2 Design synthesis

For every design point the *DesignConvergence* module, as described in Section 3.2, is used. This design is completed for every configuration. In Figure 5.3 the different design points are shown. Some payload-range-aspect ratio combinations did not converge. In the case of high aspect ratio wings the wing weight estimation method sometimes does not converge, which would crash the weight estimation module. A too small wing aspect ratio would result in a too small fuel tank, which results in an aircraft which does not converge.

Figures 5.4 shows the 3-dimensional geometry of all generated aircraft. Note that these figures are not to scale and only serve to show the multitude of generated aircraft.

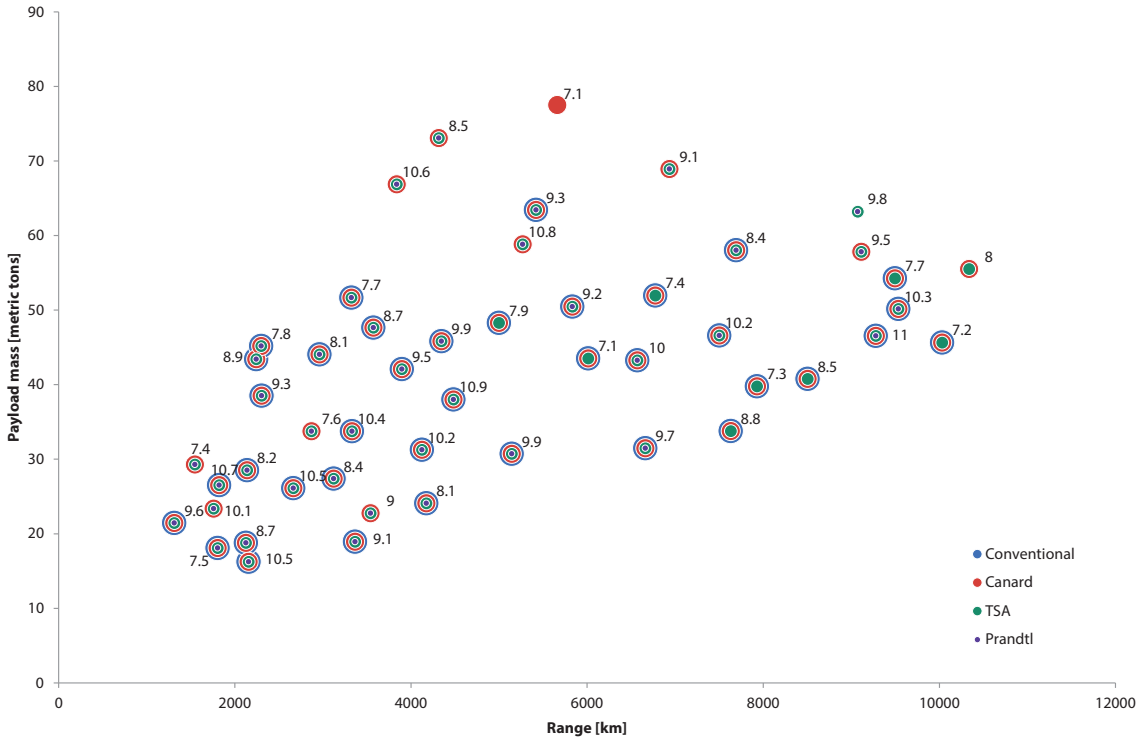


Figure 5.3: Payload - range combinations of the design runs, aspect ratio written next to the point; Missing points did not converge

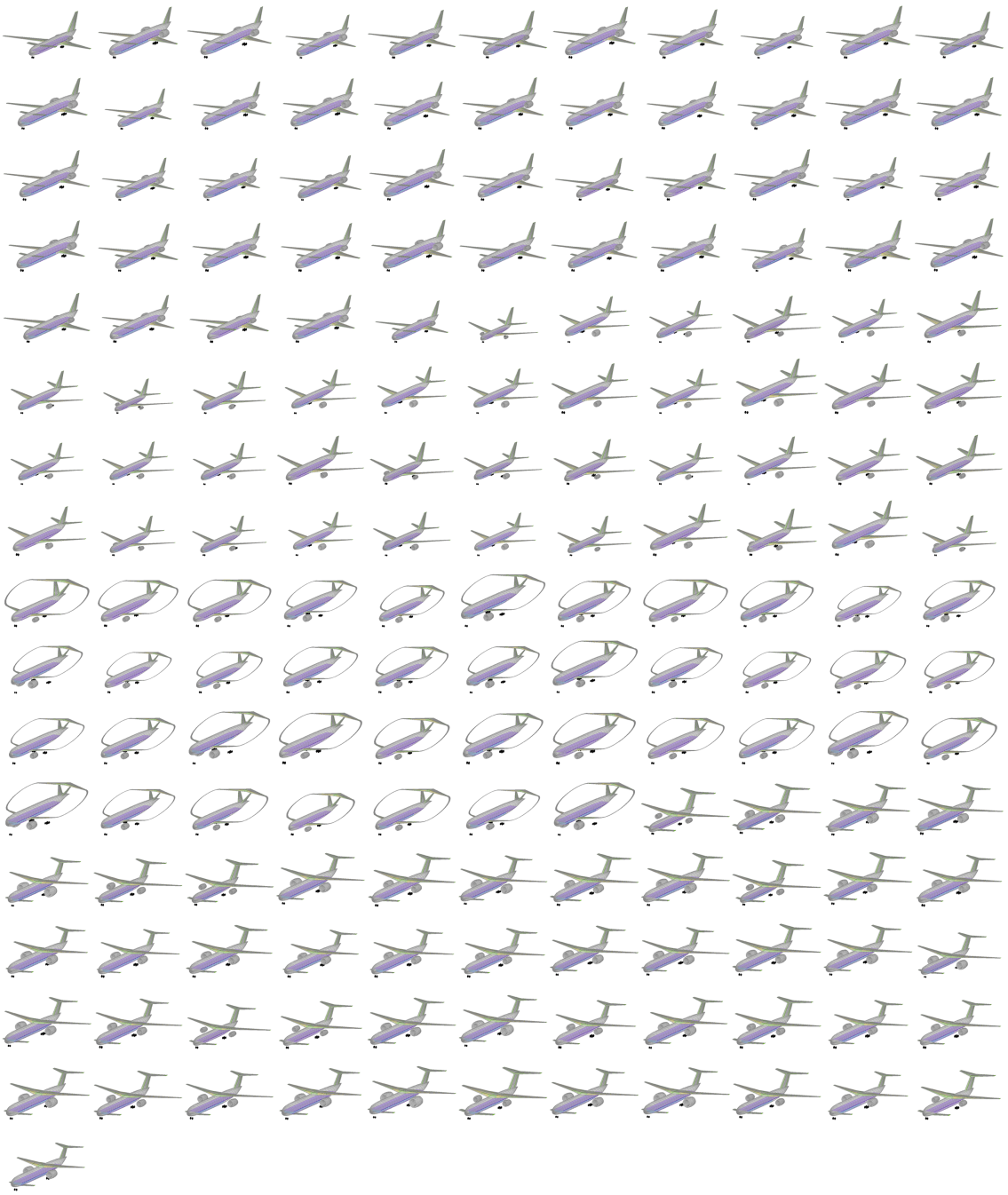


Figure 5.4: 3D renders of the generated aircraft

5.3 Key performance indicators

Key performance indicators need to be chosen to compare the different aircraft configurations. One important metric is the maximum take-off mass (MTOM). This gives an overall indication of the aircraft operating cost, since a lot the cost post are take-off mass related.

Secondly the operational empty mass (OEM) is used since this can be an indication of the aircraft purchase price, as can be seen in Figure 5.5. Note that is only valid for aircraft built at the same technology level; a weight-reduction due to more advanced materials would probably not result in a decreased purchasing price.

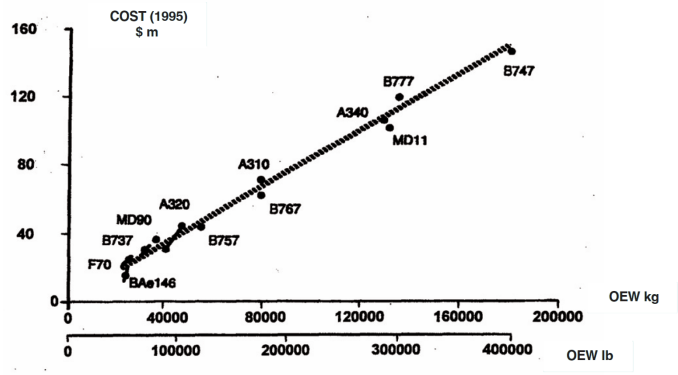


Figure 5.5: Relation between the aircraft purchase price and the Operation Empty Weight (source: [16])

The payload range efficiency (PRE) is a indication of the fuel efficiency of the aircraft and is defined as:

$$PRE = \frac{W_p \cdot R}{W_f} \tag{5.1}$$

where W_p is the maximum payload weight, R is the aircraft harmonic range and W_f is the fuel weight associated with this mission. A higher payload range efficiency indicates a lower fuel usage and therefore less noise and emissions.

The range parameter X is the first part of the famous Breguet range equation:

$$R = \frac{V \cdot L/D}{c_T} \cdot \ln \frac{W_1}{W_2} \Longrightarrow X = \frac{V \cdot L/D}{c_T} \tag{5.2}$$

where V is the aircraft speed, L/D is the lift-to-drag ratio and c_Ts the specific fuel consumption. This range parameter is a good figure of merit for the aerodynamic and propulsive efficiency of the aircraft.

5.4 Results

After the runs are completed the results are interpreted which results in a four-dimensional scattered dataset for every key performance indicator. To create continuous plots, the data is linearly interpolated.

In Figure 5.7 the selected KPIs are plotted for an conventional aircraft with an wing aspect ratio of 9 (average of the aspect ratio used to generate the aircraft). Other configurations and aspect ratios show similar trends, only the absolute values (and rates of change) differ. From Figure 5.7 can be seen that the maximum take-off mass, operational empty mass as well as the range parameter increase significantly with increasing payload mass, but are less dependent on the harmonic range. The fuel mass is more influenced by the range, as well as the payload.

In Figure 5.8 the KPIs for every configuration are plotted for increasing payload mass and harmonic range. The horizontal axis describes an increase in payload mass and harmonic range. Every point on the horizontal axis represents an unique payload versus harmonic range combination. The path through the payload-range design space can be seen in Figure 5.6.

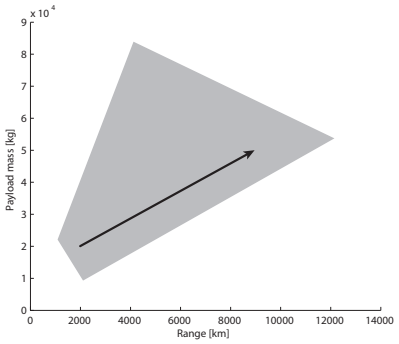


Figure 5.6: Payload mass as a function of the range as used in Figure 5.8

The coloured bands in Figure 5.8 indicate the margin of the KPI which altering the wing aspect ratio can influence. In other words: the boundaries of the coloured bands are the minimum and maximum values of the KPI which can achieved by varying the wing aspect ratio. Note that the data is not smooth; this can be improved by increasing the number of generated aircraft, at the cost of increased computation time.

In Figure 5.8a the maximum take-off mass of the different configurations is compared. As can be expected, the maximum take-off mass increases with increasing payload and range for every configuration. Interesting to see is that the conventional and the conventional aircraft and the three-surface aircraft have similar take-off masses and also the canard aircraft and the Prandtl aircraft have comparable masses. The maximum take-off masses differ on average 30 tons, which is quite considerable since this is roughly 25% of the maximum take-off mass.

Since MTOM = OEM + FM + PLM and the payload masses are the same for all configurations, the weight reduction must be in the operational empty mass and/or the fuel mass. As can be seen in Figures 5.8b and 5.8c both show a considerable reduction in mass for the canard aircraft and the Prandtl aircraft. In Figure 5.8b can also be seen that the three-surface aircraft has a reduces operational empty mass compared with the conventional aircraft. However, the fuel mass is higher for the three-surface aircraft until an harmonic range of about 8000km. In terms of fuel mass the canard aircraft has consistently the advantage over the other aircraft configurations.

The payload-range efficiency, shown in Figure 5.8d, complies with the previous figures. It shows that the efficiency of the canard aircraft and Prandtl aircraft is higher than the conventional aircraft and the three-surface aircraft, with an average difference of 1200km. The canard has clearly the highest payload-range efficiency and the conventional aircraft performs slightly better than the

three-surface aircraft until 8000km. Note that the three surface aircraft's PRE slope clearly increases with increasing range while the slope of all other lines are almost constant. This is more clearly visible in Figure 5.8f where the data in Figure 5.8d is fitted with a second-order polynomial.

In Figure 5.8e the range parameter is plotted. Surprisingly the conventional aircraft performs the best with exception for the very small ranges and payloads, where the canard aircraft performs slightly better. Again, the three-surface aircraft starts off as the worst performing aircraft for small ranges and payloads, but increases rapidly to the point where it outperforms the other configurations (with exception of the conventional aircraft) for the large ranges and payloads.

Since the conventional aircraft outperforms the other configurations in terms of the range parameter, it can be concluded that the other concepts do not get their advantage from increased aerodynamic performance (since $X = \frac{V \cdot L/D}{c_T}$). The payload-range efficiency is higher, which indicates a lower fuel mass, which is a result of the decreased operational empty mass due to the so-called snowball effect inherent to the aircraft design process.

The decreased operational empty mass could be a result of lower structural loads. In the case of the canard aircraft the root bending wing moments on the wings in lower because both the main wing and the trimming surface are both lifting: the main wing does not have to compensate the download normally present due to trimming. The Prandtl aircraft has less bending moments on the fuselage, since it distributes the weight over the two wings and therefore introduces the lifting loads more gradually into the fuselage in comparison to other configurations. Note that the Prandtl aircraft has an unfair advantage in the current implementation of the Initiator since the surfaces connecting the two parts of the box-wing are not included in the weight estimation. They are included in the aerodynamic analysis, so the loads generated by the surfaces are used in the Class II.V wing weight estimation, which results in a slightly increased wing weight.

The three-surface aircraft should also take advantage of the fact that all surfaces are lifting, which would give it the same structural advantage. This absence of could have two causes:

1. The additional weight of the extra canard surface does not weight up against the weight reduction gained by lower structural loads.

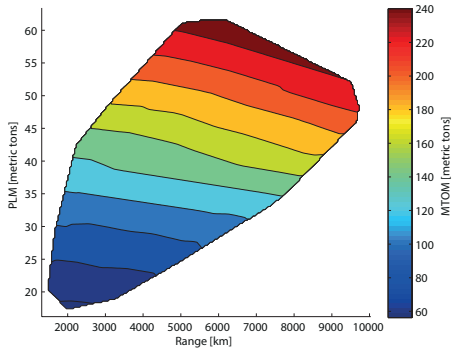
2. Trimming the three-surface aircraft is a complex problem, since there are a lot of different combinations of trimming surface angles which could trim the aircraft. Ideally the combination which results in the lowest trim drag should be selected. Currently this is not the case, the canard is fixed at a positive incidence angle, and the tail surface is used to trim the aircraft.

The trimming functionality of the Initiator is currently being improved as part of C. Huijts's Master thesis [37]. After the trimming routines are fully integrated in the design tool, the performance of the tree-surface aircraft should be re-calculated.

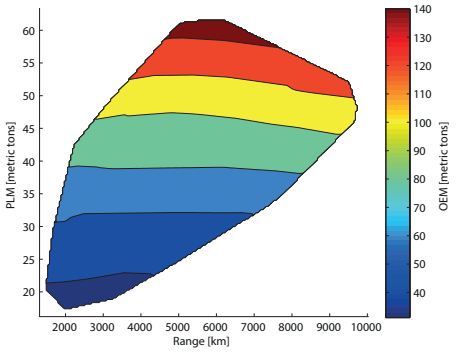
Table 5.2 list the percentual changes of each KPI with respect to the conventional aircraft.

Table 5.2: Perceptual change of each configuration with respect to the conventional aircraft

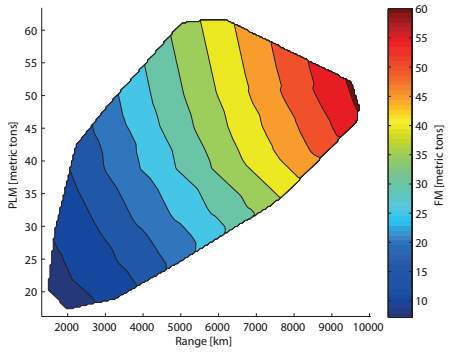
Aircraft	MTOM	OEM	FM	PRE	X
Canard	-22%	-28%	-12%	15%	-14%
Three-surface aircraft	-8%	-10%	3%	-4%	-20%
Prandtl aircraft	-26%	-31%	-8%	9%	-26%



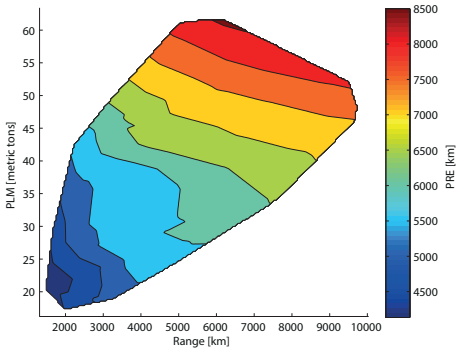
(a) Maximum take-off mass (MTOM)



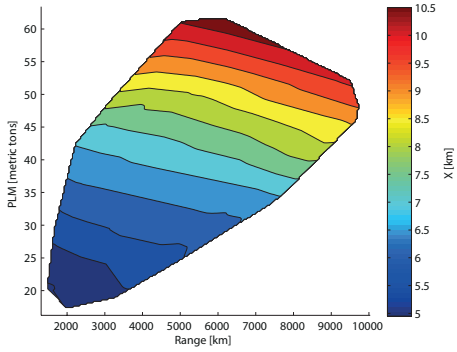
(b) Operational empty mass (OEM)



(c) Fuel mass (FM)

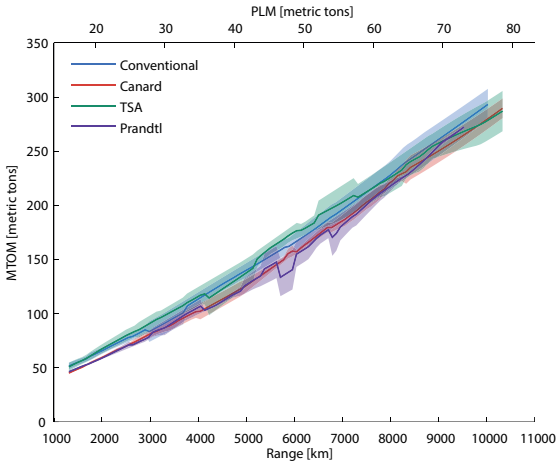


(d) Payload range efficiency (PRE)

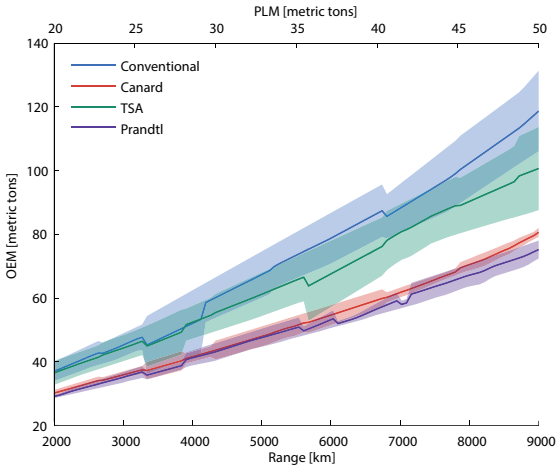


(e) Range parameter (X)

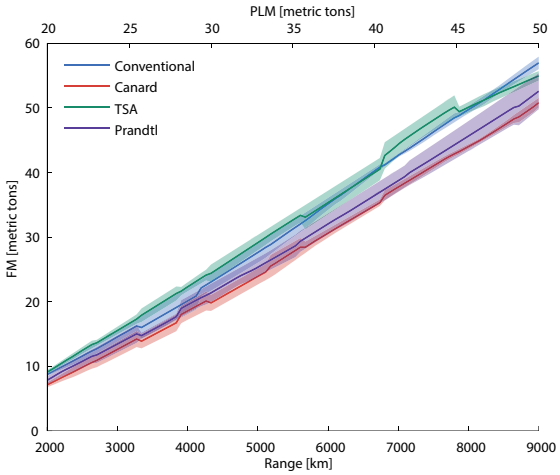
Figure 5.7: Contour plot of the KPIs for the conventional aircraft with an aspect ratio of 9



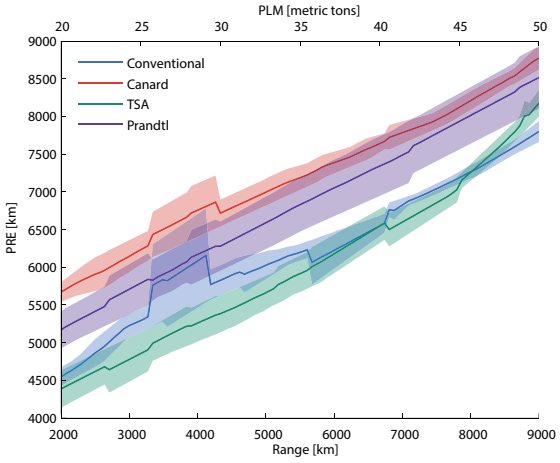
(a) Maximum take-off mass (MTOM)



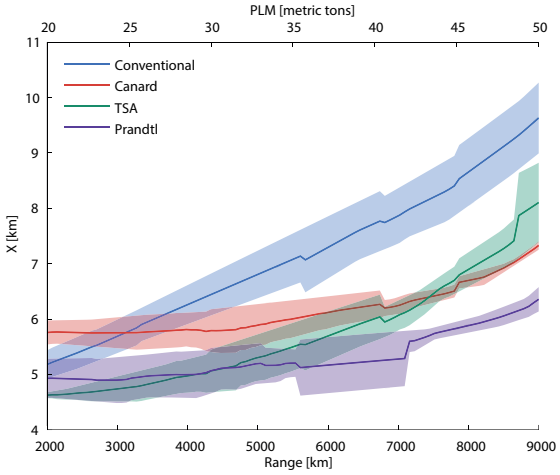
(b) Operational empty mass (OEM)



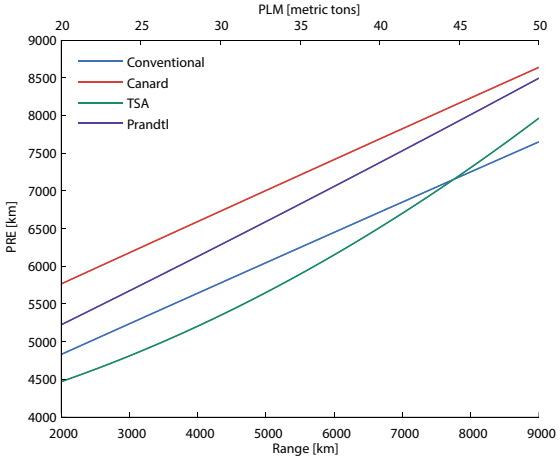
(c) Fuel mass (FM)



(d) Payload range efficiency (PRE)



(e) Range parameter (X)



(f) Payload range efficiency (PRE) fitted with a second order polynomial

Figure 5.8: Comparison of KPIs for the different aircraft configurations, coloured bands show influence of the aspect ratio on the parameters

Chapter 6

Conclusions and Recommendations

6.1 Conclusions

The goal of this thesis was to develop an automated design environment for the synthesis and analysis of conventional and unconventional aircraft configurations. The design tool is used to answer the research question: *"Which aircraft configuration has the potential to introduce a significant increase in fuel efficiency?"*

The design tool, the Initiator, implements a design process which results in a feasible aircraft design which complies to a given set of top level requirements. The Initiator is verified by comparing the output of the Initiator with existing aircraft.

The design process is proven to work, since it converges to a feasible aircraft design which complies with the top level requirements. Since process contains design loops, inaccuracies in analyses propagate easily through the whole design. By comparing the maximum take-off weights and operational empty weights, it can be shown that the generated aircraft are similar to the reference aircraft. Nine out of the thirteen generated aircraft are estimated to within 10% of the reference aircraft weights. With exception of the Airbus A380-800, all aircraft weights are either estimated correctly or slightly underestimated.

The trust-to-weight ratio is overestimated on a number of aircraft. Further investigation shows that this is caused by the absence of a constraint on the cruise lifting coefficient. The high lift coefficients result in a high vortex drag, which results in unfavourable aerodynamic performance. This happens mostly with long-range aircraft. As it turns out, the landing distance is no longer the active constraint on the wing loading for these aircraft. Currently additional constraint are developed (for example buffet onset during cruise) to avoid these high lift coefficients.

From the verification can be concluded that the geometry estimation performs quite well. Note that exactly matching the reference aircraft was never a goal of the Initiator and cannot be expected since an aircraft design problem has no unique solutions. Unfortunately creating a sizing method which generates a reasonable Blended-Wing-Body aircraft for a large variety of payload and range requirements showed to be rather difficult and need further investigation.

The Class II.V weight estimation of the fuselage overestimates on average by 43% (which is around 5% of the MTOM). The wing weight estimation underestimates on average by 32% (4% of MTOM). In the total weight estimation these two inaccuracies neutralise each other, which is currently beneficial for the design tool. Of course, this should not be a reason to refrain from improving the estimation methods.

By comparing the drag polar of an Airbus A320-200 with the vortex-lattice model it is shown that AVL can correctly estimate the drag polar (within 8 drag counts at $C_L = 0.5$), provided that the zero-lift drag is correctly estimated. Since a vortex lattice method is used, it is incapable of correctly model wing thickness effects. Therefore the accuracy of predicting the aerodynamic performance of the Blended-Wing-Body aircraft in unknown.

The profile drag is calculated with an empirical method, therefore its validity for the drag estimation of unconventional aircraft is unknown. Besides this it is shown that the profile drag is consistently overestimated by an average of 25%. Comparing the tool output with data from the Fokker 100, the zero-lift drag is underestimated by 14% when the exact geometry of the aircraft is inserted into the tool. The additional underestimation (11%) could be due to geometric differences between the 'real' and the Initiator generated aircraft.

Overall can be concluded that it is possible to model a wide range of different aircraft configurations using comparable sizing rules and analysis methods. It is shown that a design process which iterates on the aircraft maximum take-off weight and adjusts the fuel mass to match range requirements works for conventional, canard, three-surface and Prandtl aircraft. Testing the design process for the Blended-Wing-Body was unfortunately not possible with the current state of the sizing methods. Verification with existing aircraft data showed that the design tool performs as expected, differences could be traced back to the responsible modules and are listed in the recommendations below.

The design tool was used to compare the different aircraft configurations. It is shown that the canard aircraft provides a 12% reduction in fuel mass and a 28% reduction in operational empty mass in comparison with a conventional aircraft designed for the same payload and harmonic range. However, since none of the analysis methods have been validated the confidence in the results gained from the configuration comparison in low. Also the Prandtl aircraft shows promising results (-8% fuel mass and -31% operational empty weight), but since this configuration is more unconventional than the canard aircraft, the uncertainty is probably even higher.

6.2 Recommendations

From the conclusions and other insights gained during the thesis recommendations are derived for improvements to the design tool and further research.

All used analysis tools should be validated. Since the tools are used in a design loop, errors propagate very quickly and an error in one module could alter the whole design of the aircraft.

The Prandtl aircraft is currently sized with design rules which are similar to those used for conventional aircraft. A parametric study should be done to investigate the effect of the box-wing design on the aircraft performance.

The sizing of the Blended-Wing-Body aircraft is not capable enough to be used in an aircraft design loop. Currently only reasonable geometries are created for very high payload requirements. A

method of sizing the fuselage needs to be developed, since the current method of specifying a floor area and slenderness is simply not adequate for the fuselage design of the Blended-Wing-Body aircraft.

EMWET, the Class II.V weight estimation method used for the wings has trouble with wings with a high aspect ratio. Also the box-wing poses a real challenge for the analysis tool. The 'Student version' is currently used by the Initiator. The differences with the full version should be investigated and the tool needs to be adapted to be able to analyse unconventional wing shapes. The overestimation of the Class II.V fuselage weight should also be investigated. The weight of winglets and the connectors of the Prandtl wing are currently not included in the weight estimation. A weight prediction method needs to be developed for these parts.

The empirical drag estimation as currently implemented overestimates the zero-lift drag. When using the method as only aerodynamic estimation method (the way Torenbeek intended) it predicts the overall drag polar quite good. But since only the profile drag component (which would normally be compensated with an underestimated vortex drag) is used in the Initiator, it results in an overestimation of the total drag.

All currently implemented high-lift modules are either invalid for unconventional aircraft or give unreasonable results. The $C_{L_{\max}}$ of the clean wing is greatly overestimated. The high-lift devices module as currently implemented is only valid for conventional aircraft. This module should be modified to be able to model high lift devices on all implemented aircraft configurations. Once this is developed, the runway performance could be evaluated and included in the design process.

A module should be created which evaluates the aircraft stability and controllability. A more capable sizing method for tail and control surfaces should be considered instead of relying on rather crude volume coefficient methods. Also integrating effects of advanced control systems is an area a lot of research could be done.

The Initiator could benefit from a good transonic aerodynamic analysis tool. Currently there is no estimation available for the wave drag of transonic aircraft. Also modules covering aircraft cost, noise and emission still need te be developed.

In the current implementation the Initiator features a Blended-Wing-Body aircraft with an oval-shaped fuselage. A multi-Bubble fuselage needs to be added. The methods could be based on work previously done by van Dommelen [19].

Currently the sizing modules generate one aircraft based on the weight, wing loading and a set of design rules. To be able to integrate the tool with an optimiser, functionality should be added to create the possibility of handing over design variables and configuration parameters to an external module.

The Initiator could be extended to analyse other aircraft configurations. Looking back to Figure 1.3, there are still a lot of aircraft configuration currently unimplemented in the design tool.

The Initiator is currently programmed in the MATLAB environment. This allowed the rapid development of the program. However, MATLAB is a closed system and using it puts you at the mercy of the development plans of the manufacturer. During the development a lot of compatibility issues between different releases have been discovered. It could be beneficial to re-implement te program in a open and standardised programming language. One example could be Python, which together with the NumPy and SciPy packages[1] can provide an platform which is comparable, if not more capable than MATLAB .

[1] http://www.python.org, http://www.numpy.org, http://www.scipy.org

Part II

Code documentation

Chapter 7

Introduction

This document describes the functionality and implementation of a conceptual aircraft design tool called *The Initiator* described in Part I.

This Chapter gives some insight in the background from which the application is developed. Chapter 8 draws an outline of the structure of the program. Here the composition of the different components and their implementation is shown.

Chapter 9 presents the aircraft parts also known as the *High Level Primitives*. The different parameters which are required to define the parts and the methods of generating the geometry are elaborated.

In Chapter 10 the installation and ways of operating the program are explained. Also the structure of the files which are needed to operate the application is presented.

7.1 Background

The main use case of the application is the synthesis of a preliminary design of a subsonic transport aircraft from top-level requirements. A secondary use case is the evaluation of an already defined aircraft.

Besides conventional aircraft configurations, the program is designed to create and analyse designs for Canard aircraft, Three-Surface-aircraft, Prandtl-planes and Blended-Wing-Body aircraft. Since no or very little empirical data exists on these unconventional configurations, preference is given to methods which are physics-based or at least sensitive to geometry changes; within the possibility of achieving a reasonable runtime on current computer hardware.

The application is based on the conventional and Prandtl aircraft initiator written by T. Langen [17], later extended for Three-Surface aircraft design by Vaessen [10] and the Blended-Wing-Body Conceptual Design Tool written by J. van Dommelen [40] and M. Hoogreef [20].

The Initiator can be used as a stand-alone application, but is also supposed to be able to integrate into the *Design and Engineering Engine* (DEE) developed at the Delft University of Technology.

This creates the possibility to generate an aircraft from top-level requirements with the Initiator and analyse it with high-fidelity tools in the DEE. At the time of writing this whole tool-chain has not been fully developed and is out scope of this document.

Chapter 8

Program Structure

8.1 Introduction

The Initiator is written in MATLAB since this environment is widely used at universities and various analysis methods are already implemented in this language. This choice has the advantage that new students can easily extend and modify the code without learning an additional programming language, but has the disadvantage that it ties the program to a closed-source software system. The Initiator uses the fairly new object-oriented programming functionality of the MATLAB language, for more information reading the MATLAB documentation[41] is highly recommended.

The Initiator is created with extendibility in mind. At the heart of the Initiator is the so-called controller. All elements of the Initiator, including the controller are implemented is classes, which will instantiate to objects. For more information about classes and object please read any introductory text to object-oriented programming. This object handles the program flow of the Initiator. All modules use the controller to get information about other modules and the aircraft Geometry. It also keeps track of module dependencies and which modules have been completed.

When a new Initiator session is started, a new instance of the controller is created. The start-up process can be seen in Figure 8.1. The controller is tied to an aircraft definition file for which the starting point of the Initiator is deduced. After the Controller is started, depending on the input, the Initiator runs the *InteractiveMode* or the *BatchMode* module. When the module is finished, the controller is deleted and the program will exit. This can be seen in Figure 8.1.

Two starting points are currently implemented in the current version of the initiator:

1. Top-level requirements and aircraft configuration
2. Fully defined aircraft geometry

In the first case the Initiator can be used as a tool to perform preliminary sizing and analysis of the design; in the latter case the sizing process is skipped and the aircraft can be analysed directly.

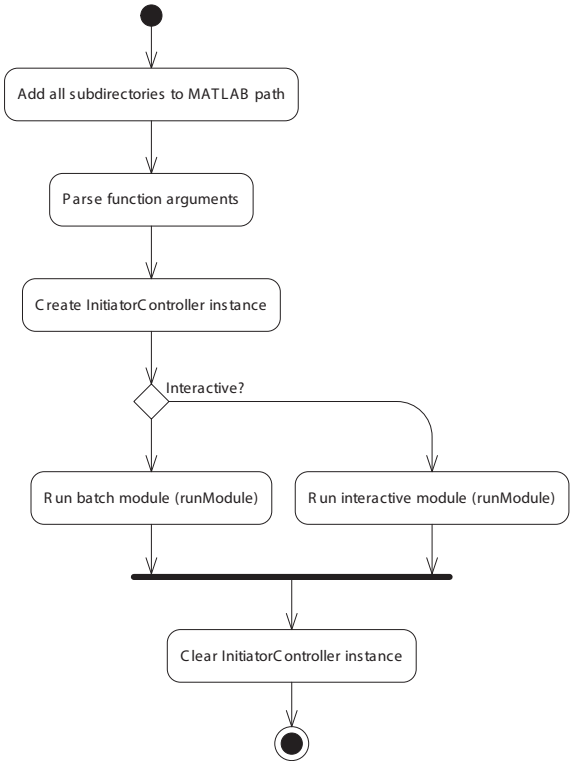


Figure 8.1: Top-level Initiator activity diagram

When the Initiator detects that geometry is defined in the aircraft definition file and can successfully generate the geometry, it will use the specified geometry. Otherwise, it will start with the preliminary sizing to generate a first estimate of the geometry.

The Initiator exists of three separate parts, the aforementioned *Controller*, the *Aircraft* object and the modules. The aircraft geometry is build up using High-Level primitives called *Parts*. All parts generate geometry which is used to drive the analysis modules and visualisation of the aircraft. An UML of the top-level classes can be seen in Figure 8.2. In Section 8.2 the workings of the *InitiatorController* is explained. Section 8.4 discusses the *Aircraft* object. The workings of the *Module* class is explained in Section 8.3, for information about the specific modules implemented in the Initiator, please read Chapter 3.

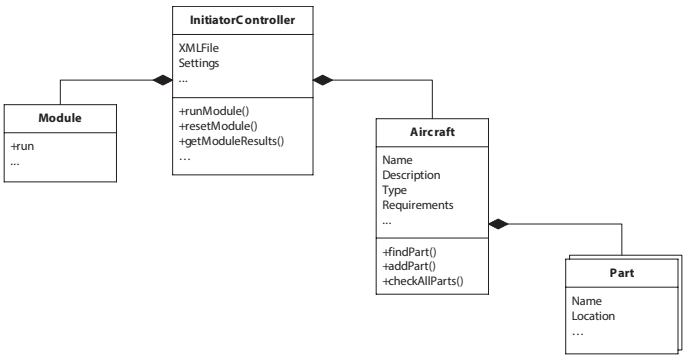


Figure 8.2: Top-level UML class of the Initiator

All in- and output of the program is written XML files. This is a plain-text file-format that is both machine- and human-readable. For reading and writing XML files the open-source TIXI library [42] developed at DLR is used. This library is written in C and has bindings for MATLAB , Python and Fortran.

8.2 InitiatorController

The Controller is the main object of the Initiator and is implemented in the *InitiatorController* class. The Controller contains an *Aircraft* object and structures which contain all the different modules. Every program session starts with the instantiation of an *InitiatorController* object. The XML files are read and the module structures are populated with the modules which are defined in the `modules.xml` file. The *InitiatorController* class instantiation can be found in an activity diagram in Figure 8.3.

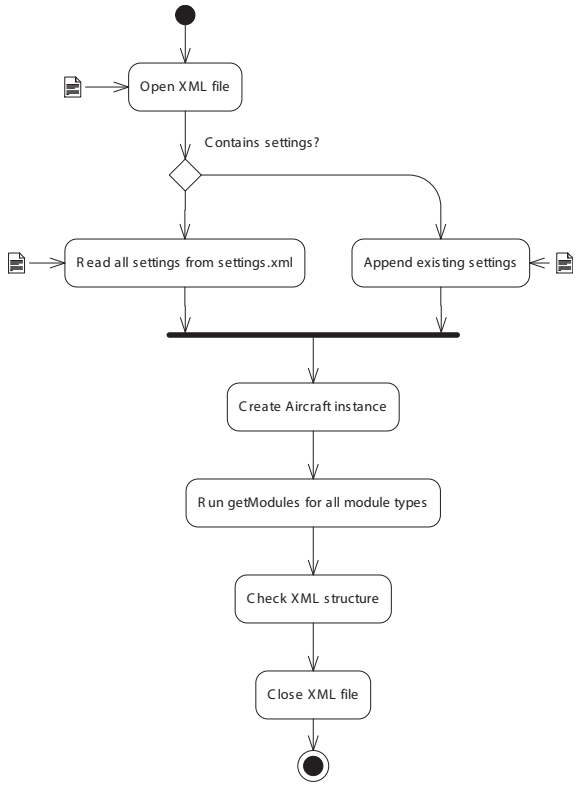


Figure 8.3: Class instantiation method of the Controller

The main method of the *InitiatorController* class is the *runModule* method. This method is called whenever a module in needed to run. The method makes sure all dependencies of the module are met and handles the switching between the different module types.

The analysis methods require the aircraft geometry. Therefore a preliminary sizing of the aircraft is needed before analysis or design methods can be used.

After start-up the controller will try to build the geometry if it does not know if a preliminary sizing is preformed. In case of a correctly pre-defined aircraft is present in the aircraft definition

file this will succeed. The controller will now be able to run analysis and design modules. In case the geometry generation fails, the controller will first run the sizing modules to perform the preliminary sizing. The full program flow of this `runModule` method can be seen in an activity diagram in Figure 8.4.

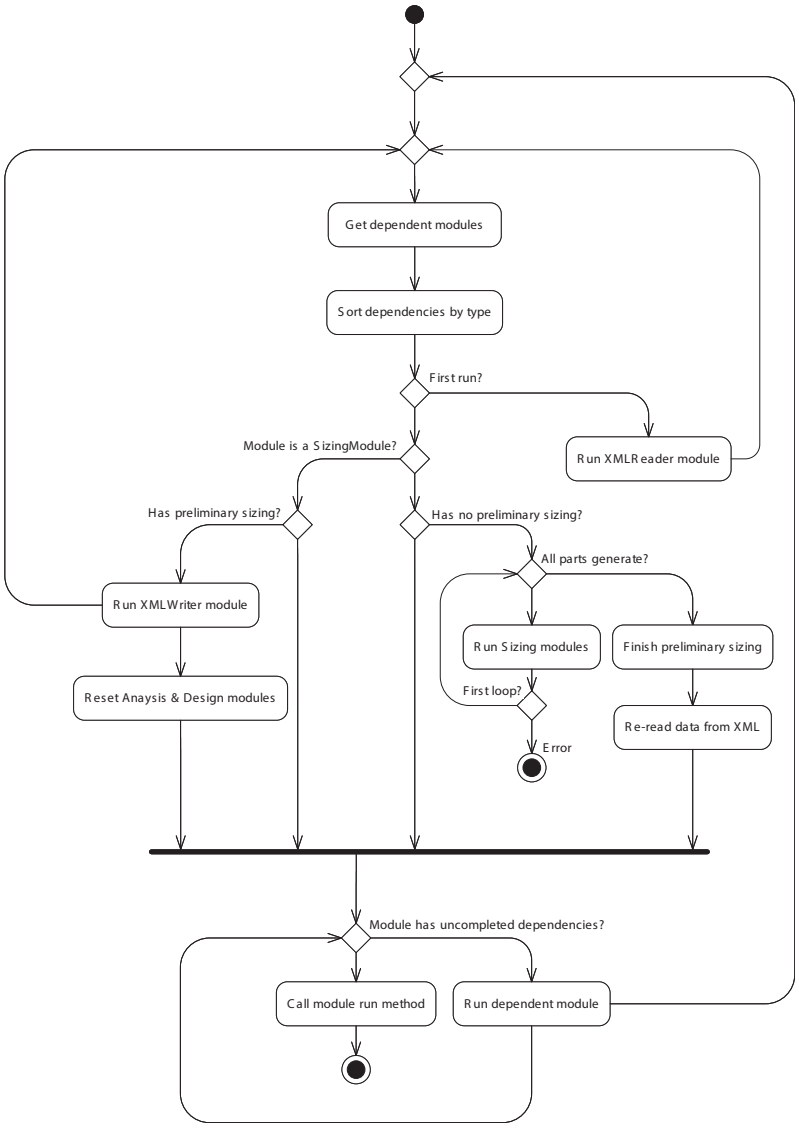


Figure 8.4: InitiatorController: `runModule` method

Other methods implemented in the *InitiatorController* class handle the settings and provide ways for modules to interact with other modules. The full list of properties methods of the *InitiatorController* can be found in Table 8.1.

8.2.1 Dependency handling

As is common in a design tool, modules are dependent on the output of other modules. The naive approach to solve this problem is by running all modules in the proper order. To downside to this

approach is the possibility that modules are called which are not needed to generate the requested end-result.

To solve this problem, when the controller is requested to run a module it only checks and, if needed, runs modules which are set as dependencies in the `modules.xml` file. This function is called recursively, until all dependent modules are completed.

In a workflow often happens that information needs to be recalculated. To accommodate this, the controller can reset a module. This also resets all modules which depend on the re-setted module; in other words: the dependencies are resolved backwards until on modules can be found which depend on the re-setted module. Again, this function is called recursively.

Table 8.1: *InitiatorController* class properties and methods

Poperty	Description
XMLFile	Name of the aircraft definition file
Settings	Cell-array containing all settings
Aircraft	Aircraft object
SizingModules	Structure containing all sizing modules
AnalysisModules	Structure containing all analysis modules
DesignModules	Structure containing all design modules
WorkflowModules	Structure containing all workflow modules
Method	**Description**
appendSettings	Merges settings from xml file and settings file
rereadSettings	Re-reads all settings from the settings file
getSetting	Returns the value of a setting
getModuleInput	Returns the module input of a specified module
getModuleResults	Return the results of a specified module
runModule	Runs a module and its dependencies
getModuleHandle	Returns a handle (pointer) of a module
resetModule	Resets a module and all modules depending on the module
runAllModules	Runs all modules of a certain type
resetAllModules	Resets all modules of a certain type
writeModuleResults	Writes module results to the aircraft definition file

8.3 Modules

All modules implemented in the Initiator are subclasses of the *Module* class. There are four different module classes defined which are used to organise the modules in groups by functionality. First there are the modules which do the preliminary sizing. These modules are all subclasses of the *SizingModule* class. Next, there are the analysis and design modules. They have the *AnalysisModule* and *DesignModule* as superclass. Finally there are the workflow-modules in which workflows, interfaces, optimisers, etc. can be implemented. These modules are all subclasses of the *WorkflowModule* class.

The module class had pre-determined structures for input and output. When the `ModuleInput` property is called, the module will check the aircraft definition file if there is input present for the

running module. This (optional) input is merged with the (again optional) default input, where preference is given for the input in the aircraft definition file. All results are stored in the `Results` property. The fields in this structure are written to the aircraft definition file after the module is completed.

When the module is completed the `Completed` property of the module is set to `true`. This property is used by the *InitiatorController* to keep track of which modules need to be called to resolve module dependencies.

The only exception to the aforementioned behaviour are the workflow modules, these modules do not have `Results` and `Completed` properties since they generally don't generate results and aren't limited to run only once (for example plotting can be done multiple times).

The different modules implemented in the Initiator can be found in Chapter 3.

8.4 Aircraft

The *Aircraft* object represents the aircraft which is currently in memory. It contains all requirements for the specified mission(s), configuration- and performance parameters and all parts which represent the aircraft geometry. It also provides methods to find parts of a certain type and check if parts are able to generate. The full list of properties methods of the *Aircraft* class can be found in Table 8.2.

Table 8.2: *Aircraft* class properties and methods

Poperty	Description
Name	Name of the aircraft
Description	(Optional) description of the aircraft
Type	Aircraft type; (Conventional, Canard, Prandtl, etc.)
MainPart	Main part of the aircraft (Part at 0,0,0)
Requirements	Harmonic range point requirements
MissionRequirements	All requirements for different missions
PerformanceParameters	Parameters related to performance
ConfigurationParameters	Parameters related to the aircraft configuration
Parts	Structure containing all aircraft parts
Method	**Description**
addPart	Add a part to the Parts structure
findPart	Find part(s) of a certain type
checkAllParts	Checks if all parts generate correctly

Chapter 9

Geometry Definition

The aircraft geometry is represented by a collection of *Parts*. Every part has its own set of properties, unique to the aircraft geometry represented by the part. This chapter presents all geometric parts implemented in the Initiator.

All objects are a subclass of the *Geometry* class, this class implements the basic geometry properties. The *Geometry* class is described in Section 9.1. The *Geometry* has three direct subclasses, the *Aircraft* class, the *Loft* class and the *Part* class which are discussed in Sections 9.1.1 through 9.2. All aircraft parts are subclasses of the *Part* class and they use the functionality of the *Airfoil*, *Loft* and *Geometry* classes to generate their geometry.

The class structure can be seen the UML in Figure 9.1. Note that for sake of clarity not all part properties and methods are shown. For a full description of the parts please review Chapter 8.

First, in Section 9.2.1, the *Wing* part is discussed, this is one of the main parts in the Initiator and is used for all wing-like parts like the main wing, tail surfaces, canards, etc. In Section 9.2.2 the *Fuselage* part is briefly discussed. This part is able to generate conventional and oval fuselages. Sections 9.2.3 to 9.2.8 treat the *BoxWing*, *Engine*, *LandingGear*, *Cargo* and *Spar* parts.

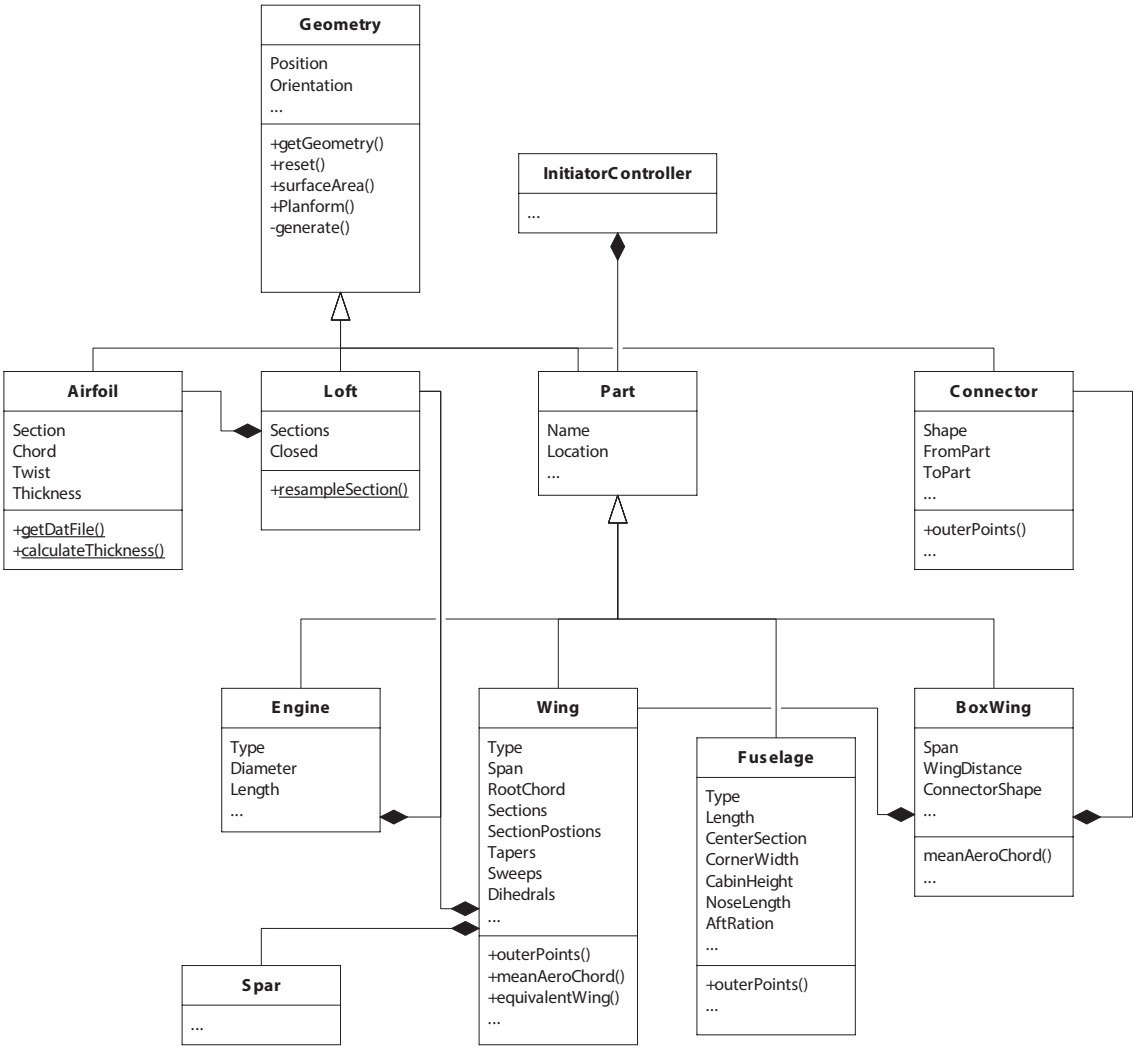


Figure 9.1: UML of all Part and Geometry classes

9.1 Geometry class

The *Geometry* class is the base class for all geometry in the Initiator. It handles the geometry positioning and orientation and makes sure the geometry is re-generated when required. The main method of the *Geometry* class is *getGeometry* (Figure 9.2). This method will call the *generate* method (Figure 9.3) when no geometry is present and returns the geometry data. The *generate* method is required for every geometry object in the Initiator . This method is expected to generate the geometry with its reference point at (0,0,0). The *generate* method of the *Geometry* class will make sure the geometry is positioned and oriented correctly. An example of how to implement geometry in the Initiator can be found in Section B.1. All properties and methods of the *Geometry* class are listed in Tables 9.1 and 9.2.

Table 9.1: *Geometry* class properties

Property	Description	Unit
Position	Position vector of the inlet centre point	m, m, m
Orientation	Rotation angles around x,y,z	°, °, °
Controller	Handle to the *InitiatorController* object	-
Generated	`true` if geometry is generated	-
Generating	`true` while generating the geometry	-
GeneratedProperties	List of properties which require the `generate` method	-

Table 9.2: *Geometry* class methods

Method	Description
getGeometry	Returns the geometry as X,Y,Z data
reset	Clears the geometry and sets *Generated* to `false`
surfaceArea	Calculates the total surface area
planform	Calculates the projected area (default in Z-direction)
generate	Generates the geometry
translate	translates X,Y,Z data by a vector
rotate	Rotates X,Y,Z data by angles around x,y,z
intersect	Checks if two Geometry parts intersect or touch

When a *Geometry* object is created, a *PostSet* listener is created for every property of the object, which can be seen in Figure 9.4. This is a MATLAB feature which creates the possibility to detect the change of a property and run a function at the event. When a property is changed, the *propertyChanged* method (Figure 9.5) of the object is called which then calls the *reset* method when this is required. The geometry will not be reset while is being generated, because this will invoke an infinite loop. The next time the *getGeometry* method is called, the geometry will be regenerated.

The *Geometry* object can also handle 'generated' properties. These are properties which are calculated by the *generate* method and are not an input slot of the object. A generated property can be created like any other property, but its name also needs to be added to the *GeneratedProperties* property of the class. The *propertyAccessed* method (Figure 9.6) makes sure the geometry is generated before the value is returned.

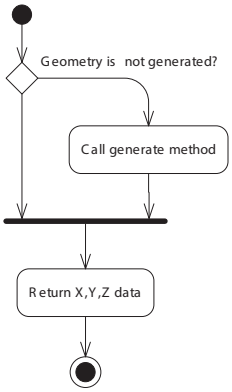


Figure 9.2: Activity diagram of the *getGeometry* method

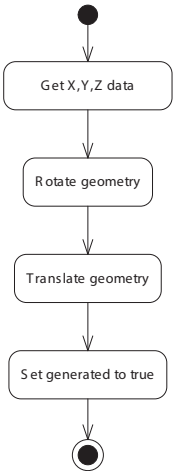


Figure 9.3: Activity diagram of the *generate* method

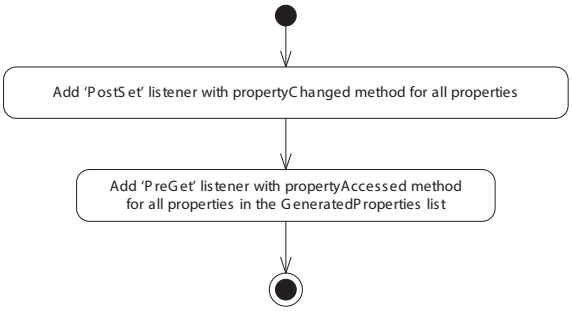


Figure 9.4: Class instantiation method of Geometry

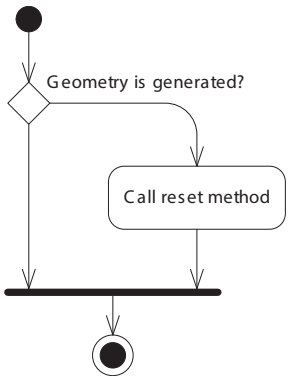


Figure 9.5: Activity diagram of the *propertyChanged* method

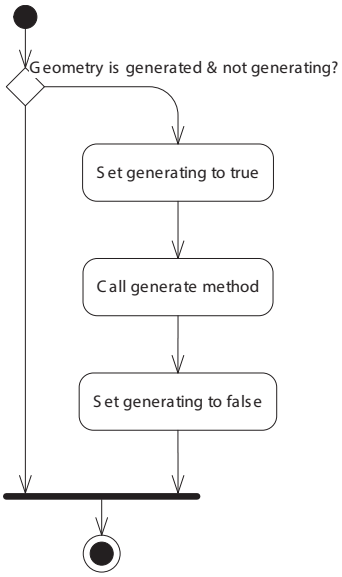


Figure 9.6: Activity diagram of the *propertyAccessed* method

How these methods work together is illustrated in Figure 9.7. Here the user changes a property of a *Wing* object. The change is noted by the property listener, the *propertyChanged* method is called and the geometry is reset if necessary. When later on in the workflow the geometry is requested, the *generate* methods are called (if necessary) and the geometry information is returned.

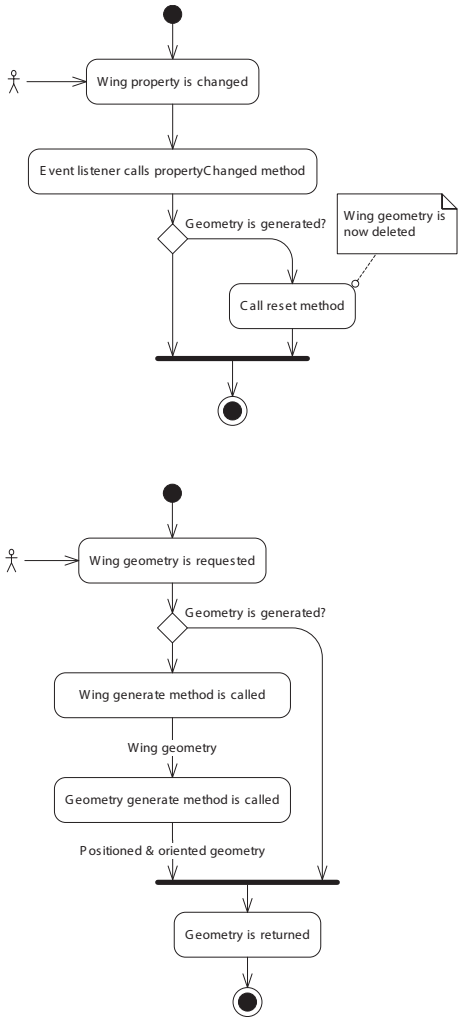


Figure 9.7: Example of a property change in the Wing object

9.1.1 Airfoil

The *Airfoil* object is used to import airfoil `.dat` files, scale them to a desired chord length, thickness and position them in 3D-space. The *getDatFile* method will sort the airfoil points from trailing edge →leading edge →trailing edge (Figure 9.9). The generate method will scale the geometry and rotate the airfoil around the $\frac{1}{4}c$ point as can be seen in Figure 9.8. All properties and methods of the *Airfoil* class can be found in Tables 9.3 and 9.4.

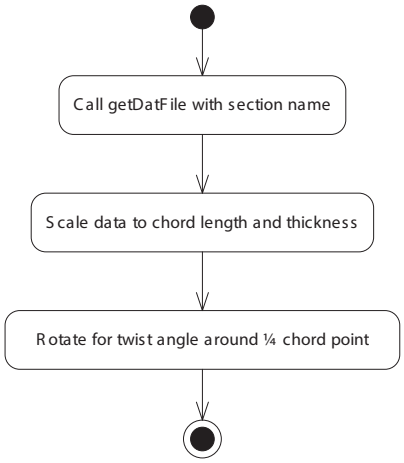


Figure 9.8: Activity diagram of the *generate* method

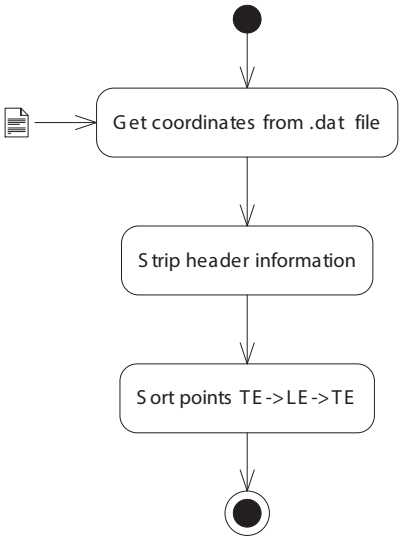


Figure 9.9: Activity diagram of the *getDatFile* method

Table 9.3: *Airfoil* class properties

Property	Description	Unit
Position	Position vector of the wing apex (inherited from *Geometry*)	m, m, m
Orientation	Rotation angles around x,y,z (inherited from *Geometry*)	°, °, °
Section	Name of the airfoil (Must match a `.dat` file)	-
Chord	Chord length of the section	m
Twist	Angle by which the airfoil is rotated around its $\frac{1}{4}c$ point	°
TcRatio	Thickness-to-chord ratio	-

Table 9.4: *Airfoil* class methods

Method	Description
getDatFile	Returns the points from a airfoil file
calculateThickness	Calculates the thickness of an airfoil

9.1.2 Loft

The *Loft* object is used to create a linear loft between two sections. It can be used with the *Airfoil* object as sections, but this no hard requirement. As long the object returns a single line in X,Y,Z the *Loft* object can use it as a section. The sections are stored in the *Sections* property. An activity diagram of the *generate* method can be found in Figure 9.10.

To be able to loft between two sections the sections need to be 'resampled' to make sure the two sections have both the same amount of points. This is done by the *resampleSection* method. The algorithm used is elaborated in Figure 9.11.

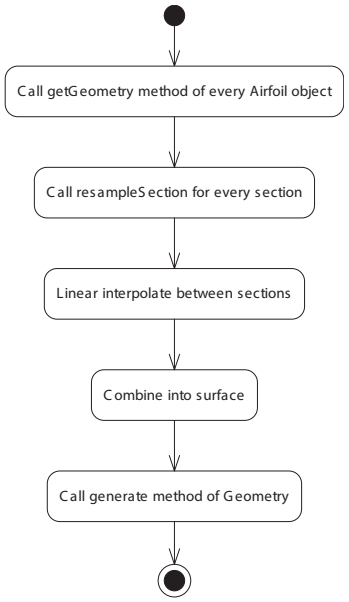


Figure 9.10: Activity diagram of the *generate* method

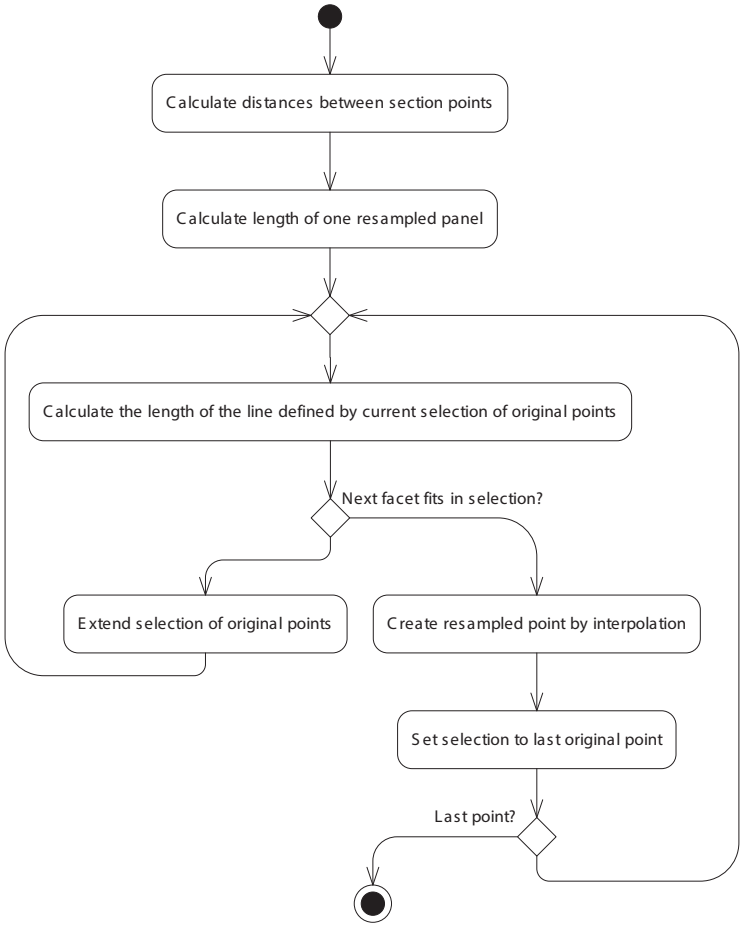


Figure 9.11: Activity diagram of the *resampleSection* method

9.2 Part class

The *Part* class is a simple extension to the *Geometry* class by adding the *Name* and *Location* properties. The *Name* property is used to identify parts and **needs** to be unique for every part. The *Location* property is set to the name of the part to which the part is connected. It defaults to the *MainPart*, but can be set to other parts (for example engines mounted to a wing).

9.2.1 Wing

The *Wing* part is used to represent all wig-like parts of the aircraft. Examples are the main wing, the horizontal tail and the vertical tail.

The wing can have any number of sections (with a minimum of 2 sections). Every section has a spanwise position, airfoil shape, chord, twist angle and thickness-to-chord ratio. Between two adjacent sections a straight trunk is generated. Every trunk has a sweep angle, taper ratio and dihedral angle. The wing can contain any number of spars which can be generated automatically or set by manually. The wing can also generate a fuel tank geometry for a set fraction of the span. Finally the wing can generate winglets with a given span, airfoil shape, taper, sweep, toe-in angle and twist.

The wing class contains methods which calculate the planform area, mean aerodynamic chord length and position, the equivalent wing and fuel volume. A complete reference of all the properties and methods of the wing can be found in Tables 9.5 and 9.6.

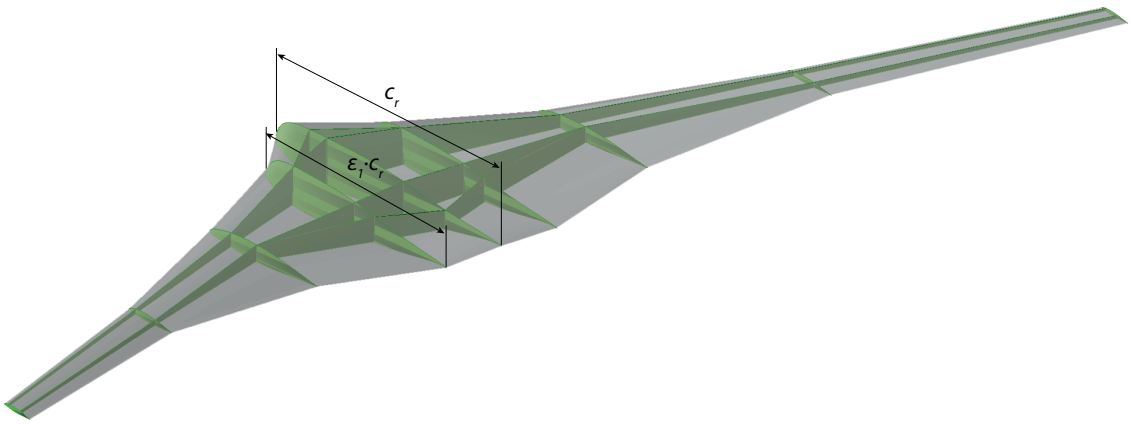


Figure 9.12: Wing geometry including spars

The wing is created by calculating the position, orientation and chord lengths of every section by using the span, taper, sweep and dihedral of every section. On every section position an *Airfoil* object is created. These *Airfoil* objects are given to a *Loft* object which creates the wing surface by interpolating between the sections. The full procedure can be found in Figure 9.13.

The spars can be generated by an algorithm or specified by the user.

The only user input for the algorithm to work are the chordwise spar positions of the outer and inner spar position. First the spar positions are calculated by using the spar positions the user provided. The tangent of the outboard trunk is then used to project a point on the inboard section.

Table 9.5: *Wing* class properties

Property	Description	Unit
Position	Position vector of the wing apex (inherited from *Geometry*)	m, m, m
Orientation	Rotation angles around x,y,z (inherited from *Geometry*)	°, °, °
Type	Wing type (Mainwing, Canard, etc.)	-
Span	Wing span	m
RootChord	Root chord length	m
Section	Cell-array of airfoil names	-
SectionPositions	Spanwise fraction of sections (0..1)	-
Twists	Section twist angles (positive is nose up)	°
TcRatios	Thickness-to-chord ratios of the sections	-
Tapers	Trunk taper ratios: $\lambda = \frac{c_t}{c_r}$	-
Sweeps	Trunk $\frac{1}{4}$c sweep angles	°
Dihedrals	Trunk dihedral angles	°
SparPositions	Chord fraction of spars (0..1) used to generate spars	-
SparLocations	Matrix containing all spar chord fractions	-
FuelTank	`true` or `false`	-
FuelTankSpanPosition	Spanwise position of start & end of fuel tank (0..1)	-
Symmetric	`true` or `false` true for a symmetric wing	-
Mirror	`true` or `false` mirrors the wing along its root	-
Winglets	`true` or `false`	-
WingletAngle	Angle between wing plane and winglet plane	°
WingletSpan	Winglet span	m
WingletSection	Airfoil name	-
WingletTaper	Taper ratio of the winglet	-
WingletSweep	Winglet sweep angle	°
WingletToeInAngle	Toe-in angle of the winglet	°
WingletTwist	Twist angle of the outboard winglet section	°

If the distance or angle falls with a certain tolerance, the projected point is used as a spar position, since this results in a straight spar, which is structurally more efficient. If the angle and distance are bigger than the et tolerance, both points are used and an auxiliary spar is created. This procedure is repeated for every trunk. A visual example can be seen in Figure 9.14, the activity diagram of the algorithm can be found in Figure 9.15.

9.2.2 Fuselage

The *Fuselage* part is able to model conventional as well as oval fuselages. It is developed by R.K. Schmidt as part of his Master thesis work.

9.2.3 BoxWing

The *BoxWing* part is used to model Prandtl-wings. It's a combination of two *Wing* objects which are connected with a *Connector*.

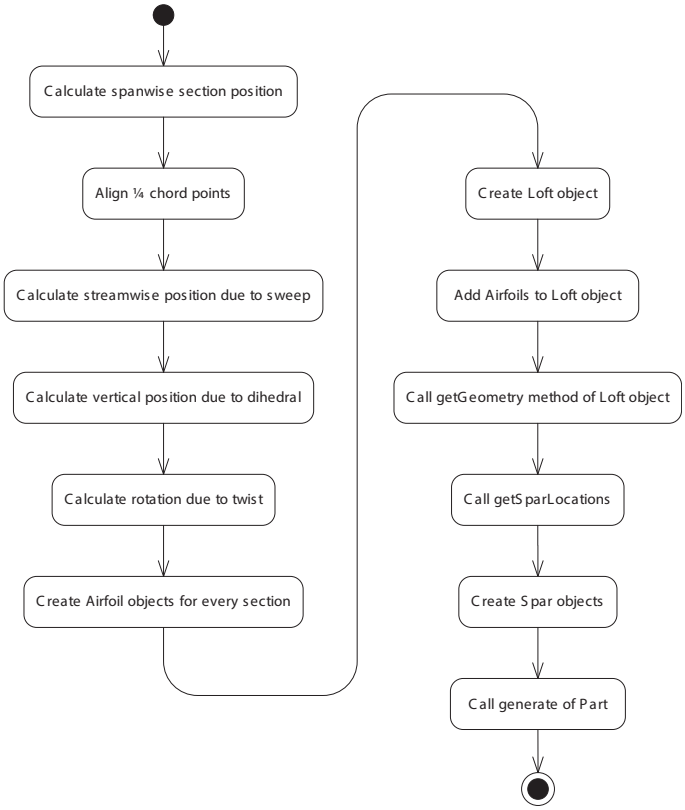


Figure 9.13: Activity diagram of the *generate* method

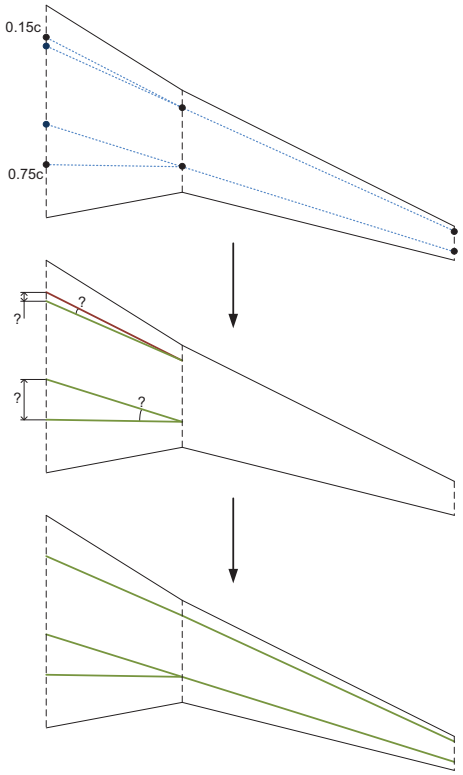


Figure 9.14: Illustration of the algorithm to generate spars

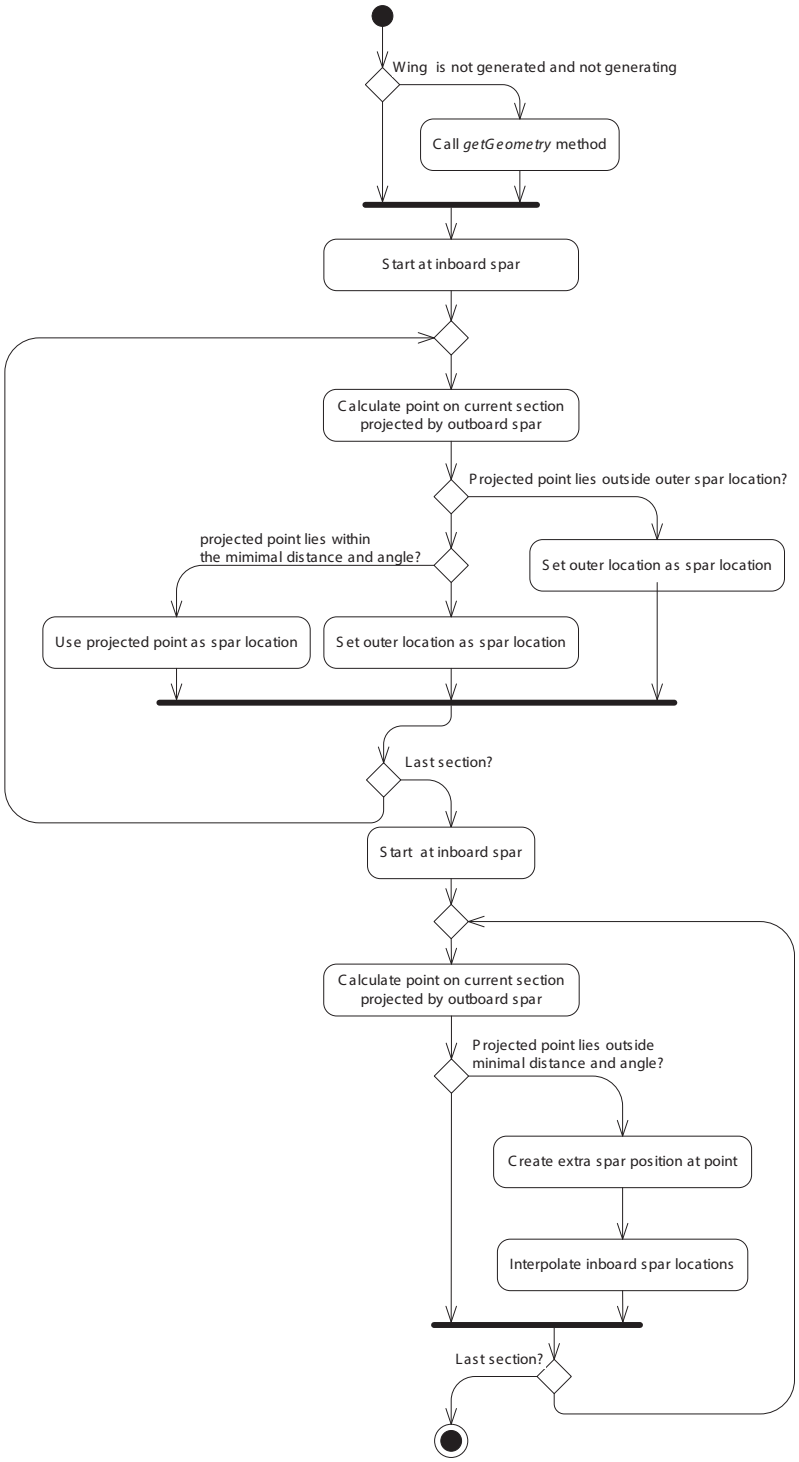


Figure 9.15: Activity diagram of the *getSparLocations* method

Table 9.6: *Wing* class methods

Method	Description
planform	Calculates the planform area (inherited from *Geometry*)
surfaceArea	Calculates the surfaceArea area (inherited from *Geometry*)
getGeometry	Returns the part surface (inherited from *Geometry*)
getChords	Calculates the chord length of every section
getWingletGeometry	Returns the winglet surface
outerPoints	Returns the leading- and trailing edge positions for every section
meanAeroChord	Returns the mean aerodynamic chord length and position
equivalentWing	Returns a structure containing the equivalent wing parameters
getCG	Calculates the wing centre of gravity
getFuelTankGeometry	Returns the fuel tank surface
getFuelTankCG	Calculates the fuel tank centre of gravity
getFuelVolume	Calculates the fuel tank volume
getPosOnLeadingEdge	Calculates a arbitrary point on the leading edge
getPosOnTrailingEdge	Calculates a arbitrary point on the trailing edge

The two wings, called `FrontWing` and `RearWing` are both instances of the aforementioned *Wing* object. The connector connects the wing tips with a Bézier curve or a straight edge. The properties of the *BoxWing* part are listed in Table 9.8.

9.2.4 Engine

The *Engine* part is used to model engines. Besides geometry information it also contains the thurst and bypass-ratio information of the engine. The properties of the *Engine* part can found in Table 9.9

9.2.5 LandingGear

The *LandingGear* part is a simple part which visualises the landing gear by generating a set of tyres. The properties of the *LandingGear* part can found in Table 9.10.

9.2.6 Cargo

The *Cargo* part is used to model a cargo bay. It can model bulk cargo and ULD cargo bays. Only in the case of ULD cargo, geometry is generated. The properties of the *Cargo* part can be found in Table 9.11.

9.2.7 ULD

The *ULD* part represents one ULD container used in the *Cargo* part. It's data is gathered from `cargo.xml`.

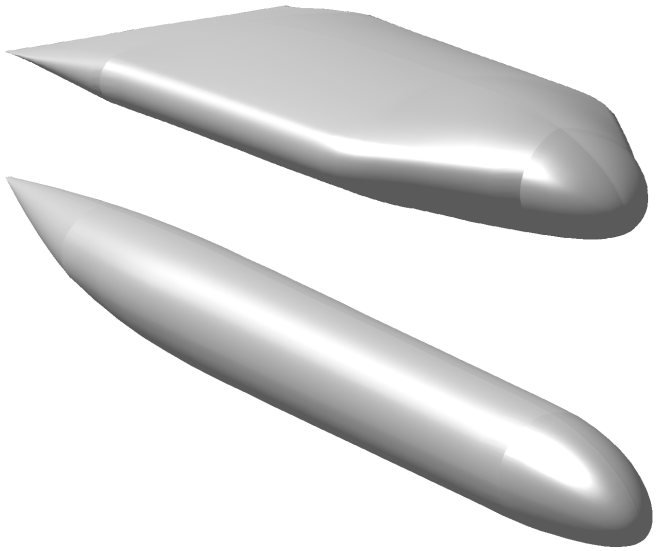


Figure 9.16: Geometry of an oval and a conventional fuselage

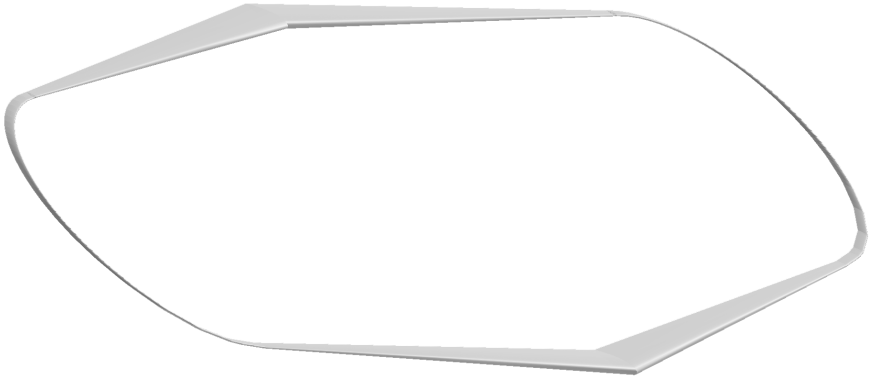


Figure 9.17: *BoxWing* geometry

9.2.8 Spar

The *spar* part is a simple part which fits a spar inside a given wing. It is positioned by specifying chordwise positions at wing sections. The spars are then modelled as straight spars between the sections. The properties of the *Spar* part can be found in Table 9.13.

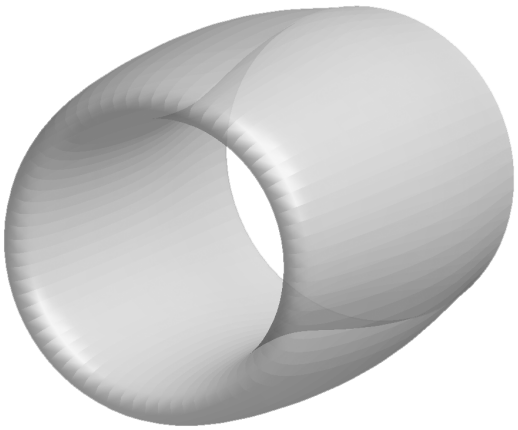


Figure 9.18: *Engine* geometry

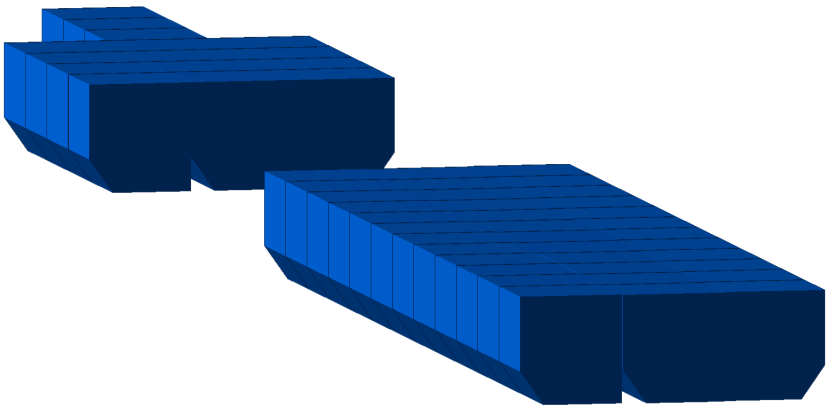


Figure 9.19: Cargo part with ULD parts

Table 9.7: *Fuselage* class properties

Property	Description	Unit
Position	Position vector of the fuselage nose (inherited from *Geometry*)	m, m, m
Orientation	Rotation angles around x,y,z (inherited from *Geometry*)	°, °, °
Type	Fuselage type	-
Length	Total length of fuselage	m
FloorZPosition	Vertical position of the floor	m
CenterSection	Vector containing: *NoseFinenessRatio AftFinenessRatio Height NoseDroop AftDroop*	-
CenterAirfoil	Airfoil name of the centre section	-
CenterBernstein	Bernstein coefficients of the centre section	-
TorusRadiusFactor	Torus radius as fraction of cabin height	-
PinchWidthRatio	Width ratio of fuselage pinch	-
PinchLengthRatio	Length ratio of pinch location	-
ForceAftPinch	Overrules pinch on non-conventional aircraft	-
Properties for conventional fuselages		
Diameter	Fuselage diameter	m
NoseFinenessRatio	Ratio between nose length and fuselage diameter	-
AftFinenessRatio	Ratio between aft length (from nose) and fuselage diameter	-
Properties for straight oval fuselages		
StraightOvalHeight	Height of the fuselage	m
StraightOvalWidth	Width of the fuselage	m
Properties for oval fuselages		
CornerWidth	Width of corner points of cabin floor	m
CornerLength	Location of corner points	m
CabinHeight	Cabin height	m
EccentricityLower	Floor eccentricity	-
EccentricityUpper	Floor eccentricity	-
NoseLength	Length of the nose	m
AftCutoff	Length of aft section	-
AftRatio	Aft fuselage taper	-

Table 9.8: *BoxWing* class properties

Property	Description	Unit
Position	Position vector of the front wing apex (inherited from *Geometry*)	m, m, m
Orientation	Rotation angles around x,y,z (inherited from *Geometry*)	°, °, °
Span	Span of the total wing-connector combination	m
WingDistance	Distance vector between the front and rear wing apex points	m
ConnectorShape	`Straight` or `Bezier`	-

Table 9.9: *Engine* class properties

Property	Description	Unit
Position	Position vector of the inlet centre point (inherited from *Geometry*)	m, m, m
Orientation	Rotation angles around x,y,z (inherited from *Geometry*)	°, °, °
Type	Engine Type (`TurboFan` or `TurboProp`)	-
Diameter	Engine diameter	m
Length	Engine length	m
Thrust	Engine thrust	N
ThrustReverser	`true` or `false`	-
BypassRatio	Engine bypass-ratio	-

Table 9.10: *LandingGear* class properties

Property	Description	Unit
Position	Position vector of the bogie centre (inherited from *Geometry*)	m, m, m
Orientation	Rotation angles around x,y,z (inherited from *Geometry*)	°, °, °
Type	`NoseGear` or `MainGear`	-
TyreDiameter	Diameter of the tyres	m
TyreThickness	Width of the tyre	m
Length	Height of the landing gear	m
NRows	Number of rows of tyres	-
NWheelPerRow	Number of wheels per row	–
XPositions	Slot to override tyre x-position	m
YPositions	Slot to override tyre y-position	m

Table 9.11: *Cargo* class properties

Property	Description	Unit
Position	Position vector of parent part (inherited from *Geometry*)	m, m, m
Orientation	Rotation angles around x,y,z (inherited from *Geometry*)	°, °, °
Type	`Bulk` or `ULD`	-
Height	Cargo bay height	m
FloorArea	Area of the cargo floor	m^2
Floors	Cell array containing the coordinates of the separate cargo floors	m
Positions	ULD positions	m
Mirrors	Boolean, `true` if containers are mirrored	-
ULDType	ULD type, as specified in `cargo.xml`	-

Table 9.12: *ULD* class properties

Property	Description	Unit
Position	Position vector of parent part (inherited from *Geometry*)	m, m, m
Orientation	Rotation angles around x,y,z (inherited from *Geometry*)	°, °, °
Height0	Height of the container	m
Height1	Height without the chamfer	m
Width0	Width of the container	m
Width1	Width without the chamfer	m
SecondChamfer	`true` if the container is chamfered at both sides	-
Depth	Depth of the container	m
MassFull	Mass of a fully loaded container	kg
MassEmpty	Mass of an empty container	kg
Volume	Usable container volume	m^3
Mirror	When `true`, geometry is mirrored on local XZ-plane	-

Table 9.13: *Spar* class properties

Property	Description	Unit
Position	Position vector of the parent wing (inherited from *Geometry*)	m, m, m
Orientation	Rotation angles around x,y,z (inherited from *Geometry*)	°, °, °
ChordFractions	Chord fraction of the spar position per wing section	-
Symmetric	`true` when spar is mirrored on local XZ-plane	-
Mirror	`true` when spar is on the other side of the local XZ-plane	-
ContainingSections	*Airfoil* objects from which the spar is generated	-

Chapter 10

User manual

10.1 Introduction

The Initiator is developed in the MATLAB environment and is tested on the latest release currently available (R2013b). To install the program the code needs to be checked out from SVN to a local directory[1] and opened in MATLAB . It is also possible to run the Initiator as an executable. For build instructions of the executable, please read Section 10.5.

10.2 Installation

The application is tested to run on Windows 7+ (32bit and 64bit), Mac OSX 10.7+ (64bit) and Linux (64bit). The version available in the SVN repository works directly on Windows operating systems. Whether it will run immediately on a Mac or Linux system is highly depended on the libraries installed on the local system. If it does not run, this probably means that the TIXI library needs to be installed or compiled and added to the MATLAB path. See the TIXI website [42] for more information and installation instructions. The Initiator is build with TIXI version 2.0.4[2], but depending on DLRs developments could also work with future versions. Also make sure the binaries of *xFoil*, *AVL* and *ESDU99031* (all present in the `External` directory) work on the current operating system.

[1] If the directory is indexed by a synchronisation service (for example Dropbox) please disable or pause this while running the program

[2] The MATLAB binding in the standard package is missing some wrapper functions, please use the `tiximatlab.c` file included with the Initiator to build the *mex* file if errors occur

10.3 Program Run

The easiest way to start the Initiator is by typing:
`>> Initiator`
in the MATLAB command prompt. This starts the application and will prompt for an aircraft definition file.

The program has a couple of optional switches which can be passed as arguments:

Table 10.1: Initiator command-line arguments

Argument	Description
`filename`	XML file
`--help`	Show help
`--interactive`	Run the Initiator in interactive mode
`--silent`	No text output to command prompt
`--debug`	Enable to break out of the interactive mode by typing debug
`--showfigures`	Shows all plots and figures during program run

All input and output done with XML files. Also the settings and other data is exchanged through XML. The only exception is the Database file which is for ease of editing an Excel file.

The main directory contains two helper-functions: *restore* and *clean* which respectively copy aircraft definition files from `CleanInputFiles` to the working directory or deletes them. The syntax of the aircraft definition file and the input which is required to run the program can be read in Section 10.4.

To function properly the Initiator needs the following XML files in its directory:

Table 10.2: XML files used by the Initiator

File	Description
`settings.xml`	Contains all settings used by the application
`modules.xml`	In this file all modules and their dependencies are set
`materials.xml`	This file contains all required material properties
`cargo.xml`	Describes the different cargo pieces (ULD containers)

10.4 XML Layout

10.4.1 Aircraft definition file

All information regarding the aircraft are collected in one file. To be able to run the Initiator from scratch one needs to set up a basic XML structure. This structure can be seen in Listing 10.1.

All data is collected in the `<initiator>` tag. Inside this main tag are four possible sub-tags, where the first two are mandatory for proper workings of the Initiator:

- <aircraft>
- <settings>
- <moduleInputs>
- <moduleResults>

Inside the <aircraft> tag all aircraft requirements, configuration parameters and performance parameters are defined.

Inside the <parts> tag all the aircraft parts (represented by high-Level Primitives) are defined. To be able to run the Initiator only empty part definitions are required, the dimensions and other parameters are calculated by the modules of the Initiator. One exception is the <engine> part which requires a location tag. Every part definition requires a name and a type tag. The name can be chosen by the user. The type needs to be a valid type for the part in question. This and other parameters for the parts can be found in Chapter 9.

The <settings> entry links to its source file through the source attribute. This is normally the settings.xml file. It is possible to overrule certain settings by redefining them inside the aircraft definition file, like is done in Listing 10.1 on lines 34-37.

The <moduleInputs> tag contains, as the name would suggest, inputs for the modules. All modules are configured to use defaults when no inputs are defined.

Inside the <moduleOutputs> tag all results calculated by the modules are written to their own result tag.

```
<initiator>
  <aircraft>
    <name></name>
    <description></description>
    <missions default="Default Mission">
      <mission name="Default Mission">
        <requirement>
          <name></name>
          <value></value>
        </requirement>
      </mission>
    </missions>
    <configuration>
      <parameter>
        <name></name>
        <value></value>
      </parameter>
    </configuration>
    <performance>
      <parameter>
        <name></name>
        <value></value>
      </parameter>
    </performance>
    <parts mainPart="Fuselage">
      <fuselage name="The Partname" type="parttype" />
      <wing name="" type="" />
      <engine name="" type="">
```

```
        <location></location>
      </engine>
    </parts>
  </aircraft>
  <settings source="settings.xml">
    <setting>
      <name>OverRuledSetting</name>
      <value>true</value>
    </setting>
  </settings>
  <moduleInputs>
    <input module="ModuleName">
      <TheInput>2013</TheInput>
    </input>
  </moduleInputs>
  <moduleResults>
    <result module="ModuleName">
      <TheResult>42</TheResult>
    </result>
  </moduleResults>
</initiator>
```

Listing 10.1: XML aircraft definition file structure

10.4.2 Settings file

The settings file contains all settings and default values of the Initiator. The settings file has a very simple structure as demonstrated in Listing 10.2.

```
<settings>
  <setting>
    <name>firstSetting</name>
    <value>1</value>
  </setting>
  <setting>
    <name>secondSetting</name>
    <value>2</value>
  </setting>
</settings>
```

Listing 10.2: Settings file structure

10.4.3 Modules file

The modules file is a XML-file which defines the modules which are implemented in the Initiator and (optionally) sets their dependencies. The file splits the modules up in four categories:

- `sizingModules`
- `analysisModules`
- `designModules`
- `workflowModules`

The dependencies are optional. When dependencies are specified the Initiator will make sure these are completed before the module is called. An example of a module definition can be seen in Listing 10.3.

```
<modules>
  <sizingModules>
    <module>
      <name>Database</name>
    </module>
    <module>
      <name>Class1WeightEstimation</name>
      <dependency>Database</dependency>
    </module>
  </sizingModules>
</modules>
```

Listing 10.3: Module definition

10.4.4 Materials file

The materials file contains all material properties are stored used by various modules in the Initiator. The Material object parses this XML-file and exposes the material properties to the MATLAB environment.

10.4.5 Cargo file

The cargo XML-file specifies the dimensions and other properties of the cargo containers used in the Initiator. These are used by the Cargo and ULD objects.

10.5 Executable Build Instructions

In the directory `Build` a `.prj` file is created which contains the configuration to compile the Initiator. To run the compiled executable the MATLAB Compiler Runtime is required on the target computer[3]. If for some reason the build project file needs to be created from scratch, please include the following directories to the shared resources:

- `AnalysisModules`
- `DesignModules`
- `SizingModules`
- `WorkflowModules`
- `External`
- `Tools`

[3]link: http://www.mathworks.nl/products/compiler/mcr/index.html

References

[1] Peeters, P. M., Middel, J., and Hoolhorst, A., "Fuel efficiency of commercial aircraft An overview of historical and future trends," Tech. Rep. November, 2005.

[2] Kroo, I., "Nonplanar Wing Concepts for Increased Aircraft Efficiency," *VKI lecture series on Innovative Configurations and Advanced Concepts for Future Civil Aircraft*, 2005.

[3] Torenbeek, E., *Advanced Aircraft Design - Conceptual Design, Analysis and Optimization of Subsonic Civil Airplanes*, Wiley, 2012.

[4] Scherer, R., "Starship dimensions," http://rps3.com/Files/Starship_Dimensions.jpg, [Online; accessed September 2013].

[5] AeroFred, "Piaggio Avanti 3-view," http://plans.aerofred.com/data/media/54/piaggio-p180-avanti-ii.jpg, [Online; accessed September 2013].

[6] NASA, "National Aeronautics and Space Administration website," http://www.nasa.gov, [Online; accessed September 2013].

[7] Schut, J. and Tooren, M. V., "Design "Feasilization" Using Knowledge-Based Engineering and Optimization Techniques," *Journal of Aircraft*, Vol. 44, No. 6, Nov. 2007, pp. 1776–1786.

[8] Rocca, G. L. and Tooren, M. J. L. V., "Knowledge Based Engineering to Support Aircraft Multidiscilpinary Design and Optimisation," *26th International Congress of the Aeronautical Sciences*, 2008.

[9] Obert, E., "Drag polars of nineteen jet transport aircraft at Mach numbers M = 0.40 - 0.60 (unpublished)," Tech. rep., 2013.

[10] Vaessen, F. and Vos, R., "A New Compressibility Correction Method to Predict Aerodynamic Interaction between Lifting Surfaces," *Aviation Technology, Integration, and Operations Conference*, 2013, pp. 1–20.

[11] Elham, A., La Rocca, G., and van Tooren, M., "Development and implementation of an advanced, design-sensitive method for wing weight estimation," *Aerospace Science and Technology*, Vol. 29, No. 1, Aug. 2013, pp. 100–113.

[12] Schmidt, R. K., *A Semi-Analytical Weight Estimation Method for Oval Fuselages in Novel Aircraft Configurations*, Master's thesis, Delft University of Technology, 2013.

[13] Roux, E., *Avions civils à réaction: Plan 3 vues et données caractéristiques*, Elodie Roux, 2007.

[14] Obert, E., *Aerodynamic Design of Transport Aircraft*, IOS Press, 2009.

[15] Nicolai, L. M., Carichner, G. E., Schetz, J. A., and Malcolm, L. M. L., *Fundamentals of Aircraft and Airship Design Volume I — Aircraft Design*, Vol. I, American Institute of Aeronautics and Astronautics, 2010.

[16] Nangia, R. K., "Efficiency parameters for modern commercial aircraft," *The Aeronautical Journal*, , No. 3068, 2006, pp. 495–510.

[17] Langen, T., *Development of a conceptual design tool for conventional and boxwing aircraft*, Master's thesis, Delft University of Technology, 2011.

[18] Vaessen, F., *Improved aerodynamic analysis to predict wing interaction of high-subsonic three-surface aircraft*, Master's thesis, Delft University of Technology, 2013.

[19] Van Dommelen, J., *Design of a Forward-Swept Blended Wing Body Aircraft*, Master's thesis, Delft University of Technology, 2011.

[20] Hoogreef, M., *The Oval Fuselage*, Master's thesis, Delft University of Technology, 2012.

[21] user), H. B. W., "De Havilland DH.106," http://commons.wikimedia.org/wiki/File:Comet_4.jpg, [Online; accessed September 2013].

[22] Dreamscape, B., "Dash 80," http://en.wikipedia.org/wiki/File:Air_to_air_photo_of_the_Dash_80_FA239925.jpg, [Online; accessed September 2013].

[23] Airbus, "Airbus company website," http://www.airbus.com, [Online; accessed September 2013].

[24] COROLLER, G., "Caravelle," http://en.wikipedia.org/wiki/File:Caravelle_12.jpg, [Online; accessed September 2013].

[25] user), R. W., "bombardier crj-900 scandinavian airlines," http://en.wikipedia.org/wiki/File:Crj900-sas.jpg, [Online; accessed September 2013].

[26] Prandtl, L., "TN-182: Induced drag of multiplanes," Tech. rep., NACA, 1924.

[27] Raymer, D. P., *Aircraft Design: A Conceptual Approach*, American Institute of Aeronautics and Astronautics, 1992.

[28] Drela, M., "AVL (Athena Vortex Lattice)," http://web.mit.edu/drela/Public/web/avl/, [Online; accessed April 2013].

[29] Torenbeek, E., *Synthesis of Subsonic Airplane Design*, Delft University Press, 1982.

[30] Slingerland, A., *Preliminary Sizing of Conventional and Unconventional Aircraft*, Master's thesis (not yet published), Delft University of Technology, 2014.

[31] Roskam, J., *Airplane design*, No. v. 1 in Airplane Design, Roskam Aviation and Engineering Corp., 1989.

[32] Melin, T., *Master Thesis A Vortex Lattice MATLAB Implementation for Linear Aerodynamic Wing Applications*, Master's thesis, Royal Institute of Technology (KTH), 2000.

[33] ESDU, "ESDU 89034: The maximum lift coefficient of plain wings at subsonic speeds." 1993.

[34] Warmenhoven, M., "A Preliminary Drag Estimation," Tech. rep., Delft University of Technology, 2012.

[35] Elham, A., *Weight Indexing for Multidisciplinary Design Optimization of Lifting Surfaces*, Delft University of Technology, 2013.

[36] ESDU, "ESDU 99031: Computer program for estimation of lift curve to maximum lift for wing-fuselage combinations with high-lift devices at low speeds." 2009.

[37] Huijts, C., *Trim drag minimization in the conceptual and preliminary design phase applied to a Blended Wing Body aircraft configuration*, Master's thesis (not yet published), Delft University of Technology, 2014.

[38] Heerens, N., *Landing gear design in an automated design environment*, Master's thesis (not yet published), Delft University of Technology, 2014.

[39] Linden, P., *Conceptual design of a passenger aircraft for in-flight refueling operations*, Master's thesis, Delft University of Technology, 2013.

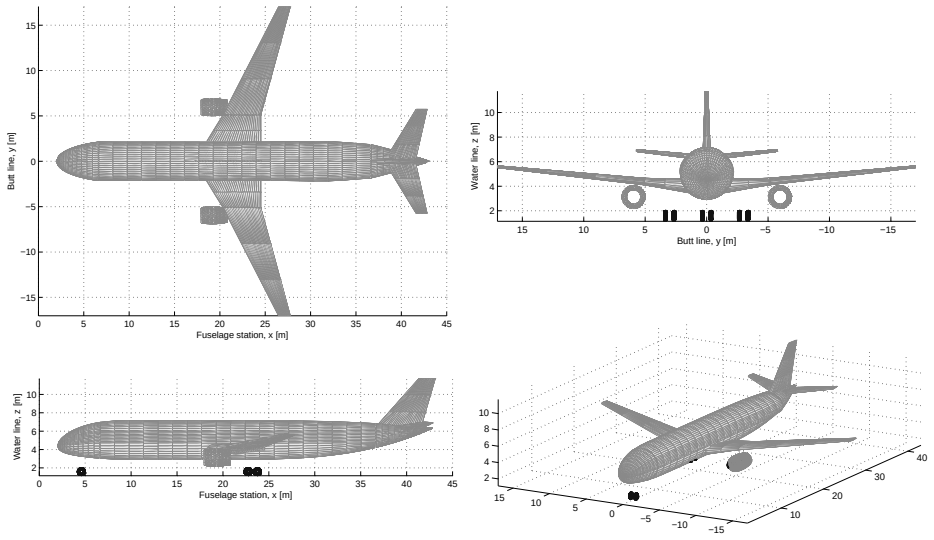
[40] Vos, R. and Van Dommelen, J., "A Conceptual Design and Optimization Method for Blended-Wing-Body Aircraft," *53rd AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics and Materials Conference*, 2012, pp. 1–14.

[41] MATLAB, *version: 8.2.0.701 (R2013b)*, The MathWorks Inc., Natick, Massachusetts, 2013.

[42] Litz, M., "TIXI: A Library for fast and simple XML Access," `https://code.google.com/p/tixi/`, [Online; accessed March 2013].

Appendix A

Aircraft report generated by the Initiator



(c) Side view **(d)** 3D view

Figure A.1: Aircraft geometry (all dimensions in meters)

A.1 General Characteristics

Aircraft "A320" generated by the Initiator version 2.0. The aircraft is a conventional aircraft with a low wing and an aspect ratio of 9.39. The aircraft is designed to transport 150 passengers with a total payload mass of 20536kg over 2870km.

A.2 Specification

Table A.1: Max payload

Pax	150	-
Payload Mass	20536	kg
Cruise Mach	0.76	-
Altitude	11278	m
Range	2870	km
Take Off Distance	2180	m
Landing Distance	1440	m

A.3 Operational Performance

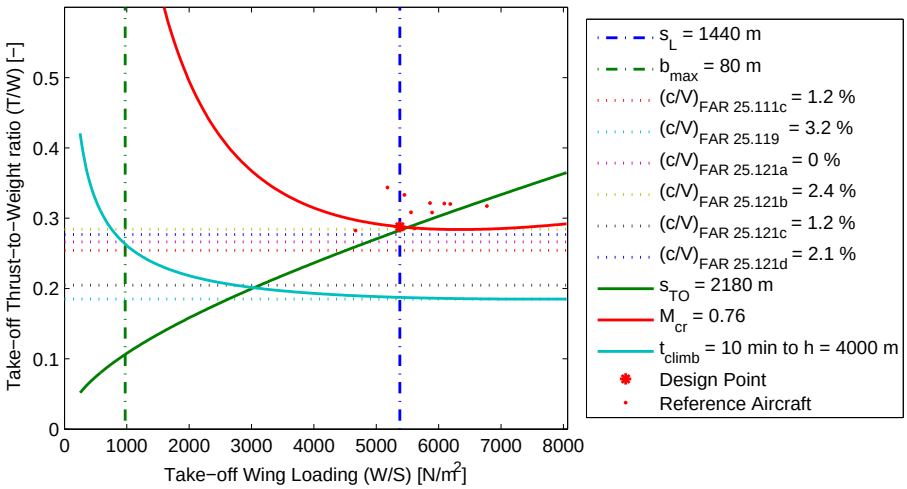


Figure A.2: Loading Diagram

Result: Wing loading at MTOM: 5378 $\mathrm{N/m^2}$
Thrust-to-weight ratio: 0.288 -

Table A.2: Performance results

L/D_{cruise}	15.9	-
Cruise altitude	11278	m
Maximum take-off mass	68000	kg
Operational empty mass	36000	kg
Payload mass	21000	kg
Fuel mass	12000	kg
Harmonic range	2880	km
Ferry range	8570	km
Maximum fuel range	8440	km

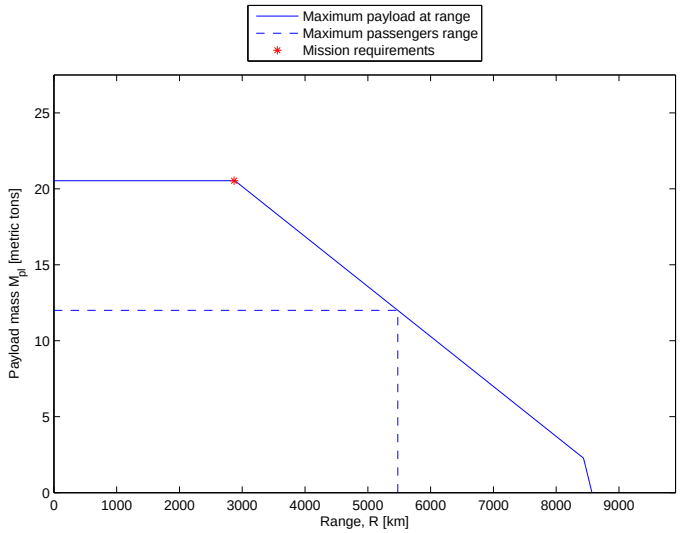


Figure A.3: Payload-Range

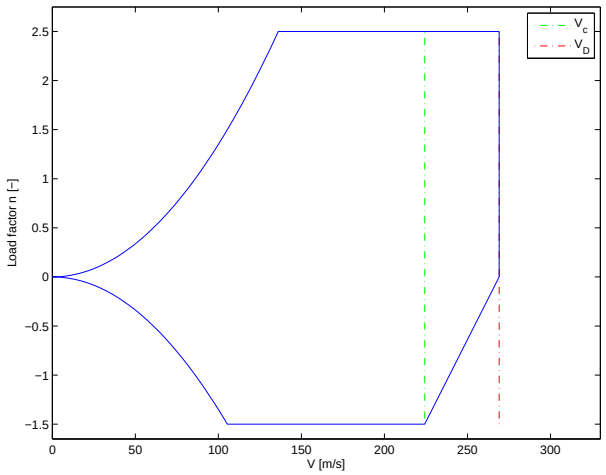


Figure A.4: Manoeuvre diagram

A.4 Weight estimation

Table A.3: Mass summary

Pax	12000	kg
Cargo	9000	kg
DLM	58000	kg
End Cruise Mass	58000	kg
FM	12000	kg
Fuel Volume	0	kg
Initial Cruise Mass	66000	kg
MLM	61000	kg
MRM	69000	kg
MTOM	68000	kg
Max FM	26000	kg
Mission FM	10000	kg
OEM	36000	kg
PLM	21000	kg
Reserve FM	0	kg
ZFM	56000	kg

Table A.4: Component masses

Engine1	2442	kg
Engine2	2442	kg
Furnishing	854	kg
Fuselage	10177	kg
Horizontal Stabiliser	604	kg
Main Gear1	1455	kg
Main Gear2	1455	kg
Main Wing	5797	kg
Nose Gear	388	kg
Vertical Stabiliser	496	kg
APU	1837	kg
Air Conditioning	1145	kg
Anti Ice	135	kg
Avionics	766	kg
Electrical	411	kg
Flight Controls	254	kg
Fuel System	132	kg
Handling Gear	20	kg
Hydraulics	1784	kg
Instruments	107	kg

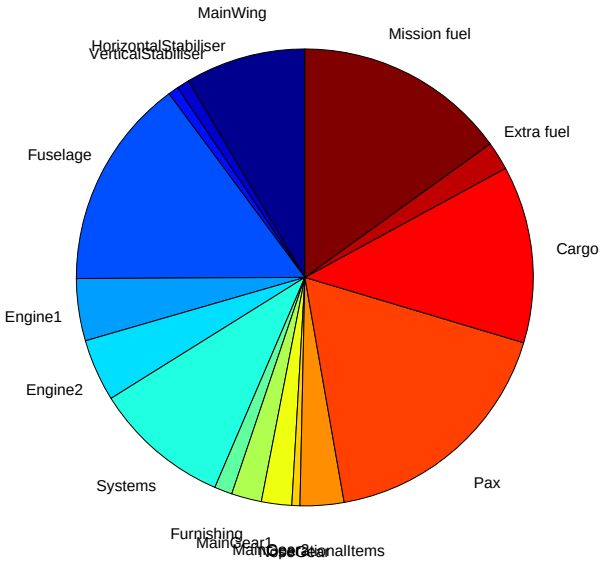


Figure A.5: Mass distribution

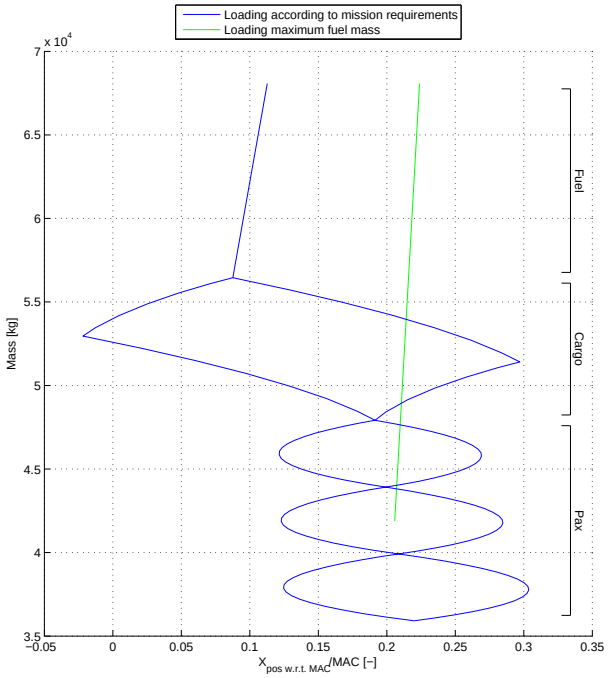


Figure A.6: Loading diagram

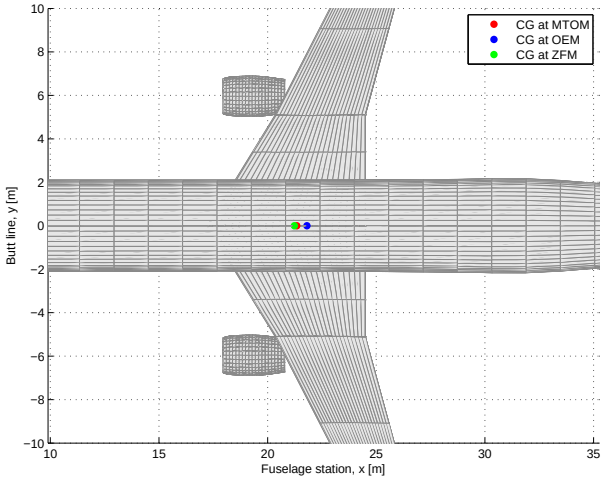


Figure A.7: CG location

A.5 Aerodynamics

Table A.5: Aerodynamic properties at cruise

$C_{L,\text{cruise}}$	0.62	-
$C_{D,\text{cruise}}$	388	cts
L/D_{cruise}	15.9	-
C_{D_0} (Clean)	231	cts
C_{D_0} (Take-Off)	576	cts
C_{D_0} (Landing)	1076	cts
Oswald factor (e) (Clean)	0.753	-
Oswald factor (e) (Take-Off)	0.803	-
Oswald factor (e) (Landing)	0.853	-
C_{L_α}	5.19	rad^{-1}
C_{m_α}	-3.49	rad^{-1}
$C_{L_{\text{max,clean}}}$	1.2	-
$C_{L_{\text{max,take-off}}}$	2.2	-
$C_{L_{\text{max,landing}}}$	3.2	-

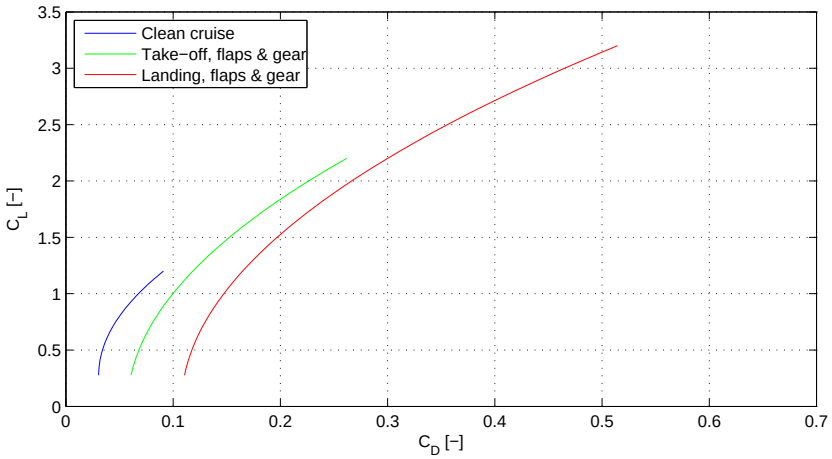


Figure A.8: Drag Polars

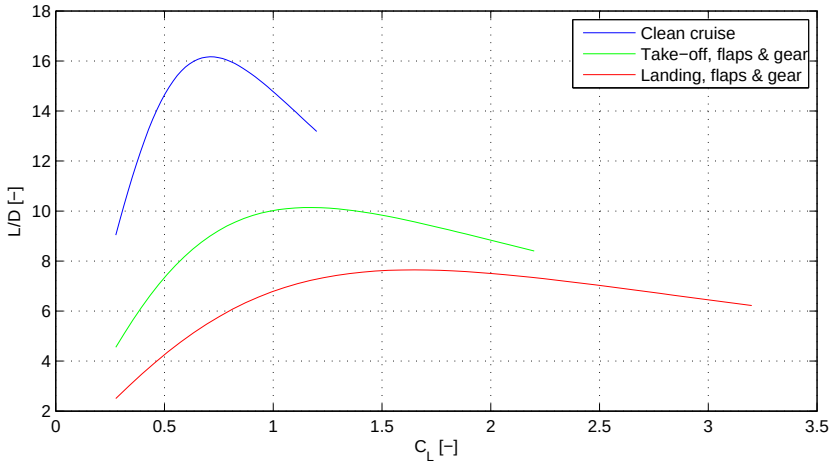


Figure A.9: Aerodynamic efficiency of the aircraft

A.6 Propulsion

Table A.6: Propulsion

Number of engines	2	-
SFC_{cruise}	0.682	h^{-1}
T_{static} (per engine)	129.8	N
Bypass Ratio	6	-
Diameter	1.83	m
Length	2.86	m

A.7 Aircraft Geometry

Table A.7: Main Wing dimensions

Span	34.1	m
Planform area	122.8	m^2
MAC	4.35	m
Root Chord	4.18	m
Root t/c	0.18	-
Tip Chord	1.28	m
Tip t/c	0.1	-
Sections (root to tip)	boeing-a, boeing-b, boeing-c	
Sweep 0.25c	24.5	°
Taper ratio	0.207	-
Twist	0	°
Dihedral	6	°

Table A.8: Horizontal Stabiliser dimensions

Span	11.5	m
Planform area	25.98	m^2
MAC	2.47	m
Root Chord	3.39	m
Root t/c	0.118	-
Tip Chord	1.19	m
Tip t/c	0.118	-
Sections (root to tip)	N0012, N0012	
Sweep 0.25c	27.4	°
Taper ratio	0.35	-
Twist	0	°
Dihedral	6	°

Table A.9: Vertical Stabiliser dimensions

Span	5.44	m
Planform area	18.3	m^2
MAC	3.66	m
Root Chord	5.04	m
Root t/c	0.118	-
Tip Chord	1.76	m
Tip t/c	0.118	-
Sections (root to tip)	N0012, N0012	
Sweep 0.25c	36.7	°
Taper ratio	0.35	-
Twist	0	°
Dihedral	0	°

Table A.10: Fuselage dimensions

Length	40.7	m
Floor Position	-58	% of fuselage height
Diameter	4.2	m
Nose Fineness Ratio	0.18	-
Aft Fineness Ratio	0.55	-
Cabin Height	1.37	m
Nose Length	4.57	m
Aft Cutoff	0.85	-
Aft Ratio	0.05	-

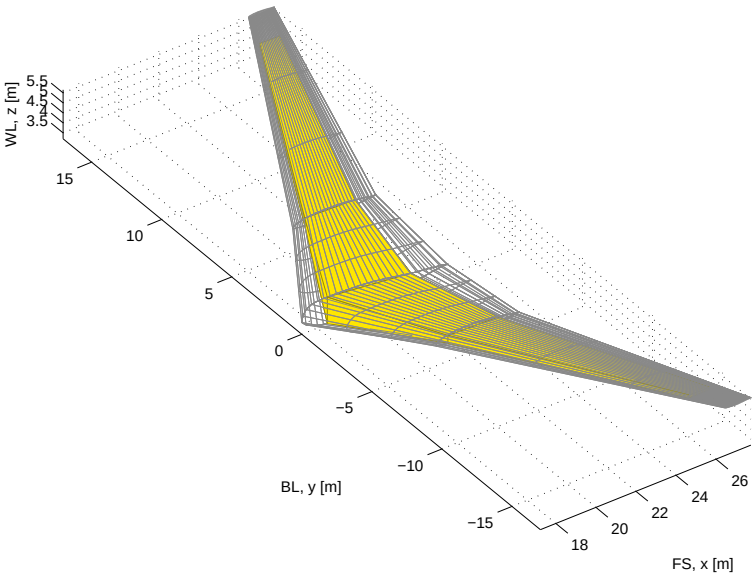


Figure A.10: Fuel tank layout

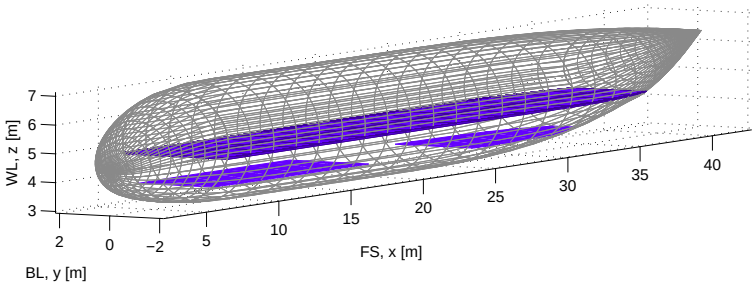


Figure A.11: Fuselage geometry; (blue = cargo ULDs, purple = floors)

Appendix B

Code examples

B.1 Part implementation

In this section a example is given how to implement a simple part into the Initiator . A simple part called *Square* will be created. The geometry is a square in the XY plane with the properties length and width.

B.1.1 File creation

First a directory for the new part needs to be created. Inside the `Geometry` directory a new directory `@Square` will be created. The '@' symbol tells MATLAB that all files inside the directory belong to the *Square* class.

Inside the `@Square` directory, two files are required to implement the part. First the file `Square.m` is created which will contain the class definition of our Part. Secondly the file `generate.m` will be created which will contain the code needed to generate the geometry.

B.1.2 Class definition file

The file `Square.m` will contain the class definition code. The Square part only has two properties; length and width. In the initiator it is a convention to write properties starting with a uppercase letter. This is done to be able to differentiate between the methods and properties. The code for `Square.m` can be found in Listing B.1.

The first line tells MATLAB that `Square` is a subclass of `Part`. Between `properties` and `end` all properties are defined, in this case `Length` and `Width`. The `(SetObservable, GetObservable)` keywords tell MATLAB that it can create listeners for the properties. This will enable the re-generation when a property is changed. The Initiator will give an error when there keywords are not set, since it then fails to create property listeners. Between `methods` and `end` all methods are defined. In this case only the class instantiation method is created, which is required. This method simply passes

the part name and a handle to the controller to the superclass. The *generate* method declaration is required since all methods in MATLAB are `public` by default. Since the `generate` method is set to `protected` in the superclass, this pattern needs to be repeated in this class definition file.

```
classdef Square < Part
%SQUARE Simple square

    properties (SetObservable, GetObservable)
        Length  % Length in [m]
        Width   % Width in [m]
    end

    methods
        function obj = Square(name,controllerHandle)
            % Call superclass instantiation method
            obj = obj@Part(name,controllerHandle);
        end
    end

    methods (Access = protected)
        generate(obj)
    end

end
```

Listing B.1: *Square* class definition file

B.1.3 Generate method

The generate method calculates the X,Y,Z points of the part. The full code is listed in Listing B.2. The `obj` variable is a handle to the Square class and can be used to access the properties and methods. Lines 5-6 calculate the corner points of the square with it's origin in the middle of the square. In line 10 the `generate` method of the superclass is called. This is required for correct functioning of the geometry objects.

```
function generate(obj)
%GENERATE Generates the square geometry

% Calculate the square coordinates, (0,0,0) is the centre
obj.X = [-obj.Length -obj.Length; obj.Length obj.Length]./2;
obj.Y = [-obj.Width obj.Width; -obj.Width obj.Width]./2;
obj.Z = zeros(2,2);

% Call superclass method
generate@Part(obj);

end
```

Listing B.2: *Square* generate method

An example instance of the *Square* part can be found in Figure B.1.

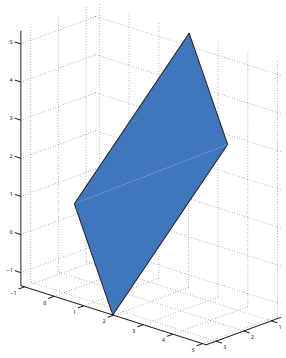


Figure B.1: *Square* geometry; `Length` = 4, `Width` = 6, `Position` = (2,2,2), `Orientation` = (45,60,15)

B.2 Module implementation

In this section a example is given how to implement a simple module into the Initiator . A design module called *SpanDoubler* will be created, which as the name suggests doubles the span of the wing(s).

B.2.1 File creation

First a directory for the new module needs to be created. Inside the `DesignModules` directory a new directory `@SpanDoubler` will be created.

Inside the `@SpanDoubler` directory, two files are required to implement the module. First the file `SpanDoubler.m` is created which will contain the class definition of our module. Secondly the file `run.m` will be created which will contain the code needed to run the module.

B.2.2 Class definition file

The file `SpanDoubler.m` will contain the class definition code. The file only contains one method, the class instantiation method. This method simply passes a handle to the controller to the superclass. The code can be found in Listing B.3.

```
classdef SpanDoubler < DesignModule
%SPANDOUBLER Doubles the span of the main wing(s)

    methods
        function obj = SpanDoubler(controllerHandle)
            obj = obj@DesignModule(controllerHandle);
        end
    end

end
```

Listing B.3: *SpanDoubler* class definition file

B.2.3 Run method

The run method contains all logic of the module and is the method which is called by the Initiator when the module needs to run. With big modules it can be advantageous to split the module into several methods and link them together in the *run* method.

The code first gets the main wings from the aircraft and then doubles the span. And then calls the *run* method of the superclass. This is required for correct functioning of the modules. The full code can be found in Listing B.4.

```
function run(obj)
%RUN Run method of the SpanDoubler module

% Get main wings
MainWings = obj.Aircraft.findPart('MainWing');

% Double span
for i = 1:length(MainWings)
    MainWings(i).Span = MainWings(i).Span * 2;
end

run@DesignModule(obj)

end
```

Listing B.4: *SpanDoubler* run method

B.2.4 Adding module

To be able to run the module inside the Initiator it needs to be added to the `modules.xml` file. This is quite straight forward, and since the module has no dependencies only its name needs to be added with a `<name>` element. Would the module have dependencies, the dependent modules need to be added with `<dependency>` elements. The relevant part of the `modules.xml` file can be found in Listing B.5.

```
<modules>
  ...
  <designModules>
    <module>
      <name>SpanDoubler</name>
    </module>
    ...
  </designModules>
  ...
</modules>
```

Listing B.5: SpanDoubler module definition

Appendix C

Sample aircraft definition file

This is the input file of an aircraft with requirements similar to the Airbus A320-200. All data is gathered from Élodie Roux - Avions civils à réaction [13].

```
<?xml version="1.0" encoding="utf-8"?>
<initiator xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:noNamespaceSchemaLocation="initiator.xsd">
    <aircraft>
        <name>A320-200</name>
        <description>Initiator file based on Airbus A320-200</description>
        <missions default="Max payload">
            <mission name="Max payload">
                <requirement>
                    <name>Pax</name>
                    <value>150</value>    <!-- [-] -->
                </requirement>
                <requirement>
                    <name>PayloadMass</name>
                    <value>20536</value>  <!-- [kg] -->
                </requirement>
                <requirement>
                    <name>CruiseMach</name>
                    <value>0.76</value>   <!-- [-] -->
                </requirement>
                <requirement>
                    <name>Altitude</name>
                    <value>11278</value>  <!-- [m] -->
                </requirement>
                <requirement>
                    <name>Range</name>
                    <value>2870</value>   <!-- [km] -->
                </requirement>
                <requirement>
                    <name>TakeOffDistance</name>
                    <value>2180</value>   <!-- [m] -->
                </requirement>
                <requirement>
                    <name>LandingDistance</name>
```

```
                    <value>1440</value>   <!-- [m] -->
                </requirement>
                <requirement>
                    <name>NumberOfFlights</name>
                    <value>100000</value> <!-- [-] -->
                </requirement>
                <requirement>
                    <name>AirworthinessRegulations</name>
                    <value>FAR-25</value>
                </requirement>
                <requirement>
                    <name>TimeToClimb</name>
                    <!-- Time [minutes] ; Altitude [meter] -->
                    <value mapType="vector">10;4000</value>
                </requirement>
            </mission>
        </missions>
        <performance>
            <parameter>
                <name>LDmax</name>
                <value>16</value>
            </parameter>
            <parameter>
                <name>SFC</name>
                <value>0.5</value>      <!-- [1/hr] -->
            </parameter>
            <parameter>
                <name>FFStartUp</name>
                <value>0.990</value>
            </parameter>
            <parameter>
                <name>FFTaxi</name>
                <value>0.990</value>
            </parameter>
            <parameter>
                <name>CLmaxLanding</name>
                <value>3.2</value>
            </parameter>
            <parameter>
                <name>CLmaxTakeOff</name>
                <value>2.2</value>
            </parameter>
            <parameter>
                <name>CLmaxClean</name>
                <value>1.2</value>
            </parameter>
        </performance>
        <configuration>
            <parameter>
                <name>WingAspectRatio</name>
                <value>9.39</value>
            </parameter>
            <parameter>
                <name>WingLocation</name>
                <value>Low</value>
            </parameter>
            <parameter>
                <name>TailType</name>
```

```
                <value>Standard</value>
            </parameter>
            <parameter>
                <name>RootAirfoil</name>
                <value>boeing-a</value>
            </parameter>
            <parameter>
                <name>KinkAirfoil</name>
                <value>boeing-b</value>
            </parameter>
            <parameter>
                <name>TipAirfoil</name>
                <value>boeing-c</value>
            </parameter>
        </configuration>
        <parts mainPart="Fuselage">
            <fuselage name="Fuselage" type="Conventional">
            </fuselage>
            <wing name="Main Wing" type="MainWing">
            </wing>
            <wing name="Horizontal Stabiliser" type="HorizontalTail">
            </wing>
            <wing name="Vertical Stabiliser" type="VerticalTail">
            </wing>
            <engine name="Engine-1" type="TurboFan">
                <location>Main Wing</location>
                <bypassRatio>6.0</bypassRatio>
            </engine>
            <engine name="Engine-2" type="TurboFan">
                <location>Main Wing</location>
                <bypassRatio>6.0</bypassRatio>
            </engine>
        </parts>
    </aircraft>
    <runList>DesignConvergence,ReportWriter,PlotTool</runList>
    <settings source="settings.xml">
    </settings>
    <moduleInputs>
        <input module="PlotTool">
            <plotModules>Geometry,DesignConvergence</plotModules>
        </input>
    </moduleInputs>
</initiator>
```

Listing C.1: XML aircraft definition file of the Airbus A320-200