Delft University of Technology

Development of a Knowledge-Based Engineering Application to Support Conceptual Fuselage Sizing and Cabin Configuration

Towards a Next Generation Multi-Model Generator

R.L.A. de Jonge

TU Delft

Development of a Knowledge-Based Engineering Application to Support Conceptual Fuselage Sizing and Cabin Configuration

Towards a Next Generation Multi-Model Generator

by

R.L.A. de Jonge

to obtain the degree of Master of Science

at the Delft University of Technology,

to be defended publicly on Thursday January 26, 2017 at 09:00.

Student number:	4005589	
Thesis registration number:	113#17#MT#FPP	
Thesis committee:	Prof.dr.ir. L.L.M. Veldhuis,	TU Delft, chairman
	Dr.ir. G. La Rocca,	TU Delft, supervisor
	Dr.ir. W.J.C. Verhagen,	TU Delft
	Prof.dr. P. Vink,	TU Delft

Source cover image: WOW Air

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

Acknowledgements

This thesis marks the final part of 7.5 years of hard work towards obtaining the degree of Master of Science in Aerospace Engineering at the Delft University of Technology. I want to use this opportunity to thank those people that have helped me in achieving this goal.

First of all, I want to thank my daily supervisor, Dr.ir. Gianfranco La Rocca, for his support, guidance and feedback over the past year. Furthermore, I would like to thank Prof.dr.ir. Leo Veldhuis, Dr.ir. Wim Verhagen and Prof.dr. Peter Vink for their willingness to be part of my graduation committee and assessing my thesis work.

I would like to thank Max Baan and Reinier van Dijk of ParaPy B.V. for their support, interest and enthusiasm.

I want to thank my parents, Jaap and Stephanie, for giving me the opportunity to pursue an academic degree and their continuous support along the way. I also want to thank my sisters, Alexandra and Fabienne, for their support.

Finally, I want to thank my friends, who have made my time here in Delft so much more valuable.

R.L.A. de Jonge
Delft, January 2017

Summary

The department of Flight Performance and Propulsion at the Faculty of Aerospace Engineering has developed the Design and Engineering Engine (DEE) concept to support Multidisciplinary Design Optimization (MDO) of complex products. The goal of the DEE is to accelerate design, analysis and optimization by automating repetitive and non-creative design activities. A central part of the DEE is the Multi-Model Generator (MMG), a Knowledge-Based Engineering (KBE) application that provides a generative modeling capability. Over the years several aircraft MMGs have been developed to support aircraft MDO.

The previous aircraft MMG was built using the KBE system GDL. Over the years, several limitations of the fuselage model included in the GDL MMG, called DARfuse, have become apparent. Therefore, this thesis describes research done into the development of a new parametric fuselage model to be included in the next generation Multi-Model Generator, currently under development within the department of Flight Performance and Propulsion. The new fuselage model is called ParaFuse and has been developed using the KBE system ParaPy. The scope of ParaFuse is limited to conventional, low-wing, passenger aircraft certified under CS 25 airworthiness requirements.

The first goal of the thesis was to develop a new parametric fuselage model. The implemented modeling approach uses longitudinal guide curves and fuselage cross sections to generate the final fuselage geometry. A separate parameterization approach is taken to provide a smooth nose end cap. Furthermore, a parameterized wing-body fairing has been implemented in ParaFuse. As a second goal, an inside-out fuselage sizing and cabin configuration method has been developed and implemented to automate the fuselage layout and sizing process. The inside-out design method can be used to automatically generate the outer fuselage geometry and interior components based on payload requirements posed by the user, such as passenger capacity and cargo type. The inside-out sizing method has been validated by reconstructing several reference aircraft using ParaFuse. On average, the error of the external dimensions of the resulting fuselages is 2% with respect to the dimensions of the fuselage of the actual aircraft. Thus, it can be concluded that the implemented inside-out sizing method can be used to accurately size the fuselage of conventional, low-wing, passenger aircraft. In addition, an outside-in cabin configuration method has been implemented in ParaFuse. This method can be used to perform cabin (re)configuration studies of fuselages with fixed external dimensions. ParaFuse is able to generate the fuselage models, including cabin interiors, within 20 seconds. This allows the user to rapidly evaluate a large number of different fuselage models.

To demonstrate the functionalities of the application, two case studies are presented in this thesis. First of all, ParaFuse has been used to generate several cabin designs for the AAR cruiser, which has been developed by the faculty as part of the RECREATE project. A second case study has been performed to evaluate the fuselage design of a regional turboprop aircraft.

The development of ParaFuse, together with the implemented fuselage sizing and cabin configuration methods, described in this thesis is a step towards a next generation aircraft Multi-Model Generator.

Contents

List of Figures

List of Tables

Nomenclature

Latin Symbols

A	Bernstein polynomial scaling coefficients	[-]
a	Semi-major axis	[m]
b	Semi-minor axsis	[m]
C	Class function	[-]
C_h	Haack series coefficient	[-]
C_p	Pressure coefficient	[-]
c	Chord	[m]
d	Diameter	[m]
h	Height	[m]
i	Index	[-]
l	Length	[m]
l_{cabin}	Cabin length	[m]
l_{pitch}	Seat pitch	[m]
l_n	Nose cone length	[m]
l_t	Tail cone length	[m]
$l_{cabin,tail}$	Length of cabin in tail	[m]
$N1$	Class function coefficient 1	[-]
$N2$	Class function coefficient 2	[-]
N_{pax}	Number of passengers	[-]
N_{sa}	Number of seats abreast	[-]
n	Order	[-]
n_{aisle}	Number of aisles	[-]
r	Radius	[m]
S	Shape function	[-]
t_{floor}	Floor thickness	[m]
t_{wall}	Cabin wall thickness	[m]
w_{aisle}	Aisle width	[m]
w_{seat}	Seat width	[m]
x	x-coordinate	[m]
y	y-coordinate	[m]
z	z-coordinate	[m]

Greek Symbols

Δ	Difference	[-]
η	normalized x-coordinate	[-]
ζ	normalized y-coordinate	[-]
θ	Body slope	[rad]
κ	Curvature	$[m^{-1}]$

Subscripts

ext	Extension
f	Fuselage
$fairing$	Fairing

le	Leading edge
n	Nose
pax	Passengers
sa	Seats abreast
t	Tail

1 Introduction

There is a large chance, that if you look up to the sky above, you will see an aircraft flying overhead. Over 100 years of research and innovation in the field of aviation has led to the aircraft that we can observe today. From a simple machine consisting of linen and wood, aircraft have evolved to immensely complicated and highly integrated vehicles [1].

The traditional aircraft design process can be split up into three distinct phases, namely the conceptual, preliminary and detailed design phase. Each of these phases in the design process comes with distinct tasks and objectives. As the name suggests, the conceptual design phase is the phase that is used to determine the baseline aircraft concept. The results of the conceptual design phase are based on simple equations and empirical data taken from reference aircraft to allow for evaluation of a large number of possible aircraft concepts. During the conceptual design phase an aircraft concept is established that fulfills requirements with regards to the aircraft layout, size, weight and performance. The conceptual design phase is highly iterative, during which, more than one aircraft concept is evaluated. Subsequently, the final concept established during the conceptual design phase is transferred to the preliminary design phase. The preliminary design phase is required to fine-tune the coarse conceptual model established in the previous design phase. Methods applied during the preliminary design phase range from physics-based computational methods to experimental testing. Finally, the detailed design phase is initiated. During this phase, a detailed design of the individual components on a subsystem level is performed. The majority of time and resources is devoted to this design phase. Furthermore, manufacturing information is generated during this design phase [2].

Figure 1.1 shows an overview of the traditional aircraft design process. From the figure it can be seen that design decisions made in the conceptual design phase significantly decrease the design freedom, while these choices are made on a limited amount of knowledge about the design. Hence, as more knowledge becomes available later in the design phase, the designer might not have the freedom to use this knowledge. This can be described as the 'knowledge paradox' [3]. In addition, the cost committed during the conceptual design phase may already be as high as 80% of the life-cycle cost [4]. Furthermore, most attention in the conceptual design phase is given to aerodynamics and propulsion, as these disciplines are the most critical in achieving performance requirements, whereas other disciplines are evaluated only later in the design process.

With the ever increasing complexity of aircraft, the traditional design approach does no longer seem suitable. According to Sobieszczanski-Sobieski et al. [5] design of complex products, such as an aircraft, involves not only trade-offs within a specific discipline but also across their boundaries as 'everything affects everything'. Several other limitations of the traditional aircraft design process have been identified. Firstly, the conceptual design phase is too short to explore the entire design space. Secondly, the analysis methods applied in the conceptual design phase are mainly based on simplified equations or statistical reference data. These methods may produce unreliable results when evaluating more unconventional aircraft concepts. Furthermore, fewer aircraft design programs are initiated and, together with a retiring workforce, this causes the loss of knowledge with respect to the aircraft design process. These observations portray the need for a new aircraft design approach [3].

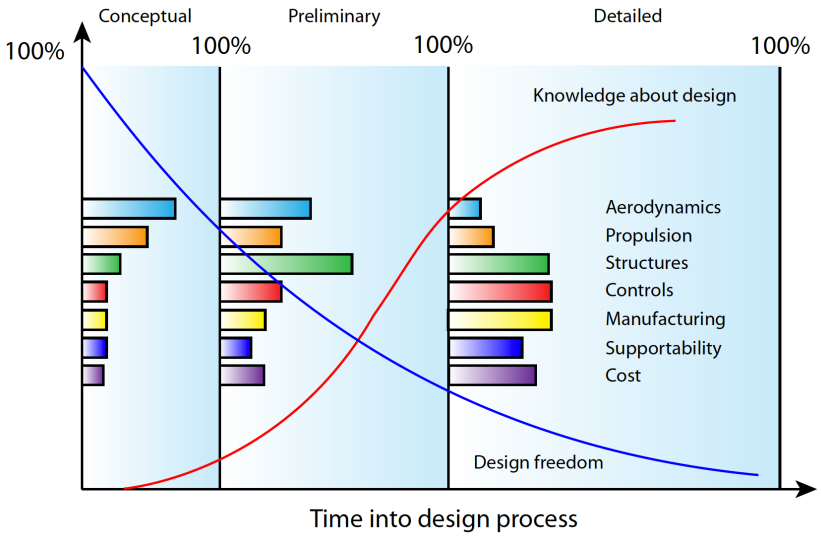


Figure 1.1: Overview of the traditional aircraft design process [6]

1.1. Multidisciplinary Design Optimization

In the past, the aircraft design process was performed manually. However, advancements in available computational power caused a shift from hand-based calculations to computerized design environments. The availability of computers and computational power first set in motion the development of aircraft synthesis software in the 1970s. With the increasing amount of available computational power a shift from the traditional aircraft design process towards multidisciplinary design optimization (MDO) could be observed in the 1980s, to overcome the shortcomings of the traditional aircraft design process [7].

As the name suggests, MDO aims to take into account all disciplines as early in the design process as possible. In doing so, the goals of MDO are to increase design freedom and available knowledge in the design process. An overview of the MDO aircraft design process and its implications on design freedom, design knowledge and committed cost is presented in Figure 1.2.

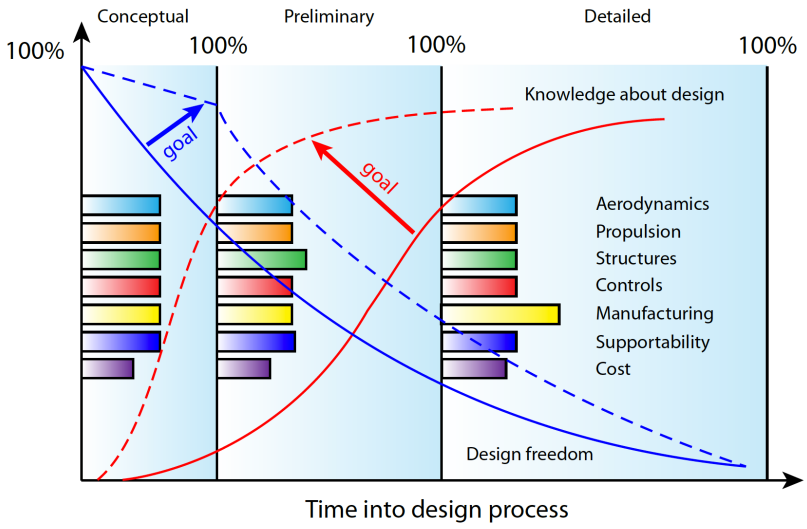


Figure 1.2: Overview of the MDO aircraft design process [6]

Several complex challenges arise when implementing a multidisciplinary design approach in the design process of complex products. As one can imagine, every discipline has its own unique optimal solution. For

example, the optimal solution with respect to aerodynamics will be different from, and possibly, conflict with the optimal solution from a structural point of view, see Figure 1.3. Hence, the MDO process will have to deal with number of different objective functions and conflicting optimal solutions. Furthermore, each discipline does not only influence itself, but might also affect the behavior of other disciplines. The MDO approach has to allow for these interactions between disciplines. Additionally, modern aircraft design is performed by multiple design teams, which works with their own set of design tools and are usually distributed over several locations. Therefore, MDO has to allow for operation in a collaborative and distributed design environment. Moreover, as the number of discipines and systems included in the model increases, the complexitity and amount of data that needs to be processed also increases very rapidly. Finally, the MDO process needs to allow for the use of Computer-Aided Design (CAD) for generation and modification of aircraft geometry during the design process. These challenges must be taken into account in order to succesfully develop a MDO framework [5].

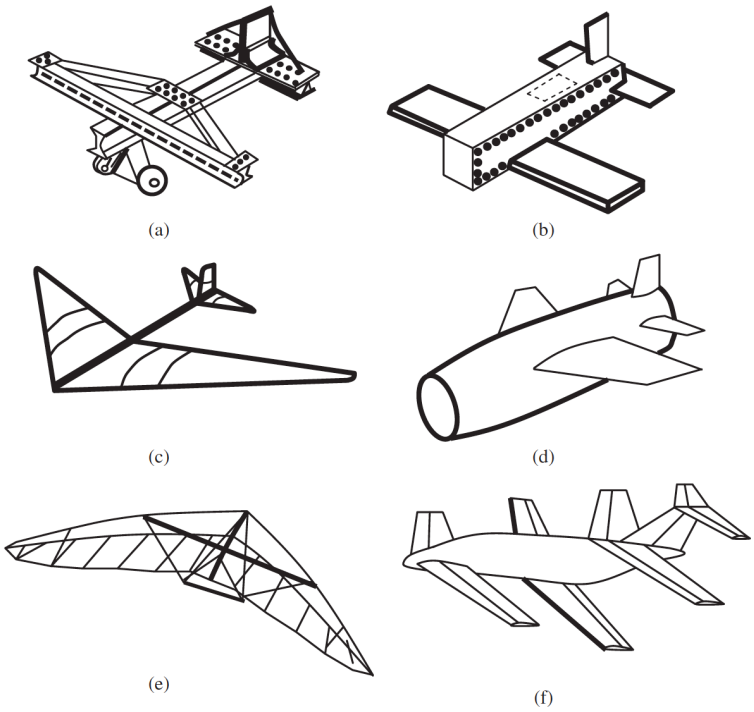


Figure 1.3: Optimal design solutions with respect to (a) Structures; (b) Manufacturing; (c) Aerodynamics; (d) Propulsion; (e) Weight; (f) Stability and Control [2]

In essence, the development of a MDO framework requires integration of a suite of analysis tools to perform analysis of the model. In addition, an optimizer should be present that can modify a set of design variables to minimize a certain cost function. An aerodynamic shape optimization application, for example, might perform some aerodynamic analysis on a wing and modify the shape of the wing with the objective of reducing aerodynamic drag. If one would want to extend this application to also take into account structural considerations, an additional interface should be added in order to allow the optimizer to communicate with the structural analysis tool. However, if the aerodynamic analysis causes the shape of the wing to change, this must also be taken into account during structural analysis. Evidently, an interface to allow communication between the different disciplinary modules within the MDO framework must be present. If the number of disciplines under evaluation increases, the number of required interfaces will increase as well. The above shows that a product model is required that can generate an instance of the design at any given time during the process [5].

A technique that is able to provide in the needs described in the previous paragraph is knowledge-based engineering (KBE). The next section will introduce KBE and show how KBE can have a role in supporting the demands of MDO.

1.2. Knowledge-Based Engineering

The previous section introduced the rationale behind a MDO approach to the design of aircraft. This section will introduce knowledge-based engineering as a technology that can support MDO of complex products. First, a definition of KBE will be given. Subsequently, a short introduction into the origins of KBE is presented. Finally, the supporting role of KBE in MDO is discussed.

Various definitions of KBE can be found in literature. La Rocca [8] identifies two papers by Chapman and Pinfold as the most cited publications in the scientific field with respect to KBE [4, 9]. According to Chapman and Pinfold, KBE can be defined as 'an engineering method that represents a merging of object oriented programming (OOP), artificial intelligence (AI) techniques and computer-aided design technologies, giving benefit to customized or variant design automation solutions.' Another definition is given by Blount, in which KBE is described as a 'true integration throughout the Product Introduction Process (PIP) supporting the ideas of concurrent engineering' [10]. According to Cooper and La Rocca, KBE is defined as 'the use of dedicated software language tools (i.e. KBE systems) in order to capture and reuse product and process engineering knowledge in a convenient and maintainable fashion' and that 'the ultimate objective of KBE is to reduce the time and cost of product development by automating repetitive, non-creative design tasks and by the support of multidisciplinary integration in the conceptual phase of the design process and beyond' [11]. An extended and complete definition of KBE is given by La Rocca in his PhD thesis: 'Knowledge based engineering is a technology based on the use of dedicated software tools called KBE systems, which are able to capture and systematically reuse product and process engineering knowledge, with the final goal of reducing time and costs of product development by means of the following: 1) Automation of repetitive and non-creative design tasks 2) Support of multidisciplinary design optimization in all phases of the design process' [3]. Many more definitions of KBE can be found in literature. Verhagen et al. [12] have performed a review of 50 papers on KBE. For a larger number of definitions the reader is referred to this paper. Verhagen et al. conclude from the literature review that the earlier definitions of KBE are more narrow and focus on geometry, whereas more recent definitions focus on the automation of repetitive research tasks [12].

From the definitions presented in the paragraph above it becomes clear that KBE is a technology that can assist the designer by automating non-creative and repetitive design tasks. To fulfill these tasks KBE applications have the ability to capture engineering knowledge. The roots of KBE can be found in the field of artificial intelligence (AI). In particular KBE stems from Knowledge-Based Systems (KBS), which emerged during the 1970s [8]. In the field of AI researchers have tried to construct systems that could solve problems in a human-like manner. However, when during the 1970s attempts at creating such systems failed, researchers in the field of AI realised that the only solution to this problem was to focus on a narrow domain in which a specific computer system would be applicable. A knowledge-based system is an example of such a system, as it is a computer system that stores knowledge in order to solve a problem in a specific field [13]. A KBS uses a reasoning mechanism to find an answer to the problem. The solution to the problem is found using some specified inputs and the knowledge stored in the system. This knowledge is stored in the KBS in a so-called knowledge base (KB). However, knowledge-based systems failed to have an impact on engineering in areas such as aerospace or the automotive industry. The main reason for this is the inability of KBSs to deal with geometry manipulation and data processing, two key aspects of a design engineering process. Knowledge-based engineering systems, which were first introduced in the 1980s, have been developed in order to overcome the shortcomings of knowledge-based systems, as they contain the knowledge and reasoning mechanisms of knowledge-based systems and posses geometry manipulation capabilities from the field of CAD [8].

As the importance of a CAD geometry engine in KBE systems cannot be denied, one might be tempted to see a KBE system as just an extension of a CAD program. The opposite is true, however. There are distinct differences between KBE and CAD systems. CAD systems are used for drafting geometries by means of 'point and click'. CAD systems allow the user to transfer their ideas to a digitized model in an interactive manner. However, the rationale behind certain design choices cannot be recorded. KBE systems, on the other hand, are able to record the ideas behind design choices. These ideas must first be translated to design rules. Subsequently, these design rules are captured in a KBE application using a programming language specific to the KBE system. Thus, KBE systems employ a code-based approach rather than the 'point and click' approach of conventional CAD systems. KBE systems use object-oriented (OO) programming languages. The object-oriented programming approach does not only provide a structure that is well suited to represent engineering problems, it also allows for high flexibility; i.e. classes, attributes and methods can be added or removed

without the need to rewrite other parts of the code and classes can be reused by creating new instances of the same class with different input parameters. More information on object-oriented programming can be found in Appendix A. Models created using KBE systems are called 'generative' models, because the application generates the model based on a set of input parameters [5].

A KBE application can be operated in two different ways. The first option is to initiate the model through the graphical user interface (GUI). The GUI allows a user to interactively manipulate the product model by modifying the input parameters. The second option, the opposite of interactive mode, is to operate the application in batch mode. That is, automatic operation of the KBE application without 'human' interference. This second type of operation is required when using the a KBE application inside a MDO framework. In that case the optimizer modifies the input parameters of the KBE application based on the results of the analysis modules [5].

The remainder of this section is dedicated to elaborating on the role of KBE to support MDO. Although the exact implementation of a MDO framework might not always be similar, typically, three distinct parts can be identified within the framework:

1. An analysis block, that triggers several analysis modules based on a given design vector.
2. A design point generator, that generates the design vector.
3. An optimizer, that searches for the design vector that minimizes the objective function.

Within a MDO framework the product model can be implemented in at least three different ways. The way in which the product model is implemented also influences the way in which KBE technology can be used to support multidisciplinary optimization. The three possibilities are listed below and their implications on the supporting role of KBE will discussed in the following paragaphs [5, 14].

1. Geometry-less
2. Geometry grid-perturbation
3. Geometry-in-the-loop

The geometry-less implementation of a product model in a MDO framework is often used for design tools that are applied in the design synthesis phase or early in the conceptual design phase. An overview of a geometry-less implementation of a MDO framework is shown in Figure 1.4. The aircraft is described using a set of mathematical equations instead of the actual geometry. The mathematical model is based on simplified equations or empirical data. Although analysis can be performed very rapidly, allowing for a large number of iterations, the applicability of such a model in case of unconventional aircraft designs is limited. The role of KBE in a geometry-less framework is limited to generation of the geometry based on the design vector that minimizes the objective function. Hence, KBE would not provide any benefits as compared to using a conventional CAD system to generate the final geometry [5].

The second MDO framework under consideration is one with a grid-perturbation model. Instead of using the geometry generator after the optimization, the grid-perturbation model requires the geometry to be generated before entering the optimization loop. An overview of the grid-perturbation MDO system can be seen in Figure 1.5. In the grid-perturbation model a grid of points is generated that represents the shape of the aircraft. Subsequently, the optimizer is able to modify the shape by changing the position of the discrete points of the grid. Several limitations arise when applying a grid-perturbation model in a MDO framework. First of all, because the position of individual points can be modified it is difficult to maintain shape control. Therefore, the distance the points on the grid are allowed to move is limited, which limits the applicability of this MDO process to fine-tuning of a configuration [14]. Furthermore, a computational grid is tailored to one main discipline. Hence, the possibility to evaluate multiple disciplines is limited. The applicability of KBE to support the optimization process would be in the generation of the geometry that is used to create the grid [5].

The geometry-in-the-loop product model is the final implementation of a MDO framework under consideration. This approach places the product model, as the name suggests, inside the optimization loop. Thus, for each iteration an updated representation of the model is created. In this case, KBE truly becomes an enabler of the MDO process. Not only can analysis be performed on a representation of the actual model, large topological variations are allowed as well. Furthermore, KBE also allows capturing of non-geometrical

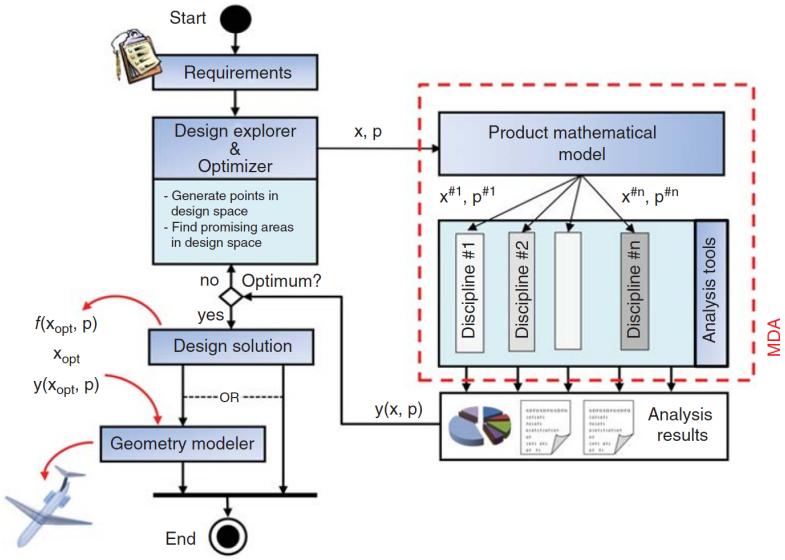


Figure 1.4: Implementation of a geometry-less MDO framework [5]

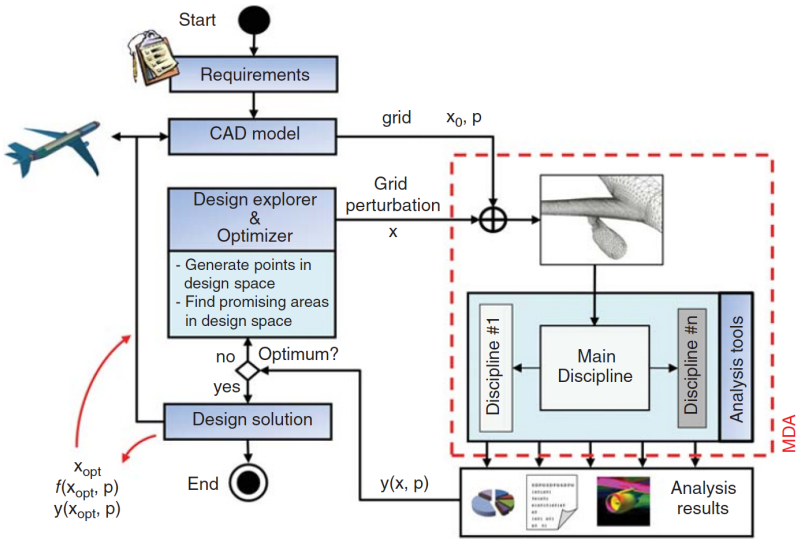


Figure 1.5: Implementation of a grid-perturbation-based MDO framework [5]

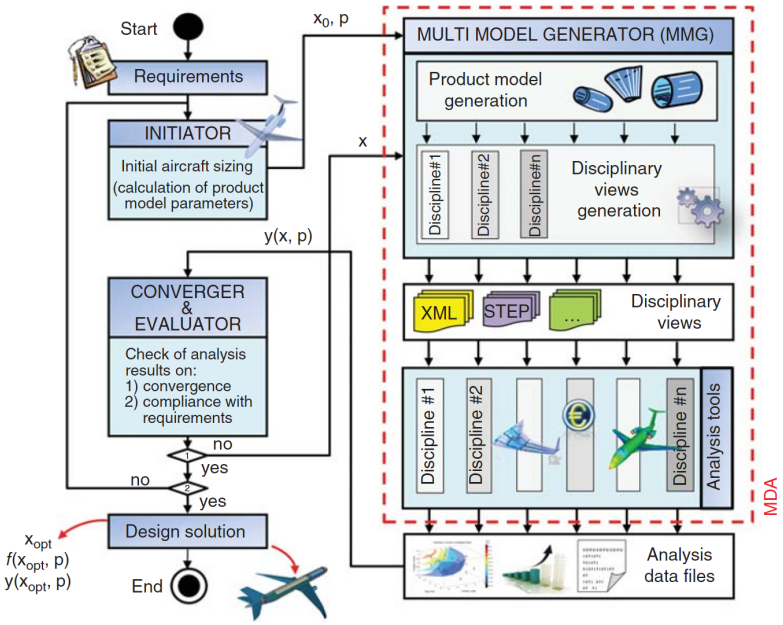


Figure 1.6: The Design and Engineering Engine (DEE) concept, a model-in-the-loop MDO framework [5]

information that might be required in the optimization process. The optimizer in the MDO framework is able to search the entire design space, generating a new model on each iteration. It is here that the code-based nature of KBE applications becomes most valuable. The main drawback is the complexity of the model generator that is required to be placed in the loop. The model generator must be robust, able to quickly generate a geometry and allow for automatic generation of discipline-specific abstractions of the product model. Several successful applications of generative models used in a geometry-in-the-loop MDO system can found in literature; Vandenbrande et al. [14] present the Boeing General Geometry Generator (GGG). Within the MOB project a Computational Design Engine (CDE) was created to support multidisciplinary optimization of a blended-wing body aircraft, this CDE used a generative ICAD model to support the geometry-in-the-loop MDO [15].

In conclusion it can be said that KBE can be seen as an enabler of MDO, when a generative model is placed within the optimization loop. The knowledge capturing and reusing mechanism of KBE allows for the generation of robust parametric models, which can be used to represent the aircraft at any stage in the design process. Furthermore, KBE is able to record both geometrical and non-geometrical information, so that the translation of the product model to data sets required for various analysis modules can be automated [5]. The faculty of Aerospace Engineering has been working on the development of a model-in-the-loop MDO system, called the Design and Engineering Engine (DEE), which will be introduced in the following section.

1.3. The Design and Engineering Engine concept

The department of Flight Performance and Propulsion of the Faculty of Aerospace Engineering at Delft University of Technology has been working on the development of the Design and Engineering Engine (DEE) concept to support aircraft multidisciplinary design optimization. Since 2002, the department has developed several DEEs for various applications. It is this DEE concept that is presented in Figure 1.6 to illustrate a model-in-the-loop MDO system. The aim of the DEE is to support and accelerate aircraft MDO by automating non-creative and repetitive design activities. The system consists of modules, that are loosely-coupled. This allows the user to tailor the design and analysis workflow to the specific needs of the design problem, i.e. some modules, that are not relevant for the design problem, may be left out [8].

The DEE concept consists of five main components. For this thesis, the implementation of the DEE concept for aircraft design is relevant. Therefore, the main components of the Aircraft DEE are introduced in this

paragraph. The first component is the Initiator, an aircraft design synthesis tool, that is under continuous development at the chair of Flight Performance and Propulsion. The aim of the Initiator is to provide an initial feasible design solution through a process called 'feasilization' [16]. The Initiator can provide an initial set of input parameters to be used in the multidisciplinary optimization process. An in-depth overview of the Initiator for aircraft design is presented by Elmendorp, Vos and La Rocca [17]. The second component of the DEE is the Multi-Model Generator (MMG). The MMG is a KBE application and the part of interest for this thesis. The next section will give a more thorough introduction of the MMG concept. The third component of the DEE is the set of analysis tools. These analysis tools span the range of disciplines involved in the aircraft design process. However, within a specific discipline, multiple analysis tool can provide the user with various levels of fidelity. Furthermore, a Converger and Evaluator is present that is used to check convergence and compliance with the requirements and constriants. This functionality is provided by an optimizer [8]. The final component is a communication framework that allows the different parts of the DEE to communicate. Over the years several communication frameworks, which were either developed in-house or commercially available, have been used for various implementations of the DEE concept [18].

1.4. The Multi-Model Generator

The Multi-Model Generator was introduced in the previous section as the KBE application that provides the model-in-the-loop functionality of the DEE concept. The MMG serves two main goals. First of all, the MMG provides a parametric modeling evironment. The second goal of the MMG is to automate the generation of abstractions required for several disciplinary analysis tools. The MMG is the enabler of MDO within the DEE concept.

Over the years, several MMGs have been developed to support various implementations of the DEE concept. For this thesis, the aircraft MMG is of interest. The parametric modeling environment is provided by the product model, as can be seen in Figure 1.6. The product model within the aircraft MMG is built up using High Level Primitives (HLP). Analogous to how LEGO bricks can be used to create a wide variety of models, one can see HLPs as the parameterized building blocks that make up the product model [3]. In case of the aircraft MMG separate HLPs are defined to build fuselage, wing, engine and connecting parts. The set of input parameters determines the final shape of the HLP. Figure 1.7 shows how the wing HLP can be instantiated in a number of different ways to create a large variety of aircraft models.

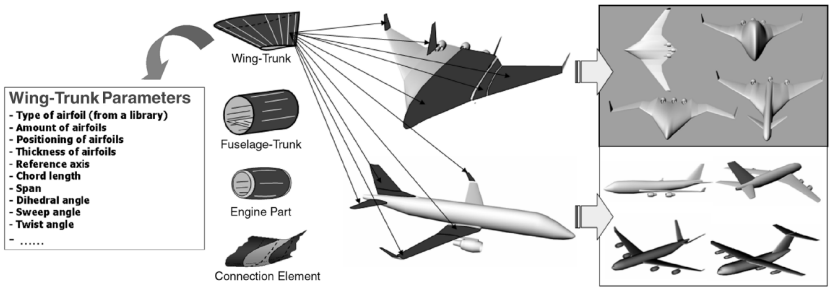


Figure 1.7: High Level Primitives of the ICAD Aircraft Multi-Model Generator [19]

The second goal of the MMG is fulfilled by capability modules (CM). These capability modules draw information from the generative product model and transform this information into abstractions that can be used by different disciplinary analysis tools. For example, a capability module may take the outer surface of an aircraft and use this surface to generate the computational mesh required for aerodynamic analysis of the aircraft. Because the different capability modules draw information from the same geometrical model consistency along the different analysis tools can be achieved.

As mentioned in the previous section the DEE concept has been under development since 2002. That the DEE concept can be used to successfully design unconventional aircraft configurations using a MDO approach was demonstrated in the European project MOB. In this project a blended wing body (BWB) aircraft was designed in a distributed environment composed of European research institutions, universities and industry [15]. This first generation MMG, used in the MOB project, was created using the KBE System ICAD. ICAD was the first commercially available KBE system and used the ICAD Design Language (IDL), which was

based on one of the object-oriented Common Lisp dialects [5]. The company developing ICAD was acquired by Dassault Systemes in 2002 and, subsequently, discontinued [8]. Therefore, the faculty of Aerospace Engineering started developing a next generation MMG, based on the KBE system GDL, produced by Genworks International and based on the programming language ANSI Common Lisp and CLOS (Common Lisp Object System) [5]. More information on the GDL-based MMG can be found in [6, 20, 21].

Now, several years after the development of the GDL-based MMG, the chair of Flight Performance and Propulsion is initiating a project to develop a next generation MMG to overcome the shortcomings of the GDL-based MMG. How the research described in this thesis report fits within this context is described in the next section.

1.5. Research motivation and thesis goals

The research presented in this thesis fits within the goal of the chair of Flight Performance and Propulsion to develop a Design and Engineering Engine to support multidisciplinary design optimization of aircraft configurations. The key technological enabler of the DEE is the Multi-Model Generator, a KBE application that provides the DEE with a generative modeling capability. Several multi-model generators have been developed to support multidisciplinary design optimization of aircraft; a MMG based on the KBE system ICAD [3] and a MMG based on the KBE system GDL [21]. Currently, a next generation MMG is under development at the chair of Flight Performance and Propulsion to overcome limitations of the GDL-based MMG.

The research presented in this thesis describes the development of a new fuselage high-level primitive. The fuselage HLP, DARfuse, of the GDL MMG was developed by Brouwers [20, 21]. DARfuse is a KBE application that is able to generate the fuselage geometry, including cabin interiors, of conventional passenger aircraft. The term 'conventional' is used here to refer to a low-winged, passenger aircraft certified under EASA CS 25 certification requirements. An example of a fuselage geometry generated using DARfuse is shown in Figure 1.8.

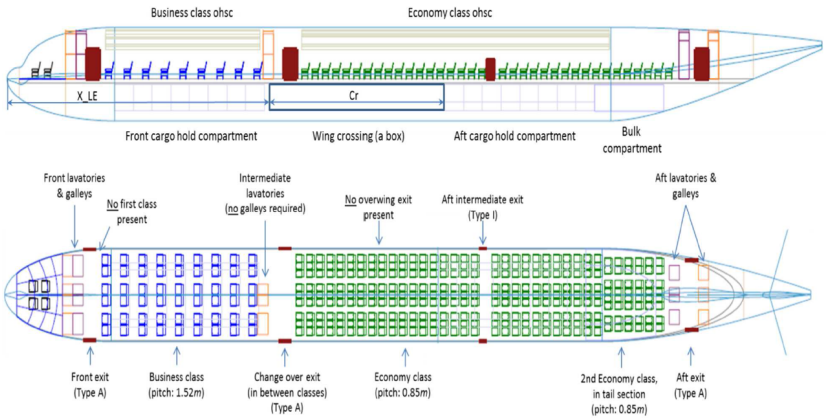


Figure 1.8: DARfuse fuselage geometry [20]

The thesis work of Brouwers consists of two parts, nameley a HLP of the fuselage and an initiator module to size the fuselage using an inside-out design approach. The inside-out fuselage design approach uses the required payload as a starting point for the fuselage sizing process. Thus, the design of the fuselage is driven by the number of passengers and cargo requirements. These payload requirements determine the amount and type of interior components. For example, top level requirements may state that a certain fuselage should be designed to carry a payload of 100 passengers, with 10% of the passenger capacity in business class and the remainder in economy class. For each passenger a certain amount of galley volume should be reserved to store food and drinks etc. Furthermore, an appropriate amount of lavatories should be placed in the cabin. Additionally, an appropriate number of emergency exits, as specified by certification requirements, should be present to allow for safe evacuation in case of an emergency. Finally, a cargo bay should be present to carry the luggage of the passengers. The amount and dimensions of each of these components is then used to determine the external dimensions of the fuselage.

Over the years, several limitations of DARfuse have been identified. First of all, the parametric definition of the fuselage in DARfuse did not produce fully satisfactory fuselage geometries, especially with respect to the nose section of the fuselage, as can be seen in Figure 1.8. These unrealistic shapes were caused by the fact that the parameterization of the nose cone used elliptical quadrants as guide curves, which caused the resulting surface to become 'blunt'. The main function of the MMG is to generate abstractions of the central product model to feed several different analysis tools. The external shape of the fuselage has a large influence on the aerodynamic performance of the aircraft. Therefore, the parameterization methods used to generate the surface must allow the user to generate realistic fuselage shapes. Thus, attention must be given to the paramerization of the fuselage to make sure that realistic fuselage shapes can be achieved.

Another limitation of DARfuse is that it is only able to generate fuselage models using the inside-out design approach. Because of this, it is not possible to generate a fuselage, with the same external dimensions as the example mentioned above, but now with a larger business class, or even an additional first class cabin section. In this case, a capability is required that allows for cabin configuration of fuselages with fixed dimensions. It must be noted that such a design case is quite common, as aircraft manufacturers usually offer a specific type of aircraft with different cabin configurations. To perform these kind of studies the application should allow for outside-in design studies. In practice, the lack of an outside-in cabin configuration method has proven to be a limitation of the GDL MMG as in some projects the faculty has been involved in, a baseline aircraft had already been designed. As DARfuse did not allow the user to perform outside-in cabin configuration studies, one had to iteratively try to match the DARfuse fuselage dimensions to the baseline requirements by modifying number of passengers, number of seats abreast or seat spacing. This is a very tedious process and constrasting with the rationale of KBE, as its aim is to automate repetative and non-creative design tasks. Therefore, to overcome this limitation, one of the requirements of the fuselage HLP described in this thesis is that it should be able to perform both inside-out and outside-in design studies.

Some limitations in the cabin configuration process of DARfuse have also been identified. First of all, the required number and type of emergency exits in DARfuse are determined based on the total amount of passengers. In case of a high density cabin configuration, i.e. one class, this is not a problem. However, when a large business class is present, the resulting fuselage contains significantly less passengers than an aircraft with the same dimensions in high density configuration. In reality, the amount and type of emergency exits are always determined using the passenger capacity of the cabin in high density configuration. The new cabin configurator should account for this fact. Furthermore, if multiple classes are present the cabin configurator of DARfuse positions exits in between the different seat classes. This is not realistic, as the emergency exits of actual aircraft are always positioned uniformly over the length of the fuselage. Another limitation of the DARfuse cabin configurator is that it does not account for the fact that possibly less seats can be placed abreast in the tapered nose cone section, but only takes this into account for the tail cone section. The cabin configurator of the new HLP should take this nose cone tapering into account.

In order to successfully use the KBE application in an optimization framework it is required that the time to generate the model is as low as possible, especially in the early design phase when a lot of iterations are required. However, the generation of a complete fuselage model in DARfuse required 2-3 minutes [20]. One of the reasons for the long runtime can be explained by the fact that DARfuse required MATLAB to perform the least-squares fitting of the main fuselage cross section. This required the application to launch MATLAB everytime a modification was made to the fuselage. Another reason can be found in the fact the generation of each different component of the model takes some time. In DARfuse, the generation of the cabin seats required the tool to create a separate instance of each individual cabin seat. For a cabin with for example 200 seats, these small amounts of time quickly accumulate to a significant amount of time. A solution to these problems should be implemented in the new fuselage HLP as in order to allow the application to be used inside an optimization framework, the runtime of the application must be reduced to the order of seconds.

Finally, a new KBE system has been introduced within the chair of Flight Performance and Propulsion called ParaPy. Unlike ICAD and GDL this system is built on top of the Python programming language rather than Common Lisp. Due to a very active Python user community a vast amount of third party libraries are available. This allows the user to incoporate for example optimization libraries directly in the application itself, rather than relying on MATLAB or other software. Furthermore, the good readability of the language will allow developers to more easily maintain the applications created using ParaPy.

As described in the introduction of this chapter, an aircraft is a highly integrated vehicle with a large number

of interdepencies. Therefore, a central generative model is needed that is able to serve the entire design process. The model should be able to serve the initialization phase of the design process, as well as the detailed design phase. With respect to the fuselage this has several implications. For example, in the initialization phase of the design process, the external dimensions of the fuselage might be most important. However, in the detailed design phase, the exact positions of the internal components are required to be able to generate the lay-out of the electrical wiring and interconnection systems (EWIS). Zhu describes the use of KBE technology to automate the design of the EWIS in his PhD dissertation [22]. The ability to automate this EWIS design process is becoming more and more relevant as the amount of electrical systems in the aircraft keeps increasing. A generative fuselage model can be used as a basis to serve both purposes. Ultimately, the aim of creating this KBE application is to enable studies, that were traditionally only taken into account later in the design process, to be performed earlier in the design process. Thus, increasing knowledge about the design earlier in the design process.

The limitations and opportunities described above have been used to set the following thesis goals:

1. Develop a parametric conventional fuselage model using the KBE system ParaPy.
2. Implement an inside-out design method to allow for fuselage sizing studies based on top level requirements, such as passenger capacity and type of cargo.
3. Implement an outside-in method to allow for cabin (re)configuration studies using predefined fuselage dimensions.

1.6. Thesis structure

In the remainder of this thesis ParaFuse, a knowledge-based engineering application to support conceptual fuselage sizing of conventional aircraft fuselages, is introduced as a step towards a next generation Multi-Model Generator. In this section an overview of the thesis report is given.

In the following chapter, **Chapter 2**, the requirements for the fuselage HLP will be introduced. The requirements will be introduced by first defining the scope of the application. Following this, the requirements of the KBE application will be stated grouped with respect the the outer geometry, fuselage interior and operational aspects of the application.

Subsequently, the parametric definition of the HLP will be presented in **Chapter 3**. This chapter will elaborate on how the fuselage surface geometry is defined in the KBE application and how the outer surface is generated

ParaFuse is able to automate the fuselage sizing and cabin configuration process. To do this, knowledge with respect to the design of the fuselage and cabin has been captured inside ParaFuse. **Chapter 4** outlines how this design knowledge has been acquired and captured.

After that, the implementation of the two sizing and cabin configuration methods, inside-out and outside-in, are described in **Chapter 5**. Furthermore, several validation cases will be presented to assess the applicability of ParaFuse.

In **Chapter 6**, two case studies are presented to illustrate the functionalities of the KBE application. This chapter will serve as an example on how ParaFuse can be used in the aircraft design process.

Finally, conclusions and recommendations with respect to the development and functionalities of ParaFuse are presented in **Chapter 7**.

2

ParaFuse HLP concept

Chapter 1 has introduced knowledge-based engineering as a technology to support aircraft multidisciplinary design optimization. Furthermore, it has outlined the research motivation and goals of this thesis. The objective is to develop a new fuselage HLP. The new fuselage HLP has been named 'ParaFuse'. The scope of ParaFuse is discussed in Section 2.1. Subsequently, a more elaborate set of requirements, in addition to the goals set in Chapter 1.5, will be established for ParaFuse in Section 2.2. Finally, an overview of the main processes describing the operations in ParaFuse are given by means of UML activity diagrams in Section 2.3.

2.1. Scope of ParaFuse

To start the development of the new fuselage HLP the scope of the application should first be established. The research motivation in Section 1.5 has outlined the limitations of DARfuse, the previous fuselage HLP developed by Brouwers. DARfuse has been used as a starting point for the development of ParaFuse. ParaFuse should therefore have at least the functionalities of DARfuse, with the addition that the limitations described in Section 1.5 are overcome.

The scope of ParaFuse is limited to conventional, low-wing, passenger aircraft certified under CS 25 airworthiness regulations. For the current implementation of ParaFuse, the scope is limited to single deck passenger aircraft only. Thus, double deck aircraft are not included in ParaFuse. Furthermore, cargo aircraft or aircraft with a combination of passengers and cargo in the upper fuselage cross section cannot be modelled using ParaFuse.

To support the design of fuselages of passenger aircraft an inside-out fuselage sizing and outside-in cabin configuration method have been implemented. The aim of the inside-out fuselage sizing method is to enable the user to evaluate the resulting fuselage dimensions based on top level requirements such as passenger capacity and cargo type, whereas the outside-in cabin configuration method can be used to determine the passenger capacity of a fuselage with fixed dimensions. Both methods should allow the designer to specify different levels of passenger comfort. This means ParaFuse should be able to model fuselages with variable seat class distributions, aisle widths, seat spacing and clearance constraints in the cabin. Furthermore, the user should be able to specify the required amount of lavatories and galleys in the cabin.

2.2. Requirements of ParaFuse

In Section 1.5 the top level goals of this thesis were introduced. This section elaborates on these goals in more detail by specifying the requirements the KBE application should satisfy. The requirements are divided in three groups, namely outer geometry, fuselage interior and operational aspects. As the name suggests, the requirements with respect to the outer geometry specify which geometrical features the tool should be able to model. The requirements related to the outer geometry are presented in Section 2.2.1. The requirements with respect to the fuselage interior deal with the components that should be included in the cabin configuration process and are presented in Section 2.2.2. The operational requirements specify how the user should be able

to operate ParaFuse and are elaborated on in Section 2.2.3. A summary of the requirements is given in Section 2.2.4.

2.2.1. Outer geometry

As explained in Chapter 1 the MMG concept consists of a generative modeling capability and various capability modules to automate the preprocessing activities required to feed various analysis tools. Especially for aerodynamic analysis, it is important that the model is able realistically represent common fuselage shapes. An investigation into existing conventional fuselages of passenger aircraft has been conducted in order to establish a set of geometrical features that must be included in the geometrical representation of the fuselage. A breakdown of the identified geometrical features of conventional aircraft fuselages is given in Figure 2.1[1].

As can be seen from Figure 2.1 a clear distinction can be made between a blunt nose section and a blended nose section. The latter case is common on newer aircraft, such as the Boeing 787 or Airbus A350. An explanation can be found in the fact that the blended windshield reduces local supervelocities on the nose section of the fuselage, as can be seen from Figure 2.2. In this figure the contours of the pressure coefficient are visualized on the nose section of the fuselage. The calculations for this figure are performed using potential flow calculations [23]. To be able to perform these kinds of studies, ParaFuse, should have the ability to model both types of nose cone sections.

With respect to the main cross section of the fuselage, four distinct shapes have been identified, namely circular, elliptical, double bubble and a more general cross-sectional shape which has been indicated as free form. Torenbeek [24] states that an increase in cross section diameter of 10% could cause an increase of as much as 3% in total drag of the aircraft. Therefore, the ability to generate aircraft fuselages with a minimum cross-sectional area, whilst satisfying cabin comfort requirements, is an important feature of a conceptual design tool as the cross-sectional area can have a large influence on the performance of the aircraft. Figure 2.3 presents the results of a case study performed by Sóbester and Forrester [25] on the cross-sectional shape of conventional aircraft fuselages. As can be seen, an increase in flexibility in the parameterization of the fuselage cross section can yield lower cross-sectional area, albeit at the cost of more function evaluations of the optimization problem. Therefore, ParaFuse should include a functionality to select the desired cross-sectional shape. Such a functionality will allow for the evaluation of the effect of the fuselage cross-sectional shape on the performance of the aircraft.

Finally, several distinct shapes of the tail cone are identified as can be seen in Figure 2.1. The most common tail cone shapes are either circular or elliptical. In case of aircraft with an entrance in the tail cone, for example in military transport aircraft, the tail cone shape is more flat. In order to evaluate the effect of the tail cone shape on the performance of the aircraft, ParaFuse, should allow for modeling these kind of tail cone shapes.

Additionally, interference effects between the wing and fuselage have a detrimental effect on the aerodynamic performance of the aircraft, causing an increase in direct operating costs. Obviously, an operator will always strive to minimize the operating cost of the aircraft in order to maximize profit. Therefore, research into reducing interference effects plays an important role in aircraft design in order to reduce drag and improve flight performance [26]. Siegel [27] presents the following three options to improve the flow around a wing-fuselage junction: 1) Optimization of the relative wing-body position. 2) Modification of the shape of the junction by using fairings or fillets. 3) Manipulation of the flow with active installations. According to Della Vecchia and Nicolosi [26] the most effective method to reduce interference drag is by using fairings between the wing and fuselage. The wing-body fairing is usually not taken into account early in the design process. Thus, the implementation of a parameterized wing-body fairing in ParaFuse would not only allow for studies into the (reduction of) interference effects, but could potentially allow these studies to be conducted earlier in the design process.

[1] Image sources: Airasia A320: planepictures.net. Cross sections: Embraer Commercial Aviation (E190), Karsten Palt (SD3-60), Ashir (A300), aircraftresourcecenter.com (A380). Tail: David Monniaux (A380), Wolfgang Kleuser (E190), Marc-Antony Payne (A400M). Nose: Kazim Alikhan (A320), Boran Pivcic (787)

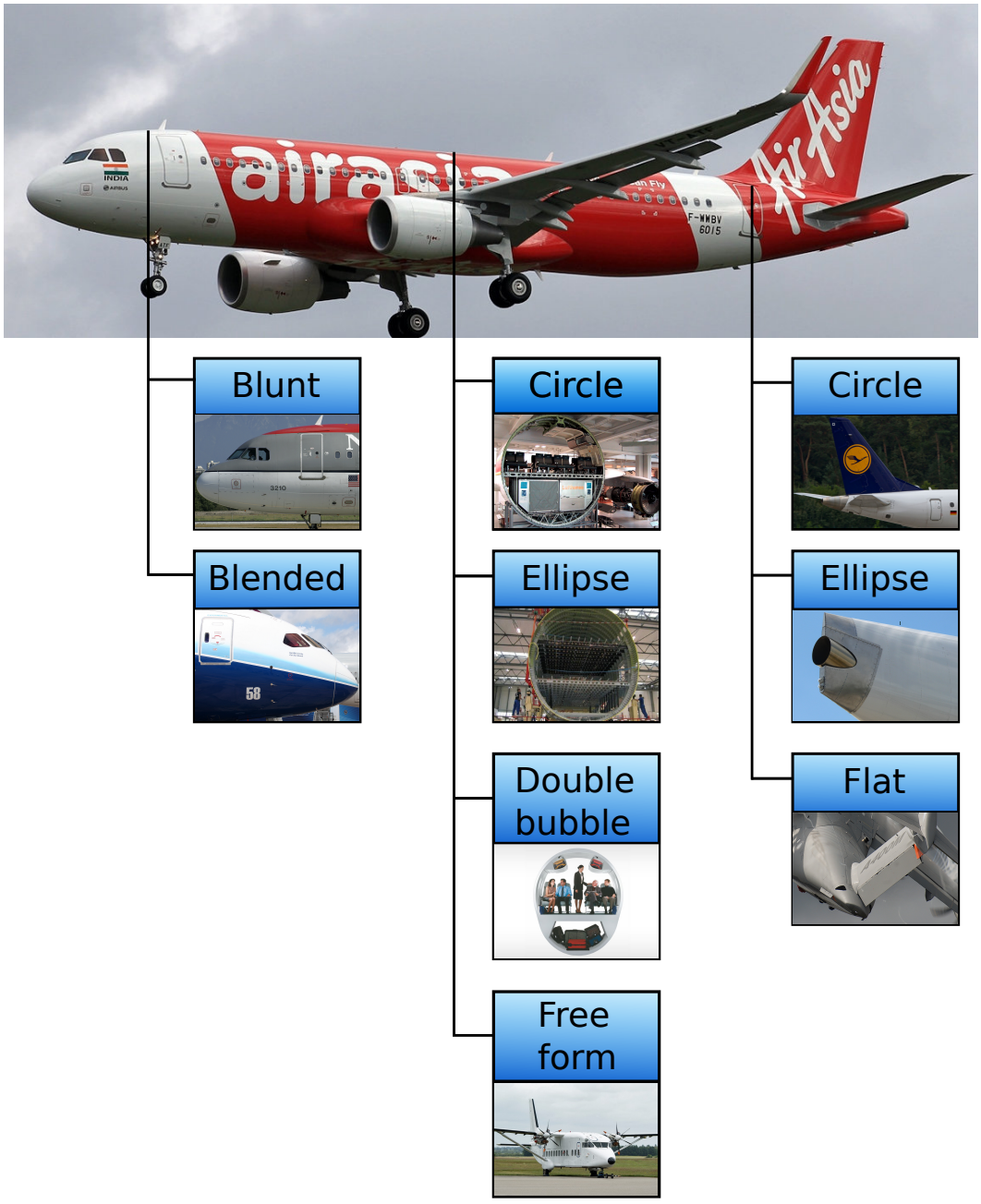


Figure 2.1: Geometrical features of conventional transport aircraft fuselages

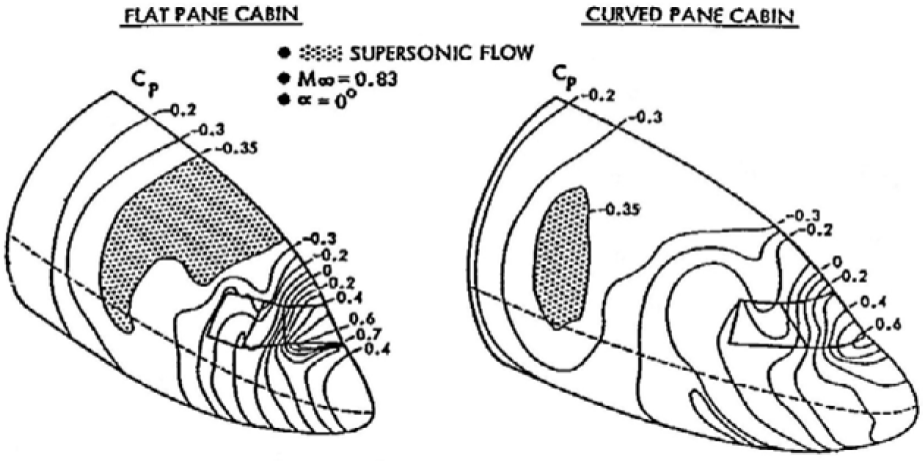


Figure 2.2: Potential flow calculations on a flat and curved pane windshields [23]

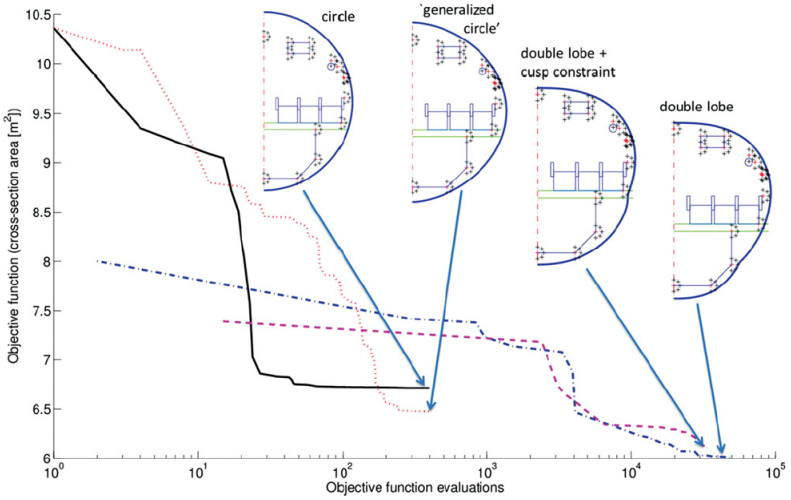


Figure 2.3: Influence of higher flexibility in cross-sectional shape definition with respect to resulting cross-sectional area and objective function evaluations [25]

2.2.2. Fuselage interior

Two approaches can be taken to fuselage and cabin design, namely inside-out and outside-in. The inside-out design approach can be used to determine the outer fuselage dimensions based on the required payload. For conventional passenger aircraft this means that the inside-out design is driven by the required amount of passengers and the type of cargo that must be stored inside the fuselage. The outside-in method can be used to perform cabin configuration studies based on a fuselages with fixed external dimensions. The need for these two design approaches was discussed in Section 1.5. In this section the requirements of these two design approaches will be elaborated on.

In order to generate correct representations of conventional passenger fuselages the components listed below need to be included in the cabin configuration process [28].

- Passenger seats in multiple class lay-out
- Cargo compartments (that allows for containerized and bulk cargo)
- Lavatories (amount determined by number of passengers)
- Galleys (volume determined by number of passengers)
- Cockpit seats and appropriate flight deck length
- Exits
- Space reserved for wing box
- Cabin walls
- Cabin floors

In addition, the KBE application must be flexible enough that it can model various types of cabins, with variable amounts of cabin comfort. Therefore, the user must be able to specify seat dimensions, clearance constraints with the cabin sidewall, aisle height and number of aisles. If these variables can be taken into account by the cabin configuration methods, ParaFuse will be able to perform several different cabin configuration studies. For example, one would be able to judge the effect of additional seat spacing on the external dimensions of the aircraft, while using the inside-out design approach. The outside-in cabin configuration method could be used to generate cabins with different seat class distributions for a fuselage with fixed dimensions. Another example of a study that would require the ability to generate different cabin layouts is presented by Fuchte [29]. In his PhD dissertation, Fuchte describes research into the influence of cabin layout on the direct operating cost of the aircraft, with a focus on airport turnaround time. The objective of his research is to determine a capacity limit above which two aisles become more efficient than single aisled aircraft.

The EASA CS-25 requirements for large aeroplanes are taken into account in the fuselage design process [30]. The certication requirements that must be taken into account in the cabin configuration process are listed below:

- CS 25.807: Emergency exits
- CS 25.813: Emergency exit access and ease of operation
- CS 25.815: Width of aisle
- CS 25.817: Maximum number of seats abreast

2.2.3. Operational requirements

This section presents requirements with respect to the operation of ParaFuse. First of all, ParaFuse should allow for flexibility in the required input parameters. In an early stage of the design one only wants to adjust a few parameters to get a general idea of the fuselage dimensions. In a later stage of the design, one might want to adjust more parameters to modify the shape of the fuselage surface. Furthermore, ParaFuse should be able to read input parameters from a separate input file. The use of separate input files provides a good overview of the chosen input parameters and, additionally, one is able to efficiently evaluate and store different sets of input parameters for different applications. If ParaFuse is to be used within an optimization framework

it is required that fuselage models can be generated in a short amount of time. Therefore, as an operational requirement, the time to generate a fuselage model with ParaFuse should be in the order of seconds.

To allow the geometries of ParaFuse to be used by different programs, a capability should implemented to export the fuselage geometry to several standard data exchange formats such as .STL, .STEP and .IGES. This functionality will allow the geometry to be used outside the ParaPy framework. As described in the introductory chapter, modern aircraft design is conducted in multiple design teams that are often not on the same location. Moreover, ACARE describes the establishment of multidisciplinary technology clusters between industry, research institutions and academia as one of the goals for aircraft research in 2050 [31]. According to Nagel et al. [32] the duration of such aircraft design projects is approximately three years and because of this the development of dedicated homogeneous software is not a feasible option. In an effort to alleviate risks and workload involved in coupling different tools in such clusters a standardized data model called Common Parametric Aircraft Configuration Schema (CPACS) has been proposed by the German Aerospace Center to serve as a common language in aircraft design. CPACS is implemented as a Schema Definition (XSD) for the Extensible Markup Language (XML) [32]. A functionality to translate the generated fuselage geometry to the CPACS data format must be implemented in the KBE application. This capability will provide opportunities to use the fuselages generated by ParaFuse by tools that have the ability to read aircraft geometries from CPACS files.

2.2.4. Summary

The requirements of ParaFuse, that have been established in the previous sections, are summarized below:

1. Outer geometry
   (a) ParaFuse should be able to model blunt and blended nose cone shapes
   (b) ParaFuse must be able to model four distinct cross-sectional shapes:
      i. Circle
      ii. Ellipse
      iii. Double bubble
      iv. Free form
   (c) ParaFuse must provide a capability to model three different tail cone shapes:
      i. Round
      ii. Ellipse
      iii. Flat
   (d) ParaFuse should be able to generate a wing-body fairing.
2. Fuselage interior
   (a) ParaFuse should allow configuration of the interior components based on both the inside-out and outside-in design approach. The components that should be included in the cabin configuration process are:
      - Passenger seats in multiple class lay-out
      - Cargo compartments (that allows for containerized and bulk cargo)
      - Lavatories
      - Galleys
      - Cockpit seats and appropriate flight deck length
      - Exits
      - Space reserved for wing box

- Cabin walls
- Cabin floors

(b) The following variables should be included in the cabin configuration process to allow studies with regard to the layout of the cabin:

- Number of aisles
- Aisle width
- Aisle height
- Cabin clearance constraints
- Seat pitch

(c) The fuselage design should be compliant with EASA CS 25 requirements

3. Operational requirements

(a) ParaFuse must allow for the use of separate input files to operate the application

(b) ParaFuse should allow for a variable amount of inputs, depending on the requirements of the user

(c) The time required to generate a fuselage model should be in the order of seconds

(d) ParaFuse should be able to export fuselage geometries using several standard data exchange formats:

- CPACS
- .STL, .IGES, .STEP

(e) To allow for quick evaluation of the generated fuselage models ParaFuse should be able to generate a summary of the main fuselage characteristics

2.3. ParaFuse concept workflow

DARfuse, and its limitations described in Section 1.5, have been used as a starting point for the development of ParaFuse. From this starting point, the scope and requirements of ParaFuse have been established. The application is required to allow for two different types of operation, either using the inside-out or outside-in design method. In this section two UML activity diagrams are presented that describe the required processes to generate the complete fuselage model, i.e. outer surface and cabin interior.

The activity diagram for the inside-out process is shown in Figure 2.4. As can be seen, the cabin interior is generated first to determine the outer fuselage dimensions. The outer fuselage surface is generated afterwards using this information. The process required for the outside-in design approach is described in Figure 2.5. This process uses information specified by the user with respect to the fuselage dimensions to reconstruct the outer fuselage surface. Subsequently, the available space in the fuselage is filled with the required interior components. After instantiation, the fuselage model, generated by both approaches, can be inspected in the ParaPy graphical user interface. The generated fuselage can be exported to several CAD formats such as .STL, .STEP or .IGES. Moreover, the fuselage surface can be exported as a CPACS file. Finally, a design report can be generated to provide the designer with a summary of the characteristics of the generated fuselage model.

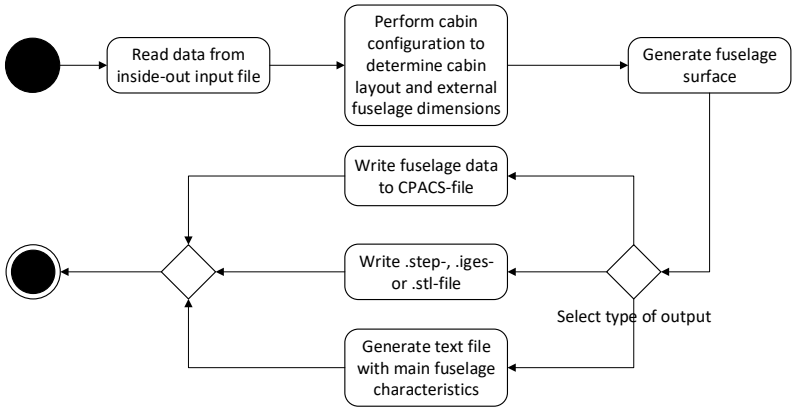


Figure 2.4: Activity diagram of the inside-out fuselage generation process

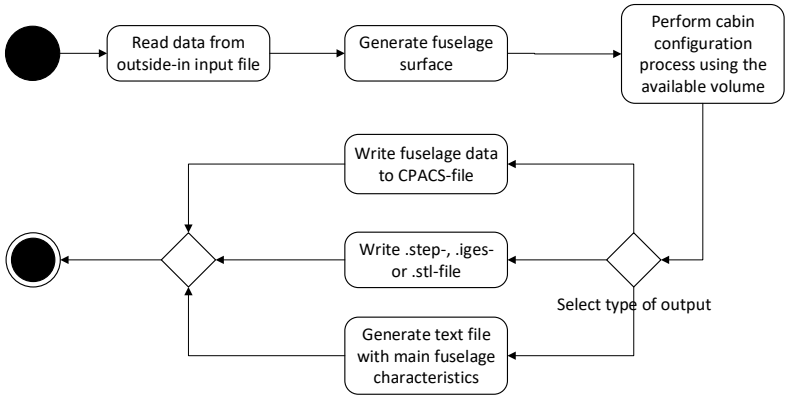


Figure 2.5: Activity diagram of the outside-in fuselage generation process

3

Parametric definition of the fuselage geometry

In the previous chapter the requirements of the fuselage outer geometry were established. These requirements have been translated to a suitable parameterization in ParaFuse. In this chapter the parameterization of the outer fuselage surface is discussed. ParaFuse uses several components to generate the outer surface of the fuselage. The main components that are used to generate the fuselage surface are shown in Figure 3.1.

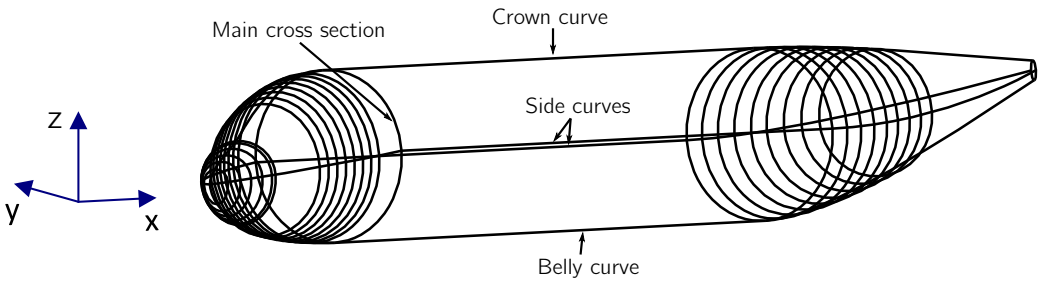


Figure 3.1: Cross sections and longitudinal guide curves

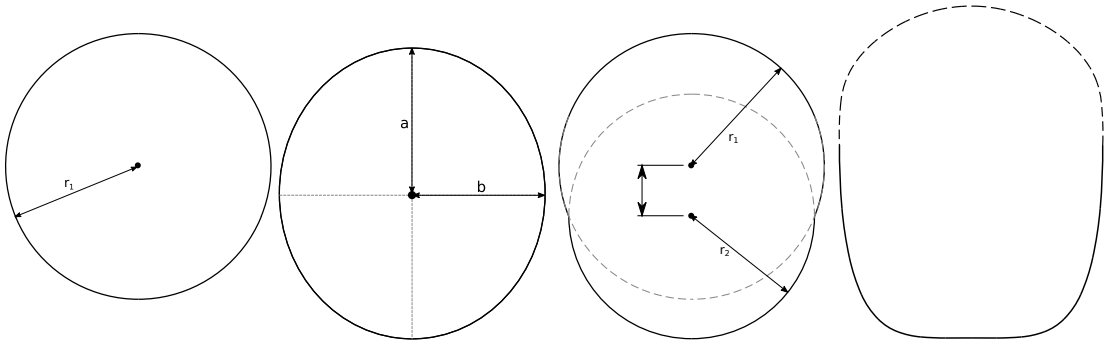
First of all, the shape of the main fuselage cross section needs to be generated. The main cross section is defined as the constant cross section of the middle section of the fuselage. The parameterization of the main cross section is discussed in Section 3.1. The main cross section is scaled at a number of locations along longitudinal guide curves. These longitudinal guide curves can be used to describe the shape of the outer fuselage. The parameterization of the longitudinal guide curves is explained in Section 3.2. After the scaling operation, the resulting cross sections are used to generate the outer fuselage surface. A thorough description of this process is given in Section 3.3. A separate parameterization approach is chosen for the generation of the nose end cap. The parametric definition of the nose end cap is given in Section 3.4. In the requirements of Section 2.2 the need for a parameterized wing-body fairing was introduced. The implementation of this functionality is discussed Section 3.5.

The values of the parameters that control the shape of the resulting fuselage model can be controlled in two different ways. When ParaFuse is used as an inside-out fuselage sizing tool, the values of the parameters are determined by performing the cabin configuration procedure. From the internal dimensions, appropriate parameter values are determined by ParaFuse so that a fuselage surface that encloses the cabin can be generated. When ParaFuse is used to perform outside-in cabin configuration studies, the parameter values need to be specified by the user by means on an XML input file, as the external dimensions of the fuselage are used to generate a cabin interior that fits within the available volume. Chapter 5 addresses how the inside-out fuselage sizing and outside-in cabin configuration methods determine the appropriate parameter values.

3.1. Main cross section

The constant middle section of the fuselage is defined as the main cross section. It is this main cross section that is used as a starting point for the generation of the fuselage outer surface. The cross section is scaled at several locations along the longitudinal guide curves. Four common cross-sectional fuselage shapes of conventional passenger aircraft were identified in the previous chapter. This section describes the parameterization approach taken in ParaFuse to model these cross sections. An overview of the four different cross sections is shown in Figure 3.2[1]. The user is able to control the cross-sectional shape of the fuselage using the parameter 'cross_section_shape'. The four options that are available are listed below:

1. circle
2. ellipse
3. double_bubble
4. free_form



(a) Circle (b) Ellipse (c) Double bubble (d) Free form

Figure 3.2: Parameterization of cross-sectional shapes in ParaFuse

The circular and elliptical cross-sectional shapes, shown in Figure 3.2a and 3.2b, are available as predefined classes in ParaPy. These shapes can be generated with a minimum amount of input parameters. When one of these cross-sectional shapes is selected by the user, ParaFuse instantiates[2] the ParaPy class `Circle` or `Ellipse`. The class `Circle` requires the definition of the radius, r_1, and the position of the center of the circle, whereas the instantiation of the class `Ellipse`, requires a value for the semi-major axis, a, semi-minor axis, b, and the position of the center of the ellipse.

Another cross-sectional fuselage shape is the double bubble and is shown in Figure 3.2c. A McDonnell Douglas MD-90, for example, has a double bubble cross section. The generation of the double bubble cross section in ParaFuse requires several intermediate steps. First of all, two instances of the ParaPy class `Circle` are generated. These two instances have a different center position and, possibly, different radii. In a subsequent step, the intersections between the two circle instances are found using a built-in function. These intersection points, together with the two circle instances, are used as an input for the class `TrimmedCurve`. This class removes the parts of the circles that fall within the outer cross section. The parts of the circles that are removed are shown as the dashed lines in Figure 3.2c. Subsequently, the remaining arcs are merged by creating an instance of the ParaPy class `ComposedCurve`. The complete procedure to generate the double bubble cross section is shown in Figure 3.3.

A more general fuselage cross section can be created by selecting the option 'free_form'. The free form cross section is defined using the Class function/Shape function transformation (CST) method proposed by Kulfan [34, 35]. This technique allows for the generation of a large variety of cross-sectional shapes using relatively few input parameters. The CST method uses a class function, $C(\eta)$, and a shape function, $S(\eta)$, to describe a

[1] These cross sections have been generated using ParaFuse. Subsequently, the cross sections have been exported as .STEP files and converted to vector graphics using CATIA V5.

[2] In object-oriented programming an instance is defined as 'a single object from a given class'. The process of creating an instance is called instantiation [33]. For more information on object-oriented programming the reader is referred to Appendix A.

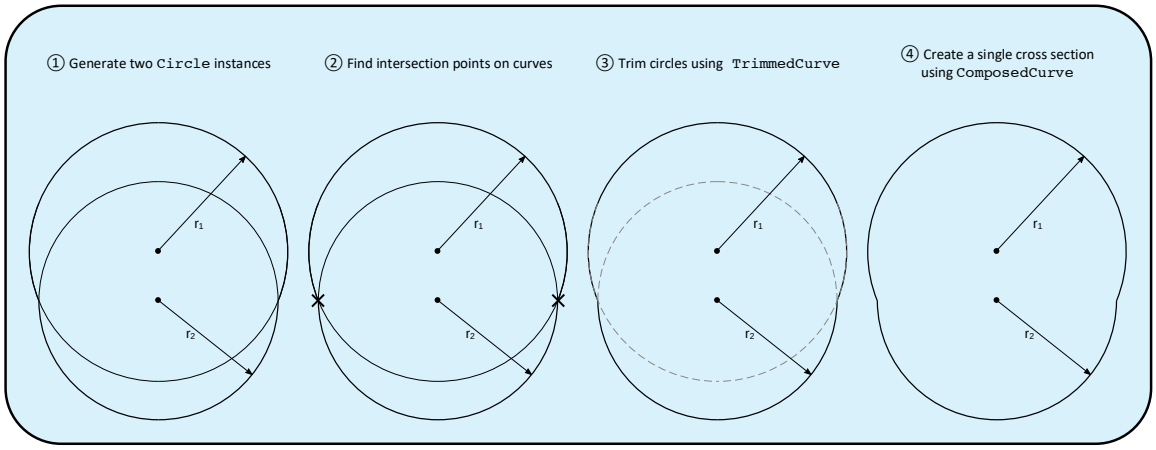


Figure 3.3: Construction of the double bubble cross section

curve as can be seen in Equation 3.1. In this equation, η, is the normalized x-coordinate and, ζ, the normalized y-coordinate.

$$\zeta(\eta) = S(\eta)\, C_{N2}^{N1}(\eta) \tag{3.1}$$

The definition of the class and shape function that compose this mathematical description of a curve are given in Equations 3.2 and 3.3, respectively. The class function coefficients, $N1$ and $N2$, can be used to modify the shape of the class function. The shape function can be modified by changing the scaling coefficients, A_i, of the Bernstein polynomial.

$$C(\eta) = \eta^{N1}(1-\eta)^{N2} \tag{3.2}$$

$$S(\eta) = \sum_{i=1}^{n} A_i S_i(\eta) \tag{3.3}$$

The shape function is a Bernstein polynomial of order n. This means that the polynomial is composed of $n+1$ terms. A single term of the Bernstein polynomial is given in Equation 3.4, where $i = 0-n$.

$$S_i(\eta) = \frac{n!}{i!(n-i)!}\eta^i(1-\eta)^{n-i} \tag{3.4}$$

An overview of the required steps to create the free form cross section is given in Figure 3.4. The CST method requires the cross-sectional shape to be split up into an upper and lower curve, indicated by the dashed and undashed lines in Figure 3.2d. Using Equation 3.1 the y-coordinates corresponding to 50 x-coordinates, distributed using a full-cosine distribution, are determined. Because of the fact that the CST method uses a normalized ordinate, i.e. $\eta \in [0,1]$, the coordinates need to be scaled to the correct width of the fuselage. An upper and lower curve is created by instantiating two `FittedCurve` instances. This class fits a B-spline curve through a list of points given as input. The final cross section is created by merging the two resulting curves using `ComposedCurve`. For more information regarding the CST method the reader is referred to Appendix B. The CST method can also be used to generate a circular or elliptical shape. However, because of the fact that the built-in ParaPy classes are able to generate the cross sections with more intuitive parameters namely, radius and semi-major and semi-minor axes, the decision has been made to use these for the circular and elliptical cross section.

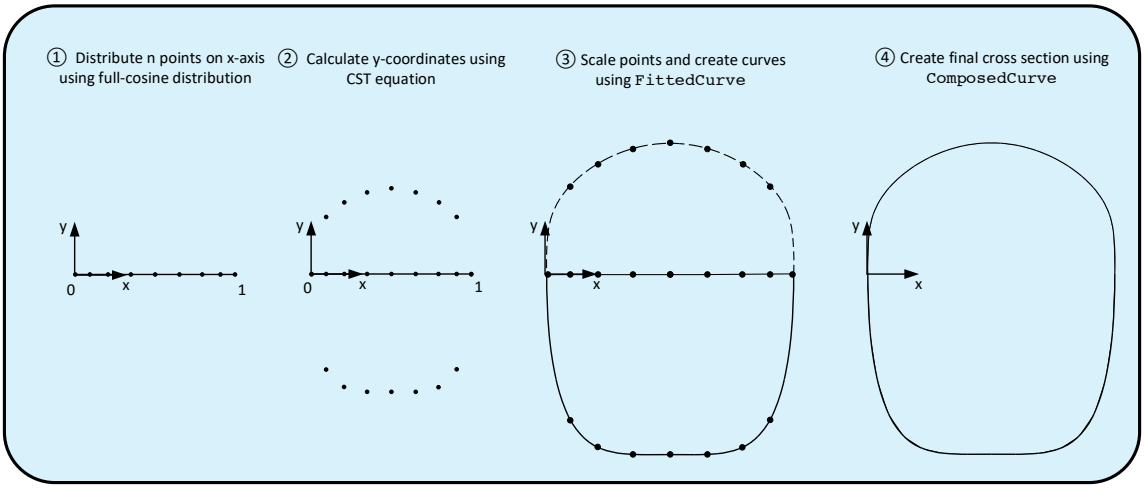


Figure 3.4: Required steps to generate the free form fuselage cross section

3.2. Longitudinal guide curves

As described in the introduction of this chapter, longitudinal guide curves are used to describe the outer shape of the fuselage. The guide curves are not used in the surface generation process itself, but are merely used to scale the main fuselage cross section, described in Section 3.1, at several locations along the length of the fuselage. It is possible to generate a surface in ParaPy using profiles and guide curves, however, during the development of this application it was found that the current implementation of this class in ParaPy is not able to fit a surface that follows the guide curves accurately enough. Therefore, the guide curves are used to scale the cross sections along the length of the fuselage and subsequently the scaled cross sections are used to wrap a surface. This section describes the generation of the longitudinal guide curves, whereas, Section 3.3 describes the generation of the fuselage surface using the scaled cross sections.

A two-step approach is used to generate the longitudinal guide curves of the fuselage. First of all, separate guide curves are generated for the nose cone, middle section and tail cone of the fuselage. For each of these sections a top, bottom, port and starboard guide curve is created. In a subsequent step, the separate guide curves of each fuselage section are merged with the guide curves of the other sections to form single top, bottom and side guide curves. The ParaPy class `ComposedCurve` is used to perform this action. In essence, the instantiation of the `ComposedCurve` class generates a single B-spline curve of the three separate curves. The guide curve generation process is illustrated in Figure 3.5.

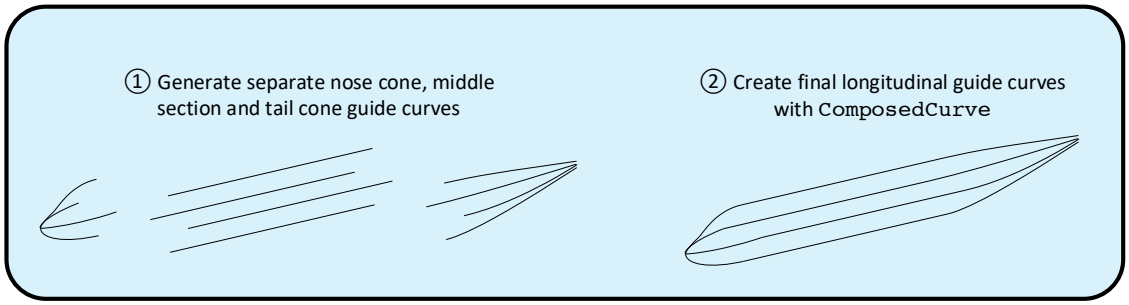


Figure 3.5: Two-step approach to longitudinal guide curve construction

The following subsections describe the parameterization of the guide curves of the nose cone, middle section and tail cone of the fuselage. The parameterization will be discussed by means of schematics. Additionally, an overview of the parameters that can be modified by the user in order to change the resulting fuselage

geometry is given.

3.2.1. Nose cone

Three distinct guide curves are generated for the nose cone section, namely a top and bottom guide curve and, because the fuselage is assumed to be symmetric, a side guide curve that is mirrored in the XZ-plane. Figure 3.6 shows the side view of the nose cone guide curves. A top view of the nose nose cone guide curves is presented in Figure 3.7. The top guide curve is constructed by generating an instance of the ParaPy class `ComposedCurve`. The three curves that form the top guide curve of the nose cone are listed below:

1. A B-spline curve, starting at the nose start point and ending at the bottom of the windshield. An additional control point, indicated as the top rail control point, can be translated along the z-axis to modify the shape of the nose. In Figure 3.6 this curve is indicated as B-spline 1.
2. A straight line segment. This line segment represents the windshield of the cockpit and is indicated as the windshield line in 3.6. This line segment is constructed using the overnose angle, windshield inclination angle and upward viewing angle.
3. A B-spline curve, starting at the top of the windshield and ending at the top rail end point. As an additional input tangency constraints are used to make sure that this curve is tangent to the top guide curve of the middle fuselage section and the windshield line. The tangency conditions are satisfied by providing the class ParaPy uses to generate the B-spline curve with the unit tangent vectors at the end of the windshield line and the start of the middle section top curve. This B-spline curve is indicated as B-spline 2 in Figure 3.6.

The position of the head of the pilot is used as the starting point for the generation of the top guide curve. This position is determined from inputs specified by the user. First of all, the position of the seat of the pilot is required. The second required input is the position of the head of the pilot, specified by $h_{pilot,eye}$ in Figure 3.6. Finally, the distance between the head of the pilot and the windshield needs to be specified. These values, together with the windshield inclination, overnose and upward viewing angles, are used to determine the position of the windshield. This windshield is used to determine the end of B-spline 1 and start of B-spline 2, respectively. After these positions are found the curves can be generated and the final top guide curve of the nose cone can be instantiated.

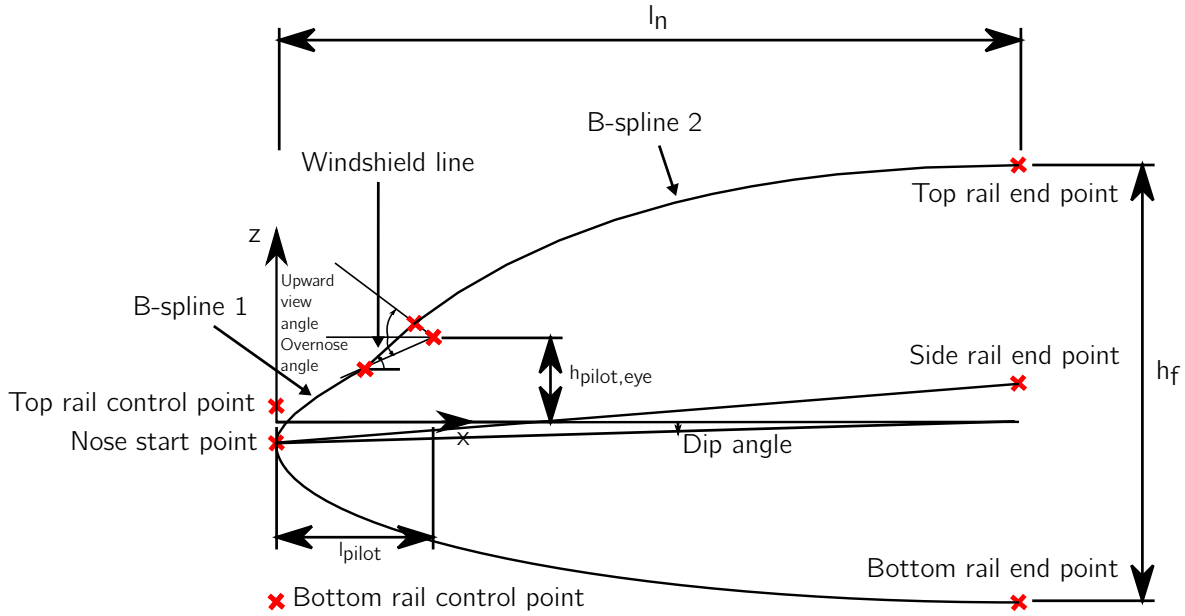


Figure 3.6: Side view of the nose cone guide curves

The bottom and side guide curves are single curve instances. The difference between the two guide curves is the fact that the side guide curve can be rotated with a certain 'dip angle' around the Y-axis to allow for a

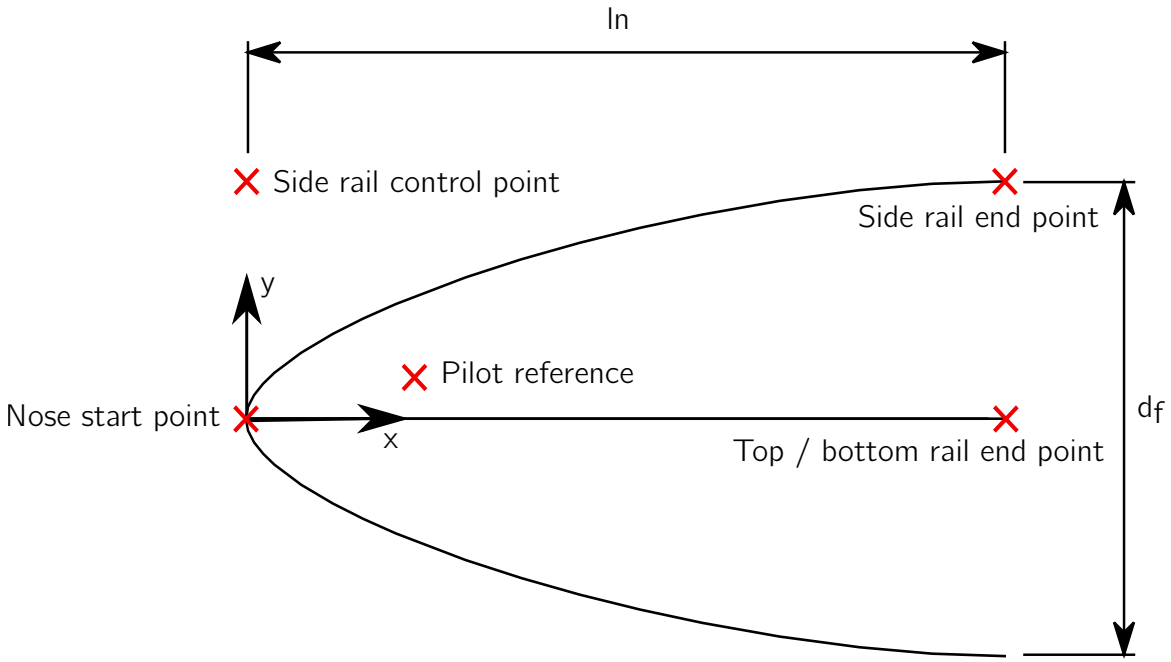


Figure 3.7: Top view of the nose cone guide curves

'drooped' fuselage nose. To generate the side and bottom guide curve the user is able to select one of three options. The generation method for these two guide curves can be selected independent of each other, i.e. method 'a' can be used for the generation of the side curve and method 'b' can be used for generation of the bottom curve. The user can specify these options using the parameters 'side_rail_generation_method' and 'bottom_rail_generation_method' in the input file of ParaFuse. The three available options are listed below:

1. 'general_bezier': This option generates a quadratic Bézier curve with three control points; nose start point, bottom rail control point and bottom rail end point in case of the bottom guide curve, as can be seen in Figure 3.6. For the side guide curve this option uses the nose start point, side rail end point and side rail control point, as can be seen in Figure 3.6. Additionally, the weights associated with each of these control points can be specified to modify the shape of the guide curve. An increase in relative weight of a control point forces the curve to move towards the corresponding control point. For example, if the list of weights given as an input equals [5, 1, 1], the curve will move more towards the nose start point.

2. 'ellipse': This option generates an elliptical quadrant. The option 'ellipse' actually uses the same parameterization as the first option, but instead of being able to select the weights of the control points manually, ParaFuse now uses fixed weights with a value of: [1, 1, 2] [36]. Although, an elliptical quadrant can be generated using option 1, if this option is selected the user is no longer required to specify the weights of the Bézier curve separately.

3. 'haack': This option generates a Haack series curve. The Haack series curve is derived from the Sears-Haack body. This curve can be constructed using Equations 3.5 and 3.6. The parameter, r, corresponds to the radius of the fuselage or the lower height of the fuselage cross section, depending on which guide curve is generated. The coefficient, C_h, can be adjusted between 0 and 0.67 to modify the shape of the resulting curve [37].

$$\theta = \cos^{-1}\left(1 - \frac{2x}{l_n}\right) \tag{3.5}$$

$$z = \frac{r}{\sqrt{(\pi)}}\sqrt{\theta - \frac{\sin(2\theta)}{2} + C_h \sin^3\theta} \tag{3.6}$$

The parameters that can be used to modify the shape of the nose cone guide curves are given in Table 3.1. Some examples of nose cone guide curves of several reference aircraft, including parameter settings, are presented in Appendix C.

Table 3.1: Inputs for the nose cone guide curves

Parameter	Type	Unit	Description
FR_n	float	[-]	Fineness ratio of the nose
x_loc_front_bulkhead	float	[m]	Location of the front pressure bulkhead
pilot_distance_to_bulkhead	float	[m]	Distance of the pilot to the front pressure bulkhead
pilot_eye_height	float	[m]	Eye height of the pilot
windshield_distance	float	[m]	Distance between the head of the pilot and the windshield
windshield_angle	float	[deg]	Windshield angle. Counterclockwise positive.
upward_view_angle	float	[deg]	Upward view angle. Clockwise positive
overnose_view_angle	float	[deg]	Overnose angle. Counterclockwise positive
nose_sharpness_factor	float	[-]	Controls the z-location of the top rail control point as a fraction of fuselage height. Value should be between 0 and 1
side_rail_generation_method	string	[-]	Either 'general_bezier', 'ellipse' or 'haack'
bottom_rail_generation_method	string	[-]	Either 'general_bezier', 'ellipse' or 'haack'
dip_angle	float	[deg]	Dip angle of the fuselage nose. Clockwise positive.
C_side	float	[-]	Haack coefficient of side rail (only required if generation method is 'haack')
C_bottom	float	[-]	Haack coefficient of bottom rail (only required if generation method is 'haack')
bezier_weights_side	list	[-]	List of floats containing the Bezier weights (only required if generation method is 'general_bezier')
bezier_weights_bottom	list	[-]	List of floats containing the Bezier weights (only required if generation method is 'general_bezier')

3.2.2. Middle section

The majority of passengers in conventional aircraft are located in the middle fuselage section. This section has a constant cross section and, therefore, the guide curves of the main cabin section are assumed to be straight lines. Similar to the nose cone section, three distinct guide curves are specified for the middle fuselage section, namely a top, bottom and side curve. Because of the assumption that the fuselage is symmetric in the XZ-plane the side curve can be mirrored in this plane to complete the set of guide curves. The side curves run along the maximum width of the fuselage cross section.

The assumption that the middle section of the fuselage is constant implies that for the resulting fuselage surface a constant section will always be present. It is believed that for conventional passenger aircraft this is a valid assumption. However, if future extensions of ParaFuse would include smaller aircraft, for example business jets such as the Piaggio P.180 Avanti, this assumption might no longer provide satisfactory fuselage shapes. A solution would be to replace the straight line segments with either B-spline or Bézier curves. This will allow for the placement of an additional control point, which could be used to generate slightly curved guide curves for the middle section of the fuselage.

The end points of the nose cone guide curves are used as starting points for the guide curves of the main cabin section. Apart from the starting points, the only required parameter is the length of the middle fuselage section. The end points, on their turn, are the starting point for the tail cone guide curves described in Section 3.2.3. Tangency between the guide curves of the middle section and guide curves of the nose and tail cone is ensured by supplying tangents of the middle section as an additional input for the guide curves of the nose and tail cone sections.

3.2.3. Tail cone

The final set of guide curves required to generate the complete fuselage guide curves are the guide curves describing the tail cone section of the fuselage. Similar to the nose cone and middle fuselage section, the tail cone guide curves consist of top, bottom and two side curves.

Figure 3.8 shows a side view of the tail cone guide curves including control points. The tail cone guide curves are all instances of the ParaPy class `BSplineCurve`. The start points of the top, bottom and side curves are the end points of the guide curves of the main cabin section. The guide curves of the tail cone end in a single point, which is extended a distance, l_{ext}, aft of the tail cone. This extension can be controlled by the user as a percentage of the tail cone length. The x-coordinate of the tail end point is determined by adding the tail cone length, l_t, and the length of the extension, l_{ext}. The z-coordinate of the tail end point is determined using the upsweep angle of the tail cone, which is also an user input. To further modify the shape of the bottom guide curve, the user is able to shift the position of the control point of the bottom guide curve along the x-axis as a percentage of the tail cone length.

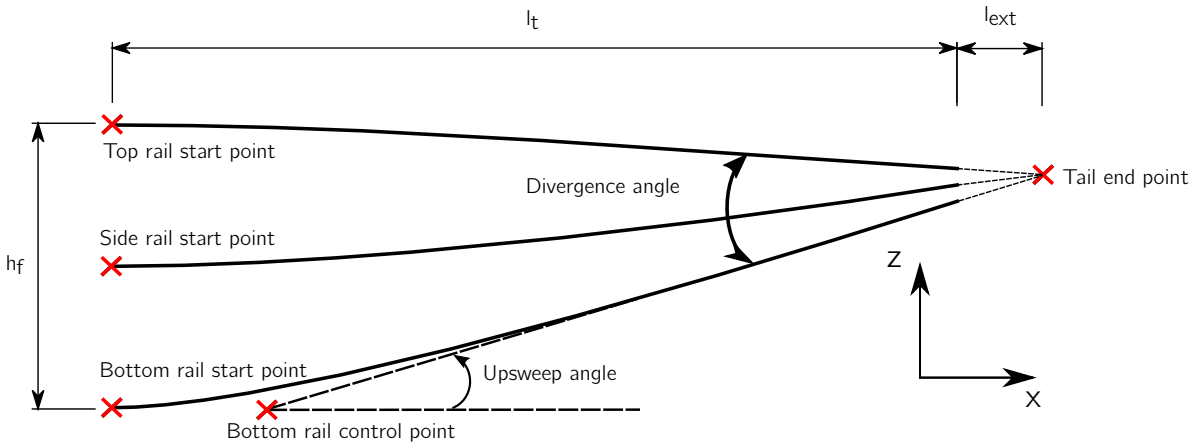
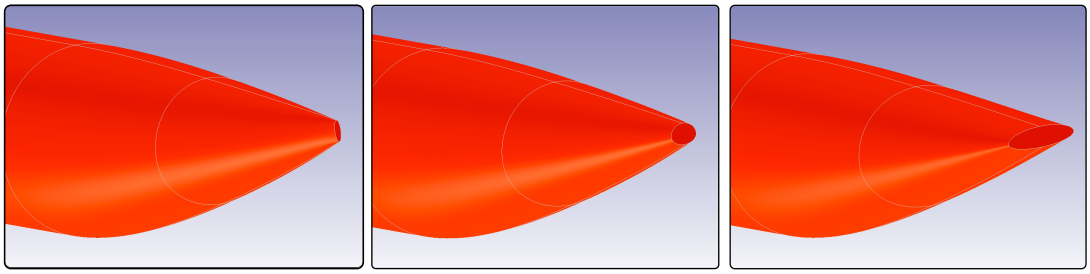


Figure 3.8: Side view of the tail cone guide curves

Three distinct tail cone shapes have been identified in Section 2.2.1. To generate these different tail cone shapes the end points of the tail cone section are changed depending on the required tail cone shape. By default, an elliptical tail cone end cap is generated by ParaFuse. However, by changing the option 'elliptical_end_cap' to False in the settings of ParaFuse the user is able to generate a rounded end cap shape of the tail cone. A 'flat' tail cone shape, as can be seen on transport aircraft with an entrance in the tail cone, can be created by increasing the 'aft_body_bluntness_factor'. This parameter translates the end point of the side guide curves a percentage of the fuselage width along the y-axis. An example of the different tail cone shapes that can be generated using ParaFuse is shown in Figure 3.9[3].

Table 3.2 gives an overview of the parameters that can be used to modify the shape of the tail cone.

[3]For clarity the figures show the resulting tail cone surface instead of only guide curves.



(a) Ellipse (b) Rounded (c) Flat

Figure 3.9: Tail cone geometries generated with ParaFuse

Table 3.2: Inputs for the tail cone guide curves

Parameter	Type	Unit	Description
elliptical_end_cap	boolean	[-]	Boolean to control generation of an elliptical end cap for the tail cone
FR_t	float	[-]	Fineness ratio of the tail cone
bottom_control_point_factor	float	[-]	Longitudinal location of the bottom control point as fraction of the tail cone length. Value between 0 and 1.
upsweep_angle	float	[deg]	Upsweep angle of the tail cone. Counterclockwise positive.
tail_cone_cut_off_factor	float	[-]	Factor to control how much the tail cone end point must be extended aft of the tail cone as a percentage of tail cone length. Value between 0 and 1.
aft_body_bluntness_factor	float	[-]	Factor to control bluntness of the tail cone as a fraction of the maximum fuselage width. Value between 0 and 1.

3.3. Surface generation

The previous sections have introduced the parameterization of the main fuselage cross section and the longitudinal guide curves. Subsequently, the main cross section is scaled at several locations along the longitudinal guide curves. The outer fuselage surface is created using these scaled cross sections only. In this section the procedure to generate the fuselage surface is described.

By default, a cross section is located at the start and end point of the nose, cabin and tail cone sections. The class `ScaleCrossSection` has been created to perform the scaling operation of the main cross section. To perform this operation the class requires the main cross section and target width and height as input. These values are derived using the longitudinal guide curves. The side guide curves follow the maximum width of the fuselage. If required, the scaling operation is able to non-uniformly scale the upper and lower part of the fuselage cross section. This scaling procedure is illustrated in Figure 3.10. The points in the figure are those locations where the guide curves intersect with the plane.

Subsequently, the scaled cross sections are used to generate a solid of each fuselage section, i.e. a separate solid for the nose cone, middle section and tail cone. The faces of these solids are sewn together to form the final fuselage surface. The decision to create three separate solids originates from the fact that 'wiggles' occur when lofting over a large length, through the fuselage profiles. A comparison between the two surface generation methods is shown in Figure 3.11. These surface 'wiggles' are shown in Figure 3.11a and the smooth surface when creating separate surfaces for each fuselage section is shown in Figure 3.11b. The algorithm that generates fuselage surface in ParaPy tries to fit the surface through all profiles. To approximate each profile as closely as possible, the algorithm increases the order of the resulting surface. Just as a polynomial of increasing order, this results in 'wiggles'. When the surface is created for each fuselage section separately, the fitting algorithm is able to generate a surface of lower order, that satisfies the required fuselage shape.

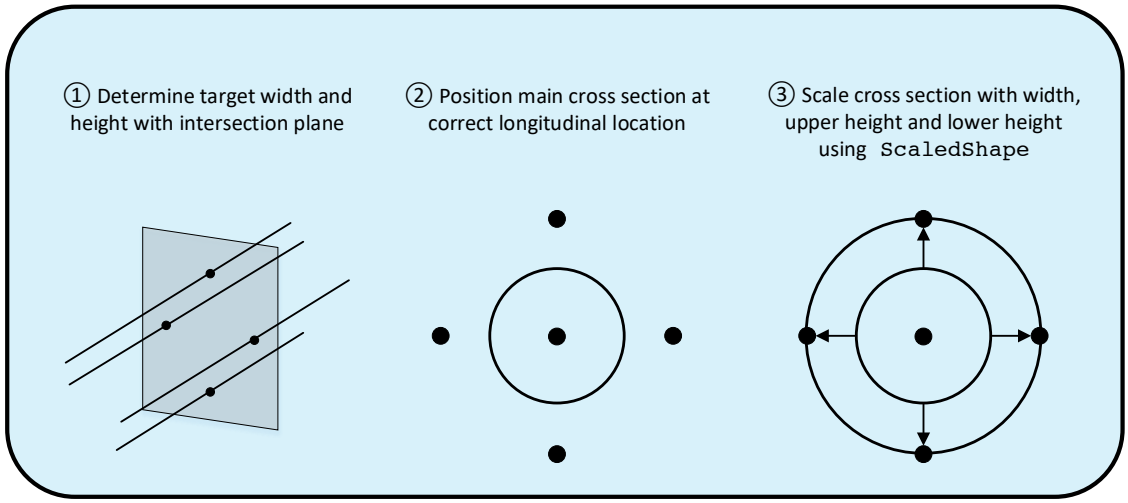
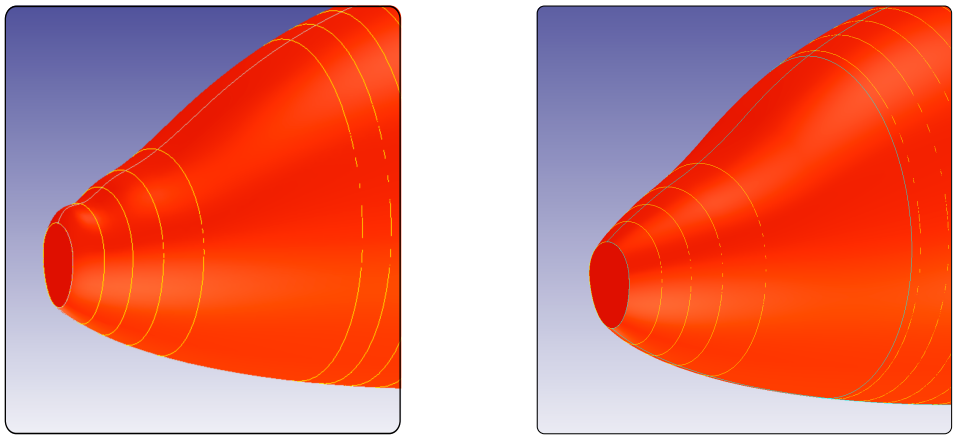


Figure 3.10: Scaling operation to scale initial cross section to guide curves

ParaPy allows the user to limit the order of the resulting surface to prevent these surface 'wiggles'. However, the result also is that the resulting surface does not follow the guide curves anymore.

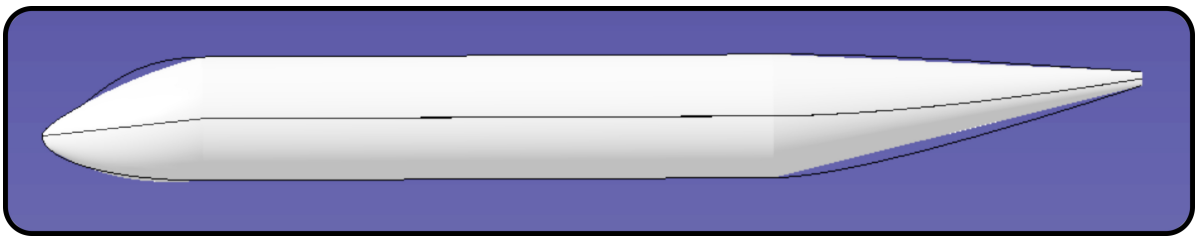


(a) One surface (b) Separate surfaces

Figure 3.11: Comparison between the two surface generation methods

Although the surface of the fuselage can be created using only the three initial cross sections, the resulting surface does not provide accurate results. Clearly, the surface does not follow the guide curves, as can be seen in Figure 3.12. This is an obvious consequence, as the surface fitting algorithm does not have sufficient information to accurately capture the shape the guide curves. This shortcoming can be alleviated by defining additional cross sections. To reduce the workload for the user, these additional cross sections do not need to be specified manually, but two classes have been created to automate this process. The first class, `CurvaturePointGenerator`, has been implemented to interpolate additional fuselage cross sections in areas where the curvature of the guide curves is high. The activity diagram in Figure 3.13 gives a schematic overview of the implemented interpolation routine. The required inputs for this class are: a curve, a list of initial section locations, the number of samples to be evaluated per segment, an interpolation threshold and the interpolation method. Two different interpolation methods are available. `CurvaturePointGenerator` can evaluate either the curvature on each sample point, or the change in curvature between samples. If the (change in) curvature at a sample location exceeds the threshold, an additional cross section is positioned at that location. This interpolation threshold is defined as a percentage of the average curvature of the segment. Furthermore, the class checks for discontinuities along the curve by using the ParaPy class

`DecomposedCurve`, which evaluates the continuity of the curve. If a different order of continuity is found the curve is split into separate arcs. Subsequently, the `CurvaturePointGenerator` class compares the tangents of the decomposed curves. If the tangents of the two arcs are dissimilar, an additional cross section is positioned at the location of the discontinuity.



(a) Side view

(b) Top view

Figure 3.12: Fuselage surface without interpolation

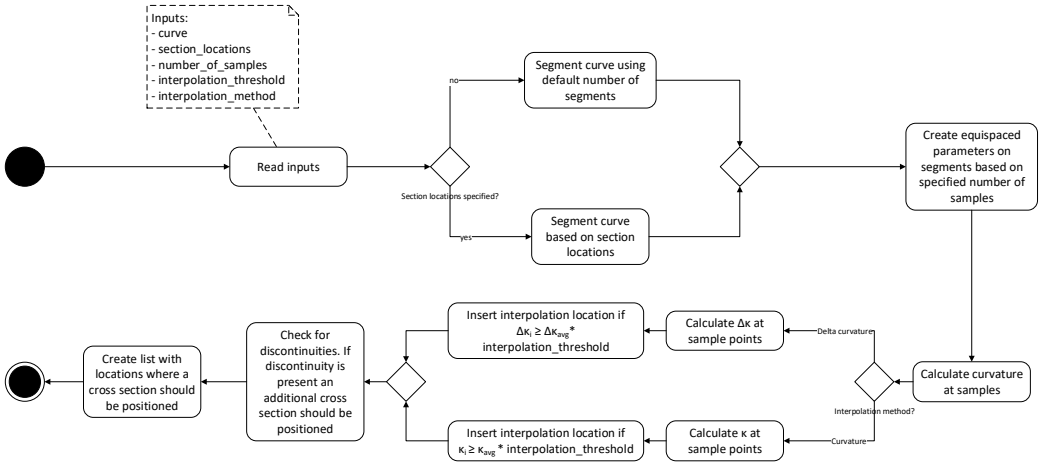


Figure 3.13: Activity diagram for the `CurvaturePointGenerator` class

The process described in Figure 3.13 is applied to all guide curves. Although the guide curves that are evaluated are all different, it might be the case that the interpolation locations are almost equal. The class `MergeInterpolatedParameters` has been created to merge the separate lists with interpolation locations and remove redundant positions. This procedure first projects one of the longitudinal side curves onto the XZ-plane. Subsequently, all interpolated locations are projected on this curve. The parameter values associated to these projected points are calculated. As a final step, a function evaluates the value of subsequent parameters. If these parameter values are within a tolerance of 1e-3[4] they are merged into one interpolation location. To illustrate the process of the `MergeInterpolatedParameters` class an example is given in Figure 3.14. Figure 3.15 presents a schematic of the fuselage guide curves and cross sections. The initial cross sections are indicated in black, whereas, the blue cross sections have been created using the interpolation

[4]The curve on which these point are projected is normalized. Hence, the parameter values range from 0 to 1

routine. The fuselage shown in Figure 3.12 is shown in Figure 3.16 as well. However, in the latter case the classes `CurvaturePointGenerator` and `MergeInterpolatedParameters` have been used to create additional cross sections. As can be seen from the figure, the resulting fuselage surface now follows the guide curves accurately.

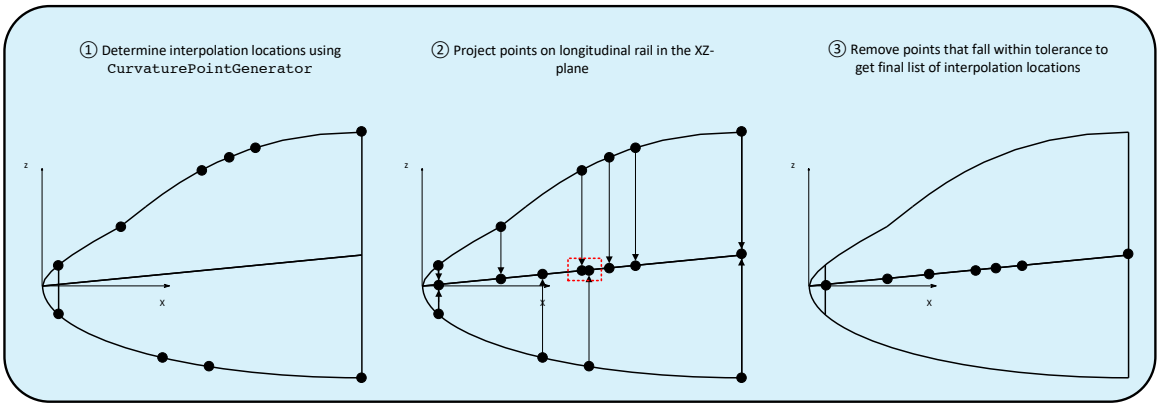


Figure 3.14: Removal of redundant parameters using the class `MergeInterpolatedParameters`

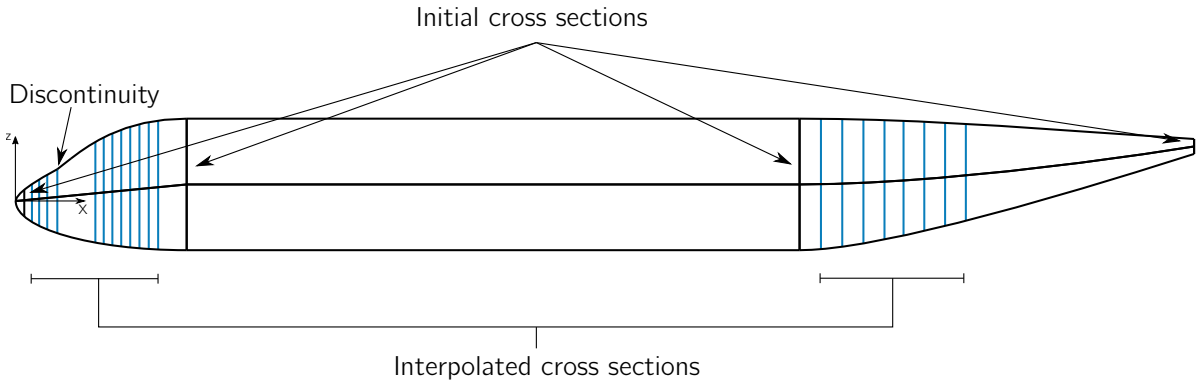
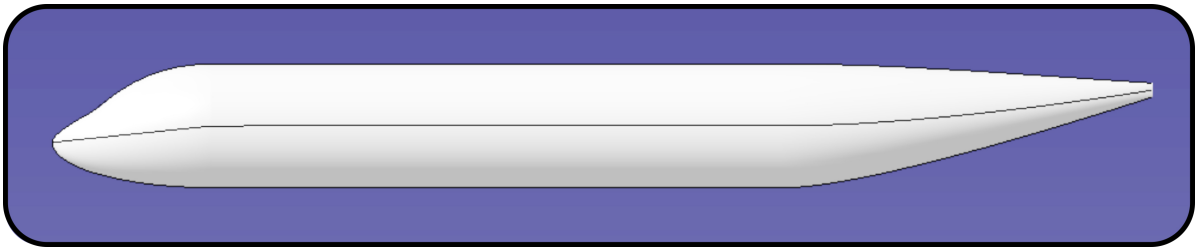
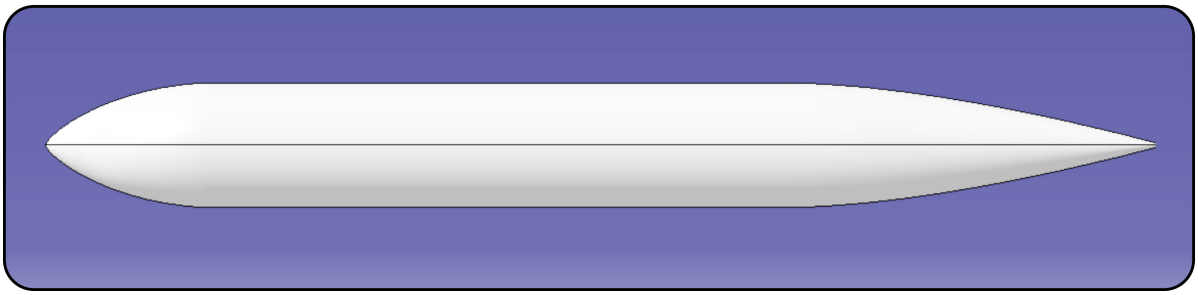


Figure 3.15: Fuselage guide curves and cross sections after applying interpolation



(a) Side view



(b) Top view

Figure 3.16: Fuselage surface with interpolation

3.4. Nose end cap

In the previous section the surface generation process of the fuselage has been described. The process of lofting through the cross sections results in flat front and aft fuselage faces[5]. If the geometry is to be used in for example aerodynamic analysis using CFD, it is important that the surface is fully defined, because aerodynamic analysis on a fuselage surface with a flat front face would not provide satisfactory results. Therefore, an additional nose end cap surface is created. This section describes the generation of the nose end cap surface.

A B-spline surface is created to represent the surface of the nose end cap. The procedure to generate the surface is shown in Figure 3.17. The control points of the nose end cap and the resulting surface are shown in Figure 3.18. The shape of the end cap can be controlled using three parameters. First of all, the amount of control points needs to be specified. These control points are distributed uniformly along the base on the nose end cap. Subsequently, these control points are translated along their corresponding isocurve[6] of the fuselage surface, as a fraction of the end cap height. To accurately capture the cross-sectional base shape this parameter is required to be large (e.g. $n = 100$). Finally, the user can control the translation of the control points in the YZ-plane as a fraction of the end cap base diameter. Using these three inputs a smooth nose end cap can be created that is tangent to the original fuselage surface.

The generated surface is tangent to fuselage surface because the control points are translated along the tangent vectors of the isocurves of the surface. Moreover, the control points in ZY-plane ensure that the extreme of the B-spline surface coincides with the starting point of the nose cone. The complete list of required input parameters for the generation of the nose end cap is shown in Table 3.3. The generated surface is sewn to the fuselage surface that was already created by lofting through the fuselage cross sections.

[5]The nose end cap is included in Figures 3.12 and 3.16
[6]Isocurves are curves that have a constant u- or v-value on a surface

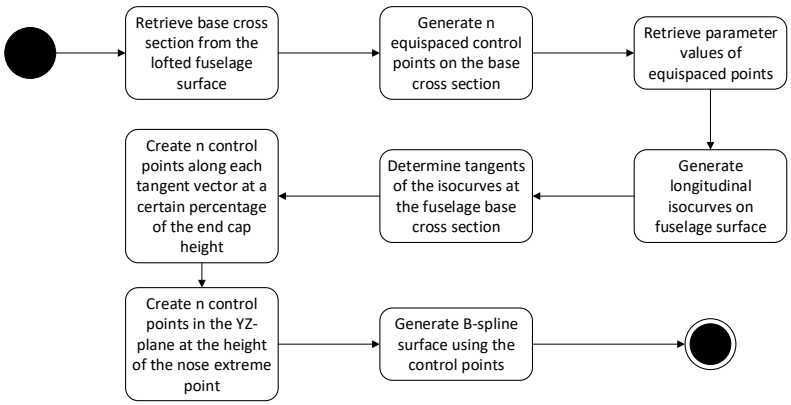


Figure 3.17: B-spline surface with control points

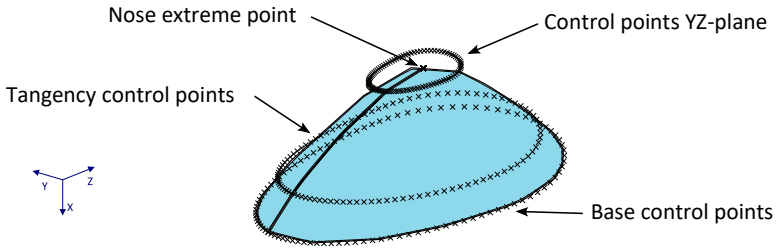


Figure 3.18: Parametric definition of the nose end cap

Table 3.3: Inputs for the generation of the nose end cap

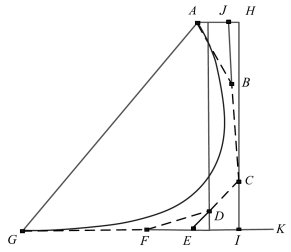
Parameter	Type	Unit	Description
n	integer	[-]	Number of control points. Must be high in order to accurately capture fuselage surface (e.g. 100)
delta_base	float	[-]	Factor to control the position of the base control points as a percentage of the height of the end cap. Value must be between 0 and 1.
delta_tip	float	[-]	Factor to control the position of the tip control points as a percentage of the base diameter of the end cap. Value must be between 0 and 1.

3.5. Wing-body fairing

Interference effects near the wing-fuselage junction can have a detrimental effect on the aerodynamic performance of an aircraft [38]. According to Della Vechia and Nicolosi the most effective method to reduce interference drag is by using a fairing between the wing and fuselage [26]. To be able to model the effect of applying a wing-body fairing early in the aircraft design process a parameterized wing-body faring has been implemented in ParaFuse.

The parameterization of the wing-body fairing follows the approach proposed by Song and Lv [39]. The fairing is defined using four sections, with a distribution that can be specified as an input, and two longitudinal guide curves. The section curves are instances of the ParaPy class `BSplineCurve` with seven control points. The definition of one section curve is given in Figure 3.19, together with a definition of the parameters that can be used to modify the shape of these section curves. These parameters could be used as design variables in an optimization problem to minimize the aerodynamic drag of the aircraft, as is illustrated in [39]. The bottom guide curve of the fairing runs in the plane of XZ-plane along the bottom of the fuselage, whereas the

top guide curve is defined in the XY-plane.



(a) B-spline curve with control points

Parameter	Definition	Physical meanings or effect
Par1	GI/GK	Magnitude of fairing extrusion
Par2	HC/HI	Longitudinal position and curvature
Par3	HJ/HA	Curvature control of upper-section profile
Par4	JB/JC	Curvature control of upper-section profile
Par5	GF/GI	Curvature control of lower-section profile
Par6	FE/FI	Curvature control of lower-section profile
Par7	CD/CE	Curvature control of lower-section profile

(b) Parameter definition

Figure 3.19: Parameterization of a section curve [39]

The position of the wing-body fairing is determined using the position of the main wing. The fairing is extended a percentange of the wing chord at the wing-fuselage intersection in front of the leading edge of the wing. The parameterization approach proposed by Song and Lv results in a surface that is located partly inside the fuselage. Therefore, the external length of the fairing surface is not known beforehand. An additional scaling operation is required to generate the fairing surface with the desired fairing length. The process to generate the wing-body fairing is illustrated in Figure 3.20. The complete list of parameters required to generate the fairing surface is given in Table 3.4.

An example of the wing-body fairing generated using ParaFuse on a short range 150 passenger aircraft is shown in Figure 3.21[7]. The current implementation of the wing-body fairing can be used to generate fairing surfaces for low-wing aircraft. However, with only a slight modification of the code, this parameterization approach could also be used to generate the fairing surface of a high-wing aircraft. This modification would include a switch that would flip the fairing section curves vertically.

In this chapter the parametric definition of the fuselage geometry of ParaFuse has been described. Chapter 5 describes how a inside-out fuselage sizing and outside-in cabin configuration method has been implemented. In order to automate the fuselage sizing and cabin configuration process several design rules need to be included in ParaFuse. These fuselage design rules are described in Chapter 4,

[7]The fuselage, wing-body fairing and wing objects have been created by ParaFuse and, subsequently exported to CATIA V5 for rendering.

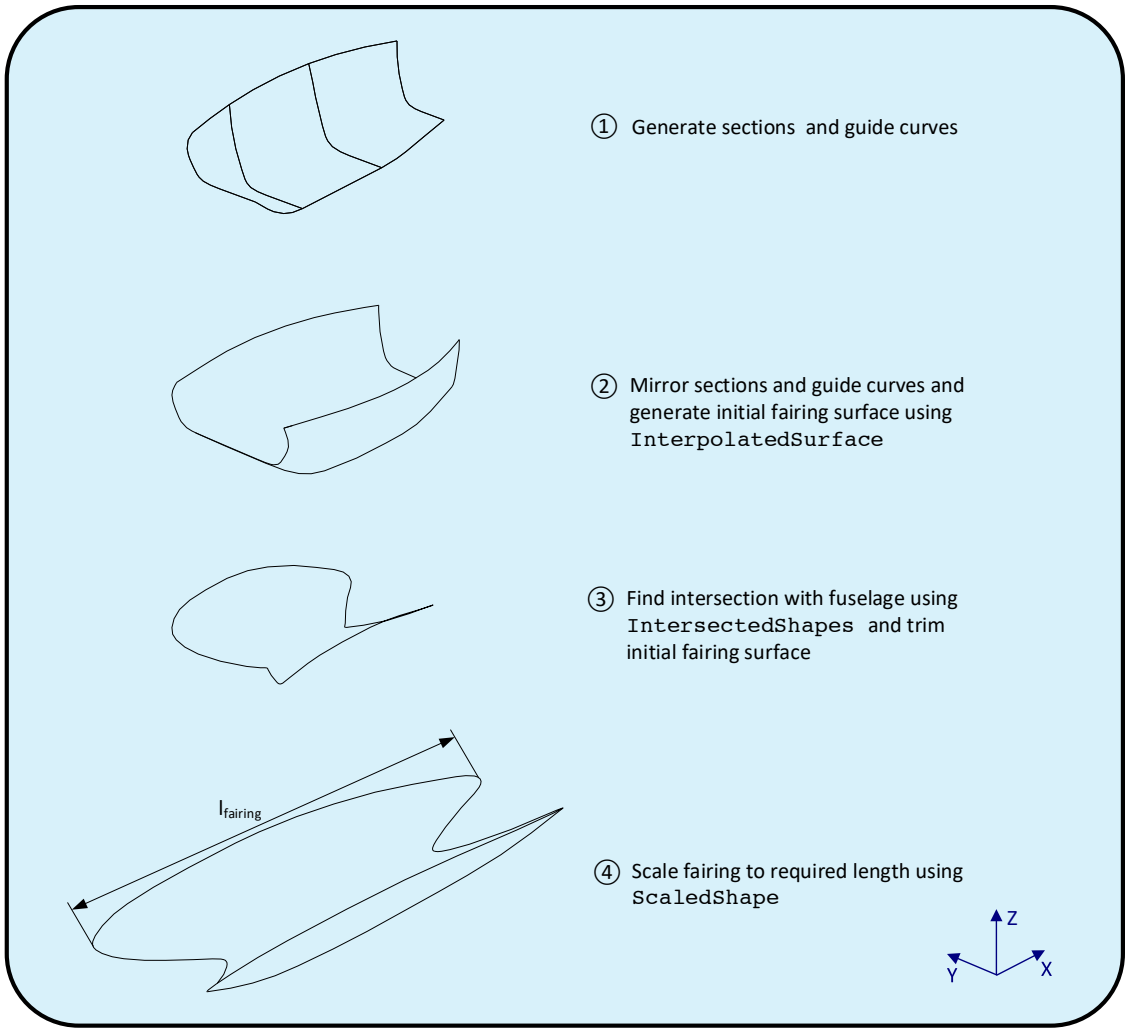


Figure 3.20: Construction of the wing-body fairing surface

Table 3.4: Required inputs for the generation of the wing-body fairing

Parameter	Type	Unit	Description
x_le	float	[m]	Wing leading edge position
fairing_length	float	[m]	Length of the wing-body fairing. Either given as an input or calculated using statistics
fairing_front_extension_factor	float	[-]	Extension of the fairing in front of the leading edge of the wing.
par1	list[float]	[-]	Magnitude of fairing extrusion. One entry for each of the section curves
par2	list[float]	[-]	Longitudinal reference of curvature. One entry for each of the section curves
par3	list[float]	[-]	Curvature control of upper-section profile. One entry for each of the section curves
par4	list[float]	[-]	Curvature control of upper-section profile. One entry for each of the section curves
par5	list[float]	[-]	Curvature control of lower-section profile. One entry for each of the section curves
par6	list[float]	[-]	Curvature control of lower-section profile. One entry for each of the section curves
par7	list[float]	[-]	Curvature control of lower-section profile. One entry for each of the section curves
top_guide_curve_control_point_factors	list[float]	[-]	Ratio of top guide curve width with respect to fuselage width
fairing_height_control	list[float]	[-]	Height control for for first and last fairing cross section. Percentage of lower fuselage height
section_parameters	list[float]	[-]	Location of the section curves. Percentage of fairing length

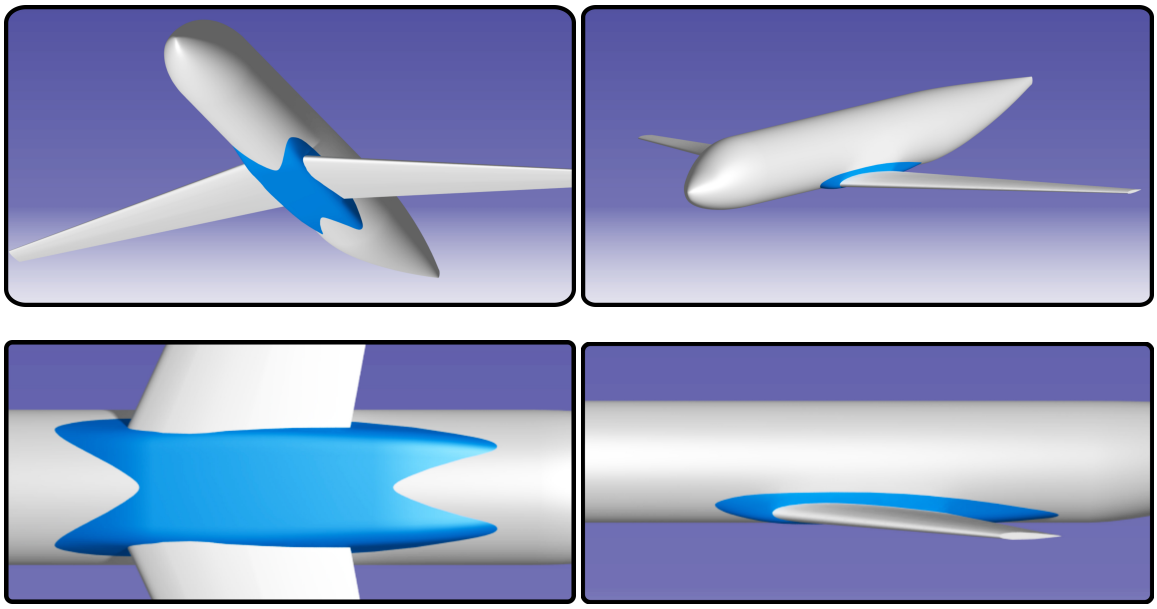


Figure 3.21: Wing-body fairing on a short range 150 passenger aircraft

4

Fuselage design rules

Knowledge-based engineering applications capture design knowledge to automate repetitive and non-creative design tasks. Therefore, knowledge regarding the conceptual fuselage design process must be collected first. This knowledge can, subsequently, be captured inside the KBE application using design rules. These design rules, together with a limited amount of user inputs, can be used to determine values for the parameters specified in the previous chapter. Design rules of aircraft at a conceptual level have been collected in a large number of well-known aircraft design handbooks [2, 24, 40–43]. Although, the design rules described in these books have been derived using reference data of a large number of aircraft, the user must be able to modify these values to allow for high flexibility of the application. Therefore, none of these values have been hard-coded in the application itself, but specified in separate input-files.

The fuselage design knowledge is a combination of statistical relationships, certification requirements and fixed component dimensions. The certification requirements are prescribed by aviation safety agencies to ensure safe air travel. In Europe the aviation authority is the European Aviation Safety Agency (EASA). The certification requirements that an aircraft has to comply with depend of the type of aircraft. Certification requirements for large passenger aircraft are specified by EASA in the CS 25 certification requirements. This documents describes requirements of the entire aircraft system, however, not every requirement applies to the design of the fuselage. Therefore, Section 4.1 gives an overview of a selection of requirements taken from EASA CS 25 certification specifications that affect the design of aircraft fuselages. After that, statistical data that is used by the KBE application is presented in Section 4.2. Finally, the implementation of this design knowledge as design rules in the KBE application is discussed in 4.3.

4.1. Certification requirements

In Chapter 2.2 an overview of the certification requirements that apply to the design of aircraft fuselages was given. Because the scope of the KBE application is the design of passenger aircraft fuselages, the design must be compliant with EASA CS 25 certification requirements. The certification requirements that apply to the fuselage design process deal with the amount and type of emergency exits, minimum aisle width and the maximum allowable number of seats abreast between aisles. In this section these requirements are discussed in more detail.

4.1.1. Emergency exits

To allow for safe passenger evacuation in case of an emergency, the amount and type of emergency exits that have to be present on the fuselage are prescribed in EASA CS25 requirements. Additionally, the minimum dimensions of each exit type are specified as well. In Table 4.1 the minimum dimensions for each type of emergency exit is given. These dimensions are taken from CS 25.807. It must be noted that this table presents the minimum exit dimensions, aircraft manufacturers usually use exits with larger dimensions, when these are also used as the main entrance to the aircraft.

Table 4.1: CS 25.807 Minimum exit dimensions

Type	Location	Width [m]	Height [m]
1	Floor	0.61	1.22
2	Floor	0.51	1.12
2	Overwing	0.51	1.12
3	Overwing	0.51	0.91
4	Overwing	0.48	0.66
A	Floor	1.07	1.83
B	Floor	0.813	1.83
C	Floor	0.762	1.22
Ventral	Bottom of fuselage	0.61	1.22

Apart from the dimensions of each exit type, CS 25.807 also specifies the amount of emergency exits that must be present on an aircraft. The amount of exits is driven by the passenger capacity of the aircraft. An overview of the required type and amount of exits is given in Table 4.2. Table 4.2 presents amount of each type of eixt that must be placed on either side of the fuselage. For example, certification specification require a the total amount 4 type 1 exits and 4 exits of type 3 to be positioned on the fuselage of a 150 passenger aircraft. If the passenger capacity of the aircraft is more than 300, all emergency exits must be of type A. In this case, the total amount of exits is found by dividing the passenger capacity by 110.

Table 4.2: CS25.807 Amount and type of emergency exits for each side of the fuselage

	Type 1	Type 2	Type 3	Type 4	Type A	Type B	Type C
$N_{pax} < 10$				1			
$10 \leq N_{pax} < 20$			1				
$20 \leq N_{pax} < 40$		1	1				
$40 \leq N_{pax} < 80$	1			1			
$80 \leq N_{pax} < 110$	1			2			
$110 \leq N_{pax} < 140$	2			1			
$140 \leq N_{pax} < 180$	2			2			
$180 \leq N_{pax} < 220$							4
$220 \leq N_{pax} < 300$						4	
$N_{pax} > 300$					$N_{pax}/110$		

4.1.2. Emergency exit access

Only providing the amount of exits in certification specification would not provide the sufficient level op safety in case of an emergency. The emergency exits must also be easily accessible from within the fuselage. Therefore, EASA CS 25.813 contains requirements with respect to the access to the emergency exits. First of all, these requirements state the the distribution of the exits must be as uniform as possible. However, the dimensions of the exits on either side of the fuselage do not necessarily need to be symmetrical. Furthermore, CS 25.813 states that there must be an unobstructed passageway leading up to each exit of type 1, type 2 or type A. The width of the passageways must be at least 0.51 m for exits of type 1 and 2. For exits of type A the minimum passageway width is required to be at least 0.91 m.

4.1.3. Minimum aisle width

In case of emergency the passengers must be able to leave the aircraft as quickly as possible. To ensure that evacuation is possible within a reasonable amount of time, CS 25.815 prescribes the minimum width of the aisles in the fuselage. Similar to the amount of exits, the minimum aisle width depends on the passenger capacity of the aircraft. In Table 4.3 an overview of the minimum aisle width requirements is given [30]. It must be noted that most aircraft manufacturers design the fuselage with wider aisles to allow for more passenger comfort.

Table 4.3: CS 25.815 Minimum aisle width [m]

	< 0.64 m from floor	≥ 0.64 m from floor
$N_{pax} < 10$	0.30	0.38
$10 < N_{pax} \leq 19$	0.30	0.51
$N_{pax} \geq 20$	0.38	0.51

4.1.4. Maximum number of seats abreast

The requirement CS 25.817 limits the amount of seats that may be placed abreast in the case that only one aisle is present in the aircraft cabin. If this is the case, a maximum of three seats may be placed next to each other. Thus, one aisled aircraft have a maximum of six seats abreast. If more seats abreast are required, an additional aisle must be positioned in the cabin [30].

4.2. Statistical data

In this section statistical data regarding the design of aircraft fuselages is collected. The data has been collected from aircraft design handbooks or collected as part of this thesis work. This data has been captured in the KBE application in such a way that these statistics are used when no other inputs are given by the user. This means that for example the number of seats abreast is calculated using a statistical relationship if this value is not specified by the user. Data has been collected on the interiors of the aircraft cockpit and cabin, cargo bay and wing-body fairing and will be presented in the following sections.

4.2.1. Cockpit interior

One of the requirements of the cockpit is that it is positioned at much as possible in the nose of the fuselage [2]. This position allows the pilot to have good visibility, which is especially of importance during take-off and landing. Typical dimensions of a cockpit for a transport aircraft in a side-by-side configuration is given in Figure 4.1. In this figure, minimum requirements for visibility angles from the pilot seat are given as well. The floor of the cockpit and cabin is positioned in the XY-plane at z=0.

The detailed design of the cockpit is beyond the scope of the current version of ParaFuse. The visibility angles, however, are used to generate the guide curves of the fuselage, as was explained in the previous chapter. However, for the cabin configuration process the total length of the flight deck must be known, to be able to derive the position of the front cabin wall. The flight deck length is defined as the the total length of the cockpit, from the front pressure bulkhead to the front cabin wall. The main driver of the flight deck length is the number of cockpit seats. An estimation of flight deck length with respect to the amount of cockpit seat can be found in Table 4.4 [43].

Table 4.4: Flight deck length with respect to number of cockpit seats [43]

Amount of seats	**Flight deck length [m]**
2	2.5
3	3.3
4	3.8

4.2.2. Cabin interior

The cabin configuration process depends heavily on the required interior components and their dimensions. One can imagine that if the seat pitch is increased by a few centimeters, this will significantly increase the length of the fuselage as this increase in pitch is added for each seat row. Therefore, statistics of typical seat dimensions have been collected to be able to accurately size the cabin. Furthermore, the diameter of the upper fuselage cross section is determined by the number of seats abreast in the fuselage. Therefore, a

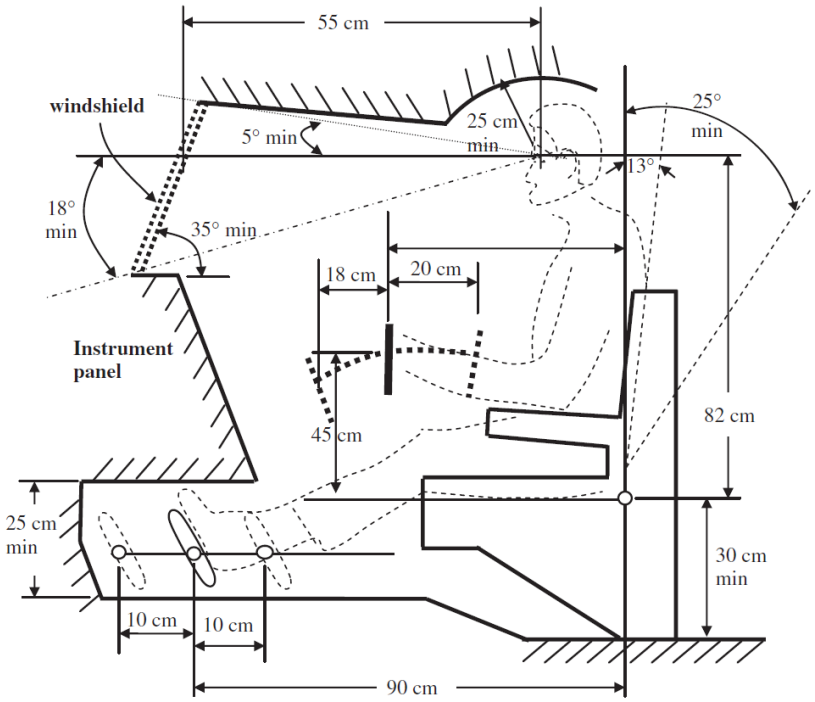


Figure 4.1: Typical cockpit dimensions for transport aircraft [2]

statistic on the number of seats abreast with respect to the passenger capacity has been collected as well. Furthermore, cabin designs require a certain amount of lavatories and galleys to provide the passengers with a certain level of service and cabin comfort. Typical figures with respect to the number of galleys and lavatories are presented in this section as well.

The level of cabin comfort is largely dictated by the dimensions of the aircraft seats and pitch of these seats. Airlines can tailor these values to fit their own objectives. For example a low-cost carrier will try to increase the passenger capacity as much as possible, to be able to offer tickets at a low price. High-end carriers, on the other hand, will often choose to have a cabin configuration that offers a higher level of comfort by for example increasing the pitch of the seats, at the cost of lower passenger capacity. These objectives often coincide with the wishes of their customers. To get an idea of typical values for the pitch and width of the seats data has been collected using the website `http://www.seatguru.com`. This website has a database on the cabin data of a large number of airlines. For the seat pitch and width the mean values have been derived for the different classes. The results are presented in Table 4.5. Histograms of this seat data can be found in Appendix D.

Table 4.5: Mean values for aircraft seat pitch and width

	Seat width [m]		**Seat pitch [m]**	
Class	**Short range**	**Long range**	**Short range**	**Long range**
Economy	0.45	0.45	0.81	0.82
Premium economy	0.48	0.48	0.97	0.97
Business	0.51	0.60	0.98	1.59
First	0.51	0.76	0.98	1.99

As mentioned before, the main driver for the upper fuselage cross section diameter is the number of seats abreast. A statistical relationship between the number of passengers in the aircraft, in a single class lay-out, and the number of seats abreast has been derived by Nita and Scholz [28]. This relation is given by Equation 4.1 and was derived by evaluating a large number of wide- and narrow-body aircraft. It must be noted that the user is able to specify the number of seats abreast. However, if the number of seats abreast is not specified

by the user, Equation 4.1 is used to determine the number of seats abreast.

$$N_{sa} = 0.45\sqrt{N_{pax}} \tag{4.1}$$

The final dimensions of the upper fuselage cross section are determined by clearance constraints around the cabin seats. The values of these clearances will also influence the level of cabin comfort. Typical values of these clearance constraints are specified in [24] and [40]. It is required that all passengers are able to freely move their head, therefore, the clearance radius between the head of the passengers and for example the fuselage wall must be between 0.2 and 0.3 m. Furthermore, the clearance between the outer cabin seats and the cabin wall must be between 0.025 m and 0.05 m. These clearance constraints determine the dimensions of the inner fuselage cross section. However, the KBE application is required to provide the outer fuselage diameter as well. This outer diameter is influenced by the thickness of the fuselage floor and thickness of the cabin wall. Torenbeek [24] presents statistics for these values. The floor thickness can be estimated to be 5% of the fuselage diameter. For non-circular fuselages the maximum width of the fuselage cross section is used. Furthermore, the outer cross section of the fuselage is approximately 4% larger than the inner cross section. Typical values for the amount of lavatories and galley volume per passenger are given in Table 4.6. Jenkinson et al. [42] states that the size of the base of a typical galley is 30 by 36 inches and the size of a typical lavatory is 36 by 36 inches.

Table 4.6: Required lavatories and galley volume [40]

		First class	**Business class**	**Economy class**
Passengers per lavatory	[-]	10 - 20	20 - 40	40 - 60
Galley volume per passenger	$[m^3]$	0.14 - 0.23	0.03 - 0.06	0 - 0.03

The position of the front cabin wall, which is determined using the estimate for the flight deck length is taken as the starting position of passenger cabin in the cabin configuration process. This passenger cabin can be located, partially, inside the nose cone of the aircraft. However, to perform the cabin configuration process the end position of the passenger cabin must also be determined. The cabin of an aircraft continues a certain length within the tail cone of the aircraft. To generate the interior of the fuselage, the length of the cabin inside the tail cone section must be determined. By evaluating a large number of reference aircraft, a statistical relationship between the length of the tail cone and the length of the cabin inside the tail cone has been derived by Morichon [44]. The length of the cabin inside the tail cone with respect to the tail cone length is given in Equation 4.2. Furthermore, the reference data and the trendline is shown in Figure 4.2

$$l_{cabin,tail} = \frac{l_{tail} - 6.3}{1.1} \tag{4.2}$$

4.2.3. Cargo

The dimensions of the lower fuselage cross section are driven by the required cargo type. Therefore, information on typical aircraft cargo containers has been collected. In the aviation industry standardized cargo containers are used. These containers are called Unit Load Devices (ULD). The use of ULDs allows for more efficient loading and handling of air cargo. Furthermore, aircraft fuselages can be designed to carry a specific type of ULD. Table 4.7 gives an overview of common ULD types and their dimensions. In Figure 4.3 the difference between half width and full width ULDs is shown. Smaller aircraft are not designed to carry ULDs, but are equipped with a bulk cargo bay. This cargo bay is fitted within the available space of the lower fuselage cross section.

4.2.4. Wing-body fairing

A parameterized wing-body fairing can be generated using ParaFuse. In order to provide a first estimate of the required fairing size statistics have been collected for a number of reference aircraft. These statistics have been collected by evaluating the technical drawings present in the aircraft characteristics and airport

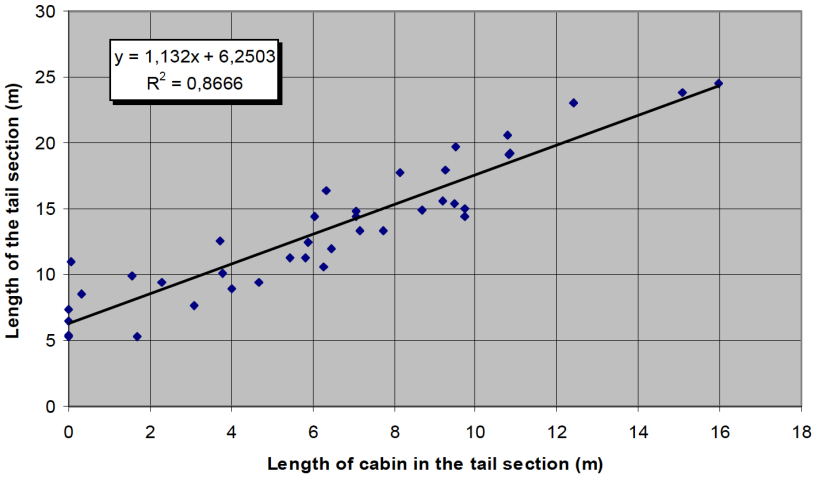


Figure 4.2: Length of the cabin in the tail cone with respect to tail cone length [44]

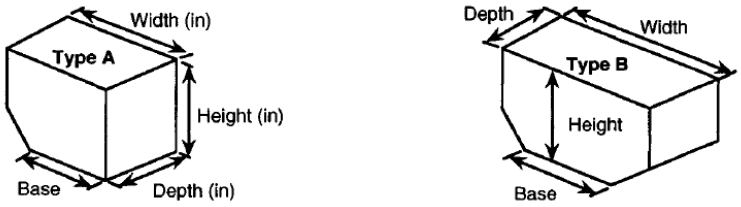


Figure 4.3: Left: Half width ULD. Right: Full width ULD [42]

Table 4.7: ULD dimensions and weight [42]

Designation	Width [m]	Height [m]	Depth [m]	Base [m]	Load [kg]	Notes
LD-1	2.337	1.626	1.534	1.562	1588	Half width
LD-2	1.562	1.626	1.534	1.194	1225	Half width
LD-3	2.007	1.626	1.534	1.562	1588	Half width
LD-4	2.438	1.626	1.534		2449	Rectangular
LD-5	3.175	1.626	1.534		3175	Rectangular
LD-6	4.064	1.626	1.534	3.175	3175	Full width
LD-7	3.175	1.626	2.032		6033	Rect/Contoured
LD-8	3.175	1.626	1.534	2.438	2449	Half width
LD-9	3.175	1.626	2.032		6033	Rect/Contoured
LD-10	3.175	1.626	1.534		3175	Contoured
LD-11	3.175	1.626	1.534		3175	Rectangular
LD-29	4.724	1.626	2.235	3.175	6033	Half width

planning documents provided by aircraft manufacturers. It is believed that the length of the wing-body fairing is determined mainly by the length of the wing chord at the wing-fuselage intersection. Therefore, a statistical relationship between the chord of the wing at the wing-fuselage intersection and the length of the fairing has been derived. Table 4.8 presents the length of the fairing and wing chord of several reference aircraft. This data has been plotted in Figure 4.4. The fairing length is related to the wing chord at the wing-fuselage intersection by Equation 4.3.

$$l_{fairing} = 0.0576c^2 + 0.3305c + 7.4533 \tag{4.3}$$

Table 4.8: Fairing data

Aircraft	Chord [m]	Fairing length [m]
A320-200	6.07	11.22
A330-300	10.56	19.31
A340-600	12.18	21.61
A350-900	13.47	23.50
A380-800	17.67	31.29
Boeing 747-400ER	14.66	24.92
Boeing 767-400ER	10.63	15.74
Boeing 777-300	13.26	20.16
Boeing 787-8	12.29	18.93
Embraer 170	4.92	10.49
Embraer 190	5.88	11.65
Fokker 100	5.67	8.54

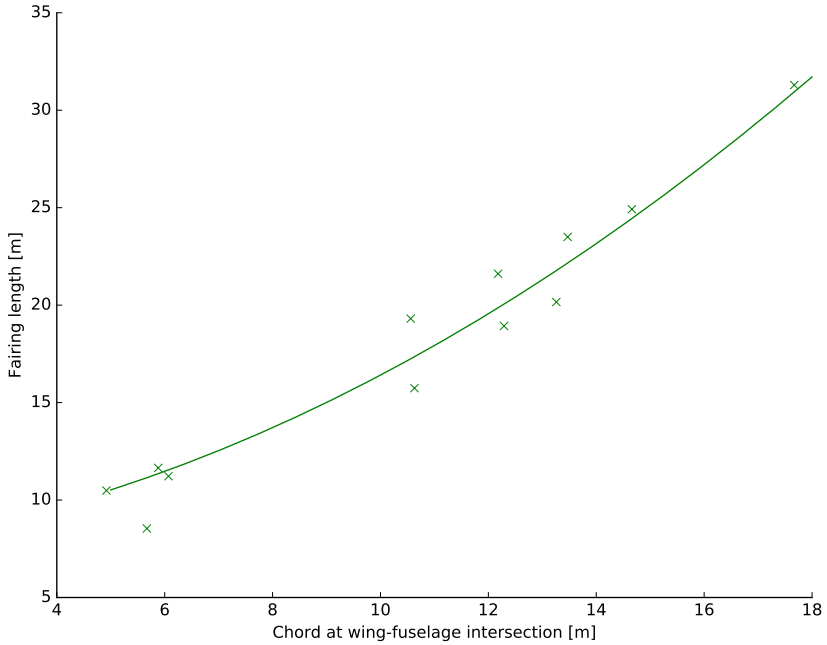


Figure 4.4: Fairing length as function of chord at wing-fuselage junction, $y = 0.0576x^2 + 0.3305x + 7.4533$, $R^2 = 0.96183$

4.3. Implementation

In the previous section statistical data regarding fuselage design has been presented. This data is either found in aircraft design handbooks [2, 24, 40–43] or has been collected using reference aircraft. Apart from statistical design rules, fixed dimensions of internal components such as galleys, lavatories, containers and seats are required to generate the fuselage design and interior components. This section gives an overview of how these design rules have been implemented in the KBE application.

These design rules and design knowledge are implemented in three different ways within the KBE application. An overview is given in the list below:

1. External data files: In the fuselage design process fixed dimensions are used for several components. For example, the dimensions of each ULD type are fixed. Separate data files have been created in the XML data format to store these values. An example of a ULD stored in the XML format is shown in Figure 4.5. This approach allows for high flexibility as these XML files can easily be extended or modified, without accessing the source code of ParaFuse. If one wants model a fuselage with a new ULD type, he or she only needs to add the dimensions of this ULD to the associated XML file. External data files have

been created for the ULDs, lavatories, galleys, exits and seats.

2. Validator functions: The fuselage model is subject to several constraints and requirements. One can define validator functions in ParaPy to ensure that inputs or attributes do not exceed certain values. For example, the aisle width may not be smaller than 38 cm according to CS25 requirements. The associated validator function is shown in Figure 4.6. A validator function evaluates the input that is given to a certain slot and if these values exceed the requirements specified in the validator function a warning is raised to the user.

3. Inputs or attributes: ParaFuse reads data from an input file to generate an instance of a aircraft fuselage. ParaPy uses input slots or attributes to read from external input files. The inputs or attributes can return a single value, however, they can also perform a calculation based on the values of other input slots or attributes. An example of the input slot that determines the number of seats abreast is given in Figure 4.7. One can see that ParaFuse first attempts to import a value for the number of seats abreast from the input file. However, if this value is not specified in the input file, ParaFuse calculates a value for the number of seats abreast based on the number of passengers in the fuselage.

```
<?xml version="1.0" encoding="utf-8" ?>
<!--All values should be specified in SI-units-->
<!--Dimensions are in [m]-->
<ulds>
    <uld type="LD-1">
        <width>2.340</width>
        <height>1.630</height>
        <height_kink>0.830</height_kink>
        <base>1.560</base>
        <depth>1.530</depth>
        <type>half</type>
    </uld>
```

Figure 4.5: Example of a ULD data stored in an XML data file

```
def aisle_width_validator(val, obj, slot):
    """Checks if aisle width complies with CS25 requirements, must be larger
    or equal to 0.380 m for aircraft with 10 or less passengers, else greater than or
    equal to 0.510.

    :rtype: Bool
    :unit: [m]
    :source: CS 25.815
    """
    return val >= 0.30 if obj.number_of_passengers <= 10 else val >= 0.38
```

Figure 4.6: Example of a validator function in ParaFuse

```
@Input
def number_of_seats_abreast(self):
    """Number of seats abreast, if not specified in input file a  statistical relation
    derived by Nita and Scholz, 2010 is used

    :return: number of seats abreast
    :type: float
    :unit: [-]
    :source: From Preliminary Cabin Design to Cabin Optimization, Nita and Scholz, 2010"""
    try:
        nsa = self.main.getIntegerElement('//general/parameters/parameter[name="number_of_seats_abreast"]/value')
    except TixiException:
        nsa = int(ceil(0.45 * sqrt(self.number_of_passengers)))
    return nsa
```

Figure 4.7: Example of an input slot in ParaFuse

5

Fuselage sizing and cabin configuration methods

In Chapter 3 the parameterization of the fuselage in ParaFuse has been discussed. To support the conceptual design process an inside-out fuselage sizing and outside-in cabin configuration method has been implemented. To automate these processes the design rules described in Chapter 4 have been implemented in ParaFuse. The inside-out design routine generates a fuselage model based on top level requirements, such as passenger capacity and cargo type. The implementation of this sizing method is described in Section 5.1. An outside-in cabin configuration method has been implemented in ParaFuse as well. The outside-in method is able to generate a cabin interior for a fuselage with fixed external dimensions. The implementation of this cabin configuration method is discussed in Section 5.2. Finally, the fuselages of several existing aircraft have been reconstructed using ParaFuse to validate the implementation of the cabin configuration methods. These validation cases are presented in Section 5.3. A quick user guide, with an overview of the required input files and the different ways to operate ParaFuse, is included in Appendix F.

5.1. Inside-out

The inside-out fuselage design procedure can be used to generate the fuselage outer surface, as well as the interior cabin layout based on top level requirements. ParaFuse can generate fuselages for conventional low-wing passenger aircraft, therefore, the main drivers of the design are passenger capacity and the type of cargo. An overview of the inside-out fuselage design procedure is given in Figure 5.1. The first activity of the inside-out design process is the generation of the main fuselage cross section. This process is described in Section 5.1.1. Subsequently, the cabin configuration process is performed to determine the external dimensions of the fuselage. The cabin configuration process is described in in Section 5.1.2. After the cabin configuration process is finished, the resulting external dimensions, together with some user inputs, such as fineness ratio of the nose and tail cone are used to generate the final fuselage surface.

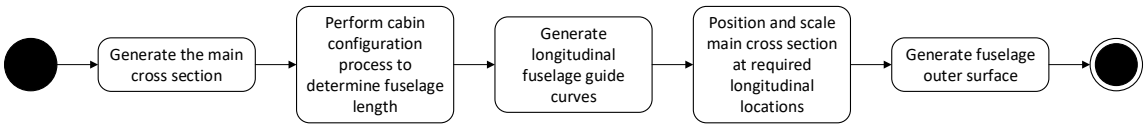


Figure 5.1: Overview of the activities in the inside-out fuselage design process

5.1.1. Main cross section sizing procedure

The inside-out sizing procedure is initiated with the generation of the main fuselage cross section. The main cross section is defined as the cross section of the constant middle section of the fuselage. In essence, the

sizing procedure of the main cross section is a two dimensional design problem that is used to determine the external dimensions of the fuselage cross section. An overview of this process is shown in Figure 5.2. In this section the required steps of this sizing procedure are explained. An overview of the required inputs for the generation of the main fuselage cross section is given in Table 5.1. The numerical values in Table 5.1 indicate the default value that a parameter takes if it is not specified in the input file. These default values have been derived using the fuselage design rules described in Chapter 4.

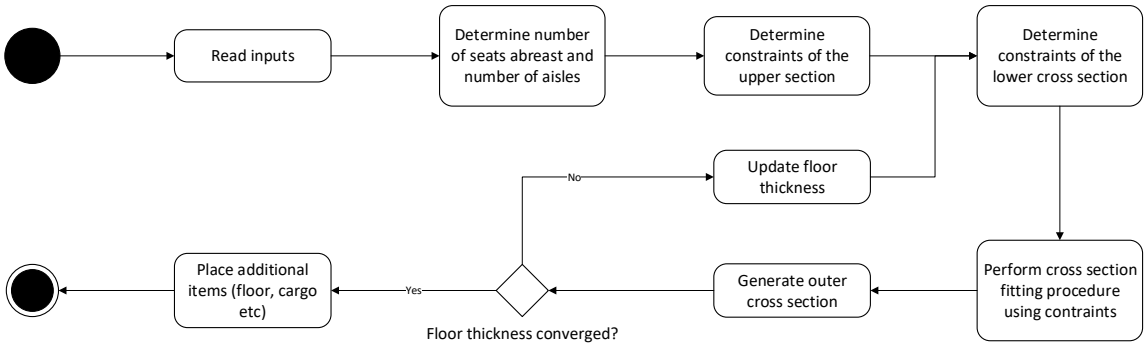


Figure 5.2: Generation of the main fuselage cross section

Table 5.1: Required inputs for the generation of the main fuselage cross section

Parameter	**Type**	**Value**	**Unit**	**Description**
number_of_passengers	int	-	[-]	Number of passengers
number_of_seats_abreast	int	-	[-]	Number of seats abreast (optional)
cross_section_shape	str	-	[-]	Cross-sectional shape: 'circle', 'ellipse', 'double_bubble' or 'free_form'
cargo_type	str	-	[-]	Cargo type: 'containerized' or 'bulk'
uld_type	str	-	[-]	Type of ULD. Only required if cargo_type == 'containerized'
minimum_bulk_bay_height	float	-	[m]	Minimum height of the bulk cargo bay. Only required if cargo_type == 'bulk'
aisle_width	float	0.50	[m]	Aisle width
aisle_height	float	1.93	[m]	Aisle height
wall_thickness_factor	float	0.04	[-]	Fuselage wall thickness as a percentage of fuselage diameter
floor_thickness_factor	float	0.05	[-]	Fuselage floor thickness as a percentage of fuselage diameter
sitting_eye_height	float	0.89	[m]	Eye height while sitting, used to position head clearance radius.
head_clearance_radius	float	0.25	[m]	Head clearance radius, used for head clearance constraints
sidewall_clearance	float	0.025	[m]	Clearance between seats and sidewall
container_clearance	float	0.01	[m]	Clearance of the containers. Only required when cargo_type == 'containerized'

ParaFuse is able to generate fuselages carrying containerized cargo or bulk cargo. An example of both cross sections is shown in Figures 5.3 and 5.4, respectively. The former figure shows a fuselage cross section with containerized cargo, whereas the latter figure shows a fuselage cross section with a bulk cargo bay. These figures will be used to elaborate on the cross section sizing process. Although, the two cross sections shown in the figures are circular, the cross section sizing procedure is able to fit circular, elliptical, double bubble and free form cross-sectional shapes.

The parameter that drives the dimensions of the upper fuselage cross section is the number of seats abreast,

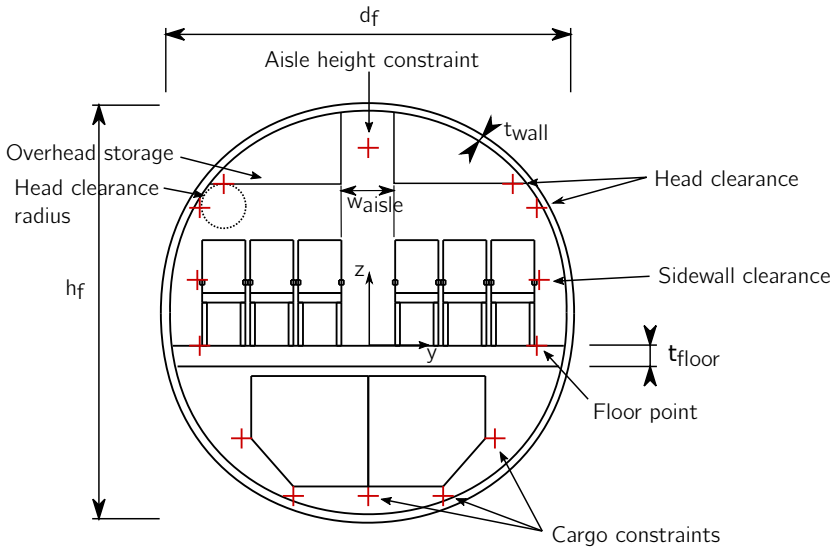


Figure 5.3: Fuselage cross section with containerized cargo generated by ParaFuse

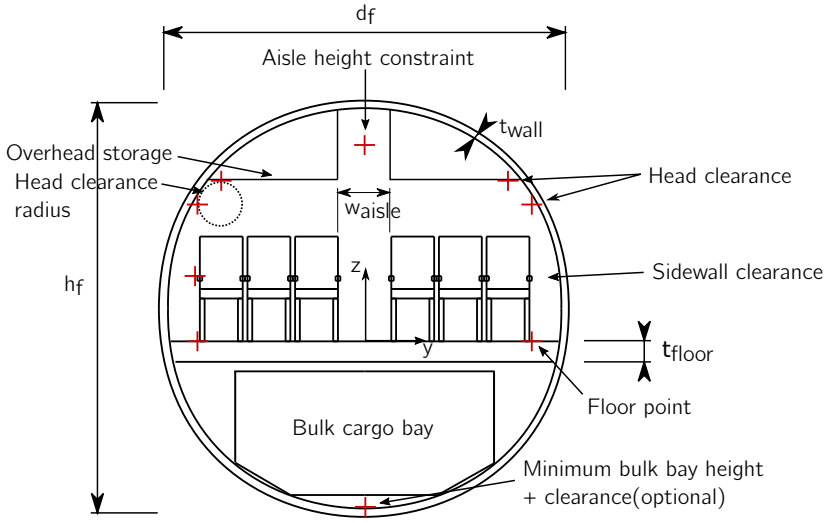


Figure 5.4: Fuselage cross section with bulk cargo generated by ParaFuse

i.e. the amount of seats that are positioned on one row. The user is able to provide this value directly as an input. However, if this parameter is not specified, the amount of passengers in the aircraft is used to determine the number of seats abreast with Equation 4.1. The fuselage has to be compliant with CS 25 requirements, therefore, an additional aisle is generated automatically when the required number of seats abreast is larger than six.

After the seats have been positioned, the constraints of the upper fuselage cross section are positioned accordingly. The constraints are indicated with the crosses in Figures 5.3 and 5.4. The fuselage cross section is assumed to be symmetric in the z-axis. For clarity, the labels indicating the different constraints are only specified on one side of the cross section. However, these labels also apply to the crosses on the opposite side of the cross section. The constraints that are taken into account for the sizing procedure of the upper fuselage cross section are listed below:

1. Head clearance: The passengers should be able to move their head freely. Therefore, a head clearance radius is used as input. The position of the passenger head must be specified as well. For a 95th percentile male passenger, this height is 0.89 m above the seat cushion [45]. Typically, the value for head clearance is between 0.2 and 0.3 meters. [24].

2. Sidewall clearance: This is the distance between the cabin wall and the armrest of the outer cabin seats. Typical values for the sidewall clearance range between 0.025 and 0.05 meters [24].

3. Aisle height: This constraint specifies the height of the main aisle in the fuselage. Generally, the aisle height is at least 1.93 meters for large passenger aircraft [40].

4. Floor points: These points specify the extremes of the outer cabin seats at z = 0.

Following the positioning of the upper cross section constraints, the constraints of the lower fuselage cross section are determined. In case of containerized cargo these constraints enclose the ULDs, with a certain clearance. If bulk cargo is required, a minimum bulk cargo bay height can be specified by the user. However, the user can also choose not to specify a minimum cargo bay height. If this is the case, no lower cross section constraints are taken into account in the cross section fitting procedure. The position of the containers or cargo bay is influenced by the thickness of the fuselage floor. However, in conceptual fuselage design the floor thickness can be estimated as a percentage of the fuselage diameter [24]. Because the goal of the sizing procedure is to determine the external dimensions of the main fuselage cross section, the diameter is not known yet, an initial value is estimated. Therefore, an iterative process is required ot determine the thickness of the fuselage floor, as illustrated in Figure 5.2.

After the constraints of the lower cross section have been positioned the diameter of the inner fuselage cross section is found by solving a constrained least squares problem using a sequential least squares programming algorithm [46]. For the circular and elliptical cross sections the algorithm tries to minimize the cross-sectional area, whilst ensuring that all constraints are enclosed by the main cross section. To generate the double bubble cross section, two separate optimization runs are performed to fit a circle around the upper and lower cross section constraints, separately. The two floor points are included as constraints in both optimization problems to ensure that the intersection points of the two circles are located at the floor points. For the free form cross section the optimizer solves a least squares problem that minimizes the distance between the constraints and the cross-sectional curve by modifying the class function coefficients ($N1$ and $N2$) and scaling coefficients of the Bernstein polynomials (A_i) of the CST function.

The outer cross section is assumed to be 4% larger than the inner cross section [24]. Hence, the inner cross section is scaled to create the outer cross section of the fuselage. Subsequently, the required floor thickness is calculated using the outer fuselage diameter. If the estimate of the floor thickness and the required floor thickness have not converged, another iteration is performed with the updated floor thickness. This process continues until the floor thickness is converged.

After the floor thickness is converged, the actual floor is generated. If a bulk cargo bay is required, the bulk cargo bay is generated as well. Finally, the overhead storage compartments are generated. The overhead storage compartments are located above the seats. The height of the overhead storage compartments is determined by adding the head clearance radius to the position of the head of the passengers. Thus, in the current version of ParaFuse the size of the overhead storage compartments is not taken into account in the optimization problem. A trend can be observed that almost no luggage is stored in the cargo bay anymore, but taken on board as carry-on luggage, especially on regional flights. Therefore, a possibility to specify minimum overhead storage compartment height might be an interesting extension to include in future versions of ParaFuse, as the overhead storage volume may become a driver for the design of passenger cabins.

Subsequently, the external dimensions and shape of the main cross section are used to generate the nose and tail cone surface. After that the nose and tail cone section are used as a starting point for the complete cabin configuration process of the fuselage.

5.1.2. Cabin configuration

After the dimensions of the main fuselage cross section have been determined, the inside-out design routine performs the cabin configuration process to determine the external dimensions of the fuselage. The cabin configuration process can be split up in three parts. Similar to the parameterization of the fuselage surface explained in Chapter 3, the cabin configuration process is performed for the nose cone, middle section and tail cone of the fuselage separately. This section will elaborate on the inside-out cabin configuration process.

As a first step in the cabin configuration process the required amount of exits, lavatories and galleys are calculated based on the required passenger capacity. The dimensions these components are taken from XML data files, as described in Section 4.3. If multiple classes are present, ParaFuse converts the number of first, business and premium economy class seats to an equivalent number of seats in high density configuration. Subsequently, the number of exits is determined using the amount of seats that would fit within the cabin in high density configuration. This is also how aircraft manufacturers determine the amount of exits. They offer cabins with different seating layout, but the amount of exits is always determined based on high density passenger capacity. The inside-out cabin configuration process first generates the surface of the tail cone and positions the interior components of the tail cone. This process is described in Section 5.1.2. After that, components of the nose cone section are positioned. An overview of the nose cone configuration process is given in Section 5.1.2. Subsequently, the remaining cabin elements are positioned in the constant middle section of the fuselage. This process is described in Section 5.1.2. An overview of the complete cabin configuration process is shown in the activity diagram presented in Figure 5.5. A complete list of the required input parameters that are used in the cabin configuration process is given in Table 5.2.

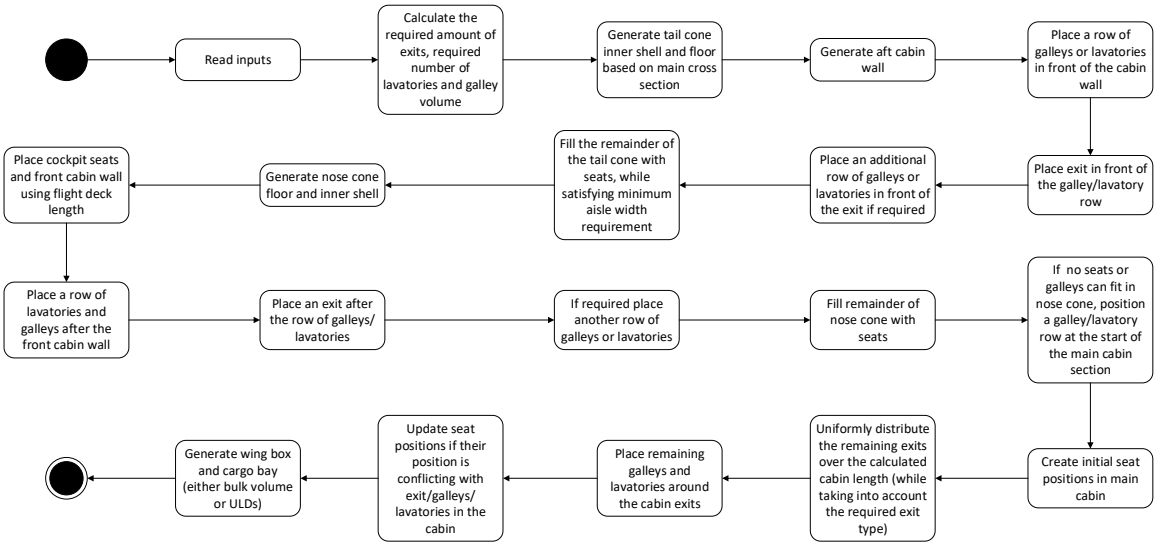


Figure 5.5: Inside-out cabin confguration process

Tail cone interior

The configuration process of the tail cone can be split up into three parts. The first step in the tail cone configuration process is the generation of the inner surface of the tail cone. The length of the tail cone is determined using the fineness ratio of the tail cone, which is specified in the input file and the diameter of the main fuselage cross section, that has been determined using the main cross section sizing procedure. Subsequently, the floor of the cabin is generated inside the tail cone surface and the aft cabin wall is positioned using Equation 4.2.

After these surfaces have been created, the inside-out cabin configurator continues by positioning a row of galleys and a row of exits in front of the cabin wall. Subsequently, another row of galleys or lavatories is positioned in front of the tail cone exits. After positioning a galley or lavatory the cabin configurator evaluates the number of installed lavatories and the installed galley volume and compares this to the required number of lavatories and galley volume. If these requirements are satisfied, the cabin configurator continues without positioning additional galleys or lavatories.

The next procedure in the tail cone configuration process is the positioning of the tail cone seats. To generate these seats ParaFuse uses the `UserPattern` class in ParaPy. This class generates copies of a seat object in a user specified pattern, instead of initiating a new seat instance for every seat in the cabin. Copying the seat instances is much less computationally expensive than the latter option, and therefore allows for faster cabin configuration. The number of seats abreast of the constant middle fuselage section is determined during

the sizing procedure of the main cross section. However, because of the fact that the tail cone is tapered, it might be possible that the minimum aisle width requirement, as specified in CS 25.815, is violated. To check this requirement, seat rails are used to position the seats in the tail cone. These seat rails originate from the position of the seats in the untapered part of the cabin and follow the contour of the tail cone floor surface. At each seat row, the distance between the two seats that enclose the aisle is evaluated. If the minimum aisle width requirement is violated, the cabin configurator does not store the position of the aisle seat in the list that contains the position of the tail cone seats. The magnitude of the aisle width violation is taken into account. If possible only one seat is not positioned in a seat row. In case of two aisles, the center seats may be repositioned to keep symmetry. If the violation is larger, the cabin configurator skips two or more seats. The procedure is illustrated in Figure 5.6. The tail cone configuration process continues until the entire tail cone is filled with seats. An example of a tail cone generated using ParaFuse is presented in Figure 5.7. The amount of seats, lavatories and installed galley volume is stored by the tail cone configurator class. Subsequently, this information is given as an input for the nose cone configuration class.

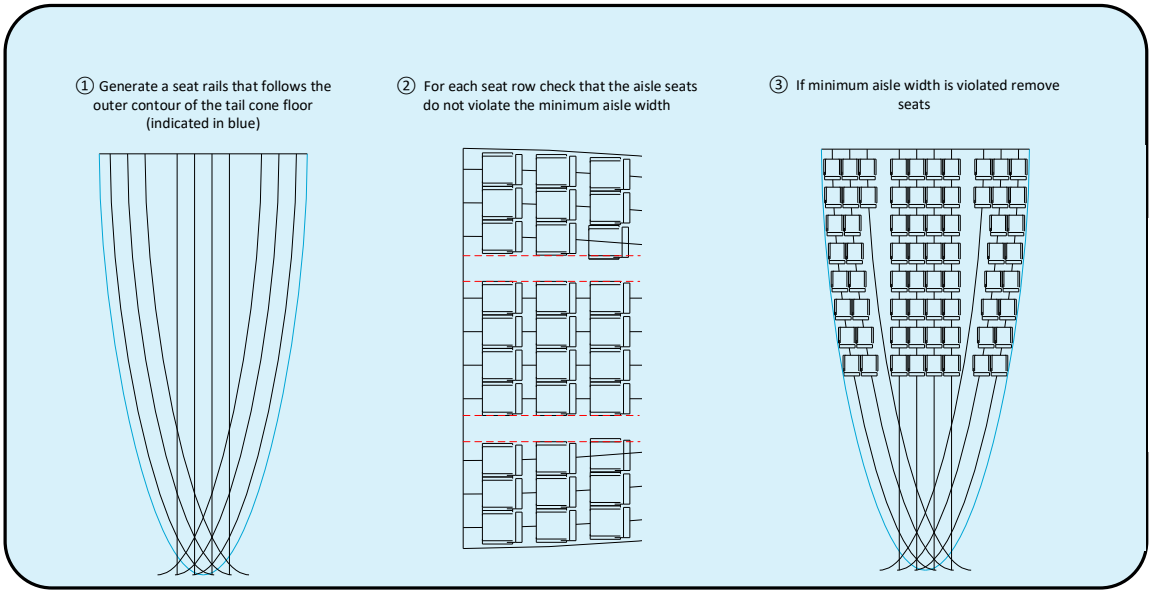


Figure 5.6: Seat removal procedure using seat rails

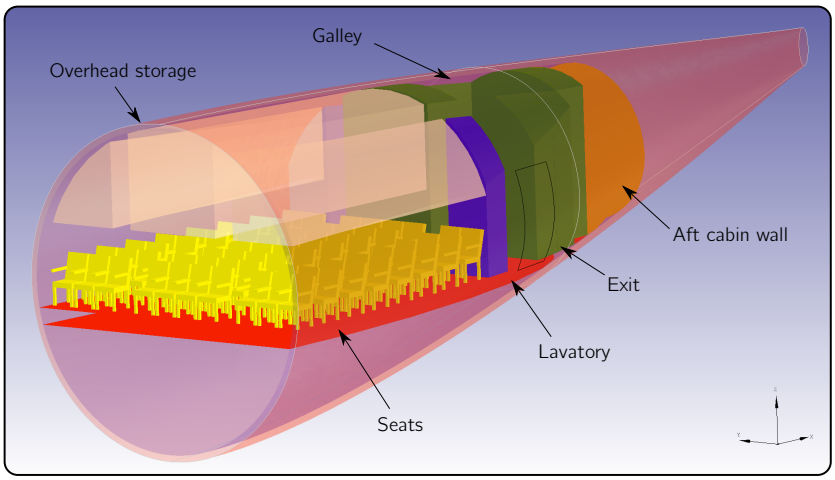


Figure 5.7: Tail cone interior generated using ParaFuse

Nose cone interior

After the interiors of the tail cone have been positioned, the nose cone is filled with the required interior components. The length of the nose cone is calculated using the dimensions of the main fuselage cross section and user input for nose cone fineness ratio. An overview of a nose cone section generated using ParaFuse is shown in Figure 5.8. First, the cockpit is positioned in the nose of the aircraft. Only the cockpit seats are positioned with respect to the front pressure bulkhead and the length of the flight deck is estimated using statistics. Susbequently, the front cabin wall is positioned and, if required, a row of galleys and lavatories is positioned in the nose cone. The cabin configurator evaluates whether additional lavatories or galleys are still required before positioning them in the nose cone. It must be noted that lavatories and galley may be positioned on the same row. After positioning the first row of lavatories or galleys, the cabin configurator positions an exit in the nose cone. If there is space and additional lavatories and galleys are still required, these are positioned aft of the nose cone exits. Following this, seats are positioned in the nose cone section. If multiple classes are present, the following order is used: first class, business class, premium economy class and economy class. In case of economy class seats the seats are positioned using the same procedure as the tail cone seat positioning procedure. For first class, business class or premium economy class seats, the amount of seats that can be positioned abreast is not known from the main cross section sizing procedure. Therefore, this value is calculated using equation 5.1 at each seat row location. This equation calculates the number of seats abreast for a specific seat type by subtracting the product of the aisle width and number of aisles from the width of the fuselage at the seat row location and dividing this value by the width of the seats that needs to be positioned. The resulting value is rounded to the lowest integer.

$$N_{sa} = \lfloor \frac{w_{fuselage,x} - n_{aisle} w_{aisle}}{w_{seat}} \rfloor \tag{5.1}$$

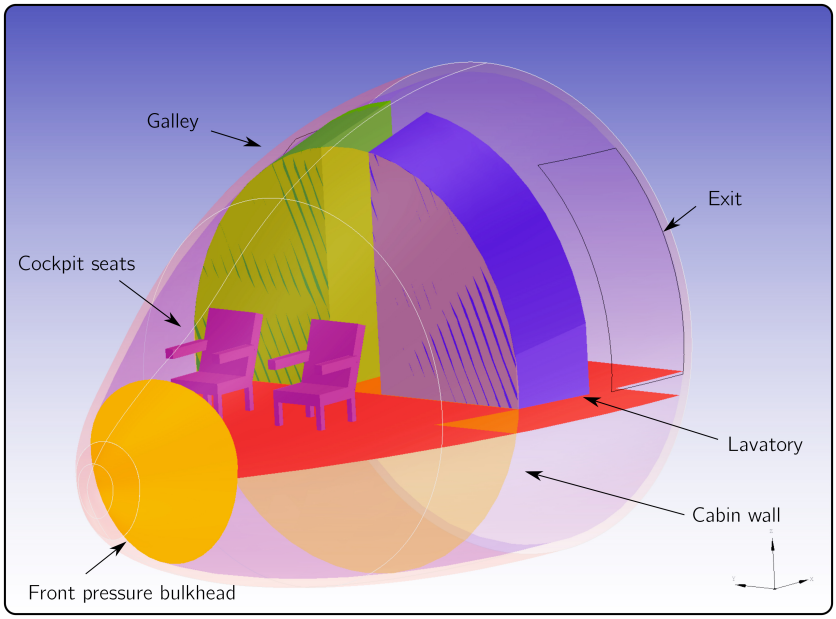


Figure 5.8: Nose cone interior generated using ParaFuse

The seat placement procedure continues until the entire nose cone is filled with seats. If this process is finished, the number of remaining seats, exits and monuments that still need to be installed is calculated. These values are transferred to the class that generates the interior of the constant middle fuselage cross section.

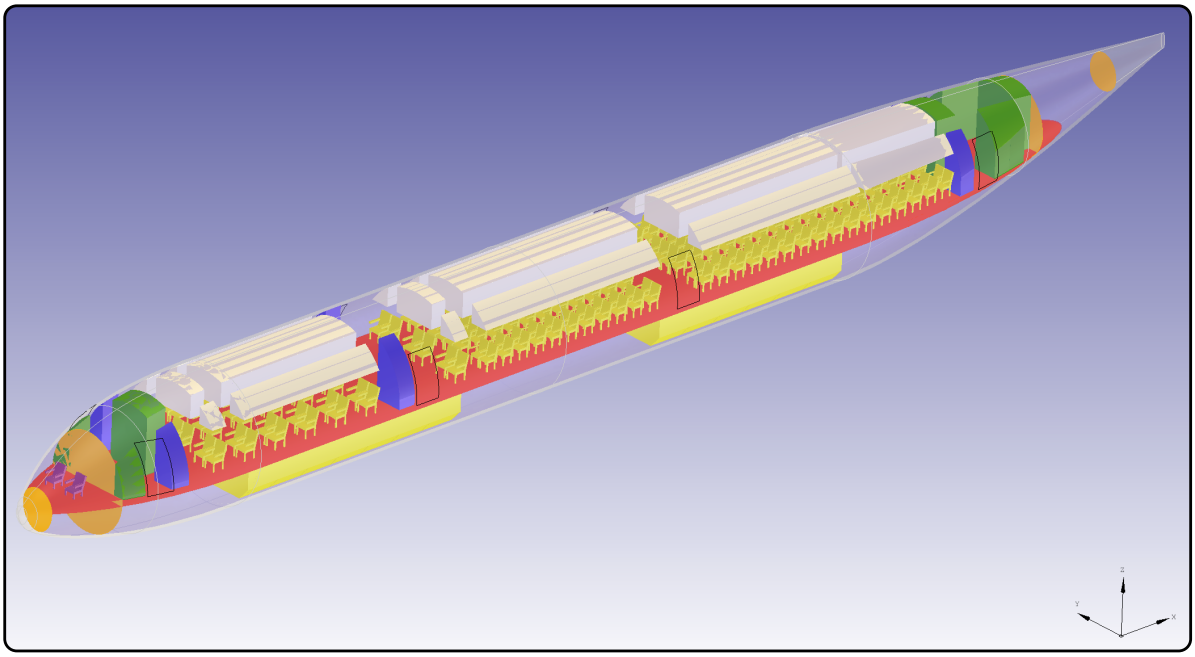
Interior of the middle fuselage section

The inside-out design approach uses the interior components to determine the dimensions of the outer fuselage geometry. Therefore, the length of the middle fuselage section is not known beforehand. All remaining

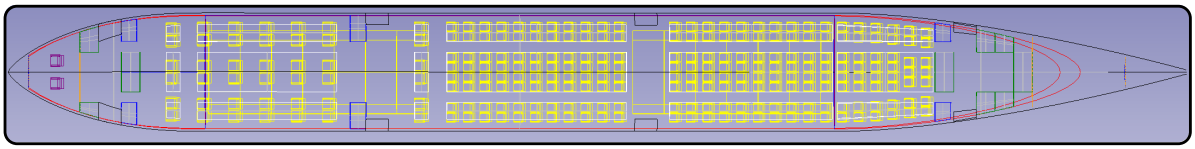
interior components are positioned in the middle fuselage cross section. However, a few intermediate step are required to perform this operation.

First of all, the remaining seats are positioned in the middle fuselage section. Subsequently, the remaining exits are distributed uniformly over the resulting length of the fuselage. The total length of the fuselage is not known yet, because it is possible that additional exits and monuments still need to be positioned in the cabin. Therefore, ParaFuse calculates the length that these components will require before positioning the exits. This length is used to distribute the remaining exits. At the same time ParaFuse uses this fuselage length to make sure that floor type exits are not positioned over the wing. The main wing position is calculated using statistics or directly retrieved from a user input. After positioning the remaining exits, ParaFuse distributes the remaining lavatories and galleys around the emergency exits. The initial seat positions are updated if these conflict with the position of the exits or lavatories and galleys. Finally, components that must be positioned in the belly of the fuselage are generated. In this process an area is reserved for the wing box and the remaining volume is filled with either cargo containers or bulk cargo bays. ParaFuse is able to calculate the resulting volume of the cargo bays. This information can be retrieved by the user via the GUI or by generating a design report, which is a text based overview of the main fuselage characteristics.

After positioning all required interior components, the length of the middle section of the fuselage is used to generate the outer fuselage surface with the correct length. Figure 5.9 shows an example of a fuselage for a two class A330 like passenger aicraft generated in ParaFuse.



(a) Isometric view



(b) Top view

Figure 5.9: A330 like passenger aircraft generated using the inside-out design method. Galleys are indicated in green and lavatories are indicated in blue.

Table 5.2: Required inputs for the inside-out design routine, including example values

Parameter	Type	Value	Unit	Description
number_of_passengers	int	150	[-]	Number of passengers
number_of_seats_abreast	int	6	[-]	Number of seats abreast (optional)
percentage_economy	float	0.88	[-]	Percentage of seats in economy class. Check to make sure that total amounts to 1
percentage_premium_economy	float	0	[-]	Percentage of seats in premium economy class
percentage_business	float	0.12	[-]	Percentage of seats in business class
percentage_first	float	0	[-]	Percentage of seats in first class
pax_per_lavatory_economy	int	50	[-]	Number of passengers per lavatory in economy class
pax_per_lavatory_premium_economy	int	50	[-]	Number of passengers per lavatory in premium economy class
pax_per_lavatory_first	int	35	[-]	Number of passengers per lavatory in business class
pax_per_lavatory_first	int	15	[-]	Number of passengers per lavatory in first class
galley_vol_per_pax_economy	float	0.03	[m^3]	Required galley volume per passenger in economy class
galley_vol_per_pax_premium_economy	float	0.03	[m^3]	Required galley volume per passenger in premium economy class
galley_vol_per_pax_business	float	0.04	[m^3]	Required galley volume per passenger in business class
galley_vol_per_pax_first	float	0.15	[m^3]	Required galley volume per passenger in first class
cross_section_shape	str	circle	[-]	Cross-sectional shape: ‘circle’, ‘ellipse’, ‘double_bubble’ or ‘free_form’
cargo_type	str	containerized	[-]	Cargo type: ‘containerized’ or ‘bulk’
uld_type	str	LD-3-45W	[-]	Type of ULD. Only required is cargo_type == ‘containerized’
container_clearance	float	0.1	[m]	Clearance of the cargo bay
minimum_bulk_bay_height	float	0.90	[m]	Minimum height of the bulk cargo bay. Optional if cargo_type == ‘bulk’
aisle_width	float	0.51	[m]	Aisle width
aisle_height	float	2	[m]	Aisle height
wall_thickness_factor	float	0.04	[-]	Fuselage wall thickness as a percentage of fuselage diameter
floor_thickness_factor	float	0.05	[-]	Fuselage floor thickness as a percentage of fuselage diameter
sitting_eye_height	float	0.89	[m]	Eye height while sitting, used for head clearance constraints
head_clearance_radius	float	0.2	[m]	Head clearance radius, used for head clearance constraints
sidewall_clearance	float	0.025	[m]	Clearance between seats and sidewall
FR_n	float	1.5	[-]	Fineness ratio of the nose cone
FR_t	float	3	[-]	Fineness ratio of the tail cone

5.2. Outside-in

In contrast to sizing the outer fuselage surface based on the required interior components, the outside-in method can be used to investigate different payload and cabin configurations of a fuselage with fixed external dimensions. This section gives an overview of the outside-in cabin configuration method. The processes required to generate the fuselage surface with interior components using the outside-in method are shown in Figure 5.10. The focus of the outside-in cabin configuration process is on the last activity block of this figure.

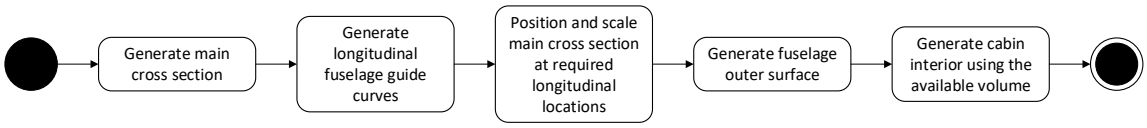


Figure 5.10: Overview of the activities in the outside-in cabin configuration process

It must be noted that, although the design approach is different, the resulting cabin is similar to the cabins generated using the inside-out approach. This is caused by the fact that the same design rules are used by both fuselage sizing methods. Therefore, the focus of this section is to describe the differences between the two methods, rather than repeating the steps taken in the cabin configuration approach. First of all, the generation of the main cross section for the outside-in design approach is discussed in Section 5.2.1. Subsequently, the cabin configuration process is decribed in Section 5.2.2.

5.2.1. Main cross section sizing procedure

Similar to the inside-out design routine, the generation of the main fuselage cross section is used as a starting point to generate the fuselage surface. The process to generate the fuselage cross section is shown in Figure 5.11. Because the external dimensions of the fuselage are used as an input, the first activity in this process is the generation of the main fuselage cross section. The cross section can be generated in two different ways. Either the diameter of the fuselage cross section is given as an input and used to generate a circular cross section or the cross section is specified as a fuselage profile in the CPACS data format. An example of such a fuselage profile is given in Listing 5.1. The second method allows for the generation of fuselage surfaces with a cross section other than a circle.

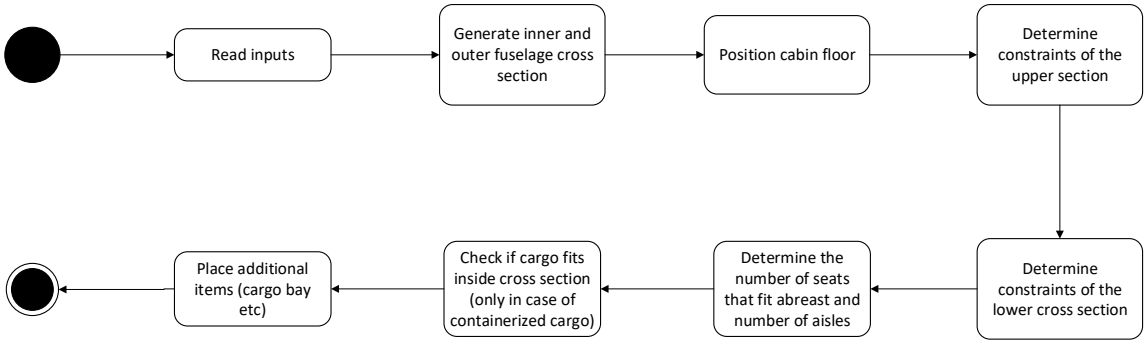


Figure 5.11: Generation of the main fuselage cross section and calculation of number of seats abreast using the outside-in method

Listing 5.1: CPACS fuselage profile

```
<!--DOUBLE BUBBLE-->
<fuselageProfile uID="fuselage_Profile50">
    <name>Profile_ParaFuse_fuselage_50</name>
    <pointList>
        <x mapType="vector">0;0;0;...;0</x>
        <y mapType="vector">1.77636e-015;-0.376963;-0.746046;...;1.55431e-015</y>
```

```
        <z mapType="vector">-2.76855;-2.74123;-2.65983;...;-2.76855</z>
    </pointList>
</fuselageProfile>
```

Because the diameter of the fuselage cross section is known, no iterations are required to determine the thickness of the cabin floor. Following the placement of the cabin floor, which is located at z=0, the constraints of the cross section are positioned. For the outside-in case, these constraints are not used to determine the dimensions of the cross section, but are used to determine the amount of seats that can be positioned abreast. If the number of seats abreast is larger than six, see Section 4.1.4, ParaFuse automatically generates a second aisle. ParaFuse uses dimensions of economy class seats to determine the number of seats that can be positioned abreast for the main cross section. The lower cross section constraints are only generated in case of containerized cargo. These constraints are used to check whether or not the ULD type specified by the user fits within lower cross section of the fuselage. If the constraints fall within the inner cross section of the fuselage, the ULDs are generated. However, if a constraint is located outside the inner fuselage cross section the ULDs are not generated and a warning is raised to notify the user. A complete overview of the required inputs, together with default values if the parameters are not specified in the input, to generate the main fuselage cross section for the outside-in cabin configuration method is shown in Table 5.3. The main cross section, together with a user specified fuselage length, is used to generate the longitudinal guide curves. Subsequently, the outer fuselage surface is generated. The fuselage shell is used as a starting point for the cabin configuration process described in the next section.

Table 5.3: Required inputs for the outside-in cross section generation process

Parameter	Type	Value	Unit	Note
fuselage_diameter	float	-	[m]	Diameter of the fuselage cross section (optional, otherwise specify CPACS fuselage profile)
z_center	float	-	[m]	Height of the cross section center position (optional, otherwise specify CPACS fuselage profile)
cargo_type	str	-	[-]	Cargo type: 'containerized' or 'bulk'
uld_type	str	-	[-]	Type of ULD. Only required is cargo_type == 'containerized'
aisle_width	float	0.50	[m]	Aisle width
wall_thickness_factor	float	0.05	[-]	Fuselage wall thickness as a percentage of fuselage diameter
floor_thickness_factor	float	0.04	[-]	Fuselage floor thickness as a percentage of fuselage diameter
sitting_eye_height	float	0.89	[m]	Eye height while sitting, used for head clearance constraints
head_clearance_radius	float	0.25	[m]	Head clearance radius, used for head clearance constraints
sidewall_clearance	float	0.025	[m]	Clearance between seats and sidewall
container_clearance	float	0.1	[m]	Clearance of the cargo bay

5.2.2. Cabin configuration

The outside-in cabin configuration method can be used to perform cabin (re)configuration studies using a fuselage with fixed external dimensions. The outside-in cabin configurator generates and positions the required interior components, such as lavatories, galleys, cargo bays and seats in the available interior space. Figure 5.12 outlines the subsequent steps of the outside-in cabin configuration process.

The cabin configuration process of the outside-in method follows the same design rules as the inside-out cabin configuration process described in Section 5.1.2. First, the tail and nose cone sections of the fuselage are filled with the required interior components. Because of the fact that the length of the fuselage is known beforehand, the exits can be distributed uniformly over the fuselage length. After that, the remaining galleys and lavatories are positioned around these exits. Finally, the remaining space in the middle fuselage section is filled with seats.

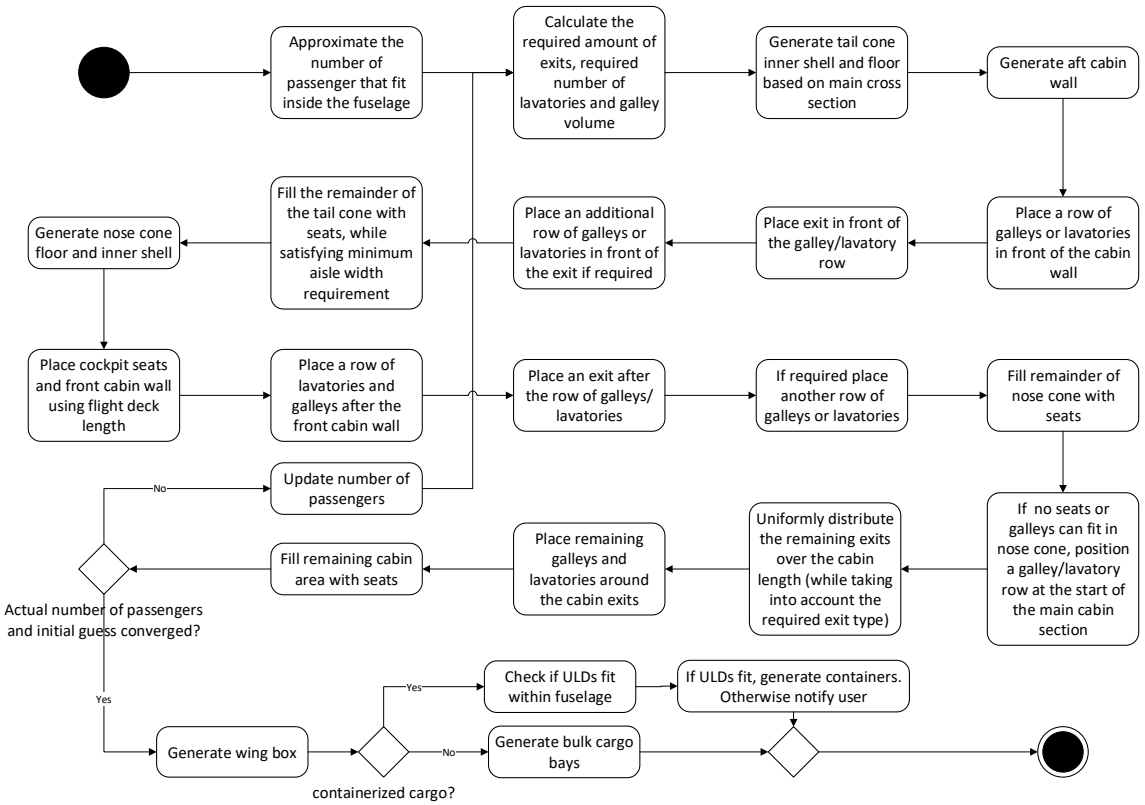


Figure 5.12: Outside-in cabin confguration process

However, the amount of lavatories, galleys and exits that need to be installed in the cabin depend on the number of passengers in the aircraft. For the outside-in cabin configuration process, the passenger capacity is not known a priori, as the external fuselage dimensions are the driver for the layout of the cabin. Therefore, the outside-in cabin configuration process is of an iterative nature. The cabin configuration process is initiated by estimating the passenger capacity of the aircraft using Equation 5.2. The passenger capacity is estimated by dividing the length of the cabin by the seat pitch and multiplying the result with the number of seats abreats in the main fuselage cross section. Equation 5.2 overestimates the passenger capacity of the aircraft, because the cabin is tapered in the nose and tail cone and some cabin area cannot be used to position seats. After one cabin configuration run, the actual amount of seats in the cabin is used as an input for the next iteration. This process continues until the number of seats converges. A complete overview of the required inputs for the outside-in cabin configuration process is given in Table 5.4.

$$N_{pax} = \frac{l_{cabin}}{l_{pitch}} N_{sa} \tag{5.2}$$

Table 5.4: Required inputs for the 'outside-in' design routine, including example values

Parameter	Type	Value	Unit	Description
fuselage_length	float	37.57	[m]	Length of the fuselage
fuselage_diameter	float	4.14	[m]	Fuselage diameter (optional)
z_center	float	0.5	[m]	Height of the cross section center position (optional)
FR_n	float	1.5	[-]	Fineness ratio of the nose section
FR_t	float	3	[-]	Fineness ratio of the tail section
percentage_economy	float	0.88	[-]	Percentage of seats in economy class
percentage_premium_economy	float	0	[-]	Percentage of seats in premium economy class
percentage_business	float	0.12	[-]	Percentage of seats in business class
percentage_first	float	0	[-]	Percentage of seats in first class
pax_per_lavatory_economy	int	50	[-]	Number of passengers per lavatory in economy class
pax_per_lavatory_premium_economy	int	50	[-]	Number of passengers per lavatory in premium economy class
pax_per_lavatory_business	int	35	[-]	Number of passengers per lavatory in business class
pax_per_lavatory_first	int	15	[-]	Number of passengers per lavatory in first class
galley_vol_per_pax_economy	float	0.03	[m^3]	Required galley volume per passenger in economy class
galley_vol_per_pax_premium_economy	float	0.03	[m^3]	Required galley volume per passenger in premium economy class
galley_vol_per_pax_business	float	0.04	[m^3]	Required galley volume per passenger in business class
galley_vol_per_pax_first	float	0.15	[m^3]	Required galley volume per passenger in first class
cargo_type	str	containerized	[-]	Cargo type: 'containerized' or 'bulk'
uld_type	str	LD-3-45W	[-]	Type of ULD. Only required is cargo_type == 'containerized'
aisle_width	float	0.51	[m]	Aisle width
wall_thickness_factor	float	0.04	[-]	Fuselage wall thickness as a percentage of fuselage diameter
floor_thickness_factor	float	0.05	[-]	Fuselage floor thickness as a percentage of fuselage diameter
sitting_eye_height	float	0.89	[m]	Eye height while sitting, used for head clearance constraints
head_clearance_radius	float	0.25	[m]	Head clearance radius, used for head clearance constraints
sidewall_clearance	float	0.025	[m]	Clearance between seats and sidewall
container_clearance	float	0.1	[m]	Clearance of the containers. Only required when cargo_type == 'containerized'

5.3. Validation of the inside-out fuselage design approach

The inside-out sizing routine generates a fuselage surface based on several top level requirements. To assess the ability of the implemented inside-out sizing routine to produce accurate fuselage designs several validation cases have been evaluated. The selected validation cases provide a comparison of a number of conventional low-wing passenger aircraft, from one to two aisled aircraft and small to large.

During the conceptual design phase it is important to be able to produce an accurate estimate of the external dimensions of the fuselage, because the size of the fuselage has a large effect on the drag and weight of the aircraft. Therefore, the accuracy of the implemented cabin configuration methods is assessed by comparing the values for length, width and height of the fuselages generated by ParaFuse to the fuselages of the actual aircraft. The results of the evaluated validation cases are shown in Table 5.5. As can be seen from the table, the inside-out design routine is able model aircraft fuselages throughout the entire design range with only a small error margin. On average the error of the resulting fuselage width is 2% with respect to reference aircraft. The error of the resulting fuselage height is on average 1% with respect to the reference aircraft. The average error of the fuselage length is on average 2% with respect to the reference aircraft.

Table 5.5: Results of the inside-out design routine compared to actual aircraft

	Actual			ParaFuse			Difference wrt to actual aircraft		
	Width [m]	**Height [m]**	**Length [m]**	**Width [m]**	**Height [m]**	**Length [m]**	**Width [-]**	**Height [-]**	**Length [-]**
A320-200	3.95	4.14	37.57	4.10	4.10	37.72	0.04	0.010	0.004
A330-200	5.64	5.64	57.51	5.64	5.64	54.5	0.00	0.000	0.052
A350-900	5.96	6.09	65.26	6.01	6.01	64.4	0.01	0.013	0.013
E190	3.01	3.35	36.24	2.92	3.4	36.12	0.03	0.015	0.003
B777-200	6.2	6.2	62.94	6.21	6.21	62.27	0.00	0.002	0.011
B787-8	5.77	5.94	55.91	5.92	5.92	54.13	0.03	0.003	0.032

A complete overview of the validation cases is presented in Appendix E. In this appendix, not only the external dimensions are given, but also a comparision of the installed galleys, lavatories and cargo. First of all, it can be said that the inside-out routine is able to accurately generate aircraft fuselages for conventional low-wing passenger aircraft. Several other conclusions can be drawn from these validation cases as well. These conclusions are listed below.

- ParaFuse seems to underestimate the amount of containers that can be positioned in the belly of the aircraft. This can be explained by the fact that in the current implementation of ParaFuse cargo bays are only positioned in the constant middle section of the fuselage. However, especially for larger aircraft, there might be enough space in the nose and tail cone sections to position some additional cargo containers. Therefore, creating a capability to position containers in the nose and tail of the fuselage as well, can be an improvement for future versions of ParaFuse.
- The design rules with respect to the required galleys are based on a required galley volume per passenger. This galley volume is based on the passenger capacity of the aircraft. However, the installed galley volume of the reference aircraft cannot be found in the documents provided by the aircraft manufacturers. Only the amount of galleys in the cabin is available. As most galleys have different sizes, it is difficult to compare the galleys only based on quantity instead of volume to the amount of galleys that are generated by ParaFuse. It must be noted that the fuselage models in ParaFuse have been generated using typical values for the required galley volume, as can be found in aircraft design handbooks [40, 42].
- The length of the flight deck is currently estimated based on the number of pilot seats [43]. A more detailed cockpit design might provide more accurate values for the flight deck length. This might be a possible extension for future versions of ParaFuse as well.

6

Case studies

The previous chapters have introduced ParaFuse, a knowledge-based engineering application to support conceptual fuselage design. In the current chapter two case studies are presented to show the functionalities of the application. The two case studies both illustrate the capability of the application to be used as an inside-out and outside-in design tool.

The first case under consideration is the AAR cruiser, a passenger aircraft with air-to-air refuelling capabilities. The AAR cruiser has a passenger capacity similar to an Airbus A330 or Boeing 767 and was investigated by Delft University of Technology within the European Commission project RECREATE. This case study is described in Section 6.1. The second case study, described in Section 6.2, concerns with the design of a fuselage for a smaller regional turboprop aircraft with a capacity of 130 passengers.

6.1. AAR Cruiser

The faculty of Aerospace Engineering was involved in the European project RECREATE[1], in which research is performed into new passenger aircraft and operational concepts. One of the proposed concepts is a transport system based on air-to-air refuelling of passenger aircraft. Within this context, the AAR cruiser concept has been developed. The AAR cruiser is designed with the payload requirements of a twin-aisle medium range aircraft, such as the Airbus A330, but with a range typical for a smaller A320-type aircraft. Because of the refuelling capability, the AAR cruiser requires less fuel capacity. The lower weight of the aircraft and the lower fuel weight, allows for the reduction of the area of the lifting surfaces of the AAR cruiser.

The baseline design for the AAR cruiser aircraft was performed using the AC-X design framework. After the initial design, the cabin design of the AAR cruiser was performed separately using DARfuse. Thus, the external dimensions of the fuselage had already been established at the start of the cabin design. One of the limitation of DARfuse is that it can only be used to perform inside-out fuselage sizing and cabin configuration studies. While this approach is well suited to perform a clean sheet design of a fuselage, it does not allow for cabin configuration studies for a fuselage with predefined external dimensions. Within the RECREATE project it was required to generate several different cabin layouts for the AAR cruiser. To perform these kind of studies using DARfuse, several intermediate steps were required, as for the inside-out design case the external dimensions follow from the required interior components. To design the different cabins with DARfuse it was required to estimate the capacity and class distribution that would result in a fuselage with the required external dimensions. One can imagine that this process required a lot of iterations to find a fuselage with the correct external dimensions. To perform these cabin reconfiguration studies properly, a KBE application that allows for outside-in cabin configuration is required.

ParaFuse is able to perform both inside-out and outside-in cabin configuration studies. As a case study the AAR cruiser has been redesigned using ParaFuse based on the RECREATE AAR cruiser requirements as specified in the PhD dissertation of Li [47]. The requirements of the AAR cruiser are specified in Section 6.1.1.

[1] For more information the reader is referred to: `http://www.cruiser-feeder.eu`. Last accessed: 09-01-2017.

Subsequently, five outside-in cabin designs have been generated, which are discussed in Section 6.1.2. After that, an inside-out design of the fuselage has been performed based on the orignal RECREATE design case, which is presented in Section 6.1.3. Finally, conclusions regarding the case studies are drawn in Section 6.1.4.

6.1.1. Requirements

The AAR cruiser has been designed with payload requirements of a medium range twin aisle aircraft, such as the Airbus A330. Because of the refuelling capability the AAR cruiser has the design range of a short range aircraft such as the Airbus A320. A side view of the baseline design of the AAR cruiser is shown in Figure 6.1.

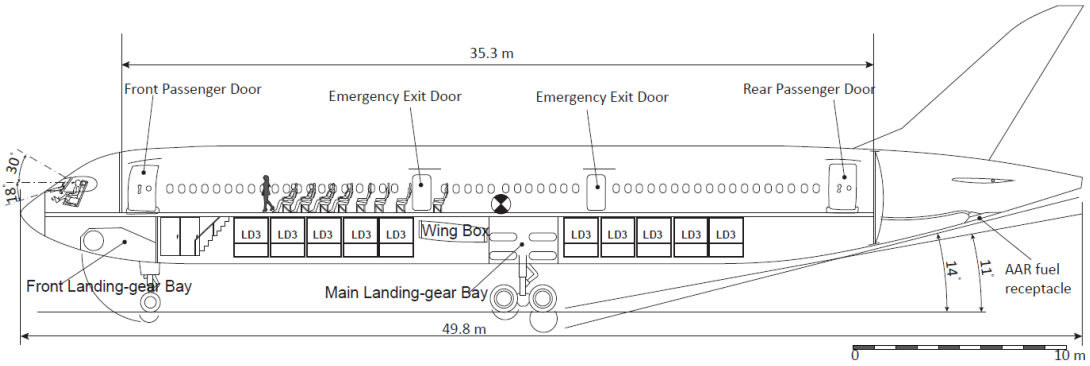


Figure 6.1: Baseline design of the AAR cruiser [47]

Just as an A330 the main fuselage diameter of the AAR cruiser is 5.64 m, whereas the length of the cruiser is 49.8 m. This allows for a main cabin cross section with 8 economy class seats abreast, and a cargo capacity of 20 LD-3 unit load devices. The required seat pitch and width for the economy, business and first class seats is shown in Table 6.1. As an additional requirement the cabin must provide room for 12 lavatories. Five different cabin designs for the AAR cruiser are presented in the PhD dissertation of Li and are shown in Figure 6.3 [47].

Table 6.1: Seat pitch and width requirements of the AAR cruiser

		Economy	**Business**	**First**
Width	[m]	0.46	0.55	0.65
Pitch	[m]	0.83	1.62	2.16
Seats abreast	[-]	8	6	4

6.1.2. Outside-in

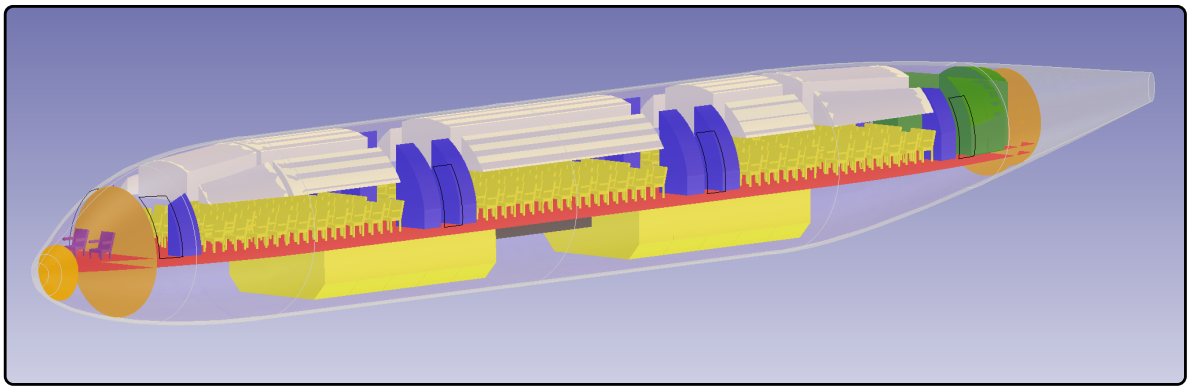
As a case study five different cabin layouts have been generated using ParaFuse: A high density case, two 2 class configurations and two 3 class configurations. The outer fuselage dimensions and cabin requirements specified in Section 6.1.1 have been used as a starting point for the design studies. The required percentage of business and first class seats is determined as percentage of the high density design case. The remaining cabin space is filled with economy class seats. An overview of the design cases is given in Table 6.2.

As mentioned before, the outside-in cabin configuration method generates a cabin within the available space of a predefined outer fuselage surface. An overview of the fuselage model visualized in the ParaPy GUI can be seen in Figure 6.2. ParaFuse positions the lavatories and galleys around the required emergency exits in the cabin. The cabin designs presented by Li allow for four lavatories on one row. Therefore, the cabin layout is not exactly similar to the cabin configurations shown in Figure 6.3. Furthermore, the cabin designs in Figure 6.3 do not always take into account additional clearance around the emergency exits. These facts explain why

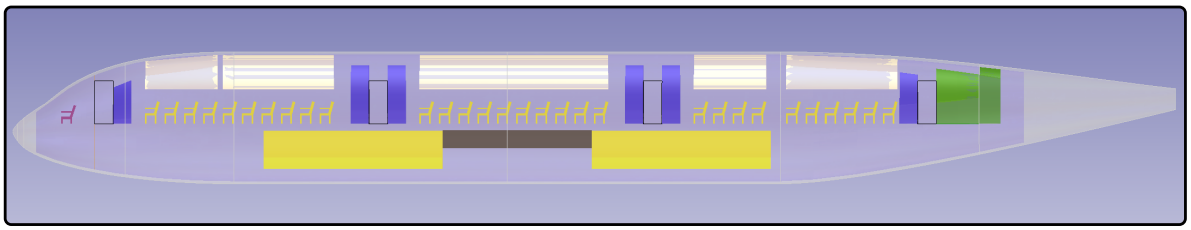
Table 6.2: Outside-in design cases

	Case	Business	First
1	1 class	0.0	0.00
2	2 class	0.13	0.00
3	2 class	0.20	0.00
4	3 class	0.10	0.03
5	3 class	0.12	0.05

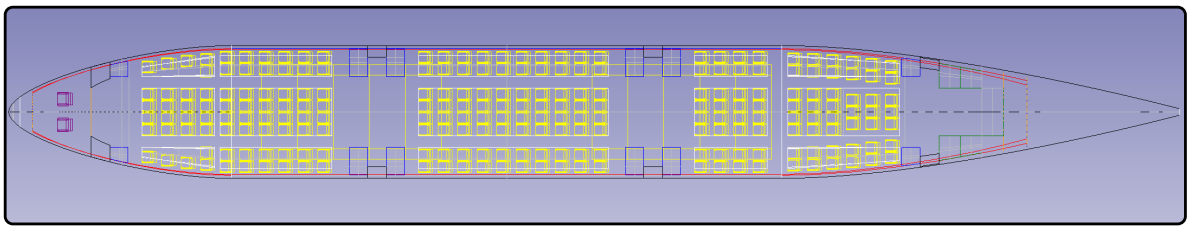
the capacity of the cabin generated using ParaFuse is 231 passengers in single class configuration, whereas the single class cabin configuration presented by Li [47] allows for 258 economy class passengers.



(a) Isometric view



(b) Side view



(c) Top view

Figure 6.2: Outside-in design for the AAR cruiser with a single class cabin configuration. Lavatories indicated in blue, galleys indicated in green. Total capacity = 231 pax

The five cases in Table 6.2 have been generated using ParaFuse. The percentage of first and business class seats has been derived from the cabin designs shown in Figure 6.3. These percentages are calculated with respect to the capacity of the high density case. An overview of the cabin layout for each design case is shown in Figure 6.4[2]. The resulting capacity for each case is given in Table 6.3. As can be seen from Figure 6.2 a galley area is located in the tail cone of the aircraft, depicted in green. Furthermore, all lavatories, the blocks that are colored blue, are positioned around the emergency exits of the main cabin.

[2]The cabin layouts have been exported as .STEP files and imported in CATIA V5 to create technical drawings.

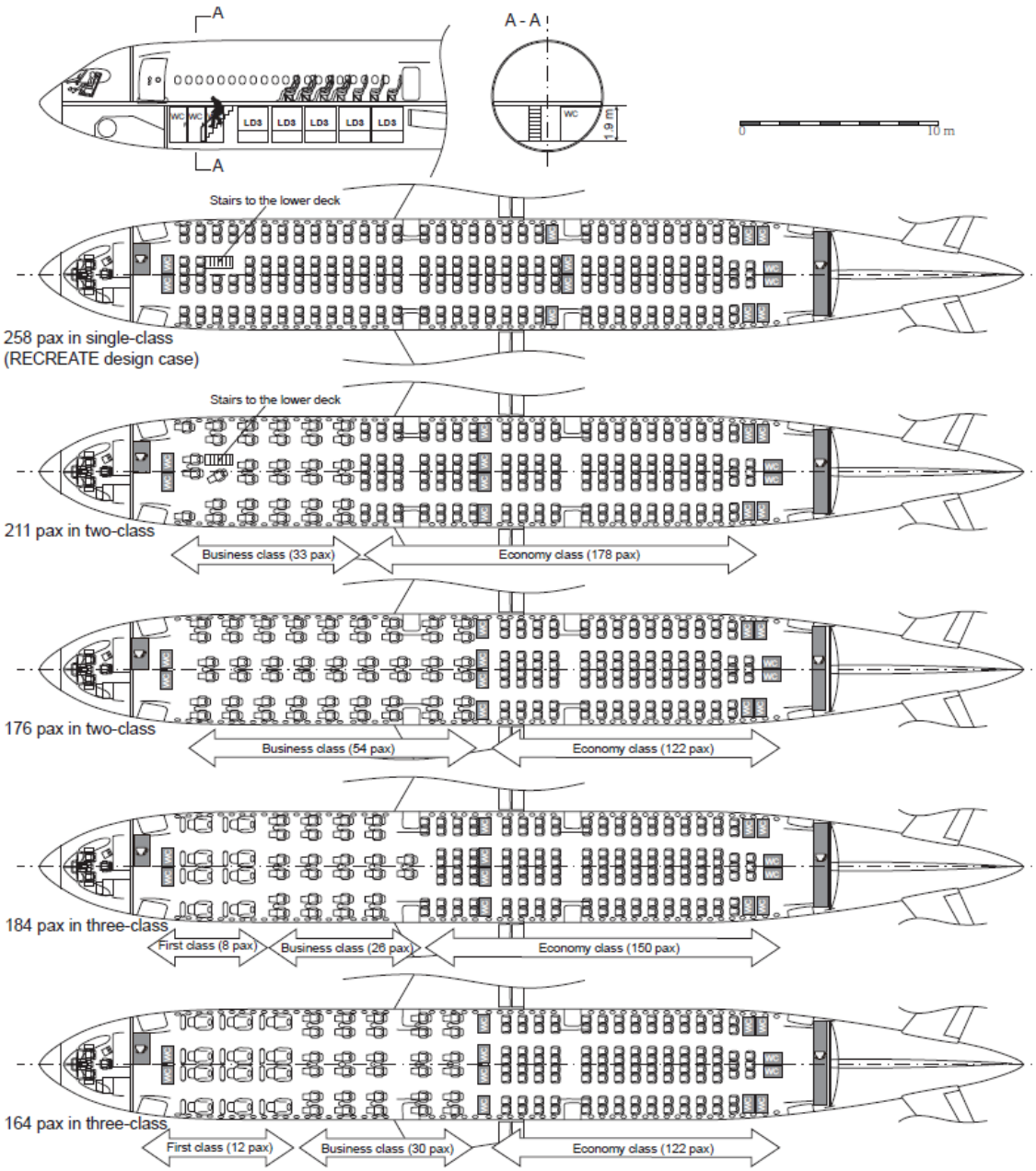
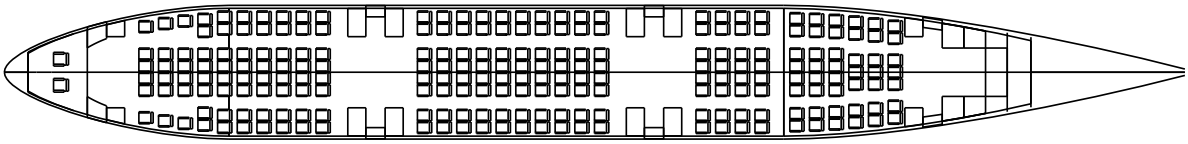


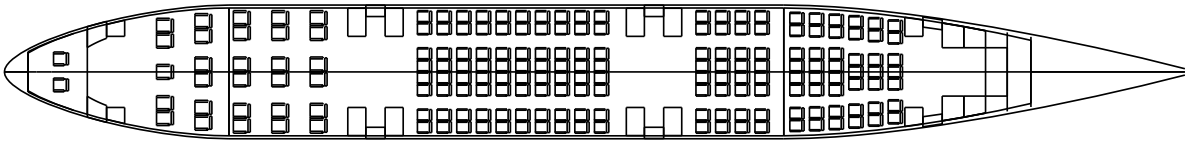
Figure 6.3: Possible cabin layouts of the AAR cruiser [47]

Table 6.3: Resulting capacity for the outside-in design cases and difference with respect to AAR cruiser design

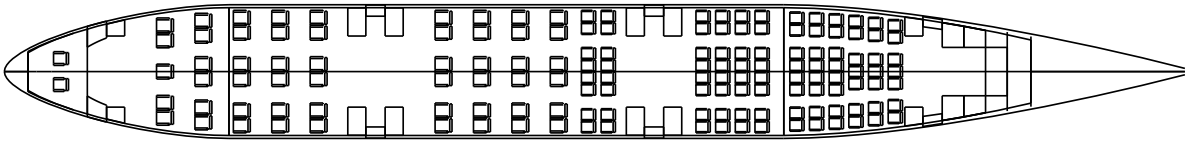
	Case	Economy	Business	First	Total
1	1 class	231 (-27)	0 (-)	0 (-)	231 (-27)
2	2 class	157 (-21)	29 (-4)	0 (-)	186 (-25)
3	2 class	93 (-29)	53 (-1)	0 (-)	146 (-30)
4	3 class	125 (-25)	24 (-2)	8 (-)	157 (-27)
5	3 class	93 (-29)	30 (-)	12 (-)	135 (-29)



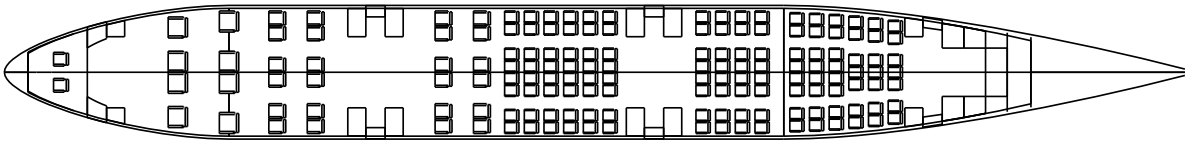
(a) Economy class: 231 pax



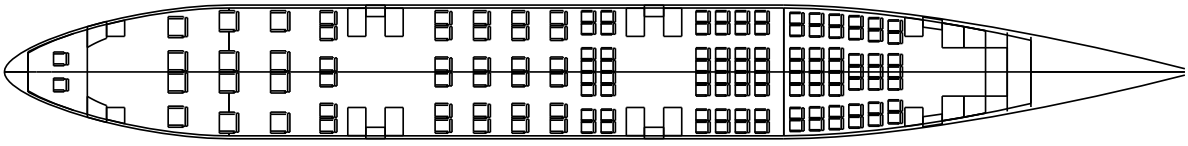
(b) Economy class: 157 pax, Business class: 29 pax



(c) Economy class: 93 pax, Business class: 53 pax



(d) Economy class: 125 pax, Business class: 24 pax, First class: 8 pax



(e) Economy class: 93 pax, Business clas: 30 pax, First class: 12 pax

Figure 6.4: Five possible cabin configurations for the AAR cruiser

In the introduction of this thesis it was explained that one of the requirements for a geometry in the loop optimization framework is speed. A MMG should be able to rapidly generate a geometric model to allow for a large number of design evaluations. The previous fuselage model, DARfuse, required 2-3 minutes to generate a complete model of the fuselage, which is not satisfactory if the model is to be used in an optimization framework. The required time to generate the surface models in the ParaPy GUI has been measured for each of the five cases. The results[3] are shown in Table 6.4. As can be seen from the table, the cabin configuration takes approximately 15 seconds, whereas the generation of outer fuselage surface takes only 1-2 seconds. The generation of the outer surface is much quicker, as no additional calculations are required because the external dimensions are given as input for the outside-in design case. It must be noted that ParaFuse can also be operated in batch mode. When operated in batch mode no visualization is required and, therefore, ParaFuse runs much faster. For example, ParaFuse is able to write the main fuselage characteristics[4], which requires the cabin configuration process to performed, to a text file in less than 10 seconds.

Table 6.4: Time (in seconds) required to generate the ParaFuse fuselage models

Variant	Outer surface	Interior components
1	2.0	15.0
2	1.5	14.0
3	1.0	16.0
4	1.0	16.5
5	1.0	14.0

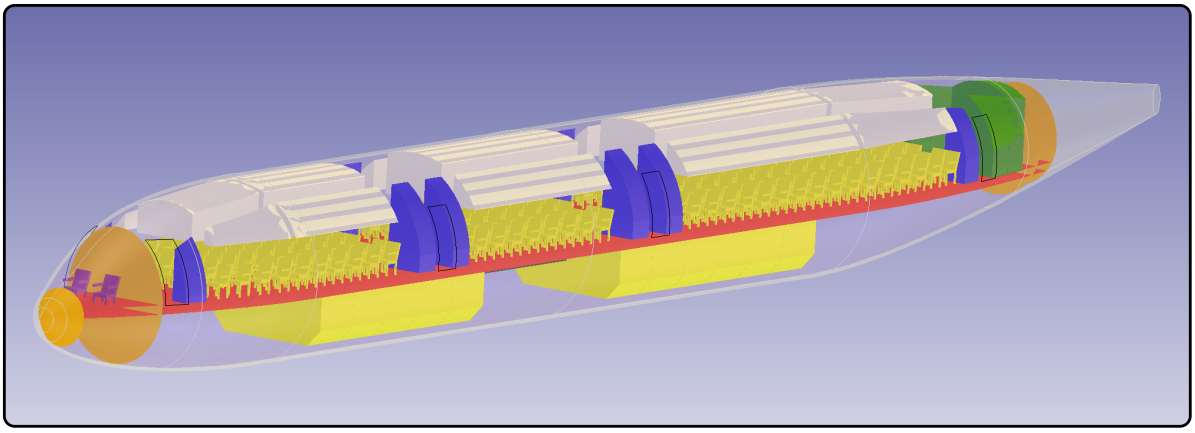
6.1.3. Inside-out

To get an impression of the required fuselage dimensions to allow for a capacity of 258 passengers, using the ParaFuse cabin configurator, an inside-out design study of the AAR cruiser has been performed. Figure 6.5 gives an overview of the resulting fuselage generated with the inside-out design for a passenger capacity of 258 in a single class configuration. The resulting main fuselage cross section is the same as for the outside-in design case and has a diameter of 5.64 meters, with 8 seats abreast. The resulting fuselage length equals 52.35 meters. It must be noted that the resulting design allows for 5 additional passengers if the option is selected that all cabin rows should be filled completely. Because of the increase in fuselage length an additional pair of LD-3 containers can be positioned in the aircraft as well. Similar to the outside-in cases, 12 lavatories are positioned in the cabin of the aircraft.

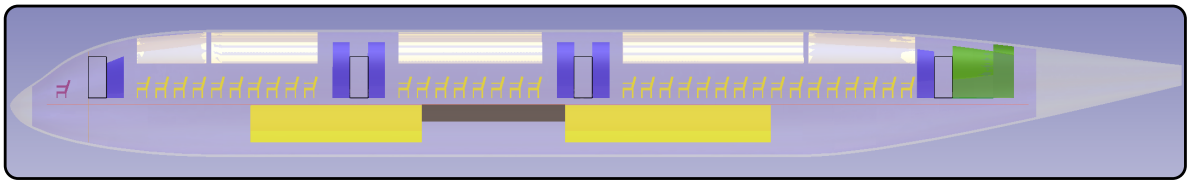
The time required to generate the inside-out fuselage model has been measured as well. The total time it takes to generate the model is almost equal to the outside-in cases, but the required time for the individual components is different. This difference is caused by the fact that ParaFuse needs to determine the location of the interior components to derive the external dimensions of the fuselage. Therefore, the generation of the fuselage outer surface takes approximately 7 seconds. Because the cabin configuration has already been performed to generate the outer fuselage surface, the interior components only need to visualized in the ParaPy GUI. Because their size and positions have already been stored in the memory of the computer, the generation of the interior components reduces to approximately 9 seconds.

[3] The results have been generated on a MacBook pro with a 2.7 GHz Intel Core i5 processor. ParaPy runs on Windows 10 on a virtual machine, which has been allocated 4 GB of RAM.

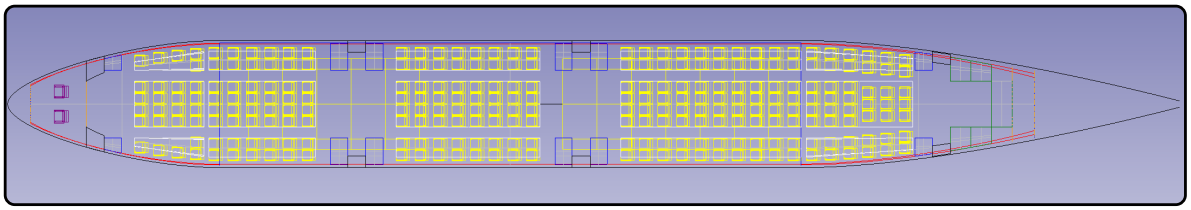
[4] These characteristics include: cargo volume, number of passengers per class, external dimensions, number of lavatories, galley volume and information regarding the emergency exits.



(a) Isometric view



(b) Side view



(c) Top view

Figure 6.5: Inside-out design for the AAR cruiser with 258 passengers in a single class cabin configuration. Lavatories indicated in blue, galleys indicated in green

6.1.4. Conclusion

As part of the RECREATE project the faculty of Aerospace Engineering has developed the AAR cruiser concept, a mid range passenger aircraft that allows for air-to-air refuelling. Within this research project the sizing of the cabin was not included into the AC-X framework, but performed separately using DARfuse. DARfuse does not allow for outside-in cabin configuration based on fixed external fuselage dimensions. As a result, the fuselages needed to be generated with the inside-out design routine using trial and error, until a resulting fuselage was found with the correct dimensions.

ParaFuse is able to generate fuselage models using both the inside-out and outside-in approach. As a case study, the AAR cruiser fuselage has been reconstructed using ParaFuse. Furthermore, five different cabin configurations have been generated. ParaFuse is able to generate the fuselage models within 20 seconds. Because of different positioning rules the resulting cabins are not entirely similar to cabins generated by Li [47]. Furthermore, no additional legroom clearance is taken into account around the emergency exits of the cabins presented in [47]. Therefore, the capacity of the cabins generated with ParaFuse are lower than the RECREATE design cases. However, this case study can be used as a demonstrator of the outside-in capability of ParaFuse. Additionally, an inside-out design of the AAR fuselage has been generated for a capacity of 258 passengers. The resulting fuselage has an overall length of 52.35 meters as compared to 49.8 meters for the RECREATE design case.

6.2. Regional turboprop aircraft

For another project, the chair of Flight Performance and Propulsion is involved in the design of a regional turboprop aircraft. The requirements for this aircraft specify a 110-130 passenger capacity in high density configuration. A baseline fuselage design with a length of 38.04 m and a diameter of 3.54 m has been provided. As a case study the fuselage of this aircraft has been generated using ParaFuse.

First of all, the outside-in method has been used to generate a model of the fuselage using the dimensions specified above. An initial cabin design with a standard seat pitch for regional aircraft of 30 inches resulted in a cabin with a capacity of 149 passengers for the given fuselage dimensions. Therefore, the seat pitch for the outside-in case has been increased to reduce the passenger capacity to 129. The outside-in case is discussed in Section 6.2.1. In a subsequent step, the fuselage of the aircraft has been redesigned using the inside-out design method. An overview of the redesigned fuselage using the inside-out routine is presented in Section 6.2.2.

6.2.1. Outside-in

For the outside-in design case, the dimensions specified in the introudction have been used to generate the a model of the fuselage using ParaFuse. Additional requirements for the regional aircraft specified a seat width of 0.46 meters and an aisle width of 0.48 meters. These parameters, in conjunction with the fuselage diameter, allow for a 5 seat abreast main cabin section.

As was mentioned in the introduction, an initial cabin configuration run using a seat pitch of 30 inches (0.762 m) resulted in an overestimation of the amount of cabin seats. In a second iteration the seat pitch was increased to 35 inches (0.89 m) to generate a fuselage with 129 seats. A summary of the main input parameters required to generate the fuselage model is shown in Table 6.5. A standard economy class seat has been used in the cabin configuration process with a width of 0.46 m and a pitch of 0.89 m.

The following figures give an impression of the generated fuselage model, as visualized in the ParaPy GUI. An impression of the main cross section is shown in Figure 6.6. Figure 6.7a shows an isometric view of the fuselage model. An additional functionality of ParaFuse is the ability to export the generated fuselage models in the CPACS file format. The fuselage model exported in CPACS and visualized in the TiGL-viewer is shown in Figure 6.7b. CPACS, a XML based data format, is proposed by the German Aerospace Center (DLR) as a central data format in aircraft design. CPACS allows for collaboration in aircraft design by decreasing the required number of interfaces when coupling different analysis tools. Therefore, a `CPACSWriter` class has been created to translate the outer fuselage to surface to a CPACS representation. A side view of the fuselage can be seen in Figure 6.8a. Finally a top view, including an overview of the seating arrangement is shown in Figure

Table 6.5: Main input parameters for the outside-in cabin configuration method

Parameter	Type	Value	Unit
Fuselage length	float	38.04	[m]
Fuselage diameter	float	3.54	[m]
Cross section center	float	0.7	[m]
Fineness ratio nose cone	float	1.19	[-]
Fineness ratio tail cone	float	3.2	[-]
Percentage economy	float	1.0	[-]
Pax per lavatory economy	int	50	[-]
Galley vol. per passenger economy	float	0.03	[m^3]
Cargo type	str	bulk	[-]
Aisle width	float	0.48	[m]
Seat width	float	0.46	[m]
Seat pitch	float	0.89	[m]
Wall thickness factor	float	0.04	[-]
Floor thickness factor	float	0.05	[-]
Sitting eye height	float	0.89	[m]
Head clearance radius	float	0.2	[m]
Sidewall clearance	float	0.025	[m]
Container clearance	float	0.05	[m]

6.8b. The main parameters of the resulting fuselage model are given in Table 6.6. The generation of the fuselage model, both the cabin and outer surface, using the ParaPy GUI takes approximately 12 seconds.

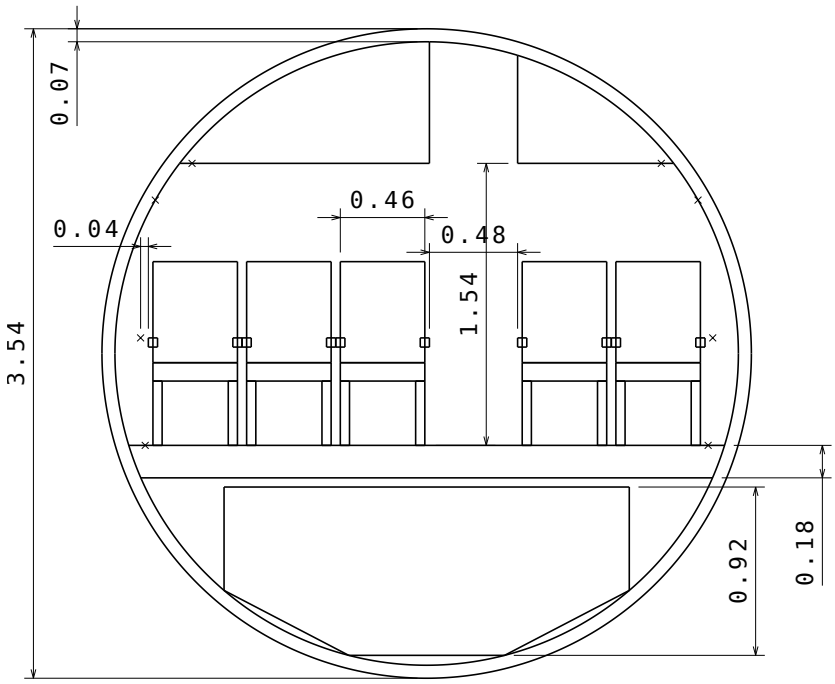
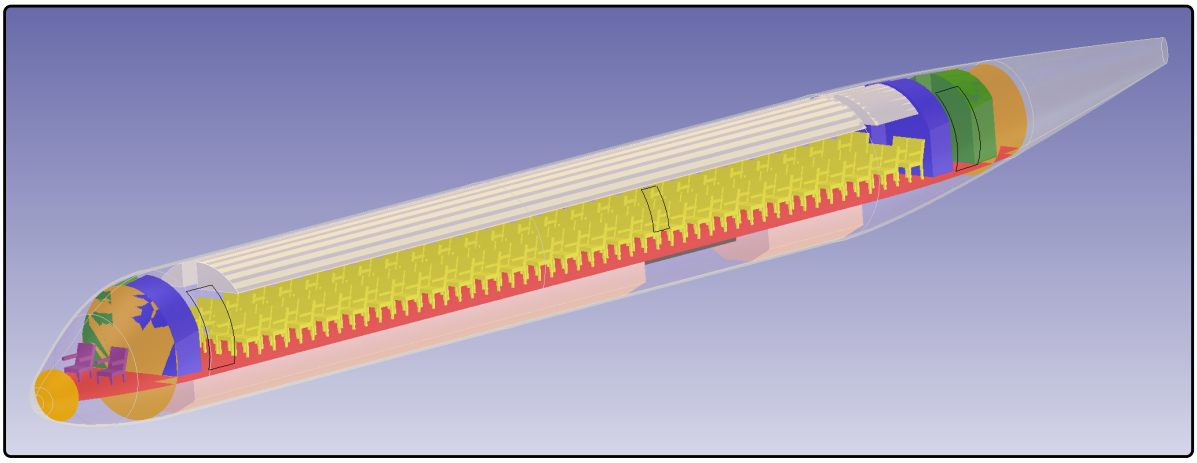
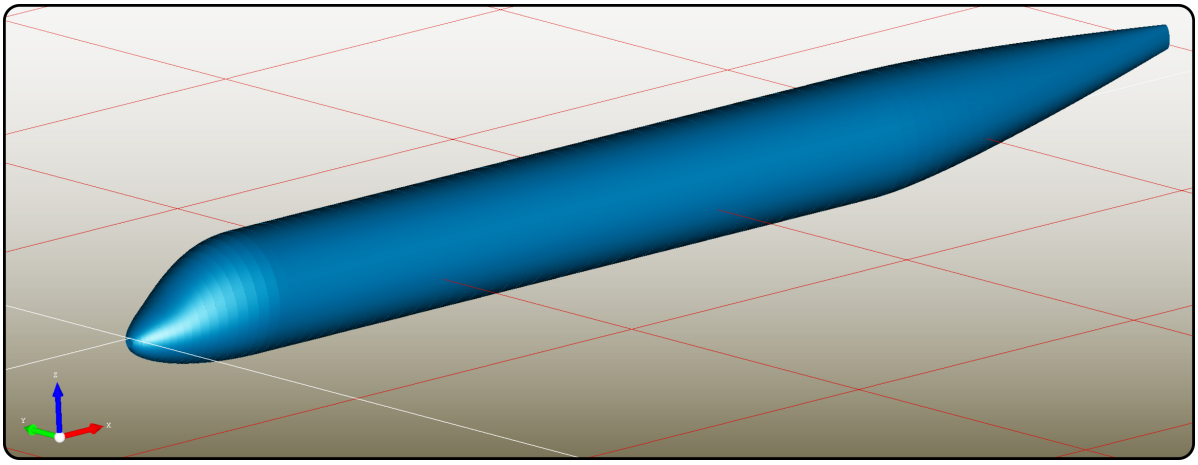


Figure 6.6: Main cabin cross section of the aircraft

The ParaFuse model has been compared with the cabin configurator implemented in the FPP Initiator and a cabin configurator developed by the University of Naples. Figure 6.9 shows the models generated by the three applications. Although the three models have equal length and width, ParaFuse is the only application that generates the cabin elements separately, whereas the other two tools only reserve some extra space to account for the required cabin elements. Therefore, the model generated by ParaFuse is expected to be more accurate. The length of the cabin in the tail cone of the baseline aircraft is not known, therefore, a statistical

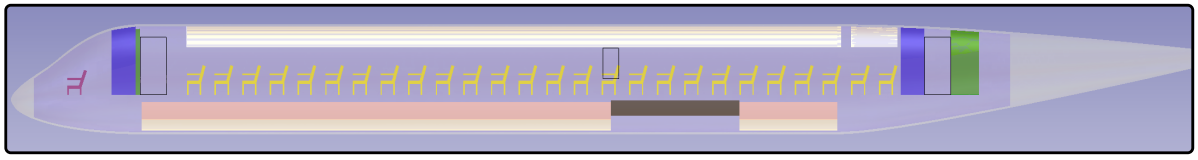


(a) ParaPy

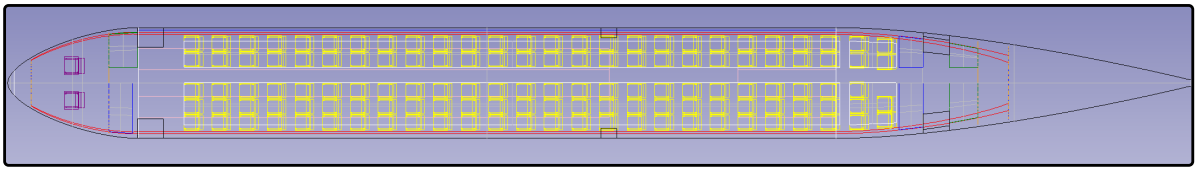


(b) CPACS

Figure 6.7: Isometric view of fuselage, generated using the outside-in routine



(a) Side view



(b) Top view

Figure 6.8: Fuselage of the aircraft, generated using the outside-in routine. Lavatories indicated in blue, galleys indicated in green

relationship has been used by ParaFuse, as described in Chapter 4. A possible explanation for the increased capacity of the ParaFuse model could be that the length of the cabin in the tail cone of the baseline aircraft is much shorter than is predicted by the statistical relationship.

Table 6.6: Design summary for the aircraft generated using ParaFuse

Parameter	Value	Unit
Design approach	Outside-in	[-]
Cross-sectional shape	Circle	[-]
Length	38.04	[m]
Diameter	3.54	[m]
Fineness ratio	10.76	[-]
Number of classes	1	[-]
Number of economy class seats	129	[-]
Required galley volume	3.87	[m^3]
Installed galley volume	4.74	[m^3]
Required number of lavatories	3	[-]
Installed number of lavatories	3	[-]
Cargo type	Bulk	[-]
Installed bulk cargo bays	2	[-]
Installed bulk volume	24.49	[m^3]
Exits	Type 1: 4, Type 3: 2	[-]

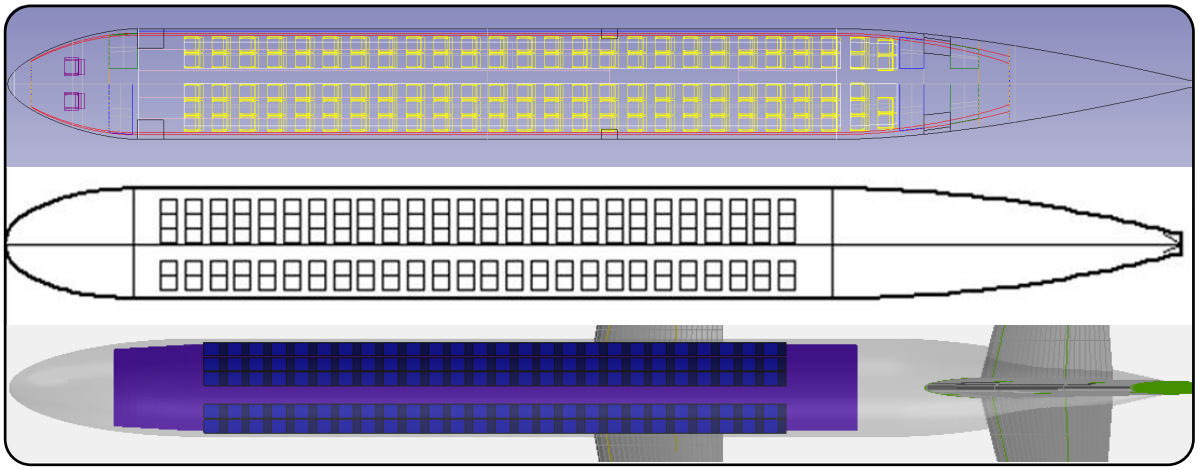


Figure 6.9: Fuselage comparison; ParaFuse (top), University of Naples (middle), Initiator (bottom)

6.2.2. Inside-out

The design project of the regional turboprop aircraft is still in the early design phase. The design choices made in this stage of the project may have a large effect in a later phase. The results of the outside-in design case give an indication that the fuselage might be too large for the required payload. Therefore, the fuselage has been redesigned using the inside-out design approach for 129 passengers with a seat pitch of 30 inches. An overview of the input parameters for the inside-out design routine are given in Table 6.7. With these parameters a redesigned fuselage model has been generated. An overview of the main features of the resulting fuselage is given in Table 6.8. The generation of the cabin and the outer surface of the fuselage require approximately 4 seconds each.

The main fuselage cross section, shown in Figure 6.10, that has been generated with the inside-out sizing method is very similar to the outside-in cross section, with fixed dimensions. It appears that a slight reduction in fuselage diameter can be achieved, as can be seen from the figure. However, the length of the redesigned fuselage is 34.67 meters. This is a reduction of almost 9% compared to the baseline fuselage design. An overview of the resulting fuselage is shown in Figures 6.10 and 6.11.

Table 6.7: Main input parameters for the inside-out redesign

Parameter	**Value**	**Unit**
Number of passengers	129	[-]
Cross section shape	circle	[-]
Number of seats abreast	5	[-]
Fineness ratio nose cone	1.19	[-]
Fineness ratio tail cone	3.2	[-]
Percentage economy	1.0	[-]
Pax per lavatory economy	50	[-]
Galley volume per passenger economy	0.03	[m^3]
Cargo type	bulk	[-]
Minimum bulk bay height	0.925	[m]
Aisle width	0.48	[m]
Seat width	0.46	[m]
Seat pitch	0.762	[m]
Wall thickness factor	0.04	[-]
Floor thickness factor	0.05	[-]
Sitting eye height	0.89	[m]
Head clearance radius	0.2	[m]
Sidewall clearance	0.025	[m]
Container clearance	0.05	[m]

Table 6.8: Design summary for the redesigned fuselage generated using ParaFuse

Parameter	**Value**	**Unit**
Design approach	Inside-out	[-]
Cross-sectional shape	Circle	[-]
Length	34.67	[m]
Diameter	3.51	[m]
Fineness ratio	9.87	[-]
Number of classes	1	[-]
Number of economy class seats	129	[-]
Required galley volume	3.87	[m^3]
Installed galley volume	4.74	[m^3]
Required number of lavatories	3	[-]
Installed number of lavatories	3	[-]
Cargo type	Bulk	[-]
Installed bulk cargo bays	2	[-]
Installed bulk volume	27.27	[m^3]
Exits	Type 1: 4, Type 3: 2	[-]

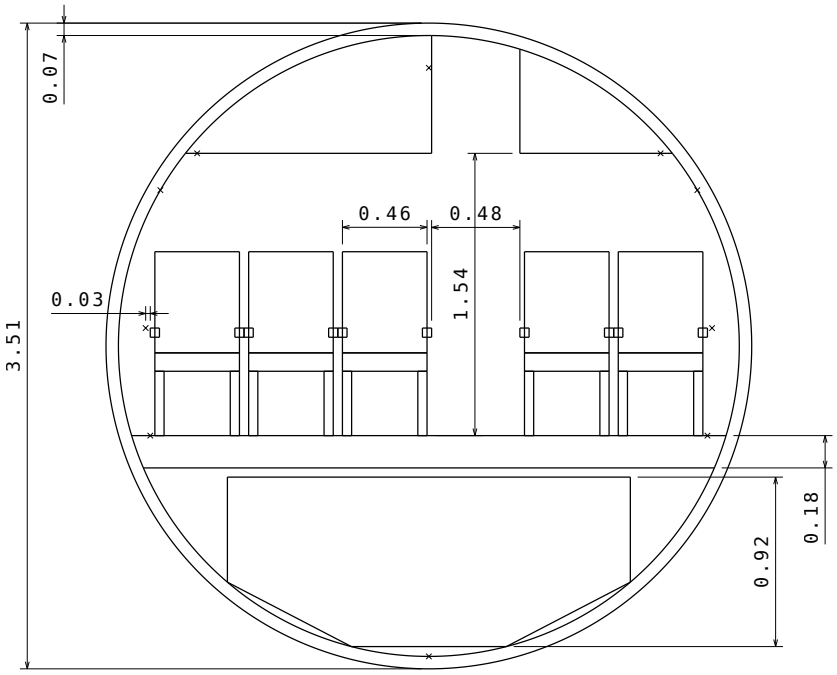
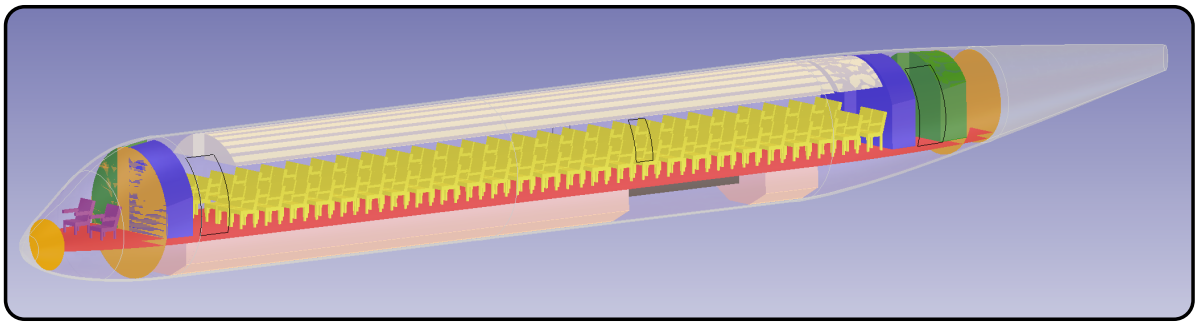
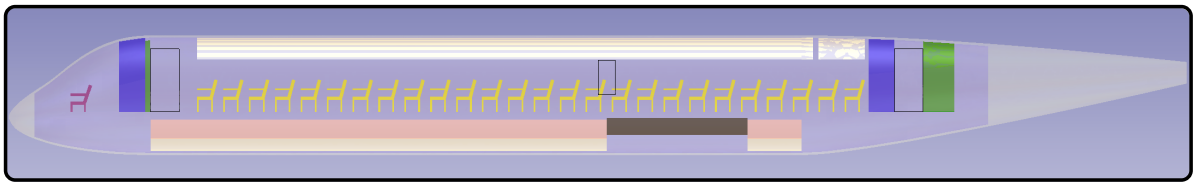


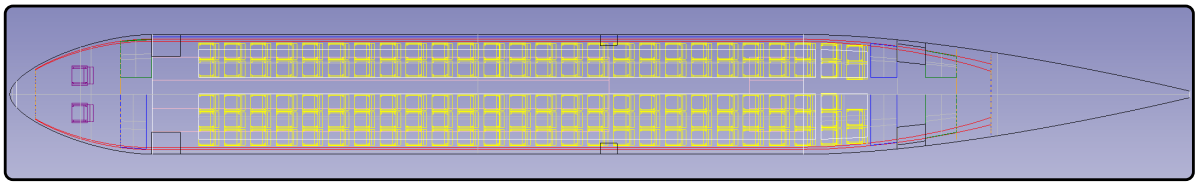
Figure 6.10: Main cabin cross section of redesigned fuselage



(a) Isometric view



(b) Side view



(c) Top view

Figure 6.11: Redesigned fuselage for the aircraft using the inside-out method. Lavatories indicated in blue, galleys indicated in green

6.2.3. Conclusion

The fuselage of the regional turboprop aircraft has been considered in this case study. As a first step, the initial fuselage and cabin has been reconstructed using the outside-in cabin configuration method. The fuselage and cabin resulting from the outside-in design study provided evidence that the fuselage dimensions might have been overestimated for the required payload. Therefore, a redesign using an inside-out design routine for a comparable fuselage with 129 passengers has been performed. This process resulted in a fuselage with an overall length of 34.67 m and a diameter of 3.51 m. Compared to the initial design, the inside-out design case results in a reduction of almost 9% in fuselage length. The reduction in fuselage length will have an effect on the weight of the aircraft, which on its turn affects the required area for the lifting surfaces. Furthermore, a reduction in wetted area of the fuselage will result in reduced drag due to lower skin friction. In conclusion it can be said that a larger than necessary fuselage can have a significant influence on the performance of the aircraft. However, at this moment it is not known if some interior components of the cabin require additional space. With this information it would be possible to draw more definitive conclusions.

7

Conclusions and recommendations

The previous chapters have described the research that has been performed into a knowledge-based engineering application to support the conceptual design of aircraft fuselages. Several conclusions can be drawn with respect to the performed research. These conclusions are presented in Section 7.1. Furthermore, recommendations for future research and development are presented in Section 7.2.

7.1. Conclusions

In the introductory chapter of this thesis, the following goals were established:

1. Develop a parametric conventional fuselage model using the KBE system ParaPy
2. Implement an inside-out design method to allow for fuselage sizing studies based on top level requirements
3. Implement an outside-in method to allow for fuselage (re)configuration studies using predefined fuselage dimensions

The KBE application ParaFuse was developed to fulfill the thesis goals stated above. ParaFuse is a KBE application that is able to generate product models for fuselages of conventional, low-wing, passenger aircraft. ParaFuse is a step towards the development of a next generation aircraft multi-model generator based on the KBE framework ParaPy. The aim of the aircraft MMG is to support multidisciplinary design analysis and optimization by reducing repetitive and non-creative design activities.

The following steps have been taken to develop ParaFuse. First of all, a study was performed to establish common shapes of existing conventional aircraft fuselages. These shapes were used as a starting point for the parametric definition of the fuselage high-level primitive. The implemented parametric fuselage model allows for the definition of a main cross section and longitudinal guide curves to generate the outer surface of the fuselage. The main cross section is defined as the cross section of the constant middle section of the fuselage. The ability to specify four different cross-sectional shapes allows for high flexibility in the type of fuselages that can be modelled. The longitudinal guide curves are constructed separately for the nose cone, middle section and tail cone of the fuselage. The implementation of Bézier curves to model the bottom and side guide curves of the nose cone section has improved the ability to model realistic aircraft nose cone shapes. The separate guide curves are merged to form the complete set of longitudinal guide curves. As an intermediate step in the process to generate the outer fuselage surface, the main cross section is scaled at several locations along these longitudinal guide curves. The location of the scaled cross sections is determined automatically by means of a class that is able to calculate interpolation locations based on the curvature of the longitudinal guide curves. Additionally, a class has been created that removes redundant interpolation locations. Following this process, the fuselage surface is created by lofting through the scaled cross sections. The resulting fuselage surface has flat front and aft faces. Therefore, a separate end cap for the nose section is required to create an aerodynamically sound outer surface. The parameterization of the nose cone end cap is fast and stable and, thus, seen as an improvement with respect to the parameterization of the nose end cap

in DARfuse. In his thesis, Brouwers states that the generation of the nose end cap is slow and unstable [20]. Additionally, a parametric wing-body fairing has been implemented in the ParaFuse fuselage model. The parameterization of the wing-body fairing is adopted from Song and Lv [39]. They also present a study where the parametric wing-body fairing has been applied to successfully reduce aerodynamic drag. To draw definitive conclusions regarding the parameterization of the wing-body fairing a similar study should be performed using the ParaFuse model.

To support the designer, two fuselage sizing and cabin configuration methods have been implemented in ParaFuse. First of all, an inside-out fuselage sizing method has been implemented to generate a fuselage model including interior components based on top level requirements, such as passenger capacity and required cargo type. The fuselage models that are generated by ParaFuse are compliant with EASA CS 25 certification specifications. The cabins generated by ParaFuse allow for the specification of multiple seat classes. The required number of emergency exits is determined by transforming the required passenger capacity and class distribution to an equivalent number of passenger in high density configuration. The emergency exits are distributed uniformly over the length of the fuselage. Furthermore, galleys and lavatories are generated according to requirements specified by the user. The position of the wing box is derived as a function of fuselage length or directly read from the user input. Finally, either containerized or bulk cargo is positioned in the available space in the belly of the fuselage. The second design method that has been implemented in ParaFuse follows the outside-in approach. The outside-in cabin configuration method first generates the outer fuselage surface based on fixed external dimensions specified by the user. Subsequently, the interior volume of the generated fuselage is filled with the required interior components. Because of the fact that the total capacity is not known a priori, an iterative process is required to determine the final cabin layout. The scope of ParaFuse is limited to passenger aircraft certified under CS 25 certification specifications. In order to improve the fuselage sizing and cabin configuration methods, the inside-out and outside-in methods should be extended to be able to model different types of aircraft, such as smaller businness jets, high-wing aircraft, double deck aircraft and freighter aircraft.

The inside-out fuselage sizing method has been validated by replicating a number of existing aircraft fuselages using ParaFuse. To assess the performance of the inside-out fuselage sizing method, the resulting fuselage dimensions were compared to the actual aircraft. On average the external dimensions of the fuselages generated by ParaFuse differ approximately 2% with respect to the dimensions of the actual aircraft fuselages. From these validation cases it can be concluded that the implemented inside-out fuselage sizing method is able to accurately size fuselages of conventional, low-wing, passenger aircraft. The outside-in cabin configuration method uses similar design rules as the inside-out fuselage sizing method. The outside-in cabin configuration method could be improved by adding a capability that allows the user to specify the exact location of the emergency exits and cabin floor, so that ParaFuse can be used to reconfigure cabins where all the other parameters have already been determined.

To demonstrate the functionalities of ParaFuse two case studies have been presented in this thesis. The case studies are based on projects the department of Flight Performance and Propulsion is involved in. The case study for the AAR cruiser showed the need for an outside-in fuselage design method to perform cabin configuration studies. This capability was not available in the previous generation MMG.

ParaFuse can be used to rapidly generate the outer fuselage surface and cabin interior of an aircraft. The process of surface generation and cabin configuration is performed in seconds. This creates opportunities to evaluate a large number of designs during the conceptual design phase. The reduction in time is a significant improvement with respect to the previous generation fuselage HLP, which took approximately 2-3 minutes to generate a fuselage model. Furthermore, the short evaluation time, and the ability to operate in batch mode as shown in Appendix F, allows ParaFuse to be used within an optimization framework, which is the ultimate goal of the development of the MMG. ParaFuse is able to export the generated fuselage model to several standard data exchange formats such as .STEP, .IGES and .STL. Additionally, the resulting fuselage surface can be exported in the CPACS data format. This functionality should allow for easy coupling with analysis tools that are able to read aircraft geometries from CPACS files.

During the development of any software application it is very important that the source code is well documented. The documentation serves as a reminder of the programming choices that were made. Furthermore, proper documentation is required if the application is to be used and, possibly, extended by others. Therefore, an extensive documentation has been created with the aid of the Python documentation generator Sphinx. The Sphinx documentation is html-based and can, therefore, easily be viewed in a web browser and hosted

on the internet (or intranet). More information on the documentation of ParaFuse can be found in Appendix G.

7.2. Recommendations

The scope of the research performed as part of this thesis was to develop a KBE application to support the conceptual design of conventional aircraft fuselages. In this section recommendations for future research are presented. These recommendations can be used to improve the application.

ParaFuse is proposed as a step towards a next generation multi-model generator. To be able to model complete aircraft, ParaFuse should be integrated with a wing HLP into an aircraft MMG. The next step would be to use this MMG in a Design and Engineering Engine to perform multidisciplinary design optimization of aircraft configurations. As a stand alone application ParaFuse can already have its value. Apart from the presented opportunities to rapidly perform conceptual fuselage sizing based on the interior requirements and cabin configuration studies for fuselages with fixed external dimensions, ParaFuse can be used for many other types of research. A few examples are presented in this paragraph. ParaPy has an integrated meshing capability that allows for the automation of preprocessing activities required for aerodynamic and structural analysis. A study that uses this capability to perform aerodynamic analysis with the panel code VSAERO is presented by Wei [48]. It is recommended that the VSAERO interface that has been developed as part of the work of Wei is combined with ParaFuse to demonstrate the ability to perform aerodynamic analysis using fuselages generated by ParaFuse. Furthermore, ParaFuse could be used to produce the surface models that are required to perform studies into the design of the electrical wiring and interconnection system (EWIS), such as presented by Zhu [22]. Another interesting research topic would be to couple ParaFuse to the FPP Initiator [17, 49]. It would be interesting to explore the ability of ParaFuse to support the fuselage geometry estimation methods of the Initiator. In this way, ParaFuse would serve one of the goals of KBE to increase the design knowledge very early in the aircraft design process.

In the remainder of this section recommendations to improve ParaFuse are presented. These recommendations are separated in two groups:

1. Parameterization approach:

    - Implement a capability to perform unlofting of generic CAD files. Such an approach would allow one to create a parametric model from a 'fixed' surface that is read from a CAD file. In order to achieve this, a dedicated unlofting class should be created. An example of an implementation of such an unlofting class is presented by Brouwers [20].

    - The scope of ParaFuse is limited to modeling conventional aircraft fuselages. It is recommended that the modeling capability of ParaFuse is extended to also be able to model unconventional aircraft cabins. Several assumptions limit the ability of ParaFuse to model unconventional configurations. First of all, the middle fuselage section is assumed to be straight. This assumption presents limitations to the modeling flexibility of ParaFuse. If a designer would like to model for example a lifting fuselage or fuselages without a straight middle section such as the fuselage of the Piaggio P180 Avanti business jet, the longitudinal guide curves should allow for continuous curvature. In Chapter 3 it was proposed to replace the straight line segments by a Bézier curve or B-spline with some additional control points to allow for curved guide curves. Additionally, a capability to generate oval fuselage cross sections should be implemented to allow more flexibility in the modeling capabilities of ParaFuse. The parameterization of the oval fuselage is described in the research of Hoogreef [50]. Furthermore, the implementation of the oval fuselage cross section should also allow for easy cross over of models generated by the Initiator into ParaFuse as this parameterization approach is also implemented in the Initiator.

    - Currently, the windshield of the aircraft is only taken into account by the top guide curve of the fuselage. This parameterization is not able to fully capture the shape of the windshield. To be able to create more accurate nose cone surfaces a separate parameterization approach should be taken to model the entire windshield. This approach could start from an ideal visibility map from the cockpit, as presented in Roskam [41] and Torenbeek [24]. Subsequently, a surface could be generated between the intersection points of these visibility vectors on the fuselage surface, to be

able to more accurately model the windshield.

- Currently, ParaFuse is able to retrieve fuselage cross sections from a CPACS file. To generate the outer fuselage surface, using the CPACS fuselage profile, ParaFuse scales the cross section at several longitudinal locations along the guide curves. A simple extension of ParaFuse would be the capability to generate a fuselage surface using only CPACS fuselage profiles. This extension would require a bypass of the `ScaleCrossSection` class, as all profiles are imported directly from the CPACS file. The remainder of the surface generation process that is implemented in ParaFuse would not require any modification.

2. Inside-out fuselage sizing and outside-in cabin configuration:

   - The current implementation of the outside-in cabin configuration method requires information about the external fuselage dimensions to perform the cabin configuration process. The outside-in approach should be modified in such a way that it only requires the external surface of the fuselage to perform the cabin configuration procedure. With this modification it would become possible to generate cabins for fuselages that have not been generated using ParaFuse, but for example have been provided in a CAD or CPACS format.

   - Currently, a statistical relationship is used to determine the length of the flight deck. To increase the accuracy of the fuselage sizing methods a more detailed cockpit interior should be implemented in ParaFuse.

   - In the current implementation of the cabin configuration process no space is reserved for flight attendants and crew resting areas. To be able to more accurately determine the size of the cabin, these should be taken into account as well.

   - The size of the overhead storage compartments is not used as a constraint in the cross section sizing procedure. With the current trends in aviation, especially on short flights, the overhead storage capacity might become more important than the capacity of the cargo bay in the lower fuselage cross section. Therefore, it is recommended that an option to also take into account the size of the overhead storage compartments is added to the cross section sizing procedure.

   - In the current implementation of ParaFuse no containers are placed within the nose and tail cone. Cargo is only positioned in the middle fuselage section. In Appendix E it was observed that the number of containers seems to be underestimated because of this assumption. Therefore, it is recommended that the cargo bays are extended into the nose and tail cone of the aircraft. Before positioning a container in the nose or tail cone section of the aircraft, a check should be performed to see whether the container still fits within the available volume because of tapering of the nose and tail cone section.

   - Additionally, the cargo capacity should be added as an input for the inside-out sizing method. Currently, the size of the fuselage is determined based on the passenger capacity of the aircraft. Subsequently, the cargo is fitted in the available space. If cargo capacity can be given as an input as well, this might be used as a design driver for the inside-out cabin configuration process.

   - Implement the ability to generate fuselages of high-wing aircraft. The position of the wing will have implications on the position of the emergency exits and on the parameterization of the wing-body fairing.

   - An extension that follows the previous recommendation, would be the ability to perform the cabin configuration procedure for aircraft that have both a high and a low-wing. With this extension one would be able to use ParaFuse to generate fuselages of Prandtl planes.

   - Currently, it is not possible to model twin deck aircraft. It is believed that the parametric definition of the external surface would not require that much modification to model twin deck aircraft. The cabin configuration processes, however, would require significant alterations. To be able to model twin deck aircraft a second floor should be created in the cabin of the aircraft. Additionally, new clearance constraints should be established for the sizing procedure of the main fuselage cross section. Furthermore, new design rules for the placement of lavatories, galleys and exits should be derived.

- The current version of ParaFuse is not able to take into account more more than two aisles in the cabin configuration process. However, for the generation of blended wing body cabins more than two longitudinal aisles and several cross aisles are required. Therefore, design rules should be derived to model more than two aisles. Additionally, design rules should be established to determine the amount and position of the emergency exits.

A

Object-Oriented Programming

The object-oriented programming approach is based upon the fact that data and operations are modeled as coupled instead of separate elements. The various entities that interact with each other during the execution of a program are called 'objects'. Objects are organized in 'classes' based upon commonalities. Objects of the same class support a common set of operations. Furthermore, an object from a class is called an 'instance' of that class. An instance can hold several pieces of data. These pieces of data are called 'attributes'. Although objects of the same class have similar attributes, the value of these attributes can be different. For example several instances of the class 'fuselage' can have a different values for the attributes length, diameter and number of passengers. All the attribute values of a particular instance grouped together are called the 'state information'. The state information of an object represents the condition of that object at a given time. It is important to note that over time the attribute values of an object may be subject to change. The instances of a class can support a number of operations. These operations are called the 'methods' and provide a mechanism for interacting with the object. Methods and attributes are grouped together in a term called 'members' [33].

In the object-oriented paradigm objects are viewed as passive entities that do not perform any actions unless the object is told to do so. In order to interact with an instance of a specific class one of its methods can be 'called' or 'invoked' by another object. The object that invokes the method is termed 'caller', whereas the instance to which the method that is invoked belongs is termed the 'callee'. A sequence of method calls can be made. If so, these calls will be performed in the order in which they are invoked [33].

In some cases additional information must be specified by the caller in order for the callee to perform the correct action. This additional information is captured in a 'parameter'. For each method it must be established beforehand how many parameters are expected, what type of information is to be sent and in what order. Additionally, a callee can be expected to sent back information to the caller after an action is invoked. This information is captured in a 'return value' that is sent back to the caller. The expected parameters and return value that can be sent by a method is collected in the 'signature' of the method. This signature should be documented in the class definition [33].

Two types of methods exist in the object-oriented paradigm. First of all, methods can be 'inspectors'. This means that the methods do not change the state of the object, but only provide information on the current state to the caller by means of a return value. Secondly, methods can cause a change to the state of the callee. These methods are called 'mutators'. Mutators do not necessarily use return values to return information to the caller. Finally, some methods do not classify as an inspector or a mutator. This can happen when an action is invoked that does not sent any information to the caller and does not change the state of the callee [33].

By identifying commonalities between classes one can organize classes into a class hierarchy. This will allow for a more efficient representation of the different classes. Although two classes may be different from each other, some attributes and methods can be similar. Therefore, it is possible to define a generic parent class in which the common methods and attributes are defined. Subclasses or child classes can inherit methods and attributes from this parent class. The relationship between a parent and a child class is described as an

'is-a relationship'. This is different from a 'has-a relationship' when a class has attributes that are instances of other classes [33].

B

Class Function/Shape Function Transformation Method

Kulfan presents a universal parametric geometry representation method in [35, 51]. The method introduces a class function/shape function transformation technique (CST) and according to Kulfan an 'essentially limitless design space' can be achieved.

Kulfan adopts the notion described in Sobieczky [52] that the geometric definition of an aircraft is based on two basic types of shapes. The complete aircraft configuration is built up using different instances of the two basic shapes. Kulfan identifies these basic shape types as class 1 and 2. Class 1 are wing-airfoil-type shapes, whereas class 2 are body cross-section-type shapes. A overview of components that can be parameterized using class 1 shapes is given below [51]:

1. Wings
2. Rotor blades (e.g. helicopter, turbomachinery)
3. Horizontal or vertical tails
4. Nacelles

Class 2 shapes can be used to parameterize the following components [51]:

1. Fuselages
2. Rotor hubs or shrouds
3. Ducts
4. Lifting bodies

A detailed description of the CST method with an application to class 1 shapes can be found in Kulfan and Bussoletti [51]. An extension to the CST method to represent class 2 shapes and three dimensional geometries is given in Kulfan [35]. An overview of the CST method is present in this section. Firstly, the basis of the CST methodology will be discussed with respect to the class 1 shape type. Subsequently, the extension of the CST method to the class 2 shapes, such as aircraft fuselages will be described.

The technique Kulfan uses to describe an airfoil geometry can be broken into 5 steps. First of all, a general mathematical equation needs to be specified that is able to describe the geometry of a round-nose/sharp aft-end airfoil. Because of the fact that the airfoil has a round nose, singularities occur in the derivatives at the nose. Therefore, the next step is to identify the source of the numerical singularity in the expression for the airfoil. Subsequently, parts of the expression need to be rearranged or transformed to eliminate the singularity. The transformation or rearrangement of the expression leads to the identification of a 'shape function' transformation technique, which when applied to the original mathematical expression causes the expression to become a well-behaved analytical function. The shape function can be used to control the key design parameters. Finally, a 'class function' was introduced to enable the application of the procedure to a

wide variety of airfoils. For the representation of an airfoil this approach leads to the general expression in Equation B.1 [35].

$$\zeta(\psi) = \sqrt{\psi}(1-\psi)\sum_{i=0}^{N} A_i \psi^i + \psi\zeta_T \tag{B.1}$$

Where $\psi = x/c$, $\zeta = z/c$ and $\zeta_T = \Delta Z_{TE}/c$. According to Kulfan [35] the square root term in Equation B.1 is the only function that will provide a round nose airfoil. $(1-\psi)$ enables a sharp trailing edge, whereas $\psi\zeta_T$ can be used to define the trailing-edge thickness. The final term in Equation B.1 is

$$\sum_{i=0}^{N} A_i \psi^i$$

and defines the shape of the airfoil between the leading-edge and trailing-edge. Kulfan [35] defines a shape function $S(\psi)$ which is derived from Equation B.1 as:

$$S(\psi) = \frac{\zeta(\psi) - \psi\zeta_T}{\sqrt{\psi}[1-\psi]} \tag{B.2}$$

Combining Equations B.1 and B.2 yields the following expression of the shape function:

$$S(\psi) = \sum_{i=0}^{N} A_i \psi^i \tag{B.3}$$

The definition of the class fiction $C(\psi)$ is given by Kulfan in Equation B.4. The class function is used to describe general geometries. For example, a general NACA airfoil can be specified with $N1 = 0.5$ and $N2 = 1.0$.

$$C_{N2}^{N1}(\psi) \stackrel{\Delta}{=} (\psi)^{N1}[1-\psi]^{N2} \tag{B.4}$$

Finally, airfoil coordinates can be derived from the shape and class function combination when Equation B.1 is written in the general form:

$$\zeta(\psi) = C_{N2}^{N1}(\psi)\, S(\psi) + \psi\zeta_T \tag{B.5}$$

The unit shape function can be represented with a Bernstein polynomial. Effectively, this breaks down the airfoil shape function in component geometries that are summed and scaled to form the final airfoil shape. A Bernstein polynomial of order n is composed of $n+1$ terms of the form:

$$S_i(\psi) = K_i \psi^i (1-\psi)^{n-i} \tag{B.6}$$

Where $i = 0-n$ and K_i are binomial coefficients. These coefficients are defined as:

$$K_i \equiv \begin{pmatrix} n \\ i \end{pmatrix} \equiv \frac{n!}{i!\,(n-i)!} \tag{B.7}$$

As mentioned before the overall shape function is obtained by summing the scaled component airfoil geometries, as can be seen in Equation B.8. Bernstein polynomials of increasing order, representing the unit shape function, are shown in Figure B.1. The first term in each series represents the LE radius, whereas the last term represents the TE boat-tail angle. The other terms shape the rest of the airfoil.

$$S(\psi) = \sum_{i=1}^{n} A_i S_i(\psi) \tag{B.8}$$

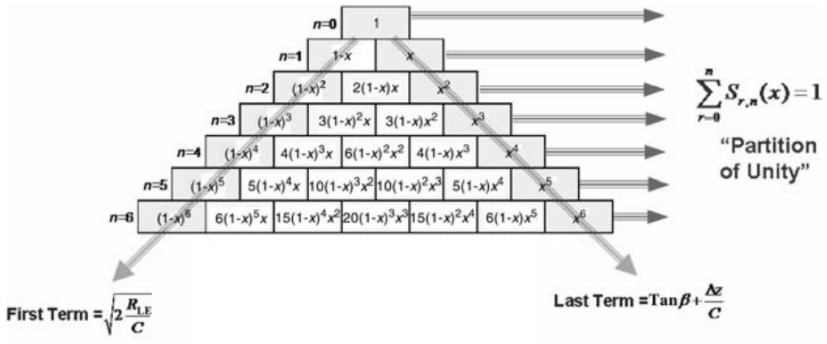


Figure B.1: Bernstein polynomials representing the unit shape function [35]

The scaling coefficients A_i can be obtained in several ways. For example, the the coefficients can be obtained by using them as design variables in an optimization problem or by performing a least-squares fit to a certain geometry [35]. Important to consider is the order of the Bernstein polynomial that is required to accurately represent an airfoil geometry. Kulfan and Bussoletti [51] have performed an extensive CFD analysis on a great number of airfoils, in which actual airfoils were compared to airfoils created with the CST method. It is concluded from this analysis that Bernstein polynomials with an order ranging from 6 to 9 closely match geometry and aerodynamic characteristics. Furthermore, results have shown that lower order Bernstein polynomials still produce geometries almost similar to the original airfoil. Kulfan suggests that even lower order Bernstein polynomials (order between 4 and 6) could produce optimum results. Hence, a high flexibility with only a limited number of deisgn variables can be achieved.

The CST method has been extended to body cross section geometries. In order represent a lateral cross section [35] uses a different shape and class function for the upper and lower part of the cross section, an approach similar to representing cambered airfoils. The general definition for the class function of class 2 shapes is given in Equation B.9. The general CST representation of the upper and lower parts of a body type cross section is given in Equations B.10 and B.11. An example using unit shape functions with different class function can be seen in B.2. More elaborate shapes, representing geometries with for example fairings, can be achieved by using different shape functions.

$$C(\eta) = \eta^{NC1}(1-\eta)^{NC2} \tag{B.9}$$

$$\zeta u(\eta) = Su(\eta)\, Cu_{NC2}^{NC1}(\eta) \tag{B.10}$$

$$\zeta l(\eta) = Sl(\eta)\, Cl_{NC2}^{NC1}(\eta) \tag{B.11}$$

Finally, Kulfan has extended the CST method to represent three-dimensional geometries. According to Kulfan [35] three-dimensional bodies can be represented as a distribution of cross-sectional shapes. The CST method can be used to describe the cross-sectional shapes and their distribution along the body axis. Kulfan has succeeded in creating a parameterization method which is able to accurately generate two- and three-dimensional geometries with only a limited number of design variables.

One of the shortcomings of the CST method presented by Kulfan is that local modifications of the geometry are not possible because Bernstein polynomials are used as a shape function. Furthermore, the an increase in the number of control points also leads to an increase in the order of the polynomial. As mentioned earlier, B-splines offer a solution to these disadvantages. Ideally, one would like to be able to control the shape of a geometry locally as well as globally. Therefore, Straathof [53] proposes to use a combination of Bernstein polynomials and B-splines. Straathof calls this approach the Class-Shape-Refinement-Transformation method (CSRT). A refinement function based on B-spline basis functions is added to the CST method proposed by Kulfan. The addition of the refinement function allows the user to locally modify the surface of the parameterized geometry. The CSRT method is defined as:

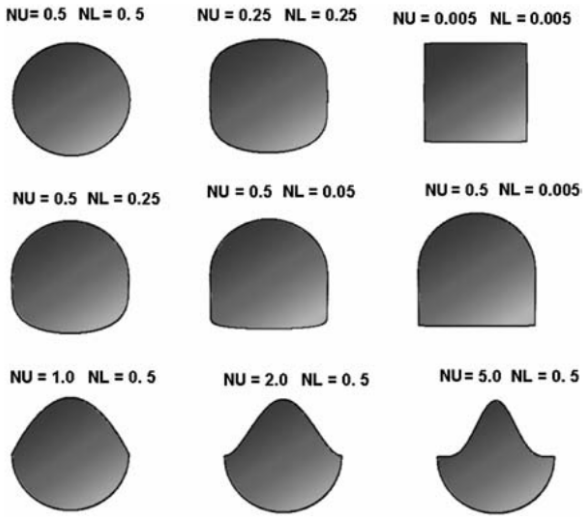


Figure B.2: Body type cross section shapes [35]

$$\zeta(\psi) = C_{N2}^{N1}(\psi)\, S(\psi)\, R(\psi) \tag{B.12}$$

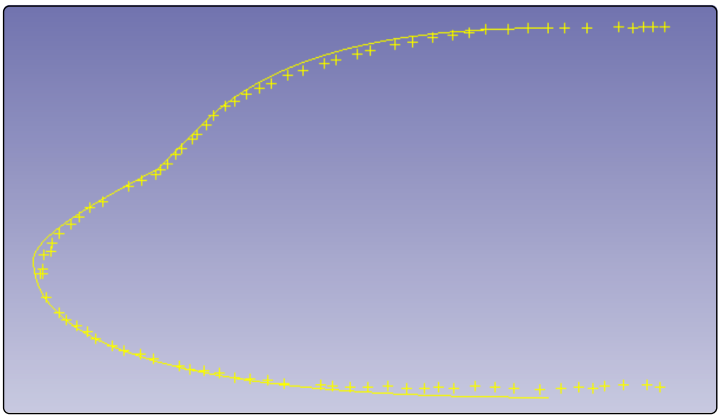
C

Nose cone shapes

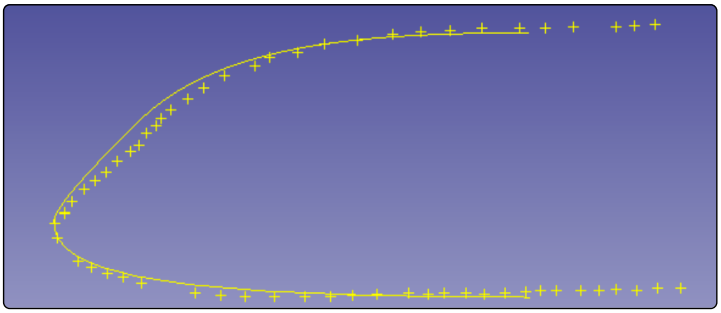
This chapter provides examples of nose cone shapes that can be modelled using ParaFuse. A side view of the resulting nose cone guide curves of several reference aircraft is presented in Figure C.1. The outline of the nose cone shape of the actual aircraft has been taken from Aircraft Characteristics documents and traced with graph digitizing software. The corresponding parameter values that have been used as input to generate a matching nose cone guide curve are shown in Table C.1

Table C.1: Parameter settings for nose cone guide curves

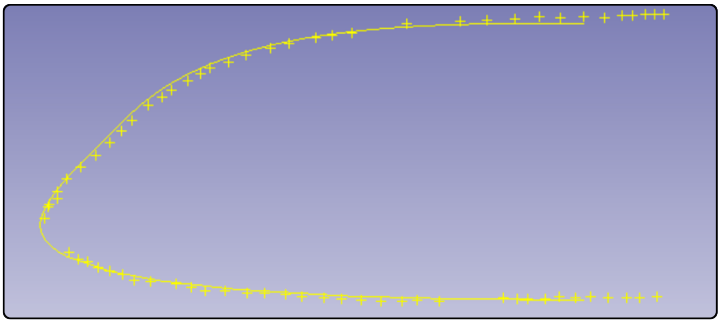
Parameter	**Unit**	**A320**	**A350**	**B787**
FR_n	[m]	1.4	1.8	1.97
x_loc_front_bulkhead	[m]	1	1	1
pilot_distance_to_bulkhead	[m]	1.3	1.3	1.3
pilot_eye_height	[m]	1.15	1.15	1.15
windshield_angle	[deg]	45	45	45
upward_view_angle	[deg]	30	27	25.4
overnose_angle	[deg]	20	20	21
dip_angle	[deg]	2	5	5
nose_sharpness_factor	[-]	0.3	0.2	0.3
bottom_rail_generation_method	[-]	general_bezier	general_bezier	general_bezier
bezier_weights	[-]	[1,1,1]	[1.6, 2, 1.6]	[1,1.1,1]



(a) Airbus A320



(b) Airbus A350

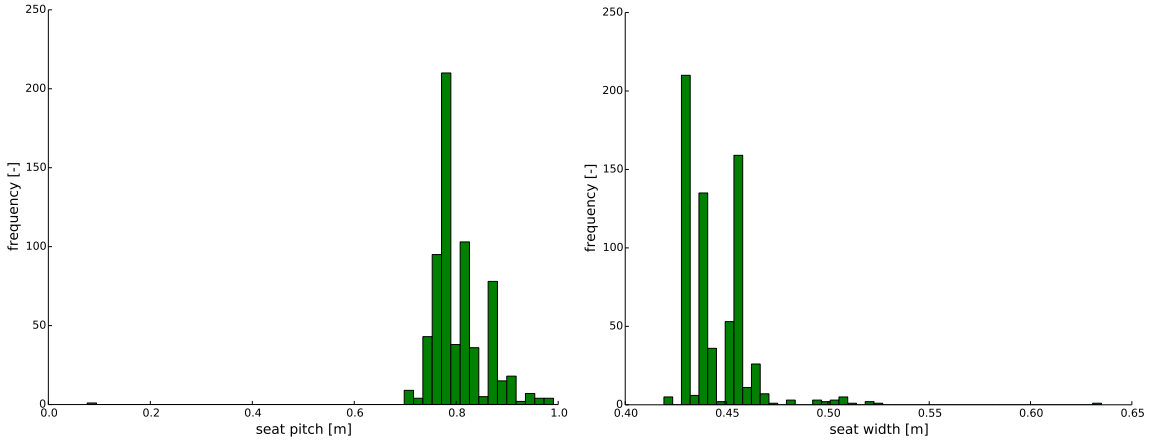


(c) Boeing 787

Figure C.1: Nose cone shapes of reference aircraft reproduced with ParaFuse

D

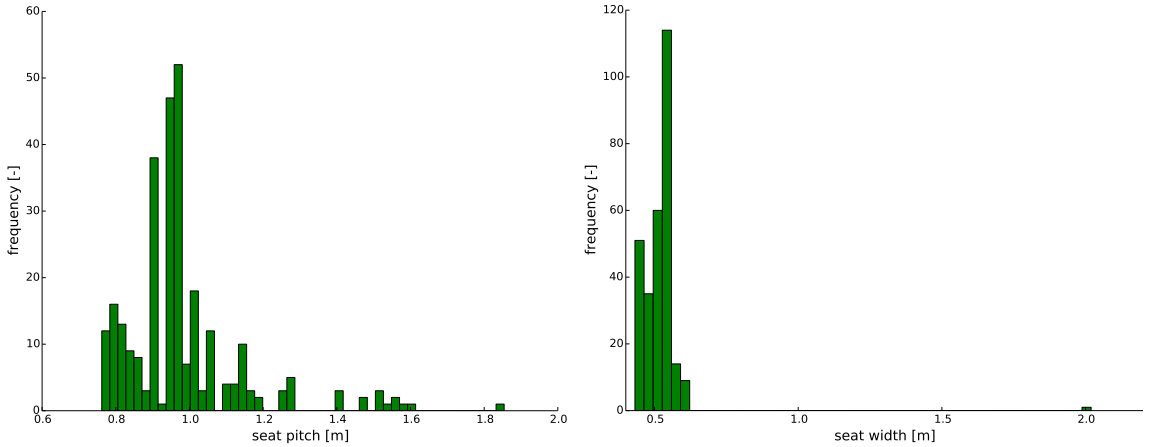
Seat data



(a) Pitch, $\mu = 0.805$, $\sigma = 0.056$

(b) Width, $\mu = 0.446$, $\sigma = 0.017$

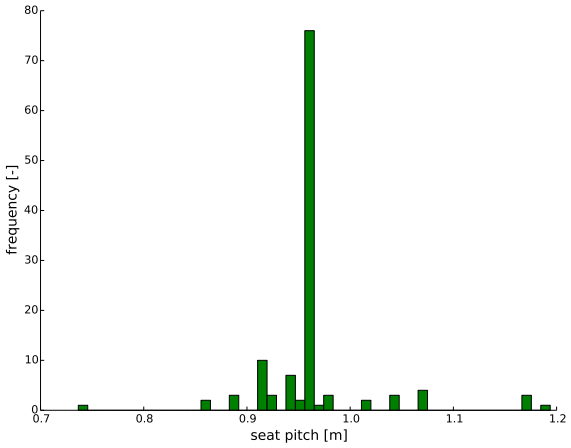
Figure D.1: Shorthaul economy class



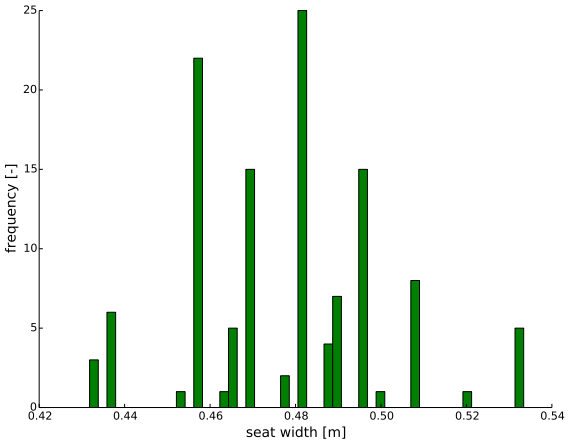
(a) Pitch, $\mu = 0.98$, $\sigma = 0.0166$

(b) Width, $\mu = 0.514$, $\sigma = 0.098$

Figure D.2: Shorthaul business/first class

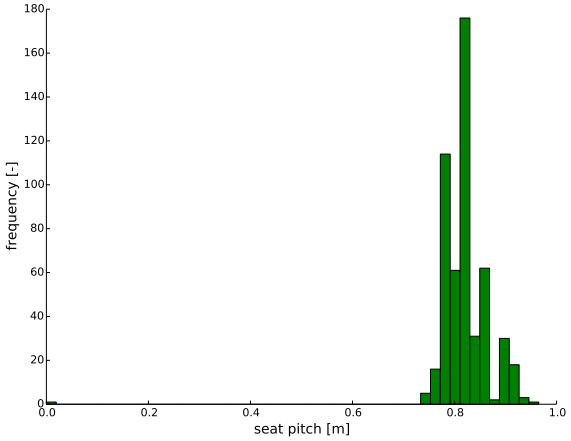


(a) Pitch, $\mu = 0.966$, $\sigma = 0.055$

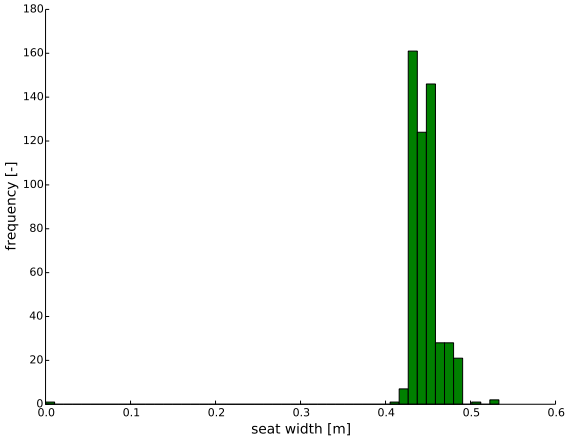


(b) Width, $\mu = 0.478$, $\sigma = 0.022$

Figure D.3: Premium economy class

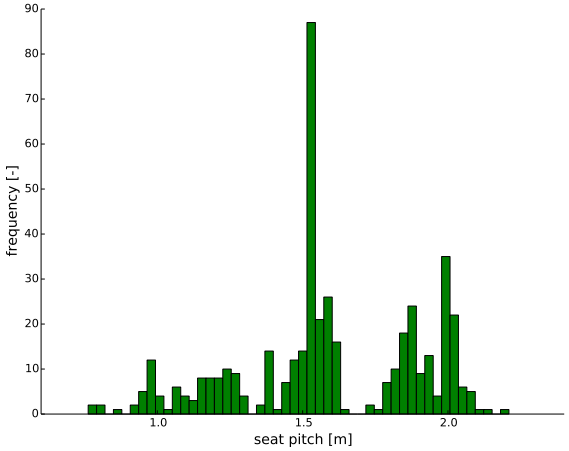


(a) Pitch, $\mu = 0.819$, $\sigma = 0.052$

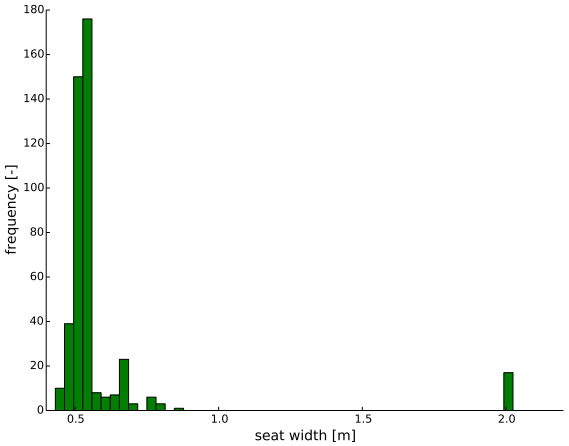


(b) Width, $\mu = 0.447$, $\sigma = 0.025$

Figure D.4: Longhaul economy class

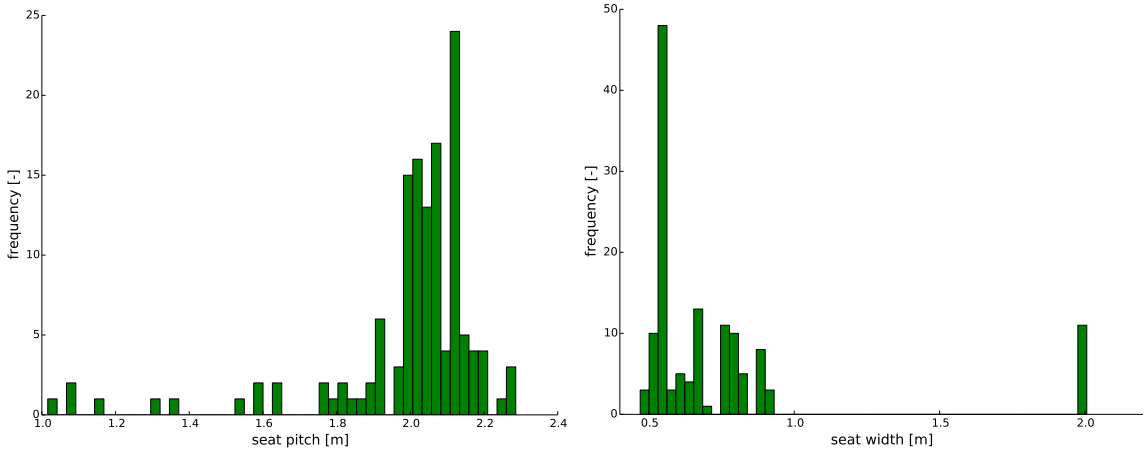


(a) Pitch, $\mu = 1.588$, $\sigma = 0.313$



(b) Width, $\mu = 0.596$, $\sigma = 0.287$

Figure D.5: Longhaul business class



(a) Pitch, $\mu = 1.985$, $\sigma = 0.22$

(b) Width, $\mu = 0.756$, $\sigma = 0.392$

Figure D.6: Longhaul first class

E

Validation cases

In this chapter the cases that have been generated to validate the inside-out fuselage sizing and cabin configuration method are presented. The focus of these validation cases is the inside-out cabin configuration method. For each of the cases, a figure with the cabin layout of the actual aircraft and the cabin layout generated by ParaFuse is included. The cabin layouts of the actual aircraft have been taken from Aircraft Characteristics and Airport Planning documents. These documents are provided by the manufacturer and contain a large amount of information regarding the aircraft dimensions and characteristics. Furthermore, a comparison of the side view of the aircraft is given as well.

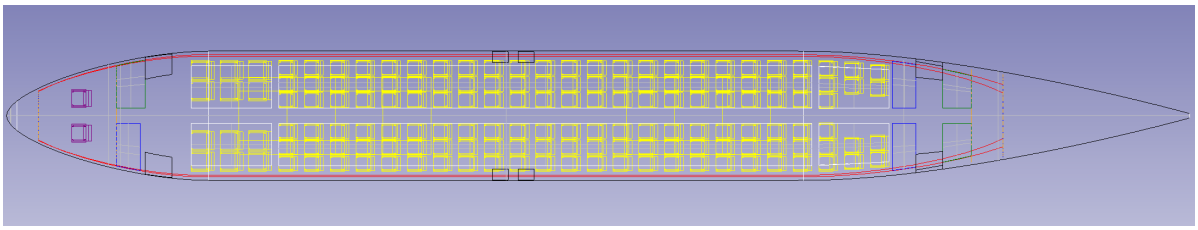
In the ParaFuse models the lavatories are indicated in blue and the galleys are depicted in green. The unit load devices are located in the lower fuselage section and are colored yellow, whereas bulk cargo bays have a pink color. An area for the wing box is reserved by the black rectangle.

The inside-out approach design a fuselage for a certain amount of passengers. Because of the fixed amount of passengers some rows in the resulting model may be partially filled. As a default setting ParaFuse completely fills these otherwise partially empty rows. Because of this, some of the generated models have more seats than they have been designed for.

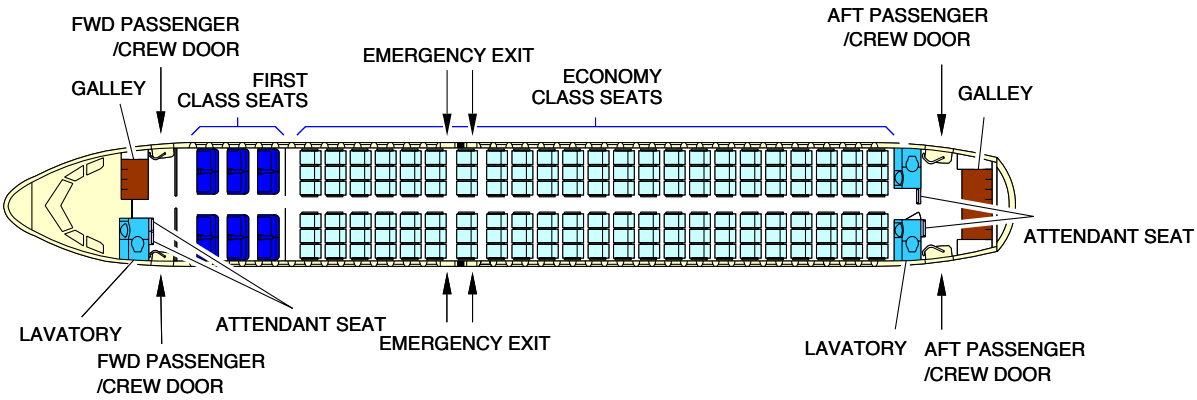
E.1. Airbus A320-200

Table E.1: Comparison between ParaFuse model and actual aircraft

		ParaFuse	**Actual**
Fuselage length	[m]	37.72	37.57
Fuselage width	[m]	4.10	3.95
Fuselage height	[m]	4.10	4.14
$N_{pax,economy}$	[-]	140	138
$N_{pax,business}$	[-]	12	12
$N_{pax,first}$	[-]	0	0
Number of galleys	[-]	3	3
Number of lavatories	[-]	3	3
Number of containers	[-]	14	14

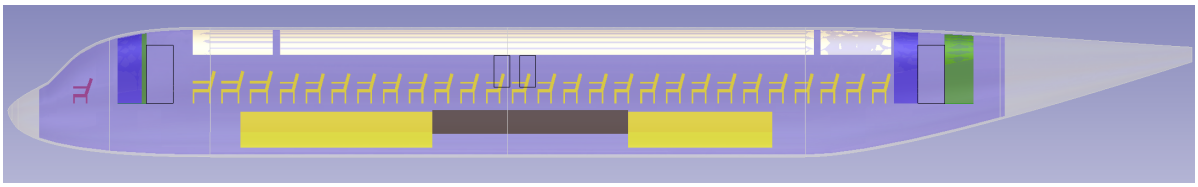


(a) ParaFuse

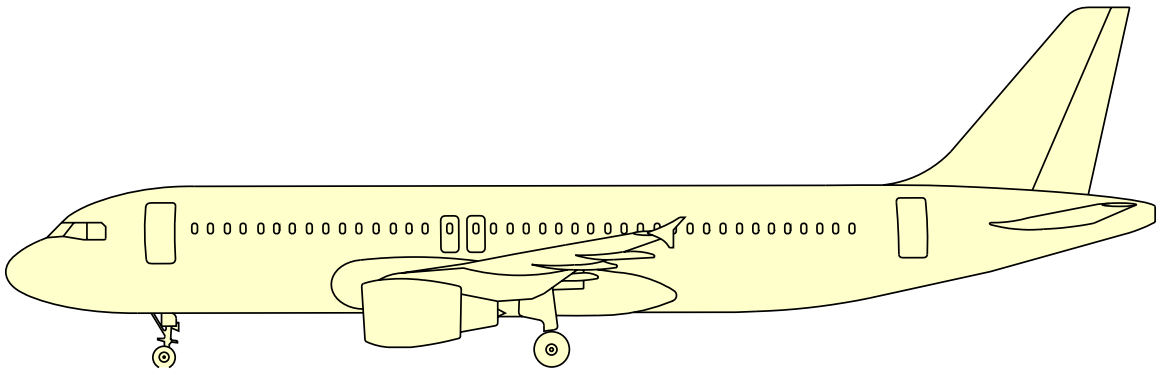


(b) Actual

Figure E.1: Airbus A320-200 cabin lay-out



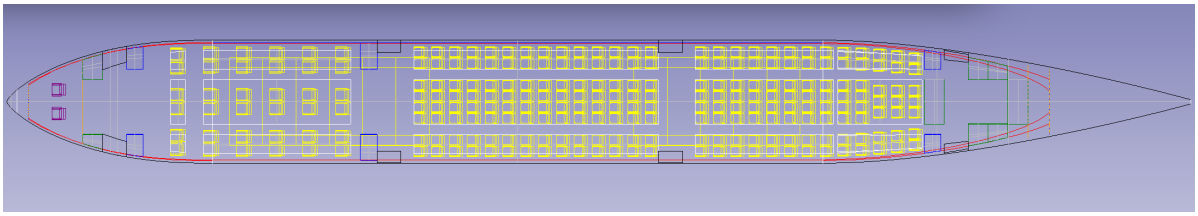
(a) ParaFuse



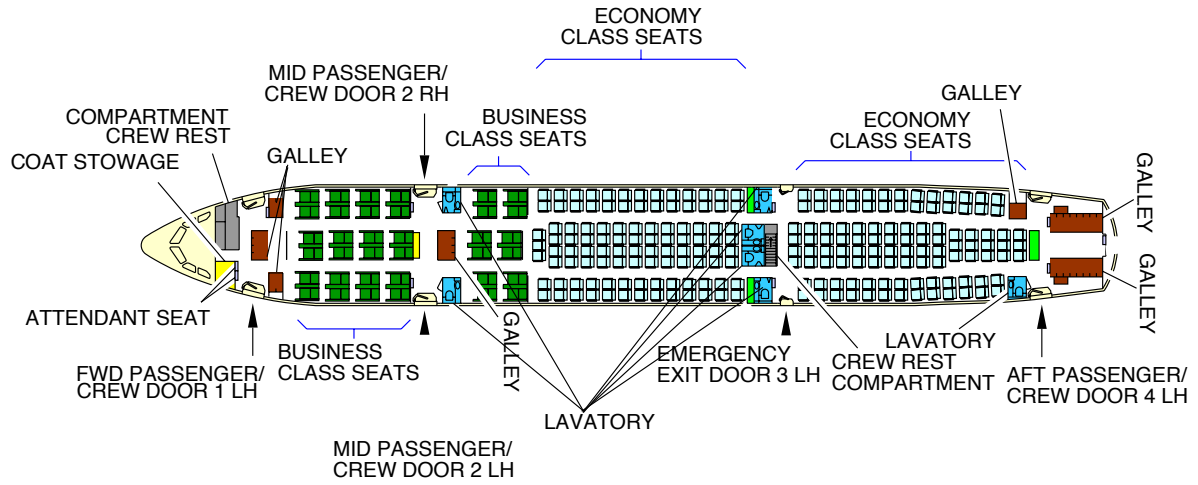
(b) Actual

Figure E.2: Airbus A320-200 side view

E.2. Airbus A330-200

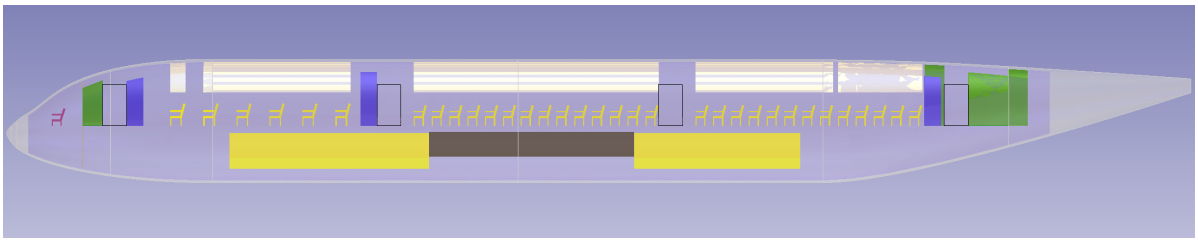


(a) ParaFuse

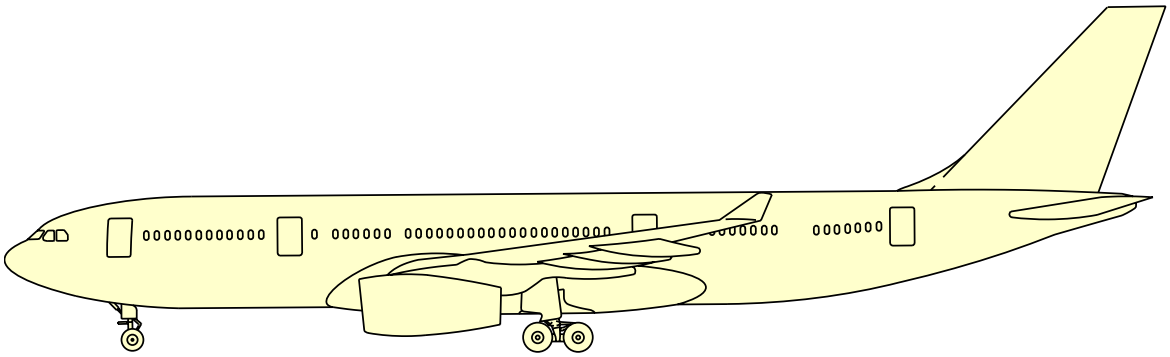


(b) Actual

Figure E.3: Airbus A330-200 cabin lay-out



(a) ParaFuse



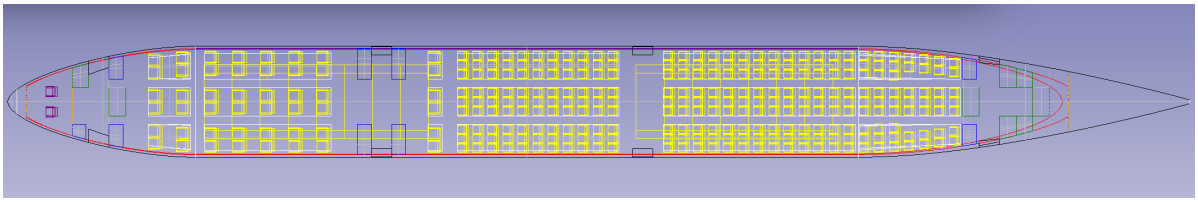
(b) Actual

Figure E.4: Airbus A330-200 side view

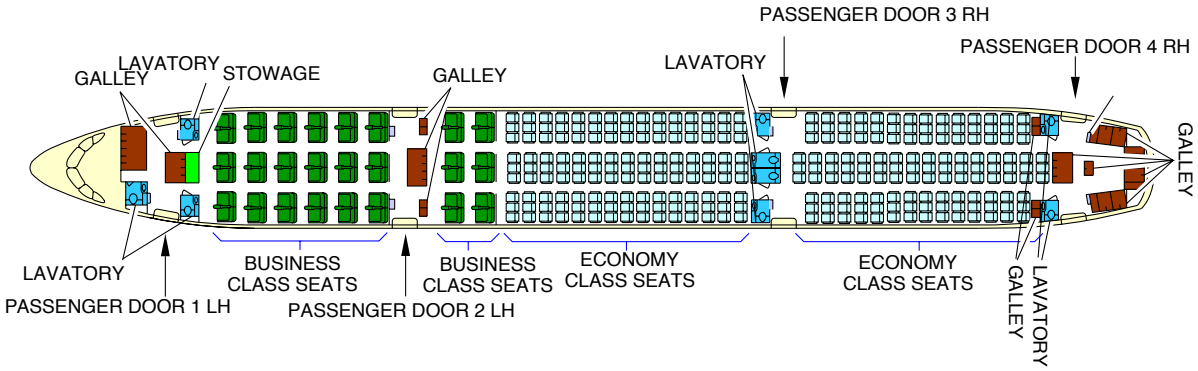
Table E.2: Comparison between ParaFuse model and actual aircraft

		ParaFuse	**Actual**
Fuselage length	[m]	54.5	57.51
Fuselage width	[m]	5.64	5.64
Fuselage height	[m]	5.64	5.64
$N_{pax,economy}$	[-]	213	210
$N_{pax,business}$	[-]	0	0
$N_{pax,first}$	[-]	36	36
Number of galleys	[-]	8	7
Number of lavatories	[-]	6	7
Number of containers	[-]	22	26

E.3. Airbus A350-900

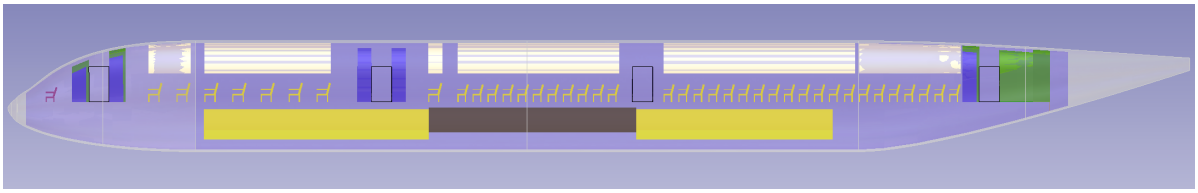


(a) ParaFuse



(b) Actual

Figure E.5: Airbus A350-900 cabin lay-out



(a) ParaFuse

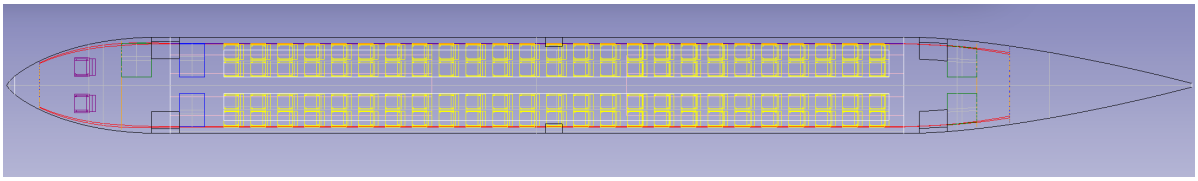
(b) Actual

Figure E.6: Airbus A350-900 side view

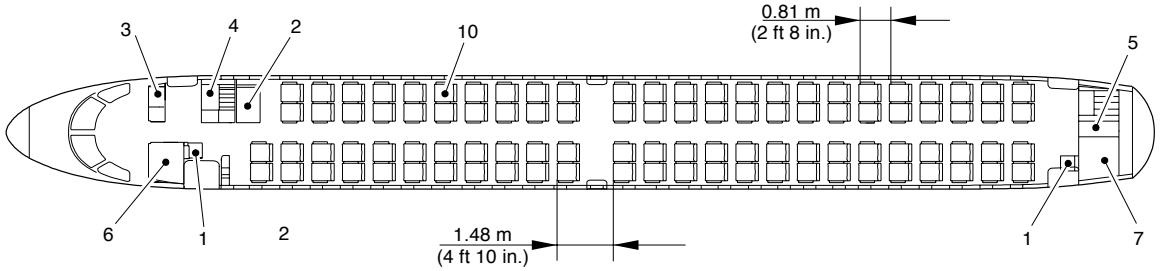
Table E.3: Comparison between ParaFuse model and actual aircraft

		ParaFuse	**Actual**
Fuselage length	[m]	64.4	65.26
Fuselage width	[m]	6.01	5.96
Fuselage height	[m]	6.01	6.09
$N_{pax,economy}$	[-]	271	267
$N_{pax,business}$	[-]	48	48
$N_{pax,first}$	[-]	0	0
Number of galleys	[-]	8	12
Number of lavatories	[-]	9	9
Number of containers	[-]	30	36

E.4. Embraer 190



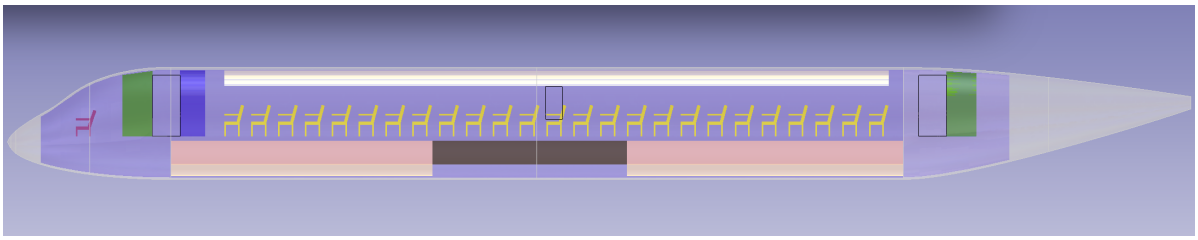
(a) ParaFuse



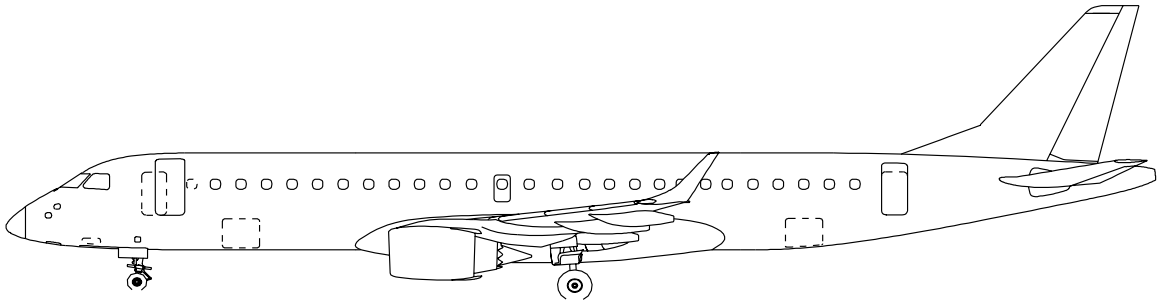
1 – FLIGHT ATTENDANT SEAT
2 – WARDROBE
3 – FWD RH G1 GALLEY
4 – FWD RH G2 GALLEY
5 – AFT RH GALLEY
6 – FWD LAVATORY
7 – AFT LAVATORY
8 – CARGO COMPARTMENT
9 – OVERHEAD BIN
10 – PASSENGER SEAT

(b) Actual

Figure E.7: Embraer 190 cabin lay-out



(a) ParaFuse



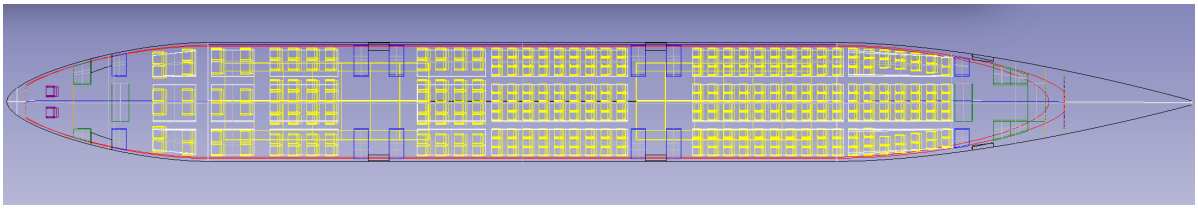
(b) Actual

Figure E.8: Embraer 190 side view

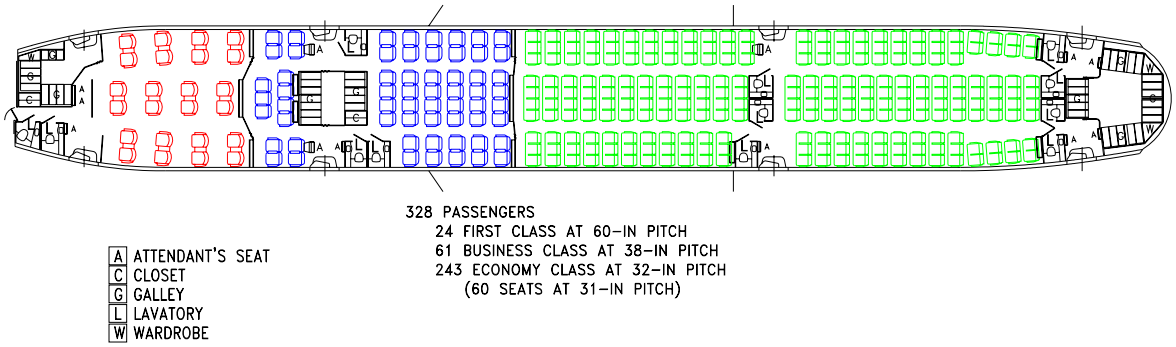
Table E.4: Comparison between ParaFuse model and actual aircraft

		ParaFuse	**Actual**
Fuselage length	[m]	36.12	36.24
Fuselage width	[m]	2.92	3.01
Fuselage height	[m]	3.4	3.35
$N_{pax,economy}$	[-]	100	98
$N_{pax,business}$	[-]	0	0
$N_{pax,first}$	[-]	0	0
Number of galleys	[-]	3	3
Number of lavatories	[-]	2	2
Bulk cargo volume	[m^3]	28.65	22.63

E.5. Boeing 777-200

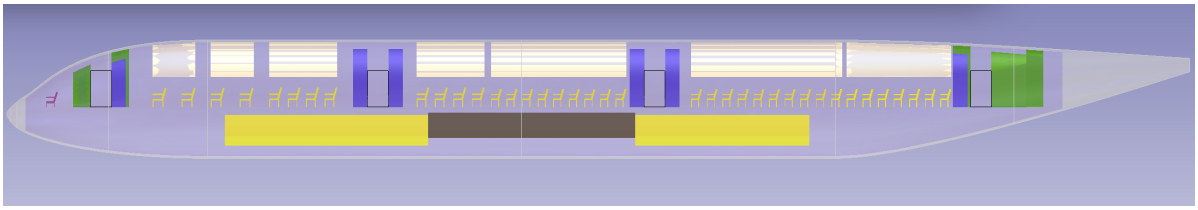


(a) ParaFuse

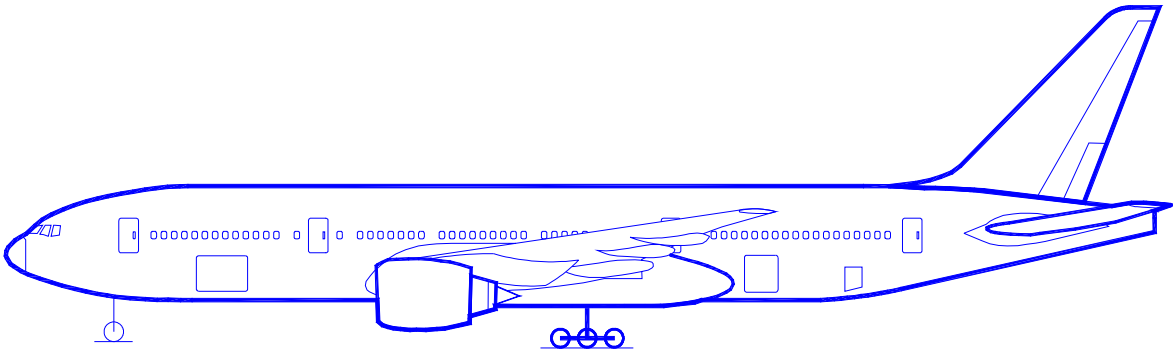


(b) Actual

Figure E.9: Boeing 777-200 cabin lay-out



(a) ParaFuse



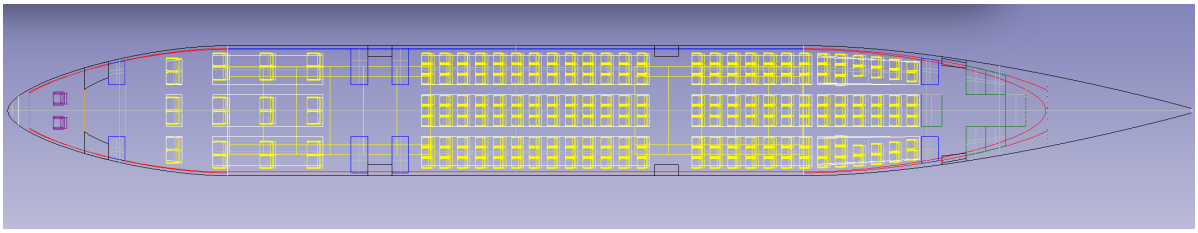
(b) Actual

Figure E.10: Boeing 777-200 side

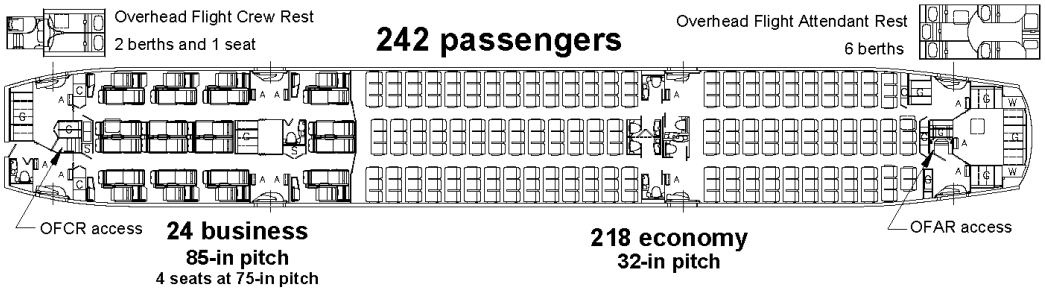
Table E.5: Comparison between ParaFuse model and actual aircraft

	ParaFuse	**Actual**	
Fuselage length	[m]	62.27	62.94
Fuselage width	[m]	6.21	6.2
Fuselage height	[m]	6.21	6.2
$N_{pax,economy}$	[-]		243
$N_{pax,business}$	[-]	64	61
$N_{pax,first}$	[-]	24	24
Number of galleys	[-]	9	9
Number of lavatories	[-]	12	12
Number of containers	[-]	26	32

E.6. Boeing 787-8

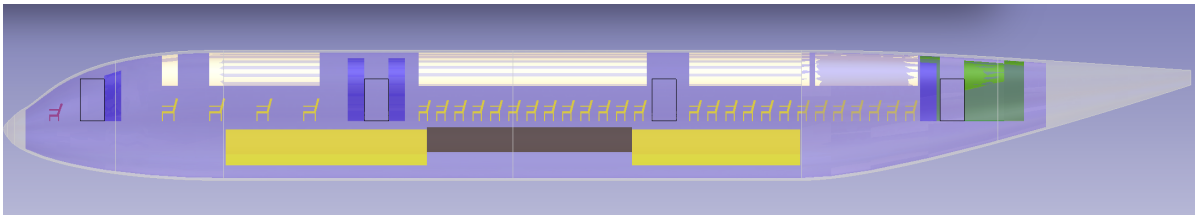


(a) ParaFuse

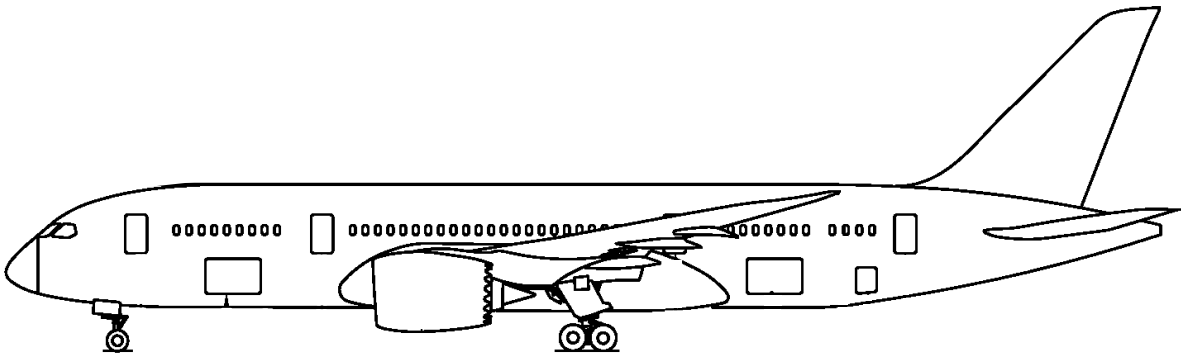


(b) Actual

Figure E.11: Boeing 787-8 cabin lay-out



(a) ParaFuse



(b) Actual

Figure E.12: Boeing 787-8 side view

Table E.6: Comparison between ParaFuse model and actual aircraft

		ParaFuse	**Actual**
Fuselage length	[m]	54.13	55.91
Fuselage width	[m]	5.92	5.77
Fuselage height	[m]	5.92	5.94
$N_{pax,economy}$	[-]	226	218
$N_{pax,business}$	[-]	24	24
$N_{pax,first}$	[-]	0	0
Number of galleys	[-]	7	6
Number of lavatories	[-]	8	8
Number of containers	[-]	22	28

F

ParaFuse user guide

ParaFuse is a knowledge-based engineering application that can be used to generate product models of fuselages of conventional passenger aircraft. To assist the designer two different fuselage sizing and cabin configuration methods are available. First of all, the inside-out design approach can be used to generate the fuselage based on top-level requirements such as passenger capacity. Secondly, the design can use the outside-in design routine to generate cabin interiors for a fuselage of fixed dimensions. This manual outlines how ParaFuse can be operated by the user.

F.1. Software requirements

ParaFuse is a KBE application built using the KBE framework ParaPy. In order to use ParaFuse the items listed below should be installed on the computer of the user. Currently, ParaPy can only be operated on Microsoft Windows. For more information on the installation of ParaPy the reader is referred to the documentation of ParaPy.

- Visual studio C++ 2012 redistributable packages
- Python and libraries
  - Python
  - Pip
  - wxpython
  - NumPy
  - SciPy
  - Matplotlib
- ParaPy
- IDE
- ParaFuse project

An Integrated Development Environment (IDE) is required to develop Python programs. One of the most popular IDE's is PyCharm by JetBrains. It is recommended that PyCharm is used to develop and edit the ParaFuse project. A free community edition is available, but with a student email address from TU Delft, one is also able to download the professional edition of PyCharm under an academic license. PyCharm can be dowloaded from `https://www.jetbrains.com/pycharm/download/`.

The ParaPy software can be obtained from the Blackboard organization 'Track 5: Flight Performance and Propulsion'. A license is required to be able to use ParaPy. This license can be obtained by sending an email

to Dr.ir. G. La Rocca[1].

ParaFuse has been developed using ParaPy version 1.0.7. To make sure that ParaFuse works properly, please use this version of ParaPy.

F.2. Project structure

The ParaFuse project is structured as follows:

ParaFuse/

- bin/
- doc/
  - source
  - build
- static/
- parafuse/
  - data/
    - ◇ toolInput.xml
    - ◇ seat_data.xml
    - ◇ uld_data.xml
    - ◇ monument_data.xml
    - ◇ exit_data.xml
  - returnDirectory/
    - ◇ design_report.txt
    - ◇ other files (.step, .iges, .stl, CPACS)
  - __init.py__
  - fuselage.py
  - other Python source code files that are used by the application

The main files of ParaFuse are contained in the parafuse module, that is located in the main project folder. The input files are stored in the subfolder 'data'. Files that are generated by ParaFuse, for example a CPACS export are stored in the subfolder 'returnDirectory'. To operate ParaFuse, the user must make sure that the the data files 'seat_data.xml', 'uld_data.xml', 'exit_data.xml' and 'monument_data.xml' are all present in the 'data' folder. Clean examples of these files can be found in the documentation of the project, which is shipped together with the parafuse module[2]

Two design approaches are available in ParaFuse. To select a design approach the user must specify the appropriate approach as a setting in the file 'toolInput.xml'. The file 'toolInput.xml' contains the inputs for the different parameter values required to generate a fuselage model. The different methods require a different type of input file. Again, clean input files for the tool input for both methods are included with the documentation of the application.

[1] g.larocca@tudelft.nl
[2] For more information on the documentation see Appendix G.

F.3. Operating ParaFuse

The ParaFuse project can be viewed and edited using PyCharm. The IDE can be used to extend or modify the code and the different input files. An overview of the main `Fuselage` class in the PyCharm IDE is shown in the figure below:

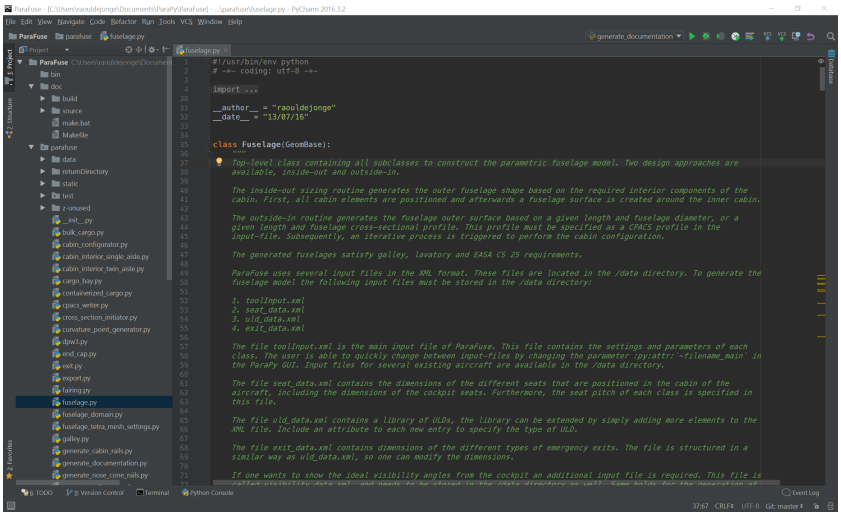


Figure F.1: ParaFuse `Fuselage` class in PyCharm IDE

ParaFuse can be used in two different ways. The first option is to use ParaFuse in an interactive way using the ParaPy GUI. To instantiate a ParaFuse fuselage model one must click the 'run' button in PyCharm, shown in the figure below.

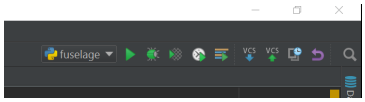


Figure F.2: Instantiate `Fuselage` class to operate ParaFuse interactively with the ParaPy GUI

When the 'run' button is pushed ParaPy initializes the GUI. This GUI can be used to inspect the `Fuselage` instance. The user is able to interactively modify the input parameters of the ParaFuse model using the GUI. A ParaFuse fuselage model generated in the ParaPy GUI is shown in Figure F.3.

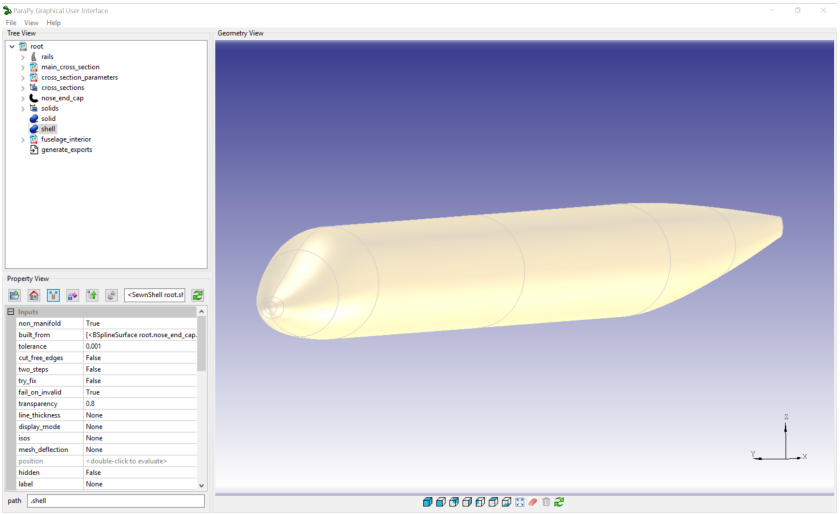


Figure F.3: ParaFuse fuselage model in the ParaPy GUI

The second option is to run ParaFuse in batch mode through an Python intepreter. The batch mode can be used to retrieve information about the product model without the using the GUI. The ability to run ParaFuse is batch mode is required if ParaFuse is to be used within an optimization framework, i.e. operation without human interaction. As an example, the piece of code below shows how one can retrieve the diameter of the fuselage cross section in batch mode. It must be noted that without the need to visualize the model in the ParaPy GUI the code runs even faster. Subsequently, the command is given to generate a design summary using the attribute `generate_design_report`, a screenshot of the resulting design report is shown in Figure F.4

Listing F.1: Example of operating ParaFuse in batch mode

```
>>> from parafuse.fuselage import Fuselage
>>> input_file = 'toolInput.xml'
>>> obj = Fuselage(filename_main=input_file)
>>> obj
>>> <Fuselage root at 0xd7479e8>
>>> obj.main_cross_section.diameter
>>> 4.12
>>> obj.generate_design_report
>>> Design report written to:
... C:\\Users\\raouldejonge\\Documents\\ParaPy\\ParaFuse\\parafuse\\returnDirectory
```

```
#############################################################################################
#                                                                                           #
#                                    ParaFuse Design Report                                 #
#                                     2017-01-10 13:18:38                                   #
#                                                                                           #
#############################################################################################

Design approach: inside-out

#############################################################################################
FUSELAGE PROPERTIES

Cross section shape: circle
Length: 36.55 [m]
Diameter: 4.13 [m]
Fineness Ratio: 8.86 [m]

#############################################################################################
PASSENGER INFORMATION

Total number of passengers: 150 [-]
Number of pax first class: 0.0 [-]
Number of pax first class: 0.0 [-]
Number of pax premium economy class: 0.0 [-]
Number of pax economy class: 150.0 [-]

#############################################################################################
MONUMENT INFORMATION

Required galley volume: 4.5 [m^3]
Installed galley volume: 6.3 [m^3]
Required number of lavatories: 3.0 [-]
Installed number of lavatories: 3.0 [-]

#############################################################################################
CARGO INFORMATION

Cargo type: containerized
ULD type: LD-3-45W
Installed ULDs: 16 [-]
Installed ULD volume: 31.63 [m^3]

#############################################################################################
EXIT INFORMATION

Exits per side {type, number}: {'1': 2, '3': 2}
```

Figure F.4: Screenshot of the ParaFuse design report

G

ParaFuse code documentation

The ParaFuse project has been documented using the Sphinx documentation generator. Sphinx is a Python package that is able to generate documentation of a Python project directly from the source code of the project. The aim of documentation is to describe to other developers how to use the code properly. To allow for easy readable comments, Python uses docstrings. These docstrings are enclosed by triple quotation marks and should be located in each separate class [33].

The docstrings can be accessed in two different ways. First of all, they can be accessed through the Python intrpreter using the `help` command. Furthermore, Sphinx uses these docstring to generate a html-based documentation, which can be accessed using a web browser. This functionality allows the user to access the documentation without the need to go into the source code of the project.

The documentation of ParaFuse is stored in the project directory and contains documentation of every single class. A screenshot of the landing page is shown in Figure G.1. The documentation of the classes contains a description of the class, supported by code snippets and flow charts. Furthermore, the inputs, attributes and parts that are defined within the specific classes are documented as well. A screen shot of the documentation of the class `CrossSectionInitiator` is shown in Figure G.2. Additionally, a user manual is included in the documentation, see Figure G.3. In this manual the requirements of ParaFuse are presented. Furthermore, the manual shows how to operate ParaFuse in batch mode as well as interactively using the ParaPy GUI. Furthermore, the input files that are required to operate ParaFuse are attached to this user manual as well, see Figure G.4.

Sphinx uses reStructuredText (RST) as markup language. For additional information on the reStructuredText syntax the reader is referred to `http://docutils.sourceforge.net/docs/user/rst/quickref.html` and `http://docutils.sourceforge.net/docs/user/rst/quickstart.html`.

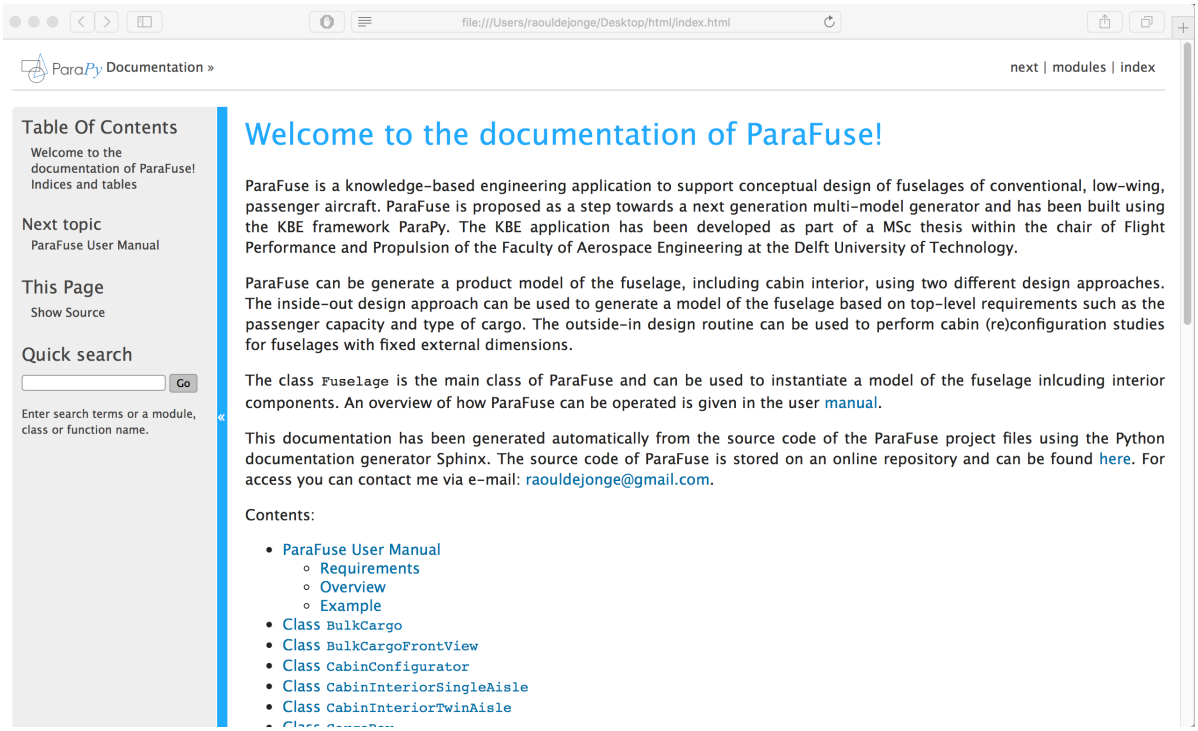


Figure G.1: Landing page of the ParaFuse documentation

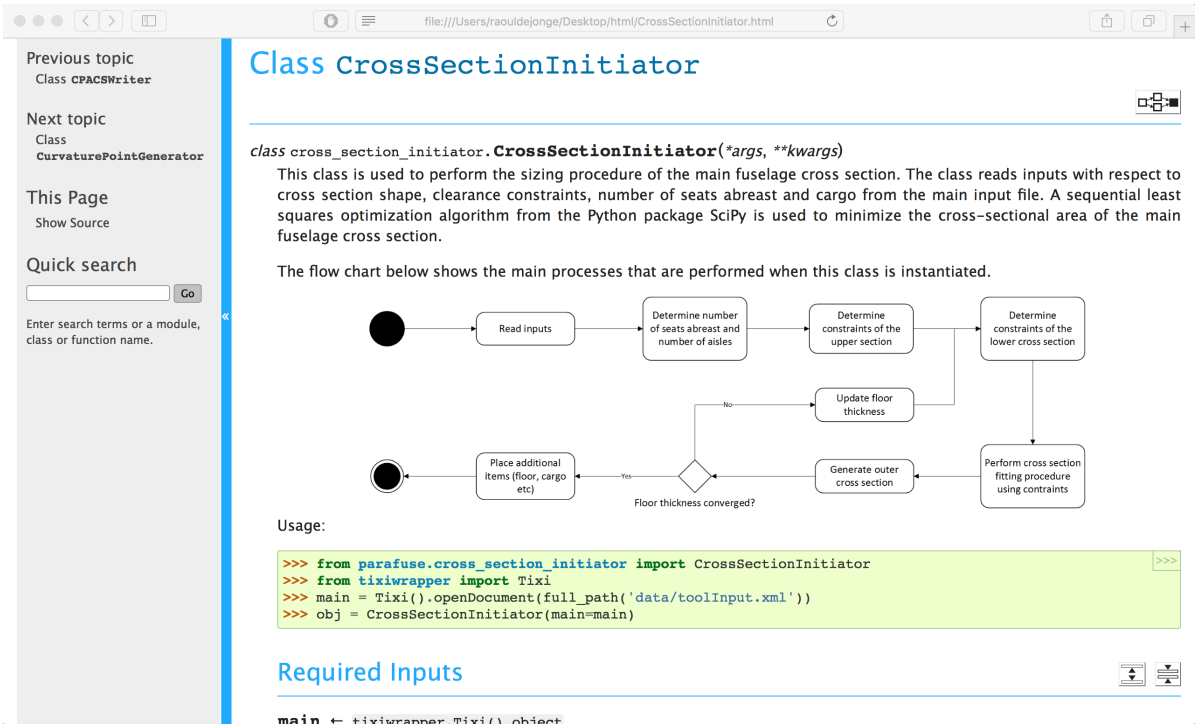


Figure G.2: Documentation page of the class `CrossSectionInitiator`

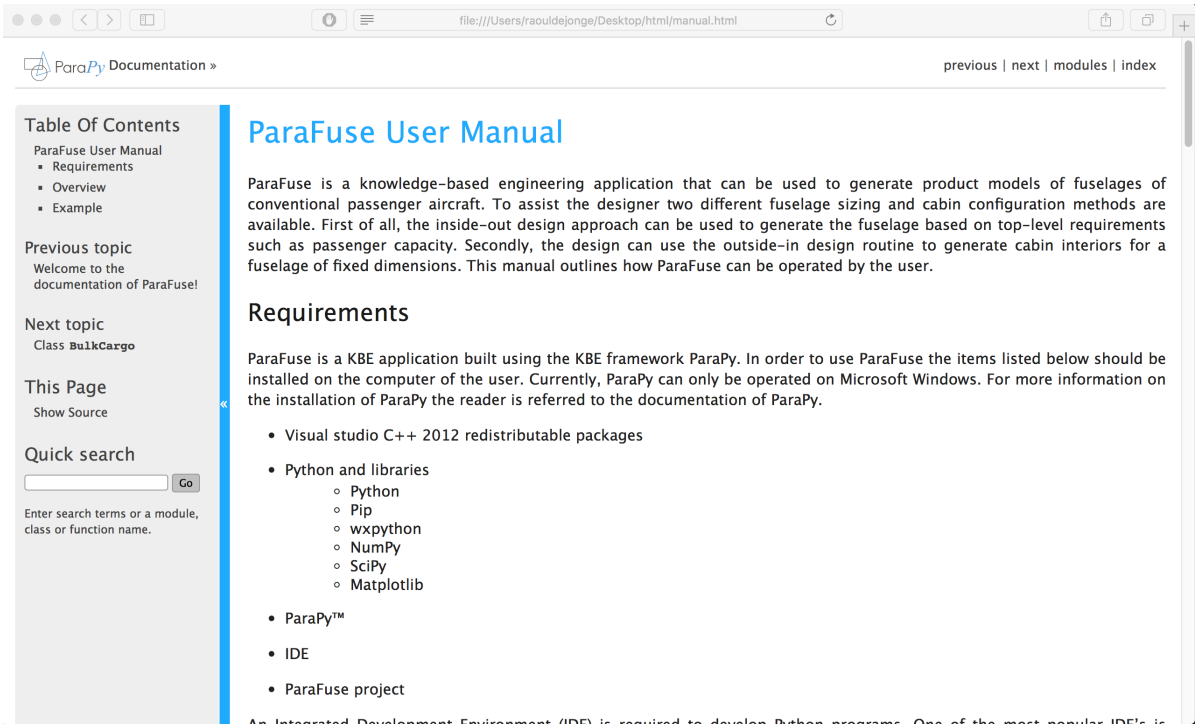


Figure G.3: ParaFuse user manual, generated using Sphinx

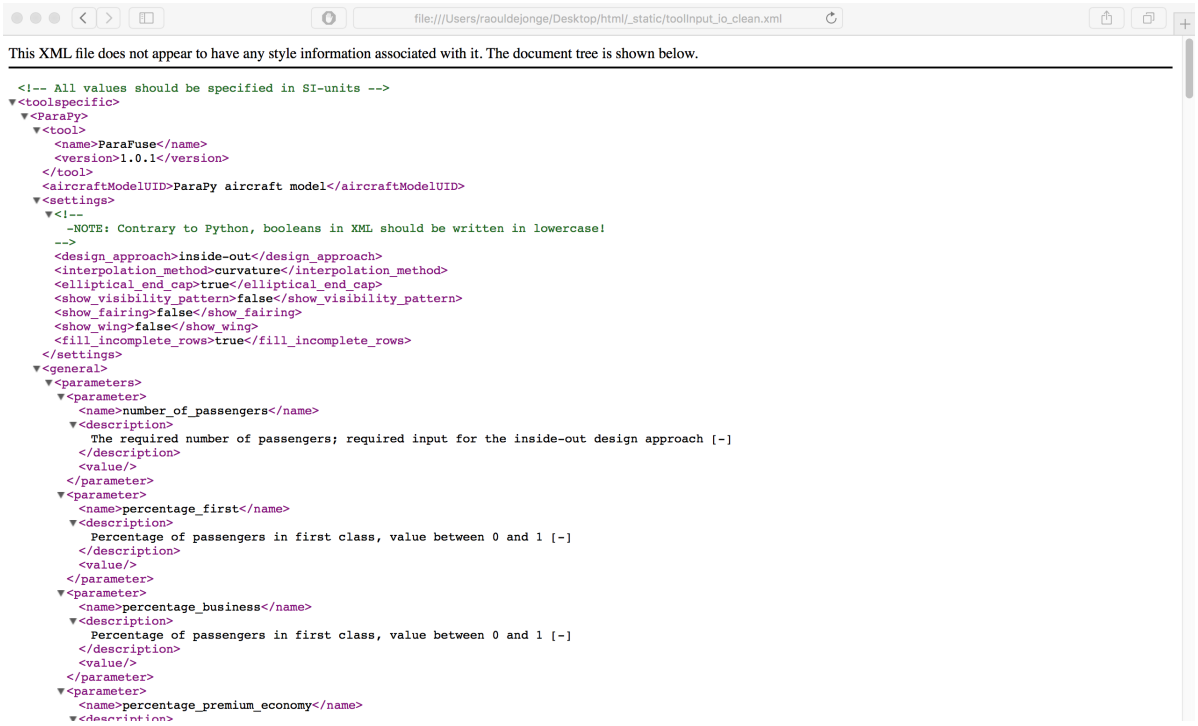


Figure G.4: Clean input file for the inside-out design approach, attached to the user manual

Bibliography

[1] K. Bowcutt, "A Perspective on the Future of Aerospace Vehicle Design," in *12th AIAA International Space Planes and Hypersonic Systems and Technologies*, (Reston, Virigina), American Institute of Aeronautics and Astronautics, dec 2003.

[2] M. H. Sadraey, *Aircraft Design*. Chichester, UK: John Wiley & Sons, Ltd, 1st ed., 2013.

[3] G. La Rocca, *Knowledge Based Engineering Techniques to Support Aircraft Design and Optimization*. PhD thesis, Delft University of Technology, 2011.

[4] C. Chapman and M. Pinfold, "Design engineering—a need to rethink the solution using knowledge based engineering," *Knowledge-Based Systems*, vol. 12, pp. 257–267, oct 1999.

[5] J. Sobieszczanski-Sobieski, A. Morris, M. van Tooren, G. La Rocca, and W. Yao, *Multidisciplinary Design Optimization Supported by Knowledge Based Engineering*. John Wiley & Sons Ltd., 1st ed., 2015.

[6] J. H. Koning, "Development of a KBE application to support aerodynamic design and analysis," 2010.

[7] E. Torenbeek, *Advanced Aircraft Design: Conceptual Design, Analysis and Optimization of Subsonic Civil Airplanes*. John Wiley & Sons Ltd., 1st ed., 2013.

[8] G. La Rocca, "Knowledge based engineering: Between AI and CAD. Review of a language based technology to support engineering design," *Advanced Engineering Informatics*, vol. 26, no. 2, pp. 159–179, 2012.

[9] C. B. Chapman and M. Pinfold, "The application of a knowledge based engineering approach to the rapid design and analysis of an automotive structure," *Advances in Engineering Software*, vol. 32, pp. 903–912, dec 2001.

[10] G. N. Blount, S. Kneebone, and M. R. Kingston, "Selection of Knowledge-based Engineering Design Applications," *Journal of Engineering Design*, vol. 6, pp. 31–38, jan 1995.

[11] D. Cooper and G. LaRocca, "Knowledge-based Techniques for Developing Engineering Applications in the 21st Century," in *7th AIAA ATIO Conf, 2nd CEIAT Int'l Conf on Innov and Integr in Aero Sciences,17th LTA Systems Tech Conf; followed by 2nd TEOS Forum*, (Reston, Virigina), pp. 1–22, American Institute of Aeronautics and Astronautics, sep 2007.

[12] W. J. C. Verhagen, P. Bermell-Garcia, R. E. C. van Dijk, and R. Curran, "A critical review of Knowledge-Based Engineering: An identification of research challenges," *Advanced Engineering Informatics*, vol. 26, pp. 5–15, jan 2012.

[13] M. Negnevitsky, *Artificial Intelligence A Guide to Intelligent Systems*. Harlow: Pearson Education Inc., 2 ed., 2005.

[14] J. Vandenbrande, T. Grandine, and T. Hogan, "The search for the perfect body: Shape Control for multidisciplinary design optimization," in *44th AIAA Aerospace Sciences Meeting and Exhibit*, (Reston, Virigina), pp. 1–16, American Institute of Aeronautics and Astronautics, jan 2006.

[15] A. Morris, P. Arendsen, G. La Rocca, M. Laban, R. Voss, and H. Hönlinger, "MOB a European Project on Multidisciplinary Design Optimisation," in *24th International Congress of the Aeronautical Sciences*, 2004.

[16] M. Tooren, E. Schut, and J. Berends, "Design "Feasilisation" using Knowledge Based Engineering and Optimization Techniques," in *44th AIAA Aerospace Sciences Meeting and Exhibit*, no. January, (Reston, Virigina), pp. 1–18, American Institute of Aeronautics and Astronautics, jan 2006.

[17] R. Elmendorp, R. Vos, and G. La Rocca, "A conceptual design and analysis method for conventional and unconventional airplanes," in *29th Congress of the International Council of the Aeronautical Sciences*, pp. 1–12, 2014.

[18] J. P. T. J. Berends and M. J. L. van Tooren, "Multi-Agent Task Environment Framework to Support Multi-disciplinary Design and Optimization," *Journal of Aerospace Information Systems*, vol. 10, pp. 258–267, jun 2013.

[19] G. La Rocca and M. J. Van Tooren, "Knowledge-Based Engineering Approach to Support Aircraft Multi-disciplinary Design and Optimization," *Journal of Aircraft*, vol. 46, pp. 1875–1885, nov 2009.

[20] Y. H. A. Brouwers, "Development of KBE applications to support the conceptual design of passenger aircraft fuselages," 2011.

[21] G. La Rocca, T. H. M. Langen, and Y. H. A. Brouwers, "The Design and Engineering Engine. Towards a Modular System for Collaborative Aircraft Design," in *28th International Congress of the Aeronautical Sciences*, pp. 1–12, 2012.

[22] Z. Zhu, *Automatic 3D Routing for the Physical Design of Electrical Wiring Interconnection Systems for Aircraft*. PhD thesis, Delft University of Technology, 2016.

[23] E. Obert, *Aerodynamic Design of Transport Aircraft*. Delft: Delft University Press, 1st ed., 2009.

[24] E. Torenbeek, *Synthesis of Subsonic Airplane Design*. Delft: Delft University Press, 1st ed., 1976.

[25] A. Sóbester and A. I. J. Forrester, *Aircraft Aerodynamic Design: Geometry and Optimization*. Chichester: John Wiley & Sons Ltd., 1st ed., 2015.

[26] P. Della Vecchia and F. Nicolosi, "Aerodynamic guidelines in the design and optimization of new regional turboprop aircraft," *Aerospace Science and Technology*, vol. 38, pp. 88–104, oct 2014.

[27] S. Siegel, "Comparison of Design Rules Regarding the Wing-Body Junction Flow of a Subsonic Aircraft," Tech. Rep. June, 2011.

[28] M. Nita and D. Scholz, "From Preliminary Aircraft Cabin Design to Cabin Optimization," in *DGLR: Deutscher Luft- und Raumfahrtkongress 2010 : Tagungsband - Manuskripte*, pp. 287–302, 2010.

[29] J. C. Fuchte, *Enhancement of Aircraft Cabin Design Guidelines with Special Consideration of Aircraft Turnaround and Short Range Operations*. PhD thesis, Hamburg University of Technology, 2014.

[30] European Aviation Safety Agency, "Certification Specifications and Acceptable Means of Compliance for Large Aeroplanes," Tech. Rep. Amendment 17, European Aviation Safety Agency, 2015.

[31] ACARE, "Flightpath 2050 Europe's Vision for Aviation. Report of the High Level Group on Aviation Research," 2011.

[32] B. Nagel, D. Böhnke, V. Gollnick, P. Schmollgruber, A. Rizzi, G. La Rocca, and J. J. Alonso, "Communication in Aircraft Design: Can We Establish a Common Language?," in *28th International Congress of the Aeronautical Sciences*, pp. 1–13, 2012.

[33] M. H. Goldwasser and D. Letscher, *Object-Oriented Programming in Python*. Pearson Education Inc., 1st ed., 2007.

[34] B. Kulfan, "Recent Extensions and Applications of the "CST" Universal Parametric Geometry Representation Method," in *7th AIAA ATIO Conf, 2nd CEIAT Int'l Conf on Innov and Integr in Aero Sciences,17th LTA Systems Tech Conf; followed by 2nd TEOS Forum*, no. March, (Reston, Virigina), pp. 1–32, American Institute of Aeronautics and Astronautics, sep 2007.

[35] B. M. Kulfan, "Universal Parametric Geometry Representation Method," *Journal of Aircraft*, vol. 45, pp. 142–158, jan 2008.

[36] N. M. Patrikalakis and T. Maekawa, *Shape Interrogation for Computer Aided Design and Manufacturing (Mathematics and Visualization)*. Springer-Verlag, 2002.

[37] W. E. Stoney, "Collection of zero-lift drag data on bodies of revolution from free-flight investigations," tech. rep., 1961.

[38] H. Schlichting and E. Truckenbrodt, *Aerodynamics of the Airplane*. U.S.A.: McGraw-Hill, 1st ed., 1979.

[39] W. Song and P. Lv, "Two-Level Wing-Body-Fairing Optimization of a Civil Transport Aircraft," *Journal of Aircraft*, vol. 48, pp. 2114–2121, nov 2011.

[40] D. P. Raymer, *Aircraft Design : A Conceptual Approach*. Washington DC: American Institute of Aeronautics and Astronautics, 2004.

[41] J. Roskam, *Airplane Design*. Kansas, U.S.A.: DARcorporation, 1st ed., 1985.

[42] L. R. Jenkinson, P. Simpkin, and D. Rhodes, *Civil Jet Aircraft Design*. London, U.K.: Arnold, 1st ed., 2003.

[43] T. Corke, *Design of Aircraft*. Pearson, 1st. ed., 2002.

[44] L. Morichon, "Selected statistics in aircraft design," tech. rep., Hamburg University of Applied Sciences, Hamburg, 2006.

[45] R. L. Huston, *Principles of Biomechanics*. Boca Raton: CRC Press, 1st ed., 2009.

[46] D. Kraft, "Algorithm 733; TOMP—Fortran modules for optimal control calculations," *ACM Transactions on Mathematical Software*, vol. 20, pp. 262–281, sep 1994.

[47] M. Li, *Conceptual Design Study for In-flight Refuelling of Passenger Aircraft*. PhD thesis, Delft University of Technology, 2017.

[48] J. H. Wei, "Parametric modelling for determining aircraft stability & control derivatives," 2016.

[49] R. Elmendorp, "Synthesis of Novel Aircraft Concepts for Future Air Travel," 2014.

[50] M. F. M. Hoogreef, "The Oval Fuselage A New Structural Design Concept for Blended Wing Body Cabins," 2012.

[51] B. Kulfan and J. Bussoletti, ""Fundamental" Parameteric Geometry Representations for Aircraft Component Shapes," in *11th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, vol. 1, (Reston, Virigina), pp. 547–591, American Institute of Aeronautics and Astronautics, sep 2006.

[52] H. Sobieczky, "Aerodynamic Design and Optimization Tools Accelerated by Parametric Geometry Preprocessing," in *European Congress on Computational Methods in Applied Sciences and Engineering*, no. September, (Barcelona), p. 8, 2000.

[53] M. H. Straathof, M. J. L. V. Tooren, M. Voskuijl, and B. Koren, "Aerodynamic Shape Parameterisation and Optimisation of Novel Configurations," *Proceedings of the 2008 Royal Aeronautical Society Annual Applied Aerodynamics Research Conference*, 2008.