INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa

UAV Flight Simulator based on ESA Infrastructure

Flight simulation models compliant with SMP standard

António Daniel Miranda de Almeida

Dissertação para obtenção do Grau de Mestre em
Engenharia Aeroespacial

Júri

Presidente: Prof. Agostinho Rui Alves da Fonseca
Orientação: Prof. Paulo Jorge Soares Gil
Doutora Ana Filipa Caetano Relvas
Vogal: Prof. Bertinho Manuel D'Andrade da Costa

Julho de 2007

Acknowledgements

I would like to express my gratitude to my supervisor and project manager Ana Relvas for providing the opportunity for my internship. Ana was permanently there to give a helping hand and she always believed that I could do the job. Her personality makes the Critical Software Lisbon offices a fantastic place to work at.

I am thankful to my supervisor at IST, Professor Paulo Gil from the Aerospace Engineering group, for his support and especially for his proofreading of this thesis.

I am grateful to my friend José Mendes, who participated in the SimUAV project with me, for always helping out when I was stuck. His determination and resourcefulness made SimUAV go further than I imagined possible.

A big collective “thank you” to the good folks at Critical Software who helped me overcome some of my limitations in the field of computer science. I couldn’t possibly name everyone, but I can say that no-one was ever too busy they couldn’t spare a few minutes. I am also grateful to those who indirectly helped, in particular by providing the resources for making SimUAV a reality.

I acknowledge Dr. William Hargrove at the Oak Ridge National Laboratory in Tennessee for answering my questions regarding the EMBYR model and my colleague José Gomes at IST for his part of the matrix processing class.

Last but not least, I am thankful to my family for having provided me with the conditions for developing during my life. I am thankful to my friends as well, to those who came and those who went, for I couldn’t have done it without you.

Resumo

A procura de aeronaves não tripuladas (*unmanned aerial vehicles* – UAVs) tem vindo a aumentar nos sectores civis e militares. Isto traduz-se na necessidade de melhores ferramentas, tais como simuladores de voo, para dar suporte ao desenvolvimento e às operações. Simuladores de voo podem ser usados para suportar o projecto de UAVs e respectivas estações de controlo, treinar operadores, e efectuar pesquisas no sector aeroespacial, oferecendo várias vantagens sobre o uso de UAVs reais.

A maioria dos simuladores de UAVs foram desenvolvidos com propósitos específicos e não são versáteis. Neste trabalho, um simulador de UAVs compatível com o *standard Simulation Model Portability* (SMP) da ESA foi desenvolvido. Este *standard* foi aplicado no campo de simulação espacial e promove modularidade e versatilidade da simulação. Uma especificação comum de modelos compatível com SMP foi estabelecida.

Esta especificação foi utilizada para desenvolver um conjunto fundamental de modelos para simular a operação de um UAV. Os modelos implementados simulam a dinâmica do veículo, aerodinâmica, propulsão, gravitação, trens de aterragem e atmosfera. Um requisito fundamental para o desenvolvimento do simulador foi o ênfase na versatilidade e capacidade de crescimento, o que simplificará a utilização de novos modelos e a melhoria e/ou substituição dos actuais. Este requisito teve impacto directo no *software* implementado. Interfaces com o *Google Earth* e o *FlightGear* foram estabelecidos para visualizar o resultado da simulação. Testes foram executados para validar a operação do simulador. A sua aplicabilidade num cenário de patrulha de incêndio foi demonstrada.

Palavras-chave

- UAV
- Simulação de Voo
- Simulation Model Portability
- SIMSAT
- STANAG 4586

Abstract

Demand for UAVs has been increasing in the civilian and military sectors. This translates to the need for better tools, such as flight simulators, to support development and operations. Flight simulators can be used to support the design of UAVs and respective control stations, train operators, and perform research in the aerospace field, offering many advantages over the use of real UAVs.

Most existing UAV flight simulators have been defined with specific purposes in mind and are not versatile. In this work, a UAV flight simulator compatible with the ESA Simulation Model Portability (SMP) standard is developed. This standard has proven applications in the field of space simulation and promotes modularity and versatility of the simulation. A common model specification compatible with SMP was established.

This common model specification was used for developing a fundamental set of models for simulating the operation of a UAV. The models implemented simulate vehicle dynamics, aerodynamics, propulsion, gravitation, landing gears, and atmosphere. A fundamental requirement for the development of the simulator was the promotion of versatility and scalability, which simplifies the addition of new models and the improvement and/or replacement of existing ones. This requirement had a direct impact in the software implemented. Interfaces with Google Earth and the FlightGear flight simulator have been established for visualizing the simulation output. A number of tests were performed to validate the operation of the simulator. Its applicability to a scenario of fire monitoring support was demonstrated.

Keywords

- UAV
- Flight Simulation
- Simulation Model Portability
- SIMSAT
- STANAG 4586

Contents

List of Figures

List of Tables

List of Acronyms

AFCS	Automatic Flight Control System
DOF	Degree of freedom
EGM96	Earth Gravitational Model 96
ESA	European Space Agency
GIS	Geographic Information Systems
ISA	International Standard Atmosphere
MMI	Man-Machine Interface
NATO	North Atlantic Treaty Organization
SIMSAT	Software Infrastructure for Modelling Satellites
SMI	Simulation Model Interface
SMP	Simulation Model Portability
UAV	Unmanned Aerial Vehicle
UDP	User Datagram Protocol
USAF	United States Air Force
VSM	Vehicle Specific Module
XML	Extensible Markup Language

Chapter 1

Introduction

Flight simulation of Unmanned Aerial Vehicles (UAVs) is an area of growing importance, in both civilian and military markets. UAV flight simulators can be used to support scientific research, vehicle and control station design and validation, and for training purposes. The development of a generic UAV flight simulator promoting versatility and supporting the ESA Simulation Model Portability (SMP) standard offers numerous advantages over the conventional approach of building a simulator for a specific application.

1.1 Background and Motivation

UAVs provide a series of advantages over conventional aircraft. By dealing away with the pilot, a great deal of constraints no longer applies to the vehicle. It becomes possible to operate flying vehicles with a very broad range of sizes and shapes, which allows for a wider range of missions to be performed. Additionally, UAVs do not need to carry life support equipment, which potentially reduces both production and maintenance costs.

There are also immediate advantages to the absence of the pilot itself. Long, repetitive missions become more effective, since the UAV does not become tired or distracted as a pilot would. The most important reason in many situations, however, is the reduction of the endangerment of human life.

Present issues with the use of UAVs in civilian roles are mostly tied to the current lack of certifications for the operation of UAVs in commercial airspace, as well as the added financial costs that often accompany emerging technologies. The European Civil Unmanned Air Vehicle Roadmap identifies the “closed loop dilemma” on Figure 1 [1].

The UAV industries are waiting for certification from the civil regulators, the regulators are waiting for firm evidence from users of UAVs that they are interested in fielding the vehicles, and the users are waiting for certified vehicles from the UAV industry.

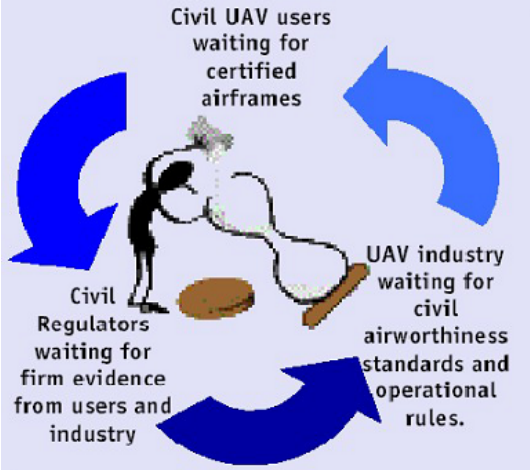

Figure 1: The UAV "closed loop dilemma" [1]

A more fundamental problem with UAVs lies within the limitations of current artificial intelligence. UAVs are not capable of taking decisions at the same level as a pilot can, which is important in contingency situations. The natural evolution of science and technology will continuously improve the autonomy of UAVs, but such issue raises the age-old question of how far can artificial intelligence go.

Due to the fact that UAVs can have a very wide range of configurations, they can be adapted to fit very specific needs. As such, each UAV is often designed with a specific range of applications in mind, which can be civilian or military. The most common applications are activities of surveillance and observation. Specific civilian applications range from border patrol, public event security, maintenance and security of oil and gas pipelines, communications, and power lines, early warning against forest fires, coastline patrolling, search and rescue support, and environmental observation [1]. Figure 2 depicts possible applications of UAVs in civil monitoring operations.

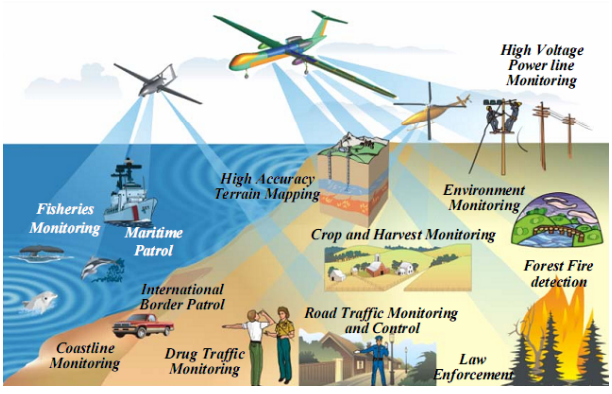

Figure 2: Applications of UAVs in civil monitoring operations [1]

Regarding system certification, a UAV flight simulator can be used for testing the behaviour of the vehicle but, perhaps more importantly, for testing the UAV control station. This offers an alternative to

the use of real UAVs in these applications, possibly reducing their cost. As a design tool, a UAV simulator can be a precious aid in the early phases of the project of a vehicle and all throughout the project of a control system. Also, since it is possible to create a UAV flight simulator that a control system operator cannot distinguish from the real vehicle [2], a UAV simulator can be used as a training tool. This has the advantages of allowing training on the most various situations, simulating failure scenarios, and keeping real UAVs in active duty.

The UAVNET thematic network is an initiative funded by the European Commission with the purpose of optimising research efforts of civil UAV applications. It is responsible for the production of the European Civil Unmanned Air Vehicle Roadmap [1], which establishes the motivation and the guidelines for bringing Europe to the front of civilian UAV research and development. This document identifies the areas where Europe is lagging behind, especially when compared to the United States of America, as well as the political and technological challenges that lie ahead. UAV flight simulation is identified as an important area of investment for overcoming these challenges.

A standard for the implementation of simulation models has been developed by ESA: the Simulation Model Portability (SMP) standard. It is widely supported by the European space industry and has been used on space simulations [3]. It promotes the encapsulation of the various components of a simulation in models that are as independent as possible, while still having a common form of interaction.

The use of the SMP standard promotes the modularity and versatility of the simulation, which means that changes to an existing model will have little or no impact on other models. This allows for easier development and maintenance. For these motives, compliance with the SMP standard is a specific requirement for the simulator developed in this thesis. An additional benefit of supporting the SMP standard is that the UAV simulator will be easier to integrate with existing space simulators.

1.2 State of the Art of Flight Simulation

Flight simulation is an art as old as aviation itself. The first flight simulators were used as a training tool without endangering the pilot or the aircraft. In the beginning, they were very simple non-powered aircraft replicas with restricted movements and a limited range of simulated features [4].

With advances in aviation, especially in World Wars I and II, the need for better and cheaper pilot training grew, which made the demand for simulation grow as well. This resulted in the advancement of simulation, with the introduction of actuators for cockpit motion, as implemented in the famous Link Trainer in Figure 3 [5].

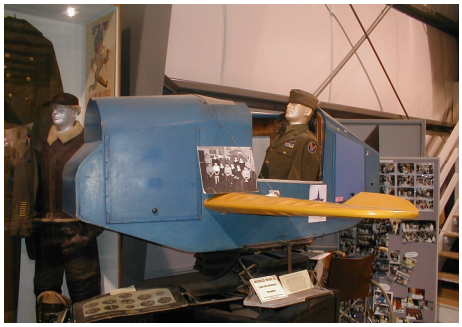

Figure 3: The Link Trainer Flight Simulator [5]

As flight simulation advanced, the need for the use of computers arose. Eventually, the first six degree-of-freedom (DOF) flight simulators were implemented. The introduction of computers in flight simulation resulted in greater freedom for implementing more and more features, resulting in more accurate simulations. Modern flight simulators are very complex and expensive machines, such as the NASA Advanced Concepts Flight Simulator [6] (see Figure 4), a six DOF simulator of generic commercial transport aircraft.

Figure 4: Advanced Concepts Flight Simulator [6]

Nowadays, the expression “flight simulator” has come to mean more than a machine used for aircrew training. A flight simulator can be a fully computational implementation, which is used not as a regular “cockpit flight simulator” but as research tool or computer game, for example. Figure 5 displays a screenshot of Microsoft Flight Simulator X, a popular computer game [7].

Since UAVs have only achieved popularity in recent years, the field of UAV simulation is still taking its first steps, making it ripe with opportunity. An example of this is how, during operation Desert Storm, the Pioneer UAV training systems did not come equipped with a simulator, which required that all training took place using real UAVs, taking vehicles away from active mission duties [8].

Figure 5: Screenshot of Microsoft Flight Simulator X [7]

The technological lag of Europe in the field of UAVs, as identified in the European Civil UAV Roadmap, also applies to the specific aspect of UAV flight simulation [1]. The existent projects in this area are often restricted to specific vehicles and applications, and are often military in nature.

An example of a UAV simulator applied to a research project is the MultiUAV simulator [9], which was developed by the United States Air Force (USAF) for investigating cooperative control between multiple UAVs operating to achieve a common objective. MultiUAV is a Simulink implementation featuring a six DOF flight dynamics model and inter-vehicle communication, as well as interfaces for post-simulation analysis using external tools.

A project similar to MultiUAV is [10]. This project is also a Simulink implementation of multiple UAVs. The vehicles are used along with a fire propagation model for the purpose of studying an algorithm for patrolling the fire front. Note that a similar scenario was implemented in this thesis, for the purposes of demonstrating the functionality of the UAV flight simulator.

Multiple UAV simulators such as the ones described above have been implemented and most have the same limitation: they are simulators designed with a specific scenario and research objective in mind and cannot be used as a generic simulation platform, which could serve in a multitude of applications by tailoring existing modules and adding new ones.

There are also multiple military applications of UAV simulators. These are usually designed for operator training purposes or for research in the field of management of multiple military units in a common scenario. An example of this is the Ness UAV simulator [11], which covers the complete range of tasks that a UAV operator would have to perform. It features a station for the operator-in-training, a station for the instructor, and a tool for generating simulation scenarios. Ness is a global company with headquarters in the USA and Israel.

Another example of a military training application of UAV simulation is the Predator UAV Synthetic Task Environment [12]. This is a system designed for training a Predator UAV operator in three specific

tasks: manoeuvring, landing, and area reconnaissance. It is built on top of a generic training platform for the Predator UAV.

Australia is also home to the development of UAV simulators for military training purposes. CAE Australia is responsible for a training station consisting of a UAV flight simulator and an operator station [13]. This simulator has been developed both for civilian and military applications, be it research or training. The system provider claims that this simulator supports modularity, but no data was found on how this is achieved, and to what degree.

In the industrial area, the RT-LAB UAV Engineering Simulator (see Figure 6) is a solution oriented for UAV design tasks, based on a hardware platform resembling a UAV control station [14]. The UAV flight simulator software features a certain degree of modularity. It supports multiple UAVs, as well as hardware in the loop operation. The RT-LAB Engineering Simulator comes close to the goals of this work, but lacks the advantages provided by the compatibility with the SMP standard.

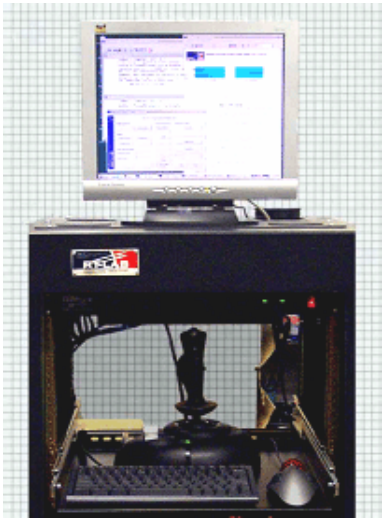

Figure 6: RT-LAB UAV Engineering Simulator [14]

There is some evidence of European research in the field of UAV simulation, trying to fill the underdevelopment of the continent in this area. An example of this is a project of the Warsaw University of Technology which developed a UAV simulator featuring an environment with atmospheric phenomena, terrain, and computer generated air traffic [15]. The goal was to study mission planning algorithms.

Another European research effort in UAV simulation was the development of a UAV fight simulator for use with a control station [16]. The goal was supporting the definition of the operational requirements for the control station, as well as testing new ideas. Multiple scenarios were generated and the performance of the control station operators was evaluated, as well as their feedback.

1.3 Scope of the Thesis

In this thesis, a series of components of an SMP compliant UAV flight simulator, SimUAV, were developed. The objective of SimUAV was to address the major shortcomings of typical UAV simulators, which are developed with a particular purpose in mind, such as military training for a specific UAV or scientific research in a particular scenario. SimUAV achieves its objective by providing a generic simulation platform – which can be used in a multitude of scenarios –, and a versatile software architecture, which allows for easy adding, updating, and removing of components.

The versatility of SimUAV is achieved through its separation into components, which consist of simulation models and interfaces with external visualization tools. Each component has a very clearly established interface for communicating with other components, and is as independent as possible. This facilitates the replacing of a particular simulation model by another version describing the same process with different precision (e.g. replacing the atmospheric model by a more accurate one). Alternatively, a particular model can be replaced by another that simulates a different reality (e.g. replacing the atmospheric model with a model for the Martian atmosphere), as long as the same inputs and outputs are provided.

The component based architecture of SimUAV was possible thanks to the support for the ESA SMP standard, which defines how the models interact with each other. Supporting SMP also ensured that SimUAV was compatible with SMP-compliant simulation platforms, such as SIMSAT, which was used in this project. This allowed for focusing on the development of the simulation models and not on the infrastructure for executing the simulation. Furthermore, supporting SMP means that SimUAV will be easier to integrate with existing space simulations, for creating complex simulation scenarios.

The specific components of SimUAV developed in this thesis consist of simulation models and interfaces. The models simulate vehicle dynamics, aerodynamics, propulsion, gravitation, terrain reactions, atmosphere, fire propagation, and an algorithm for fire front following. The interfaces serve to communicate with the tools Google Earth [17] and FlightGear [18], allowing for easier interaction with the simulation as well as a clear understanding of the simulation outputs. The interfaces also provide a valuable validation tool. The remaining components necessary for SimUAV to operate were developed in [19]. They consist of an on-board automatic flight control system, a communications interface between the simulated UAV and a control station, and simulation models for the sensors and actuators.

The aforementioned fire propagation and fire front following models, which were developed in this thesis, serve to demonstrate the applicability of SimUAV to the scenario of fire fighting support. Examples of the use of UAVs in this scenario are described in [10] and [20]. UAVs can be very helpful for fire-fighters in the field by providing them with aerial imagery of the fire zone, allowing them to better coordinate their efforts and improving their efficiency.

The work developed in this thesis was executed during an internship at Critical Software and is part of an internal research project. Due to this, the development of SimUAV followed the software development life cycle process implemented by Critical Software [21], which establishes a procedure to be followed when developing software that promotes the quality of the product. In accordance with this process, a series of tests were executed in order to validate the simulator.

1.4 Thesis Layout

The development of SimUAV followed the software development life cycle implemented by Critical Software, which is briefly described in Chapter 2. An overview of the various stages is given, describing the procedures taken and the motivation behind them. Chapters 3 to 5 represent project development specific stages corresponding to the software life cycle, and are presented in the chronological order in which the project progressed.

Chapter 3 describes the requirements needed to attain the objective of a generic SMP compliant flight simulator. The functionality that is expected from the simulator is described in a very clear and unequivocal way. The simulator is broken down into major components and the functionality that is expected of each is clearly stated. Additionally, specific technological requirements regarding standards and tools are also presented.

Chapter 4 is dedicated to describing in detail each of the software components implemented to answer the requirements described in Chapter 3. An overview of each component is given, as well as a detailed description of its functionality.

Chapter 5 presents the results of the validation tests that SimUAV was subjected to. The first part of the tests consists of a series of small-scale, specific tests which concentrate on the functionality offered by a single model or a small number of models at a time. The second part of the tests focuses on scenarios that involve all components of SimUAV.

Chapter 6 contains conclusions and recommendations for further work.

Chapter 2

Software Development Life Cycle

2.1 Introduction

This chapter presents a brief overview of the software development life cycle as implemented by Critical Software. The process described here was the support for this work. The rationale behind the standards and procedures that comprise the software life cycle is presented, as well as an overview of each of the relevant phases of the cycle.

2.2 Overview

Software development in the industrial world follows sets of guidelines and procedures that establish how the development process should take place in order to ensure that it meets certain requirements for efficiency and quality. The concept of life cycle in the software development process is widespread throughout the industry, but each company implements it according to its needs, as well as the needs of its customers. Inside each company, the process itself is tailored to each specific project.

Since the present work was part of a project of Critical Software, the procedures established by that company were followed. These procedures include the definition of the software development process [21], which is part of the quality management system of the company. The software development process is compatible with the standards used by ESA and is based on the ISO 12207 standard, which contributes to ISO 9001 compliant status attributed to Critical Software..

By following an established process, a team of software developers has a clear guideline on how to perform, from a high-level perspective. This provides for increased confidence for team members and reduces time spent planning and organizing. It ensures that the development process occurs in a standardized manner, with all the advantages brought by that. The final product – be it a software product or a maintenance activity, for example – will meet certain requirements in terms of quality,

and will perform accordingly to needs and expectations of the customer. This gives the customer confidence in the company and its products.

2.3 Phases of Software Development

The software development process implemented by Critical Software is divided into a series of phases, each with specific inputs and outputs, as well as objectives and milestones. The milestones are instants in time where the work produced in a phase is formally reviewed. The organization of the phases used by Critical Software is inspired in standards from the European Cooperation for Space Standardization, an effort involving ESA, national space agencies, and the European space industry [22]. The phases and milestones are (see Figure 7):

- Kick-off meeting (KOM): this milestone defines the beginning of the project.
- Requirements engineering phase: the specific objectives of the project are clearly stated in this phase, to a high level of detail. The preliminary design review (PDR) milestone defines the end of this phase.
- Design engineering phase: the solution that will answer to the requirements defined in the previous phase is clearly defined and implemented. This phase ends with the critical design review (CDR) milestone.
- Validation phase: the solution implemented in the previous phase is subjected to a series of tests that will determine whether it answers to the requirements defined in the requirements engineering phase. The milestone of this phase is the qualification review (QR).
- Acceptance phase: the goal of this phase is to test the implemented solution in the environment of the customer, and demonstrate that the solution meets his demands. The acceptance review (AR) milestone marks the end of this phase.
- Operations and maintenance: this phase starts as the product is accepted by the customer. During this phase, support is provided to the customer to ensure the correct functioning of the solution provided.

It is important to note that each phase represents only the main focus of the work performed at that time. The fact that a project is in a specific phase at a certain time does not prevent work related to other phases from being developed. It may even be developed in support of the current phase. Also, note that each phase may begin while the previous one is still taking place, as some work in each phase does not necessarily require the whole output of the previous phase.

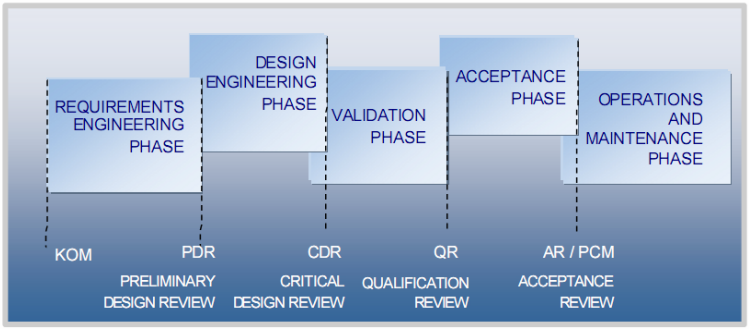

Figure 7: Phases and milestones of the software development process [21]

As previously mentioned, the software life cycle is tailored to the specifics of each project. For example, a project where maintenance is provided for an already existing piece of software will only have the operations and maintenance phase. All phases except for the acceptance and operations and maintenance phases were present in the project described in this thesis.

2.3.1 Requirements Engineering Phase

The requirements engineering phase begins with the kick-off meeting. Although every project has an objective, it is necessary to determine the exact specific functionality that is required from the software so that the objective will be met. The schedule and budget for the following phases is also defined at this time, as well as the implementation plans for the software to be developed. This mandates that the high-level architecture of the software be defined in this stage as well.

A fundamental objective of this phase is that both the customer and the design team fully understand and agree on the functionality to be offered by the software that is produced. The customer is the entity who commissions the production of the software and to whom the software is delivered – in the case of the present work, the customer is Critical Software itself, since it was an internal research project.

The output of the requirements engineering phase is a set of documents clearly stating the decisions made in this phase, as well as the motivation behind those decisions. The main document features the listing of the software requirements, usually in a tabular form, categorized according to software component and type of requirement. There are seven types of requirements that apply to the present work. They are thoroughly described in [23].

- Functional requirements specify a function that a system or system component must be able to perform.

- Performance requirements define certain computational performance parameters, preferably in numeric form, that a component must respect. A component that provides certain functionality is useless if it cannot provide that functionality when it is required by the system.

- Interface requirements mandate that a system be able to interact or communicate with certain hardware or software components. These can be databases or visualization tools, for example.

- Operational requirements focus on the interaction of the user with the system, specifying the functionality of the components of a system that are responsible for interacting with the user.

- Verification requirements apply to the design of the software, but are related to the validation phase. They may define that the software be tested in a particular way or that it provides the functionality for a particular test.

- Maintainability requirements affect the ease with which a change is made to the software, be it an upgrade, a bug fix, a change in behaviour in order to be compatible with another system, etc.

- Computer resource requirements specify the hardware platforms that the software shall support, as well as technical specifications of those platforms in terms of components.

The requirements are formally documented in table form. Each software component has a separate table for each of the types of requirements described above. Each row of the table represents a requirement. The columns of the table present a unique identifier for the requirement, a detailed description of the requirement, and indicate whether it is mandatory to implement it, how it shall be tested and how hard it will be to meet that requirement.

The work developed during this phase is documented in [24] and [25], and is described in Chapter 3.

2.3.2 Design Engineering Phase

The design engineering phase has the objective of producing the detailed software architecture and the corresponding code that will implement the requirements specified in the previous phase.

The detailed architecture specifies exactly how each software component is to be implemented, defining its characteristics as well as the processing that it will perform. This information comprises the software detailed design document. The following characteristics were identified as relevant for each software component implemented in the present work:

- Type: a brief identifier of the technology to be used to implement the component, e.g. C++ code class.

- Purpose: a very brief description of the purpose of the component.

- Function: an enumeration of the functionalities provided by the component, each described with some level of detail.

- Subordinates: the list of components that directly depend on this component for their normal function.

- Dependencies: the list of components that this component directly depends on for its normal function.

- Interfaces: a list of the interfaces used by the component, as well as the data that is transmitted through each interface, complete with the information necessary to implement the interface.

- Processing: a complete description of the processing performed in the component. The level of detail is as complete as it is possible, so that the developer may implement the component solely based on the information contained in the software detailed design document.

- Data: an enumeration of the internal data stored in the component that is relevant to understanding its operation.

Since the detailed architecture of a piece of software is closely linked to its implementation, it is only to be expected that these two activities occur simultaneously. As the architecture of a component is defined, the component is implemented in order to validate the architecture. This may lead to changes in the architecture, which in turn will reflect on the implementation. This iterative process takes place until a satisfactory solution is found.

The actual implementation of the software itself is only part of the process of software development, and, quite often, it is neither the hardest part nor the longest. The implementation must follow certain rules that define how the code is written in terms of style, since there are always many ways to implement a software component. These rules affect aspects of the code such as variable names and code organization, and impose restrictions on the use of certain functionalities provided by the programming language used.

Since different projects have different needs, it follows that each project may have its own set of coding standards. The C++ coding standards of Critical Software were followed in the present work

and are described in [26]. In certain applications, the standards are defined specifically for the project, as was the case with the C++ coding standards adopted for the development of software for the Joint Strike Fighter Air Vehicle [27].

The rationale behind the application of coding standards is that they lead to code that is reliable, maintainable, testable, and overall safer. These benefits come not only from the fact that the standards enforce better, proven ways of using certain aspects of the programming language, but also because the code will be easier to understand by the people who did not write it.

The work developed during this phase is documented in [28] and described in Chapter 4.

2.3.3 Validation Phase

The validation phase of a software project is not optional. Software that has not been validated has no use, since no guarantees exist that it correctly meets the purposes it was created for. Validated status is achieved when there is clear evidence that the software meets the requirements laid out in the requirements engineering phase. This evidence is given through the results of a series of tests performed on the software.

The tests to be executed are clearly defined in terms of which requirements they test, what environment they must be performed in, and the procedure that is to be taken, as well as the expected outcome of the test. The result of each test will therefore be classified as “passed” or “failed”. If the latter occurs, modifications will be made to the architecture and implementation and the test will be run again, until the result is as expected.

The software testing procedure implemented by Critical Software specifies four types of tests [29]. Although the four types are executed mostly in sequence, there can be some overlapping between them, as shown in Figure 8.

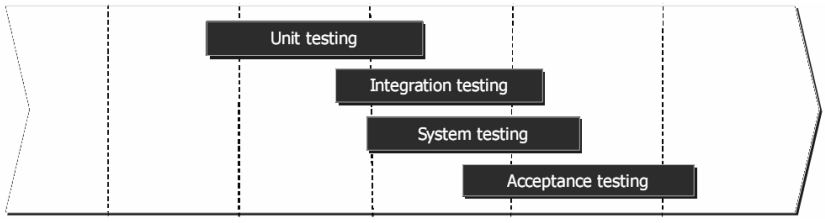

Figure 8: Timeline of the software testing procedure [29]

The types of tests are:

- Unit testing is the testing of each basic software component, in order to verify that it has been implemented according to the architecture specification. This type of testing often occurs while the design engineering phase is still ongoing.

- Integration tests are performed on groups of software components, so that the interaction between the components is tested.

- System testing verifies whether the software, as a whole, meets the requirements laid out in the requirements engineering phase.

- Acceptance testing occurs at the same time as the acceptance phase (see Section 2.3) and often consists of the execution of a subset of system tests in the presence of the customer. Its only purpose is to demonstrate the compliance of the software to the customer and was not performed on the present work.

The work developed during this phase is documented in [30] and described in Chapter 5.

2.4 Synopsis

An overview of the software development life cycle as implemented by Critical Software was presented, as well as the tailoring performed in the scope of this work. The various phases that were relevant to the project were referred.

The need for a clear definition of the objectives of the software was established in the description of the requirements engineering phase. The process of defining the architecture of the software and implementing it was presented in the section dedicated to the design engineering phase. The justification for the use of coding standards was also presented in this section. Finally, the motivation behind software testing and an overview of how it is conducted and divided into testing phases was established in the description of the validation phase.

Chapter 3

Requirements Analysis

3.1 Introduction

The requirements for the development of SimUAV are briefly described in this chapter, as well as the motivation behind each of them. The requirements are not presented in table form (as they were documented, see section 2.3.1) due to limited space. The actual implementation of the requirements and the technical solutions associated are presented in Chapter 4.

The definition of the requirements establishes in detail how the objective of a generic, SMP compliant UAV simulator can be achieved. An overview of the requirements of the simulator as a whole is presented, followed by the technological requirements for the development. These technological requirements influence the decisions taken in the high-level architecture, which is also described. A brief overview of the requirements for each of the simulation models is given, as well as for the two interfaces established for visualization of simulation parameters. Finally, the requirements for the forest fire monitoring demonstration scenario are presented.

3.2 UAV Simulator Requirements Overview

As described in Chapter 2, the requirements engineering phase of the software life cycle consists on the identification of the specific objectives of the project. The fundamental objective for the project was the development of a generic, SMP compliant UAV flight simulator, but it was necessary to determine which functionalities the simulator would support. The design and engineering of the software to implement the requirements is described in Chapter 4.

From a high-level perspective, it was defined that it would be possible for the user to specify a flight plan for the UAV based on waypoints, each with a specific latitude, longitude, and altitude, and have the simulated UAV follow that flight plan. The UAV would also be able to loiter over a specified area, as well as following discrete commands such as climbing to a certain altitude. Additionally, the

applicability of the simulator to the scenario of fire fighting support was to be demonstrated by having the UAV follow a simulated fire front.

All communications between the user and the UAV would take place through a data link compliant with the NATO STANAG 4586 standard [31]. In the scope of the same Critical Software project, but not of this thesis, a STANAG 4586 UAV console was developed for interacting with the UAV flight simulator [19].

3.3 Technology Requirements

3.3.1 Simulation Model Portability Standard

The involvement of ESA in space simulation throughout the years has produced several simulators for its missions. These simulators are used for research, engineering operations preparation, and training. The models that comprise them are often developed in different operating systems and for running in different simulation platforms. A simulation platform is the software responsible for managing the running of a simulation and coordinating the models. As such, the models developed for a particular application are often incompatible with each other. This greatly limits the possibility of reusing them in other missions.

A new simulator often requires models that have already been implemented in another simulator, such as the model for the dynamics of a satellite. A great deal of effort could be saved if these models were reused in the development of other simulators for different missions, which requires compatibility between them. The Simulation Model Portability (SMP) standard was developed in order to standardize the architecture of the models of a simulation, enhancing their reuse [32].

The SMP provides a standard interface called the Simulation Model Interface (SMI). The SMI provides a minimal set of services for the models and enables them to be fully integrated into simulation environments such as SIMSAT, ROSE, and EuroSim. The general concept for an SMP-compliant system, using the SMI, is illustrated in Figure 9.

The SMI effectively acts as an adapter between the models and the simulation environment. The models interact with the SMI and the SMI interacts with the simulation environment. The SMI has been specifically designed to be compatible with as many of the existing simulation environments as possible.

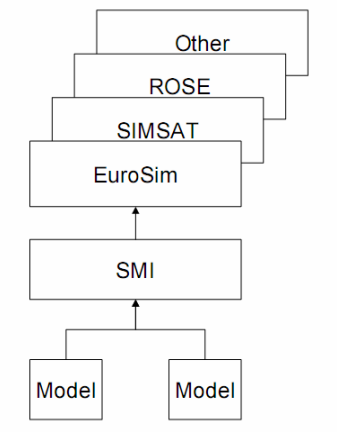

Figure 9: SMP compliant interaction of simulation models and platforms [32]

Supporting the SMP standard in a UAV flight simulator enables it to run on a number of simulation platforms, to be reused in other projects, to be enhanced by adding existing SMP-compliant models, and to be used alongside other SMP-compliant simulators. Furthermore, the compatibility with a popular standard reduces development risk and cost since the standard defines proven processes, components and interfaces.

From the perspective of model interaction, SMP provides two functionalities: publishing and services. Publishing is the providing of a simulation parameter to the simulation as a whole. The parameter is available for any model requiring it. Services are operations or procedures that a model may execute. This execution may be triggered by the other models, and may include the transfer of data between the models.

3.3.2 SIMSAT

The SIMSAT simulation platform version 3.0 was selected for use in this project because it is provided free of charge by ESA to the European space industry. It is used by ESA in space simulation applications and is an established simulation platform. Version 3.0 is the latest and most advanced version of SIMSAT. It supports the SMP standard, which is a fundamental requirement for this project.

SIMSAT is a software product that can be separated in two components: the man-machine interface (MMI) and the kernel [3]. The kernel is the engine of the simulation, taking care of the low level tasks such as creating and running processes, storing and reading data, providing standard simulation services used by most simulations (such as message logging), scheduling events and managing time. The kernel takes care of loading models and providing the MMI with the data it requests.

The MMI is the interface between the user and the simulation. For the user, it is the visible part of SIMSAT. It is used to control and study the running system during development and use. More than one MMI may be attached to a simulation at any time, and each MMI does not need to be located on the same machine as the kernel. Each MMI communicates only with the kernel and does not communicate directly with any models in the simulation.

Figure 10 shows the relationship between the kernel, the MMIs, and the simulation models. In the figure, objects with white background represent the infrastructure provided by SIMSAT. Objects with grey background identify mission-specific components which are not provided by SIMSAT but may be developed and integrated into the simulation environment. The simulation models fall into this category. Finally, objects with black background are the basic simulation models provided with SIMSAT. These models are included in SIMSAT because they are used by many simulations, but are entirely optional. They were not used in this project.

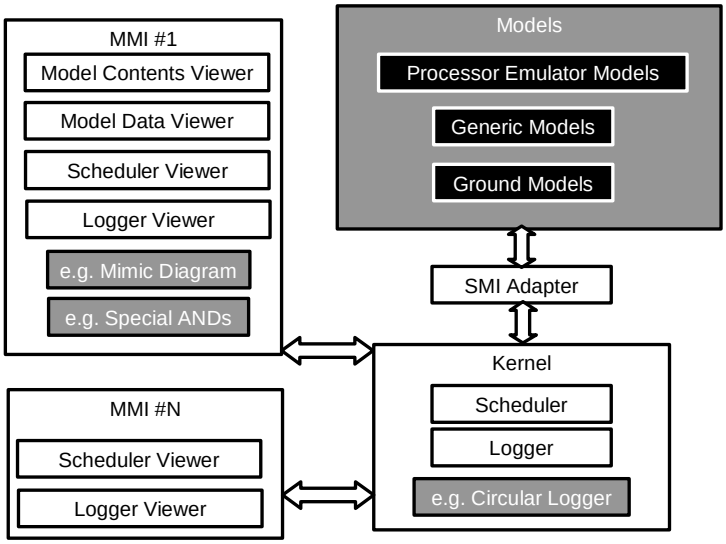

Figure 10: Relationship between SIMSAT simulation components [3]

By providing a standard simulation framework (kernel) and a user interface (MMI), SIMSAT allows the simulation developers to concentrate on the developing the models, not having to worry about the simulation platform. Figure 11 is a screenshot of the SIMSAT MMI, displaying the list of simulation models to the left and a graphical data viewer to the right. The MMI provides the user with a series of functionalities that allow the interaction with the simulation. The most relevant of these, for the common user, are described in Table 1.

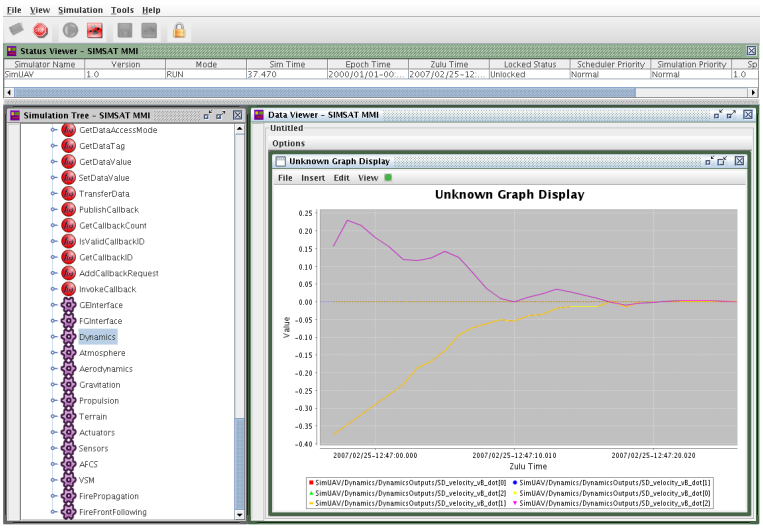

Figure 11: Screenshot of SIMSAT MMI

Functionality	Description
Open a simulation	The MMI must open a simulation before it is able to interact with it. As stated before, the simulation may be running on another computer.
Start, stop, pause and resume	These allow the user to control the running of the simulation.
Change speed factor	It is possible to accelerate or slow down the operation of a simulation.
Simulation tree	The tree provides a graphical display of the various models that comprise a simulation, indicating their respective variables, as well as the functionality they offer.
Log and message viewers	These provide a means for the user to monitor the operation of the simulation.
Commander	The commander is used to interact directly with the models that provide such functionality.
Data viewer	It is possible to visualize the data provided by each model in numeric or graphical form.

Table 1: Relevant functionalities provided by SIMSAT

3.4 High-Level Architecture

The requirements engineering process of the software development life cycle, as previously mentioned, mandates that the specific functionalities to be offered by the simulator be described in the form of a detailed set of requirements. In order to define the requirements in a detailed fashion, it is necessary to first determine the high-level architecture of the software.

The two fundamental requirements for SimUAV were compliance with the SMP standard and promotion of modularity and versatility of the software. The verification of these requirements is promoted by the encapsulation of each reality in a model that is as independent from other models as it is possible. Each model will receive the data it needs from other models, simulate the reality it represents by processing the data, and output the values of that processing.

The first task in the definition of the high-level architecture was the identification of the various simulation models, while promoting independence between them. The modules identified are displayed in Figure 12.

Although this does not reflect in the implementation, three components are identified for the sake of simplicity: the Air Component, the Visualization Interface Component, and the Environment Component. A detailed description of the models implemented in this work is given in Sections 3.5 through 3.7, while an overview of each is presented below.

- Dynamics: this model has the responsibility of receiving the various forces and torques applied to the UAV and determining the evolution of the position of the UAV through time. It must also report the values of certain parameters such as velocities and accelerations.

- Aerodynamics: it is the responsibility of this model to simulate the aerodynamic force and torque acting on the UAV. It does that from the current UAV states that it receives from the Dynamics model and from specific UAV aerodynamic data.

- Propulsion: the propulsion model simulates the force and torque generated by the UAV propulsion system.

- Gravitation: effects of gravitational attraction are simulated in this model, which provides its results to the Dynamics model.

- Terrain: this model simulates the existence of the ground by determining the force and torque generated by the landing gears. The Dynamics model uses these results in its calculations.

- Atmospheric model: an International Standard Atmosphere (ISA) is simulated by this model, along with the wind. The results are used throughout the simulation.

- Fire Propagation model: this model has the purpose of implementing a simple probabilistic algorithm for the propagation of fire in large areas. This model was developed in order to support a demonstration scenario.

- Fire Front Following: based on the results of the Fire Propagation algorithm, this model determines a flight plan that will result in the UAV following the front of the fire.

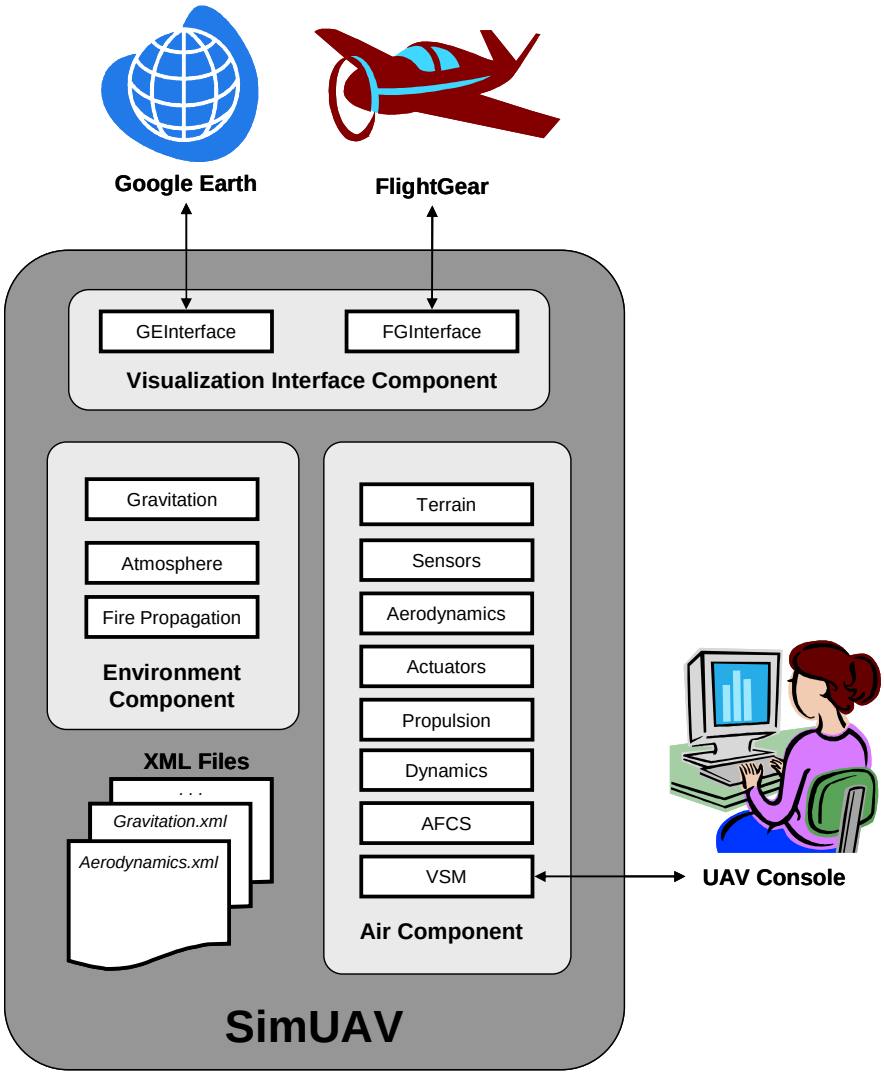

Figure 12: High-level architecture of SimUAV

As mentioned in Section 1.3, other models are necessary for the complete UAV simulator and were implemented in the scope of the same Critical Software project as the present work [19]. They are briefly described here:

- Automatic Flight Control System (AFCS): this model implements the on-board flight computer. Its functionality involves the implementation of a flight controller for the UAV, but also the management of the commands issued by the operator.

- Vehicle Specific Module (VSM): the task of the VSM is to establish the communications between the UAV and its control station, which are standardized by the NATO STANAG 4586 document [31].

- Actuators: the actuators of the UAV are simulated by this model. The values of the positions of the rudder, ailerons, and elevators are provided to the Aerodynamics model.

- Sensors: various parameters are gathered by this model, which simulates the functionality of their respective sensors and provides them to the models that require them, such as the AFCS model.

Finally, it was determined that two visualization interfaces would be implemented so that the results of the simulation could be easily observed.

- Google Earth interface: an interface with this popular viewer for Earth maps allows for visualizing the exact position of the UAV in the planet, in terms of latitude, longitude, and altitude.

- FlightGear interface: this open-source flight simulator can be used solely for visualization purposes, displaying the position and attitude of the UAV.

Also shown in Figure 12 are Google Earth and the FlightGear flight simulator, as well as the XML (Extensible Markup Language) files that hold simulation data such as aircraft inertial properties and aerodynamic data. XML is a general purpose markup language, meaning that it can hold generic data in a format that is easy to interpret by a computer. XML was chosen for the storage of data in this project due to its versatility, popularity, and cross-platform compatibility.

3.5 UAV Flight Models

The UAV flight models developed in this work are the Dynamics, Aerodynamics, Propulsion, Gravitation, and Atmospheric models. An overview of the specific requirements for each of these components is provided in the sections that follow.

There are, however, requirements that apply to all models. The compatibility with SMP is mandatory, as is the respect for the Critical Software code conventions. The models, as a whole, must provide a baseline for the simulation of the flight of a UAV in the scenario of fire monitoring. They must be independent of the UAV platform, reading specific UAV data from XML files, and modifiability and scalability of the simulation must be promoted.

The simulated UAV must follow discrete commands, e.g. climb to a certain altitude or fly at certain airspeed. It must also follow waypoints and loiter over a specified point. It is also necessary to perform a certain set of operations on the simulation, such as starting and stopping it, visualizing the parameters of the models, executing commands, logging generic simulation data, etc. All this functionality is immediately offered by SIMSAT.

Regarding the common functionality of the models, it was determined that each one would encapsulate the functionality necessary for its operation as much as possible, having a minimum of interaction with other models. The models must provide error messages and information about their internal operation as necessary.

3.5.1 Dynamics Model

The Dynamics model is responsible for receiving forces and torques applied to the UAV by the Aerodynamics, Propulsion, Gravitation, and Terrain models. It will determine the evolution of the position of the UAV through time, reporting to other models the values of certain parameters such as velocities and accelerations.

Since this model can be considered the heart of the simulation, it was decided that it would be as generic and as a powerful as possible, allowing for its future reuse. Therefore, the UAV is modelled as a rigid body according to the formulation in [33]. The user shall be able to specify the use of the explicit Euler or 4^{th} order Runge-Kutta methods for propagation of the states in time. The implementation of other methods shall be supported, and will require minimal changes to the Dynamics model.

UAV data such as inertial properties and initial states shall be read by this model from an XML file.

3.5.2 Aerodynamics Model

The resulting Aerodynamic force and torque acting on the UAV at each instant is provided by this model to the Dynamics model. In order to determine these parameters, the Aerodynamics model must obtain the current flight conditions from the Dynamics model, the actuator positions from the Actuators model, and the wind velocity and local air density from the Atmospheric model.

In order to improve the value of simulation results by increasing their accuracy, both linear and non-linear aerodynamic data will be supported. This data will be described in XML files.

3.5.3 Propulsion Model

This model simulates the behaviour of the UAV propulsion system by providing the Dynamics model with the value of the generated force and torque. The value of the current desired throttle setting is read from the AFCS model.

Due to the wide range of UAV engines and propellers available and the complexity associated with their modelling, it was decided that, in this work, a simple propulsion model would be implemented in the form of a thrust force that is the product of the throttle setting (in percent) by the maximum thrust generated by the UAV being simulated. This maximum value is read from an XML file. To simulate a delay in the response of the engine, the first order lag filter described in [19] is used, with the time constant read from an XML file.

3.5.4 Gravitation Model

The Earth gravitational attraction is simulated by this model. The UAV is considered as a point mass, since the alternative of considering higher terms of the gravitational potential is important in the case of satellites but not of UAVs. The gravitational attraction force is calculated and sent to the Dynamics model.

In order to improve simulation accuracy, an algorithm based on the Earth Gravitational Model 1996 (EGM96), referred by [33] as delivering a very good level of accuracy, is implemented. In this case the algorithm uses only the largest coefficient of the potential function of EGM96. This algorithm is used instead of considering the Earth as a concentrated mass in order to promote the reuse of the Gravitation model in future applications requiring this level of precision.

3.5.5 Terrain Model

The Terrain model is responsible for providing the Dynamics model with the total force and torque generated by the UAV landing gears. For this, it must read from the Dynamics model the current position of the UAV and must have information available on layout of the terrain where the UAV is intended to land. The ultimate goal of this model is to allow the possibility of simulating take-off and landing in specified runways.

To simulate the landing gears, this model shall read from an XML file the necessary data such as landing gear spring and damping constants, as well as position.

3.5.6 Atmospheric Model

The values of the atmospheric parameters are necessary throughout the simulation. Current density is used by the Aerodynamics model, as well as by the Sensors model. Temperature and pressure are also used by the Sensors model. An ISA model is implemented in order to obtain these parameters. For this, the value of the altitude of the UAV is read from the Dynamics model.

The existence of wind in the simulation is performed by reading a constant wind speed and heading from an XML file. This is used by the Aerodynamics model.

3.6 Visualization Interfaces

The visualization interfaces established with Google Earth and FlightGear allow the simulation user to better understand the results of the simulation. Without these interfaces, the user would be limited to observing values and graphs in the SIMSAT MMI. The interfaces implemented are extremely powerful and were an invaluable tool in the development and testing of the simulator.

3.6.1 Google Earth Interface

Google Earth is a popular 3D mapping application that provides detailed satellite imagery of the Earth (see Figure 13). The user can interact with the software by placing icons called placemarks on the map, marking a certain position for future reference. It is also possible to draw a path on the map and to place an image over a specified area. Other features are supported but not relevant for the present work.

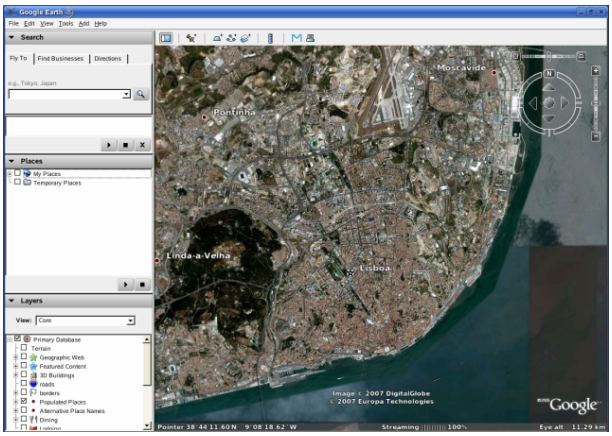

Figure 13: Screenshot of Google Earth

In this work, the position of the UAV at each instant will be automatically displayed on the map by a placemark, and a path will be drawn indicating the trajectory of the UAV over time. This data will be retrieved by the Google Earth interface model from the Dynamics model. It will also be possible for the user to manually set a placemark defining the initial position of the UAV in the simulation in terms of latitude, longitude, and altitude. For the definition of a flight plan for the UAV, the user will be able of setting several placemarks in a specified order, with each placemark defining a waypoint. Finally, the data from the Fire Propagation model will also be shown on Google Earth for the purposes of validating the Fire Front Following model. The interface between SimUAV and Google Earth will be established through Google Earth KML files, which are a language based on XML.

3.6.2 FlightGear Interface

FlightGear (see Figure 14) is an open-source flight simulator released under the GPL license [34]. For the average computer gamer, FlightGear is similar in functionality to the popular Microsoft Flight Simulator [7], although lacking advanced graphics. It is, however, much more versatile, because of its open-source nature.

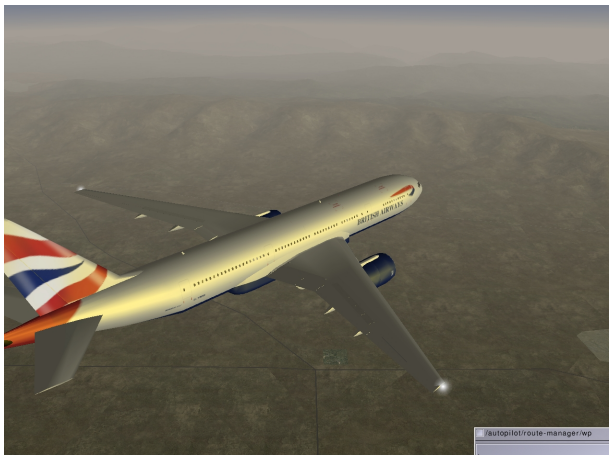

Figure 14: Screenshot of FlightGear [18]

Although the authors refer to FlightGear as a flight simulator, it concentrates on providing the graphics and the user interaction, leaving the simulation of the physics of the vehicle (dynamics, aerodynamics, etc.) to an external piece of software such as JSBSim, which calculates vehicle position, attitude, etc. and provides it to FlightGear [35].

JSBSim is a collection of simulation models for the various aspects of the flight of an aircraft, brought together under an extremely simple simulation engine. Although JSBSim is an open-source project released under the LGPL license [36] (which allows for more freedom than the GPL license), its codebase was not considered for reuse in SimUAV due to the fact that is not compatible with the requirements for modularity, portability and versatility that form the basis of the present project.

SimUAV will therefore be used as a replacement for JSBSim in FlightGear, providing UAV position and attitude at each instant, allowing the visualization of the flight of the UAV. Communication between SimUAV and FlightGear shall take place through a User Datagram Protocol (UDP) connection, which is supported by FlightGear.

3.7 Application Scenario: Forest Fire Monitoring

A civil application for UAVs that is rising in popularity is the support of fire fighting activities through the monitoring of forest areas. Examples of applications of UAVs in this scenario are described in [10] and [20]. UAVs can provide for effective surveillance of large areas, as mentioned in Section 1.1, warning of fires in their early stages as well as keeping the fire-fighters in the field up to date with the evolution of the fire front, allowing them to better coordinate their efforts and improving their efficiency.

3.7.1 Fire Propagation Model

This component shall be responsible for simulating the propagation of a fire in an area, using a very simple algorithm. The algorithm selected was the EMBYR model [37], which is adequate for use in large-scale fires. EMBYR does not simulate the physics behind a forest fire, instead dividing an area into cells and relying on a probabilistic model for the propagation of fire from cell to cell.

EMBYR is used instead of a more complex model because the actual simulation of fire behaviour is not the essence of this project. The fire propagation data is only necessary for the purposes of the Fire Front Following model.

The data generated by this model shall be provided to the Fire Front Following model as well as to the Google Earth interface, for visualization purposes.

3.7.2 Fire Front Following Model

Based on the data generated by the Fire Propagation model, this model shall determine, at each instant, a waypoint the UAV must fly to in order to follow the front of the fire. The path of the UAV will be a line approximating the shape of the front.

The generated waypoint shall be sent to the AFCS model, which will command the UAV to fly to that location. As the UAV moves and updates its visualization of the fire front area, new waypoints are generated, which results in the UAV being continuously pursing a moving waypoint. The main objective of this model is to demonstrate the applicability of SimUAV in a specific scenario.

3.8 Synopsis

A description of the requirements for the development of SimUAV was presented, as was the motivation behind those requirements. The high-level architecture of the simulator was identified and each simulation component was described in terms of the functionality it must deliver.

The SMP standard and the SIMSAT platform were introduced and addressed in the aspects relevant for this work.

Chapter 4

Architecture Design and Model Specification

4.1 Introduction

The detailed aspects of the implementation of SimUAV are presented in this chapter. The architecture that responds to the requirements established in Chapter 3 is described, followed by the implementation features that are common to all models.

An overview is given of the processing that is executed in each simulation model and the interactions between the various models are enumerated so that a global understanding of the operation of SimUAV is possible.

4.2 Detailed Architecture

As mentioned earlier, the detailed software architecture and its corresponding code are the outputs of the Design Engineering phase. The implemented code will answer the requirements specified in the requirements engineering phase.

Figure 15 presents an overview of the detailed architecture of SimUAV, which reflects the implemented code. This figure is a detailed version of Figure 12, which describes the high-level architecture of the simulator.

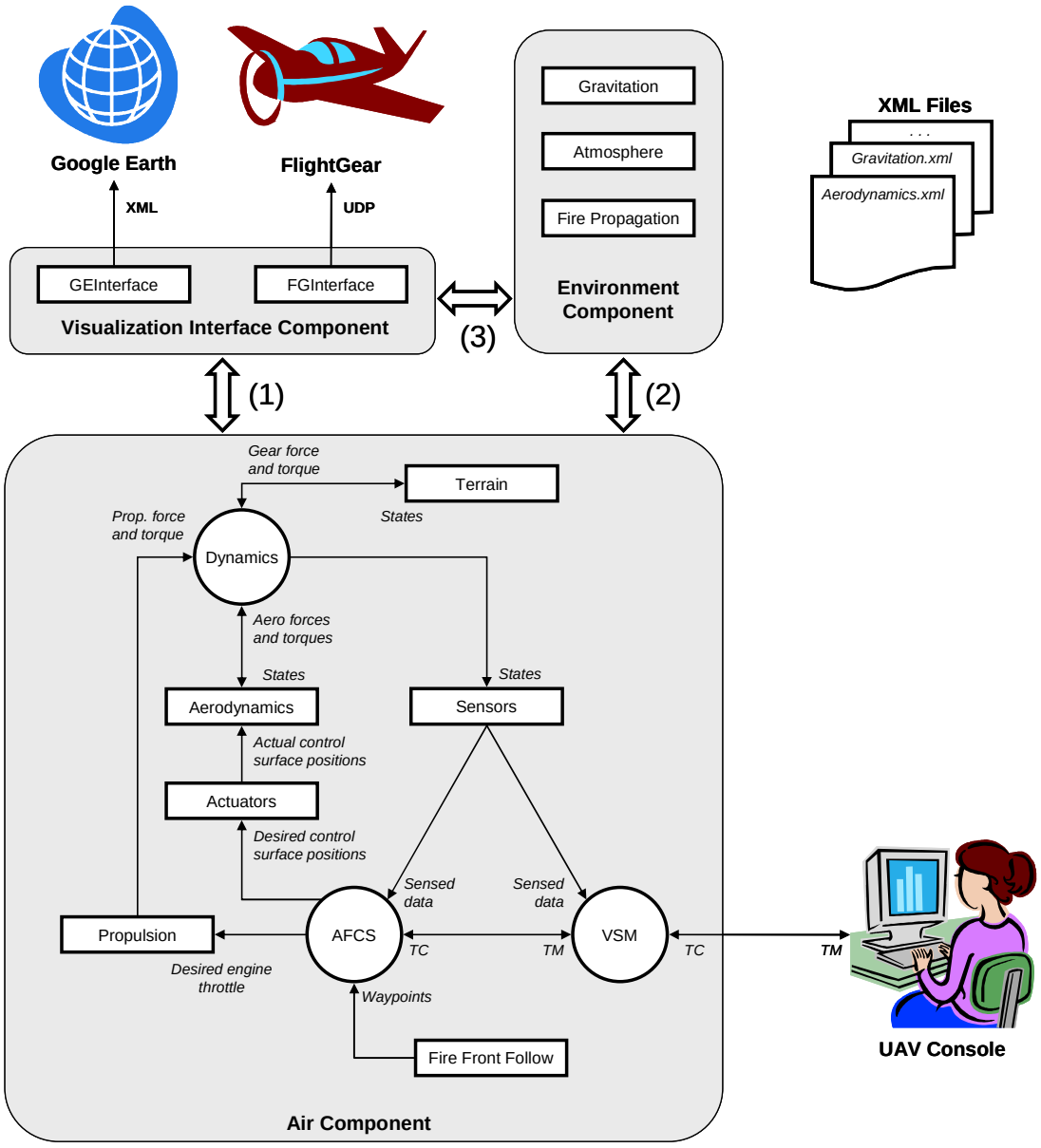

Figure 15: Detailed architecture of SimUAV

As was the case with Figure 12, three components are identified for the sake of simplicity: Air Component, Visualization Interface Component, and Environment Component. The most important factors influencing this architecture were:

- The requirements established in the requirements engineering phase, which include the support for SMP, the requirements for modularity, the decision on which models to implement, etc.

- The process of implementation itself, which naturally yields a better understanding of the problem at hand, changing previous architectural decisions. This results in an iterative process between architecture definition and code implementation until a satisfactory solution is reached.

As was also the case with Figure 12, Google Earth, FlightGear, and the XML files used for loading the simulation data are also shown. The UAV console developed in [19] is also depicted, and, as mentioned before, it will establish a STANAG 4586 compliant data link to the VSM model. This link, as per STANAG 4586, will support the TCP/IP and UDP protocols, but is not part of the scope of this thesis.

In contrast with Figure 12, the interfaces between the various models have now been identified. In compliance with one of the main requirements defined, these interfaces are SMP compliant. The white arrows marked with (1), (2), and (3) in Figure 15 represent interfaces between the Air Component, Environment Component, and Visualization Interface Component. The grouping of models into these three components is merely for clarity of the image, and it is not reflected in the implementation. Therefore, each white arrow simply stands for a set of normal, SMP compliant interfaces between several models. The interfaces that arrow (1) stands for are depicted in Figure 16.

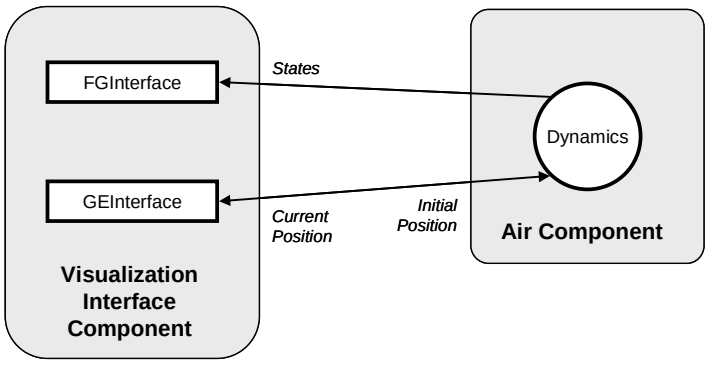

Figure 16: Interfaces between Visualization Interface and Air Components

Arrow (2) represents the interfaces in Figure 17.

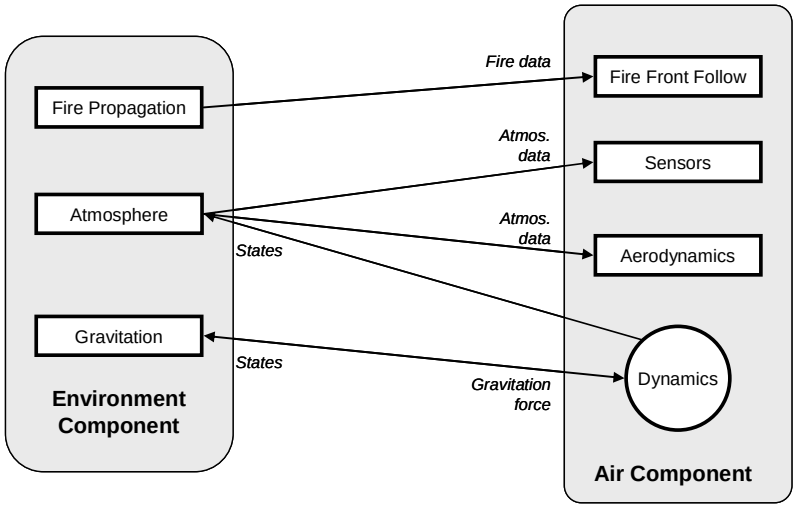

Figure 17: Interfaces between Environment and Air Components

Finally, Figure 18 describes the interfaces that arrow (3) stands for.

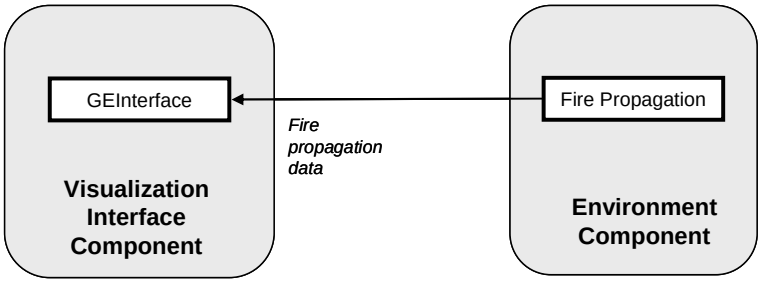

Figure 18: Interface between Visualization Interface and Environment Components

4.3 Common Model Specification

The requirements of SMP compliance, modularity, and versatility apply to all simulation models. Therefore, it is only to be expected that there are some common implementations in all models, especially for the SMP compliance code. Initial experimentation with SMP focused on establishing a common model specification. This specification consisted, in essence, of a generic SMP compliant template model which would serve as the basis for the creation of each new model.

An opportunity for the promotion of modularity and reuse was identified at the stage of definition of the architecture of this template model. There is an updated version of the SMP standard called SMP 2. It was not used in this project because it is not supported by SIMSAT 3.0, but it will be supported

by the upcoming SIMSAT 4.0. The common model specification was therefore engineered towards promoting a future reuse of the developed models in an SMP 2 environment.

This required encapsulating the physical reality of each model in a C++ class that does not make use of any simulation standard. This class is referred to as an “implementation class”. It will require an adapter that supports the SMP standard, establishing an interface between implementation of physical reality and SIMSAT. If the model is to be made compatible with another standard, such as SMP 2, little or no changes will have to be made to the implementation class. Only the adapter class will have to be replaced. This architecture is illustrated in Figure 19.

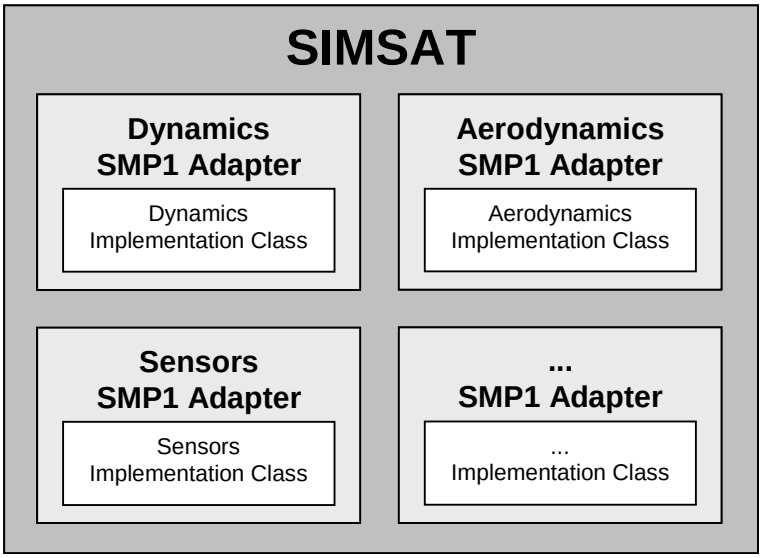

Figure 19: Common model architecture

The common model architecture is based on two C++ classes: one for the implementation and another for the adapter. This will further improve the reuse capabilities and longevity of SimUAV, by supporting the most modern simulation standards and environments.

4.4 Detailed Design

In the following sections, the implementation of each component of the simulation is presented. An overview of the objective of the component is given, followed by the fundamental formulation that is at the core of its functionality. Finally, a description is given of the processing executed upon initialization of the component and at each simulation time step. Physical quantities are in SI units, except where indicated.

4.4.1 Dynamics Model

Overview

The Dynamics model has the responsibility of receiving the various forces and torques applied to the UAV, determining the evolution of the position of the UAV through time and reporting the values of certain parameters such as velocities and accelerations. This model can be considered the heart of the simulation. For this reason, it was implemented as generic and as a powerful as it was possible, promoting its future reuse.

Formulation

The Dynamics model implements the algorithm described in [33]. Only an overview of the implementation is given here. First, it is necessary to introduce the four coordinate frames used in this model:

- ABC: Aircraft Body Centred Frame: this frame is centred at the centre of gravity of the UAV and has a fixed alignment relative to the body of the UAV.

- ECI: Earth-Centred Inertial Reference Frame: this is an inertial frame, centred at the Earth centre of mass, with one axis along the Earth spin axis and the remaining two in the equatorial plane. The frame does not rotate with the Earth. It is used for including the effects of the Coriolis force in the model, which is in accordance with the aforementioned objective that the Dynamics model be as generic and powerful as possible, since Coriolis effects must be accounted for in long-range flights (e.g. because of fuel consumption effects).

- ECEF: Earth-Centred Earth Fixed frame: similar to ECI, but rotates with the Earth.

- NED: North-East-Down Frame: this serves as an intermediate frame between the ECI and the ABC. It is useful for understanding the coordinate transformation between the ECI and the ABC, and also to visualize the attitude of the aircraft relative to the surface of the Earth.

The translational and rotational motion of the UAV is described by a set of four equations. These equations are referred to as state equations. Their derivation is described in [33] and a brief overview of each is presented next. Equation (4.1) represents the attitude of the ABC frame (represented by the letter '*b*') relative to the ECI frame (represented by the letter '*i*')

$$\dot{\mathbf{q}}_{b/i} = \frac{1}{2}\,\boldsymbol{\Omega}_q\,\mathbf{q}_{b/i} \tag{4.1}$$

where $\mathbf{q}_{b/i}$ is the quaternion representing the attitude between the ABC and the ECI frame. Attitude is often expressed in the form of Euler angles but these offer little advantage over quaternions and are extremely error-prone because of the singularities that take place e.g. for 90º pitch angle. Using quaternions promotes the implementation of the specific requirement of a powerful and generic Dynamics model. Additionally, using quaternions greatly helps in the development process since it allows unit tests (see Section 2.3.3) to be performed without worries of reaching critical values of Euler angles. The term $\mathbf{\Omega}_q$ in Equation (4.1) is given by

$$\mathbf{\Omega}_q = \begin{bmatrix} 0 & -P & -Q & -R \\ P & 0 & R & -Q \\ Q & -R & 0 & P \\ R & Q & -P & 0 \end{bmatrix} \tag{4.2}$$

where P, Q, and R are the three components of $\boldsymbol{\omega}_{b/i}^{b}$, representing the angular velocity of the ABC frame relative to the ECI frame. The position of the ABC frame relative to the ECI frame, in ECI frame components, $\mathbf{p}_{CM/O}^{i}$, is given by

$${}^{i}\dot{\mathbf{p}}_{CM/O}^{i} = \mathbf{C}_{i/b}\mathbf{v}_{CM/e}^{b} + \boldsymbol{\omega}_{e/i}^{i} \times \mathbf{p}_{CM/O}^{i} \tag{4.3}$$

where $\mathbf{C}_{i/b}$ is the transformation matrix from ABC to ECI frame, $\mathbf{v}_{CM/e}^{b}$ is the velocity of the ABC frame relative to the NED frame, and $\boldsymbol{\omega}_{e/i}^{i}$ is the rotational velocity of the Earth, relative to the ECI frame. The velocity of the ABC frame relative to the NED frame is

$${}^{b}\dot{\mathbf{v}}_{CM/e}^{b} = (1/m)\mathbf{F}_{A,T}^{b} - \left(\boldsymbol{\omega}_{b/i}^{b} + \mathbf{C}_{b/i}\boldsymbol{\omega}_{e/i}^{i}\right) \times \mathbf{v}_{CM/e}^{b} + \mathbf{C}_{b/i}\mathbf{g}^{i} \tag{4.4}$$

where $\mathbf{F}_{A,T}^{b}$ is the resultant force applied to the UAV, in ABC coordinates, $\boldsymbol{\omega}_{b/i}^{b}$ is the angular velocity of the ABC frame relative to the ECI frame (as previously mentioned), and $\mathbf{g}^{i}$: gravitational acceleration vector. The angular velocity of the ABC frame relative to the ECI frame is

$${}^{i}\dot{\boldsymbol{\omega}}_{b/i}^{b} = \left(\mathbf{J}^{b}\right)^{-1}\left[\mathbf{M}_{A,T}^{b} - \boldsymbol{\omega}_{b/i}^{b} \times \left(\mathbf{J}^{b}\boldsymbol{\omega}_{b/i}^{b}\right)\right] \tag{4.5}$$

where $\mathbf{J}^{b}$ is the inertia matrix of the body and $\mathbf{M}_{A,T}^{b}$ is the resultant torque applied to the body, in ABC coordinates. The state equations are propagated in time using either the explicit Euler

(implemented because of its simplicity) or the 4th order Runge-Kutta method [38] (implemented because of its popularity). The choice of which algorithm to use among all that apply is beyond the scope of this thesis. What falls in the scope of this thesis is the modularity implemented in the state equations which allows for easy implementation of other methods, which will be specified in future applications given to this model.

The algorithm implemented can be summed up in the following sequence of steps:

- Resulting force and torque is determined from the various contributions, except for gravitational attraction, which is handled explicitly in the state equations.
- State equations are propagated in time.
- $\mathbf{C}_{b/i}$ transformation matrix (transforms from ECI to ABC) is calculated from the $\mathbf{q}_{b/i}$ quaternion obtained from Equation (4.1).
- UAV latitude, longitude, and altitude are calculated from $\mathbf{p}^{i}_{CM/O}$ obtained from Equation (4.3).
- $\mathbf{C}_{n/i}$ transformation matrix (transforms from ECI to NED) is calculated from latitude and longitude.
- $\mathbf{C}_{b/n}$ transformation matrix (transforms from NED to ABC) is calculated from $\mathbf{C}_{b/i}$ and $\mathbf{C}_{n/i}$.
- The Euler angles are calculated from the $\mathbf{C}_{b/n}$ matrix. No calculation is performed using Euler angles, they are necessary only for visualization.

Processing

Upon initialization, UAV data such as inertial properties and initial states shall be read by this model from an XML file. At each time step, the values of generated force and torque are read from the following models: Aerodynamics, Propulsion, and Terrain. The value of gravitational acceleration is read from the Gravitation model. The algorithm described above is used to determine the evolution of the states of the UAV in time.

4.4.2 Aerodynamics Model

Overview

Aerodynamics: this model is responsible for simulating the aerodynamic force and torque acting on the UAV based on current UAV states and specific UAV aerodynamic data.

Formulation

The mathematical formulation implemented in the Aerodynamics model is based on [33], and a brief overview is given here. The notation used in [33] applies.

The first step in the determination of aerodynamic forces and torques is the inclusion of wind, which results in the determination of the velocity of the UAV relative to the air mass. The aerodynamic angles are calculated in the next step. Each component of the resulting aerodynamic force or torque is calculated in a similar fashion. The three components of the aerodynamic force (lift, *L*, drag, *D*, and cross-wind, *C*) are

$$\begin{bmatrix} L \\ D \\ C \end{bmatrix} = qS \begin{bmatrix} C_L \\ C_D \\ C_C \end{bmatrix}. \tag{4.6}$$

The three components of the aerodynamic torque (rolling moment, *l*, pitching moment, *m*, and yawing moment, *n*) are

$$\begin{bmatrix} l \\ m \\ n \end{bmatrix} = qS \begin{bmatrix} b\, C_l \\ \bar{c}\, C_m \\ b\, C_n \end{bmatrix}. \tag{4.7}$$

In Equations (4.6) and (4.7), q represents the dynamic pressure, S represents the wing surface, b represents the wingspan, and $\bar{c}$ represents the mean aerodynamic chord. Each of the six aerodynamic coefficients remaining in Equations (4.6) and (4.7) are specific to the UAV and may vary with the flight conditions. SimUAV supports both linear and non-linear models for the determination of these coefficients. In the linear model, each coefficient is (or *can be*) a function of a constant value plus linear contributions from angle of attack α, roll rate p, aileron deflection δ_a, etc. A given coefficient C_x can be calculated from

$$C_x = C_{x_0} + C_{x_\alpha}\alpha + \ldots + C_{x_p} p \frac{b}{2V_T} + \ldots + C_{x_{\delta_a}} \delta_a + \ldots \tag{4.8}$$

In Equation (4.8), V_T represents the modulus of the velocity of the UAV relative to the air mass. Other contributions that can affect the determination of C_x, such as the aerodynamic angle β, were left out of Equation (4.8) for clarity. The coefficients C_{x_y} are assumed constant for the UAV, under all flight conditions. In the non-linear model, the C_x coefficients are looked up in a table, according to the flight

conditions. A typical non-linear model will consist of a series of such tables, containing e.g. the drag coefficient C_D as a function of α and the elevator deflection δ_e, the rolling moment coefficient C_l as a function of α and δ_a, etc. Non-linear models provide better accuracy than linear models since they do not assume a linear variation of a certain aerodynamic coefficient throughout the range of each flight parameter.

Processing

Upon initialization, both linear and non-linear aerodynamic data is loaded from XML, as well as aircraft data such as wing surface and mean aerodynamic chord. At each time step, the states of the UAV are read from the Dynamics model. Wind data and the value of density are obtained from the Atmospheric model. The positions of the control surfaces come from the Actuators model. The formulation described above is used to determine the resulting aerodynamic force and torque. The values of these two parameters are sent to the Dynamics model.

4.4.3 Propulsion Model

Overview

This model simulates the force and torque generated by the UAV propulsion system.

Formulation

The propulsion force in the ABC frame is calculated through a simple linear relation

$$\mathbf{F} = \begin{bmatrix} 0 \\ 0 \\ \frac{k}{100} F_{\max} \end{bmatrix}. \tag{4.9}$$

The F_{max} value represents the maximum thrust produced by the propulsion system and k represents the current throttle setting, ranging from 0 to 100. The modelled propulsion system exerts no torque.

Processing

Upon initialization, this model reads the F_{max} value from an XML file, as well as the continuous process time constant for the first order lag filter described in [19]. At each time step, the value of k in Equation (4.9) is read from the AFCS model and is used to calculate **F**. The value of **F** is subject to the first order lag filter which will simulate the temporal delay of the response of the engine. The resulting value is sent to the Dynamics model.

4.4.4 Gravitation Model

Overview

Effects of gravitational attraction are simulated in this model. The Earth Gravitational Model 1996 (EGM96) is used in order to promote future reuse of this model in a high precision UAV simulation or even in a satellite simulation. This is a step towards the goal of a versatile and generic simulation.

Formulation

An algorithm based on the Earth Gravitational Model 1996 (EGM96) is implemented according to [33]. The value of the exerted gravitational acceleration in the ECI frame is

$$\mathbf{G} = -\frac{\mu}{r^2}\begin{bmatrix}\left[1+1.5J_2\left(a/r\right)^2\left(1-5\sin^2\psi\right)\right]p_x/r\\ \left[1+1.5J_2\left(a/r\right)^2\left(1-5\sin^2\psi\right)\right]p_y/r\\ \left[1+1.5J_2\left(a/r\right)^2\left(3-5\sin^2\psi\right)\right]p_z/r\end{bmatrix}, \qquad (4.10)$$

where μ represents the standard gravitational parameter, J_2 is an EGM96 constant, a is a WGS-84 constant, and r is the length of the **p** vector describing the position of the UAV in the ECI frame, whose components are p_x, p_y, and p_z. The value of the geocentric latitude is represented by ψ. Since the UAV is considered as a point mass, there is no torque due to gravitational effects. The complete EGM96 model contains 130,767 coefficients and is used in high precision applications [33]. The contribution of the J_2 coefficient to (4.10) is already quite small but, as mentioned above, it has been included because it is a step towards the goal of a versatile and generic simulation.

Processing

Upon initialization, J_2, a, and μ are read from an XML file. At each time step, the values of **p** and ψ are read from the Dynamics model and used in Equation (4.10). The resulting value for the gravitational acceleration is sent to the Dynamics model.

4.4.5 Terrain Model

Overview

This component is responsible for simulating the reaction force that exists when the UAV is in contact with the ground. It shall provide functionality so that take-off and landing can be simulated.

Formulation

The ultimate goal of this model is the determination of the force and torque generated by each landing gear. Each gear is modelled as having a spring with constant k and a damper with coefficient c. The

gear is connected to the UAV at a point given by the vector **e**. The direction of the gear is given by the unit vector **n**. For each gear, the first step is the determination of whether the wheel is in contact with the ground, which is described by a plane. The point of intersection is **i**. It follows that

$$\mathbf{i} = \mathbf{e} + A\,\mathbf{n} \tag{4.11}$$

where A represents the length that the landing gear would have if it were touching the ground, as illustrated in Figure 20.

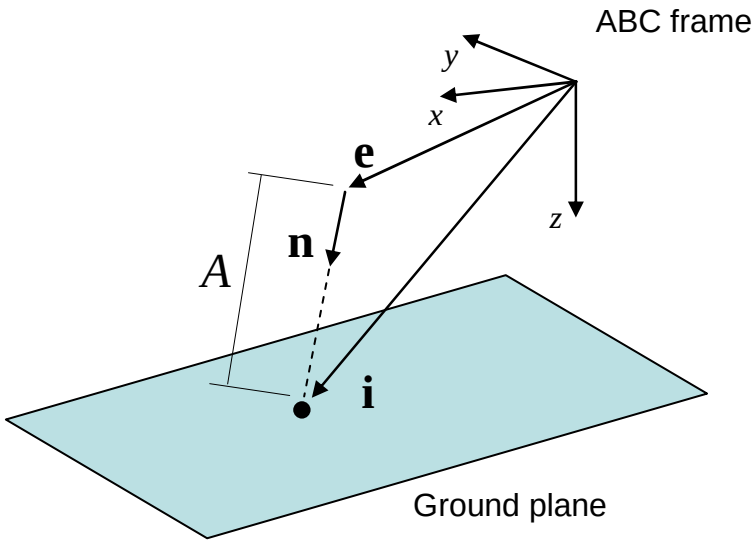

Figure 20: Determination of point of intersection between landing gear and ground

Equation (4.11) yields three equations and four unknowns (A and **i**). Since the ground is modelled as a plane, then point **i** is contained in this plane and must respect the plane equation. Therefore

$$a\,i_x + b\,i_y + c\,i_z = d \tag{4.12}$$

which gives, along with Equation (4.11), a system of four equations and four unknowns. The system is solved, giving the position of point **i** as well as the length A. If A is greater than the nominal length of the landing gear, then no further processing is done since no force is being exerted. If A is smaller, then the spring is being compressed and the generated force is

$$\mathbf{F} = \left[-k\left(A_{\text{nominal}} - A\right) + c\,v\right]\mathbf{n} \tag{4.13}$$

where v represents the compression velocity, which is obtained through differentiation between two simulation time steps. Note that in Equation (4.13) **F** acts along **n**. Since no wheel friction force is being considered at this time, the force exerted by the plane on the wheel, $\mathbf{F_n}$, must be normal to the plane. Since **n** is not necessarily normal to the plane, **F** cannot be equal to $\mathbf{F_n}$. The force of deflection of the landing gear, $\mathbf{F_d}$, will combine with **F** so that the resulting force is normal to the plane

$$\mathbf{F_n} = \mathbf{F} + \mathbf{F_d}\,. \tag{4.14}$$

$\mathbf{F_n}$ is calculated by stating that it acts along the normal of the plane and that **F** and $\mathbf{F_d}$ are normal to each other

$$\mathbf{F} + \mathbf{F_d} = \alpha\,\mathbf{n}_p \tag{4.15}$$

$$\mathbf{F}\,.\,\mathbf{F_d} = 0 \tag{4.16}$$

where $\mathbf{n}_p$ is the normal to the plane and α is a constant. Equations (4.15) and (4.16) form a system of four equations and four unknowns, which is solved for $\mathbf{F_n}$. Tire friction is modelled in a simple fashion by calculating the velocity of the wheel relative to the ground and applying a Coulomb friction model along the direction of the velocity. The force resulting from all the contributions mentioned above is used to calculate the torque exerted by the landing gear on the UAV

$$\mathbf{T} = \mathbf{i} \times \mathbf{F_{total}}\,. \tag{4.17}$$

Processing

Upon initialization, landing gear data such as spring constant and location is read from an XML file, as is the equation describing the ground plane. At each time step, the current altitude of the UAV is read from the Dynamics model in order to determine the total force and torque generated by the landing gears. The values of these two parameters are calculated and sent to the Dynamics model.

4.4.6 Atmospheric Model

Overview

The International Standard Atmosphere (ISA) is simulated by this model, along with the existence of wind. The results for the atmospheric parameters are used by the Aerodynamics and Sensors models. The wind value is used by the Atmospheric model.

Formulation

The ISA model is implemented in order to obtain values for temperature T, density ρ, and pressure p from current altitude h. The model is widely available in the literature. The reference used in this thesis was [39]. No modelling of wind is developed – it is assumed to be a constant value.

Processing

Upon initialization, the value of the wind speed and direction is read from XML. At each time step, h is read from the Dynamics model and used to calculate the various parameters of the ISA model.

4.4.7 Fire Propagation Model

Overview

This model has the purpose of implementing a simple probabilistic algorithm for the propagation of fire in large areas. It was developed in order to support a demonstration scenario. As previously mentioned, the algorithm selected was the EMBYR model [37], which is adequate for use in large-scale fires. EMBYR is a probabilistic model and therefore does not simulate the physics behind a forest fire. EMBYR is used instead of a more complex model because the actual simulation of fire behaviour is not the essence of this project.

Formulation

The EMBYR model is described in [37]. A summary of the features of the model as they were implemented is given next. Features such as terrain slope effects and influence of wind were left out for simplicity.

EMBYR begins by dividing a region into cells of a specified size. Each cell can be initialized as burnt, not burnt, or burning. In each time step, each of the burning cells has a probability of spreading fire to an adjacent cell that is not yet burnt. This probability value can be a function of forest type, wind, terrain slope, etc. but in this project was considered as a constant value for the simulation. At the end of each time step, burning cells are automatically extinguished. The newly ignited cells will spread fire in the next time step, which makes the fire spread throughout the region.

Processing

Upon initialization, necessary parameters for the EMBYR model are read from XML, such as the fire spread probability value and the size in cells of the area of the simulated fire. It is possible to specify the cells that are initially set as burning or alternatively to load an image file in the same format as the file that is output by this method for visualization in Google Earth. At each time step, the fire is spread throughout the region according to the formulation above. The data generated is provided to the Fire Front Following model as well as to the Google Earth interface, for visualization purposes.

4.4.8 Fire Front Following Model

Overview

Based on the results of the Fire Propagation algorithm, this model determines a flight plan that will result in the UAV following the front of the fire. The algorithm is based on [10]. The main objective of this model is to demonstrate the applicability of SimUAV in the particular scenario of fire monitoring.

Formulation

The algorithm begins by reading from an XML file the location of the area that is being processed by the Fire Propagation model, i.e. the area that *might* be on fire. No processing is done if the UAV is outside this area. If the UAV is inside the area – which is represented as a grid whose cells can be burning, burnt, or not burnt, as previously mentioned – then its position in the grid is determined, i.e. the cell which it is occupying at the moment.

A subset of the grid containing Fire Propagation data is read. This subset has a specified dimension, e.g. 10 cells per 10 cells. This specified dimension is intended to represent the range of the fire detection sensors, which are not simulated. The subset is centred at the cell currently occupied by the UAV. Thus, a local map of current fire data is obtained.

A linear interpolation is performed on the coordinates of the currently burning cells in the local map of fire propagation data. Figure 21 provides a diagram illustrating this situation. The UAV is located at the black cell in the centre, and its velocity is represented by the arrow. Here, the UAV sensors have a range of four cells in each direction (a square sensor range is used for simplicity). The other coloured cells represent the front of the fire (which is not continuous) and the line represents the linear interpolation providing a first-order approximation to the front of the fire. This line is intersected with the edges of the local map, yielding two points which are the candidates for the next waypoint to be sent to the AFCS model for the purpose of following the front of the fire. These points correspond to values of latitude and longitude, not to cells.

All that is left is to determine which one of these points is to be followed. This is done by determining the change in direction of the UAV that is required to reach each of the points and selecting the point that required the least change. In the case of Figure 21, the point on the left would be selected as the next waypoint.

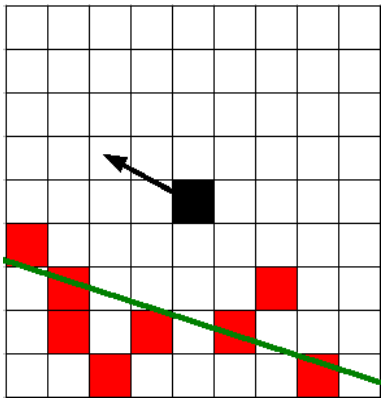

Figure 21: Representation of Fire Front Following algorithm

The coordinates of the generated point are sent to the AFCS model, which will command the UAV to fly to that location. As the simulation progresses, the UAV changes position and the fire front evolves. This means that, at each time step, a new waypoint must be determined. As such, the UAV is continuously pursing a moving waypoint, which results in its trajectory approximating the shape of the fire front. Note that the trajectory approximates the shape the fire front had at the instant that the UAV flew over the area in question (since the fire front evolves).

Processing

Upon initialization, the data necessary for the operation of this model is read from XML, such as the range of the sensors. At each time step, fire data is read from the Fire Propagation model and the UAV position and velocity are read from the Dynamics model. This data is used to implement the formulation described above, giving a waypoint that the UAV must follow. The coordinates of this waypoint are sent to the AFCS model, which will command the UAV to fly to that waypoint.

4.4.9 Google Earth Interface

Overview

An interface is established with Google Earth, a popular viewer for Earth maps. Among other features, this interface allows for the visualization of the exact position of the UAV in the planet, in terms of latitude, longitude, and altitude.

Formulation

Data exchange with Google Earth takes place through files in the KML format. This format is an implementation of the XML language. There are several tasks performed by the Google Earth interface. The most important is visualizing the position of the UAV in the planet. For this, the coordinates (latitude and longitude) as well as altitude are written to a properly formatted KML file. This file is periodically read by Google Earth, which automatically displays the position of the UAV

at the present time. Furthermore, the previous positions of the UAV are written to another section of the file, which allows Google Earth to draw a line representing the path described by the UAV in is flight.

The second task supported by the Google Earth interface is the setting of a placemark to represent the point in space where SimUAV will initially place the UAV. The user sets a placemark in Google Earth and saves it as a KML file. The file is automatically loaded by the simulation at start-up and the latitude, longitude, and altitude of the placemark are read. The Dynamics model is initialized with these values.

The definition of a flight plan for SimUAV is the third task supported by this model. The user of Google Earth sets a series of placemarks representing waypoints and saves them as a KML file which is read by SimUAV, as would be done for the definition of the initial position of the UAV, described in the previous paragraph. The waypoints read from the KML file are assembled into a flight plan which is sent to the AFCS model.

The fourth and final task supported is the visualization of the results of the Fire Propagation model. The Google Earth interface model is capable of outputting the fire data as an image file, each pixel of the image representing a cell. The colour of the pixel defines the state of the cell: red for burning, black for burnt, transparent for not burnt. This image file is loaded into Google Earth via a KML file, which defines the location where the image is placed on the surface of the Earth.

Processing

Upon initialization, the coordinates of the starting point of the UAV are loaded into the simulation, as well as an initial flight plan. At each time step, the UAV position is read from the Dynamics model and fire data is read from the Fire Propagation model. This data is displayed in Google Earth according to the formulation above.

4.4.10 FlightGear Interface

Overview

The open-source FlightGear flight simulator is used for displaying position and attitude, allowing the user of SimUAV to better understand the results of the simulation. At the time of software construction, the opportunity was identified for displaying the deflections of the control surfaces, and this was implemented as well.

Formulation

The data retrieved from the Dynamics model that is to be sent to FlightGear is assembled in a packet whose structure is defined in the FlightGear documentation [18]. This packet is capable of holding

data for various aspects of the visualization, but SimUAV only places values for position, attitude, and control surface deflection in the packet. A UDP connection with FlightGear is established through a network and the packet is sent periodically, being processed by FlightGear upon reception. FlightGear reads the data sent in the packet and positions the UAV accordingly.

Processing

Upon initialization, the IP address of the computer where FlightGear is running is read from an XML file, as is the port number where FlightGear is listening for connections. At each time step, UAV position and attitude data are read from the Dynamics model and sent to FlightGear through UDP according to the formulation described above.

4.5 Synopsis

The detailed architecture corresponding to the implementation of SimUAV was developed and presented. Relevant aspects of the various components were introduced, including the processing algorithm they implement.

The motivation behind the separation of the simulation models into SMP code and non-SMP code was established, and the resulting common model architecture was presented.

Chapter 5

Validation Results

5.1 Introduction

This chapter presents some of the tests performed in the validation phase of the software life cycle. The results of these tests are a critical part of the development of the software, demonstrating whether it serves the purpose it was created for.

Results of unit and integration tests are provided, focusing on the functionality offered by small groups of components. Following those results, the output of the system tests is presented, demonstrating the behaviour of SimUAV as a whole. Not all tests performed in this work are described due to space constraints.

5.2 Unit and Integration Testing

As introduced in Section 2.3.3, unit tests verify each basic software component, in order to ensure that it has been implemented according to the architecture specification. Integration tests are performed on groups of software components, so that the interaction between those components is tested. Although that approach was followed in this work, the results presented in this section do not. All tests are important for validating the software but, due to limited space, only the most relevant results are presented here.

Each of the following sections represents a particular test, or a series of very similar tests. Regarding the organization of the sections, note that some components (e.g. Dynamics model) have been broken up into parts, each representing an important aspect of the component from the point of view of the simulation user (e.g. the Dynamics model is broken into Translational Dynamics, Rotational Dynamics, and Latitude/Longitude Navigation). The title of each section identifies the aspect of the simulation being tested, followed by the title of the test.

5.2.1 Test 1 – Latitude/Longitude Navigation and Google Earth Positioning: Defining the Initial UAV Position

This test verifies the latitude and longitude navigation features of the Dynamics model and the capacity of the Google Earth interface to position the UAV. Using Google Earth, a placemark was set at the following coordinates: 38º 44' 14.51'' (38.737364º) north and 9º 8' 17.86'' (9.138294º) west, at an absolute altitude of 100 m (see Figure 22). The placemark was saved as a KML file and was loaded into SimUAV as the initial UAV position.

Figure 22: Test 1 – Initial Google Earth UAV position matching requested position

The SIMSAT MMI reported the following values of latitude, longitude, and altitude: 38.737364º north, 9.138295º west, and 100 m. These values match the ones entered in Google Earth, aside from a rounding error. This internal data was then used automatically by the Google Earth interface model to position the UAV in Google Earth. The result is also visible in Figure 22, where the "initial position" placemark is visible, with the placemark representing the UAV superimposed. This shows that the placemark data was correctly loaded into the Dynamics model and that the data from the Dynamics model was correctly output back into Google Earth.

5.2.2 Test 2 – Translational Dynamics and Propulsion: Thrust Force

This test verifies the generation of propulsion force and the response of the Dynamics model to that force. Using only the Dynamics and Propulsion models (all others disabled), a rigid body was simulated and a throttle value of 100% was requested from the engine immediately after the simulation began. Simulation data is presented in Table 2.

Figure 23 presents the values of thrust force, velocity, and position. The effect of the first order lag filter on the thrust force can clearly be seen as a delay in the response, as desired. The velocity grows linearly with time (there are minor inconsistencies in the graph due to the low level of detail of the graphs drawn by the MMI). The slope of the line matches 1 (the value of the applied force) to a great precision, and the graph of the position versus time can be shown to be a parabola, as expected.

Parameter	Value
Mass	1 kg
Inertia matrix (Kg.m^2)	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$
Maximum thrust	1 N
Throttle	0 %

Table 2: Test 2 – Simulation start-up data

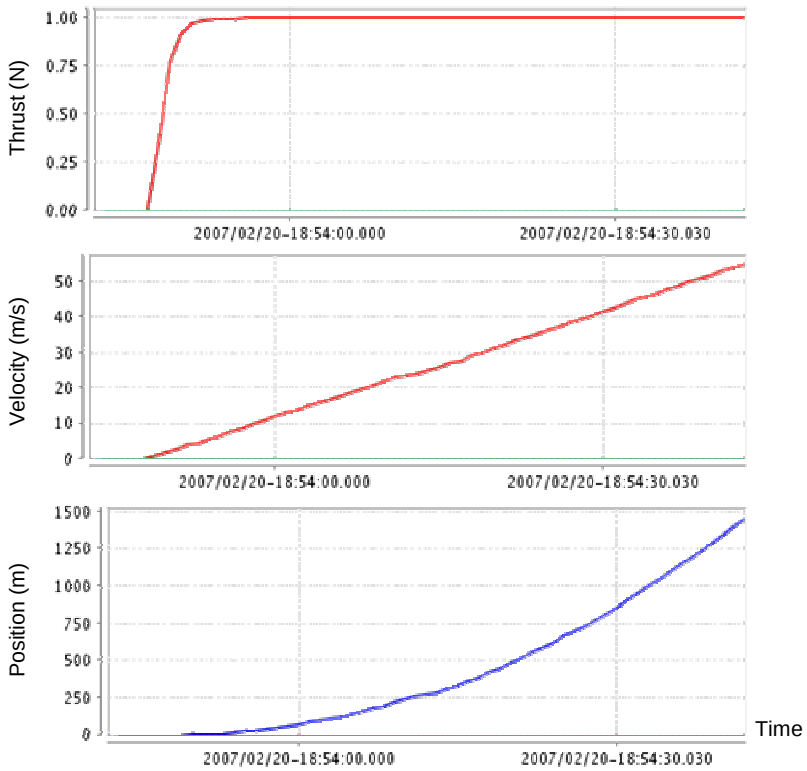

Figure 23: Test 2 – MMI screenshot showing applied thrust force, velocity and position when throttle is set to 100%

5.2.3 Test 3 – Rotational Dynamics: Applied Torque

This test verifies the rotational dynamics feature of the Dynamics model. Using only the Dynamics model (all others disabled), a torque was applied to a rigid body. Simulation data is presented in Table 3. Figure 24 presents the values of angular velocity and angular position. The angular velocity grows linearly with time (as before, there are minor inconsistencies in the graph due to the low level of detail of the graphs drawn by the MMI). The slope of the line matches 0.005 (the value of the applied torque) to a great precision, and the graph of the angular position versus time can be shown to be a parabola, as expected.

Parameter	Value
Mass	1 kg
Inertia matrix (Kg.m^2)	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$
Applied torque	0.005 N.m

Table 3: Test 3 – Simulation start-up data

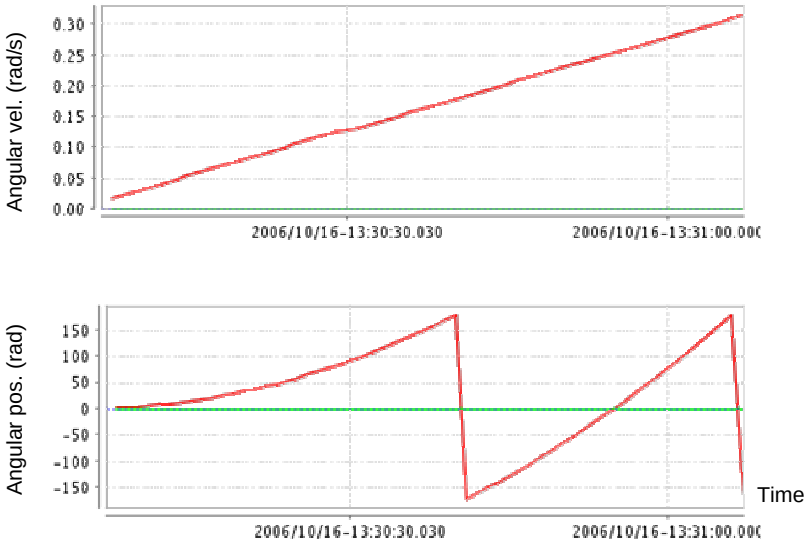

Figure 24: Test 3 – MMI screenshot showing angular velocity and angular position when subjected to constant torque

5.2.4 Test 4 – Translational Dynamics and Gravitation: Orbital Mechanics

This test verifies the translational dynamics feature of the Dynamics model and the operation of the Gravitational model. A circular orbit around the Earth was simulated, which is possible due to the generality of the Dynamics model. Using only the Dynamics and Gravitational models (all others disabled), a body was placed at a distance from the centre of the Earth, with a velocity in the equatorial plane normal to the displacement vector from the body to the centre of the Earth. According to Equation (4.10), the modulus of the acceleration g is constant throughout the orbit

$$g = \frac{\mu}{r^2}\left[1+1.5J_2\left(a/r\right)^2\left(1-5\sin^2\psi\right)\right]. \tag{5.1}$$

Replacing the variables in (5.1) by the values in Table 4 [33] gives

$$g = 9.7471\ m/s^2. \tag{5.2}$$

A simple force balance gives the value of the initial velocity v

$$m\frac{v^2}{r} = mg \ \Leftrightarrow\ v = 7898\ m/s. \tag{5.3}$$

A circular orbit was selected because of the ease with which it can be validated. Figure 25 shows the simulated orbit superimposed with the theoretical circular orbit. The observed differences are due to limitations of the graphing software, as the error was found to be very small. The distance to the centre of the Earth should have remained at 6400 km, since the gravitational acceleration was constant because the orbit was in the equatorial plane. The error in this value grew approximately linearly throughout one orbit, and was found to be 0.015% at the end.

Parameter	Value
Distance to centre of Earth - r	6400 km
Standard gravitational parameter - μ	3986004.418×10^8 m^3/s^2
Constant from EGM96 - J_2	1.0826267×10^{-3}
Constant from WGS84 - a	6378137.0 m
Latitude - ψ	0 rad

Table 4: Test 4 – Simulation start-up data

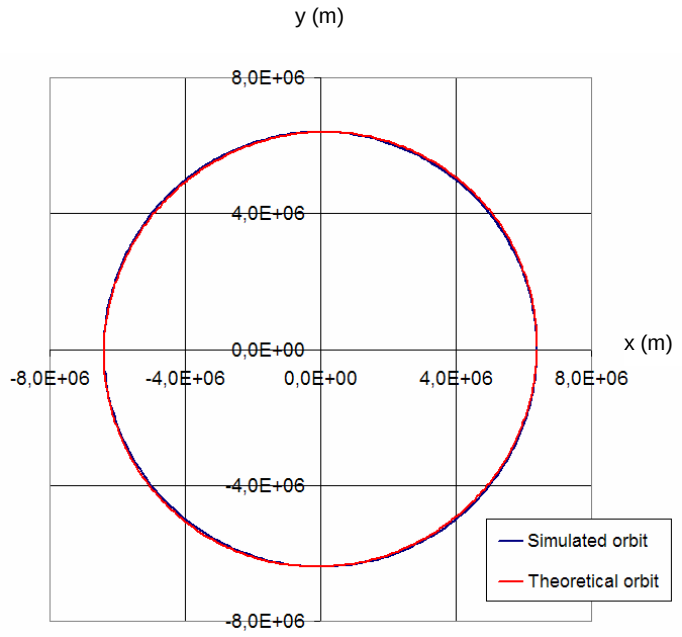

Figure 25: Test 4 – Simulated circular orbit superimposed on theoretical circular orbit

5.2.5 Test 5 – Atmospheric Parameters: Values at Several Altitudes

This test verifies the simulation of atmospheric parameters by the Atmospheric model. Using only the Dynamics model and Atmospheric model (all others disabled), the UAV was placed at several altitudes within the range supported by the ISA model implemented in the Atmospheric model. The values of pressure, temperature, and density at each of these altitudes were recorded and compared with [39]. The values are not reproduced here since they coincided to a great precision.

5.2.6 Test 6 – Aerodynamic Force and Wind Effects: Body Facing the Wind

This test verifies the generation of an aerodynamic force and the effects of wind. Using the Dynamics, Aerodynamics, Propulsion, and Atmospheric model (all others disabled), the simulation was initialized with the data in Table 5. Throttle was set to 100% immediately after simulation start-up. The velocity of the body was expected to increase due to the thrust force until equilibrium with the aerodynamic force is reached. The velocity at that instant can be determined from the balance between thrust force T and drag force (see Section 4.4.2)

$$T = C_D q S \tag{5.4}$$

which gives a value of V equal to 4.04 m/s. Since the wind is at 0.5 m/s in the same direction that the body is moving, then the ground speed at equilibrium will be 4.54 m/s.

Figure 26 shows the evolution of the drag force until the value of -1.00 N, in order to balance the thrust force. The graphic of the drag force starts from a value of zero because, at simulation start-up, before the first step, the Aerodynamics model has not yet run, and therefore is outputting the default value of zero for the aerodynamic forces. Figure 26 also shows the evolution of the velocity from the initial value of 2 m/s until the equilibrium value of 4.04 m/s. (The velocity does not start from zero as the drag force did because it is loaded at the start-up of the Dynamics model – the model does not need to run once in order to output the value of the velocity.)

Parameter	Value
Mass	1 kg
Drag coefficient	0.1
Wing surface	1 m^2
Density	1.225 kg/m^3
Velocity of body	2 m/s
Velocity of wind (in the same direction as the velocity of the body)	0.5 m/s
Maximum thrust force	1 N

Table 5: Test 6 – Simulation start-up data

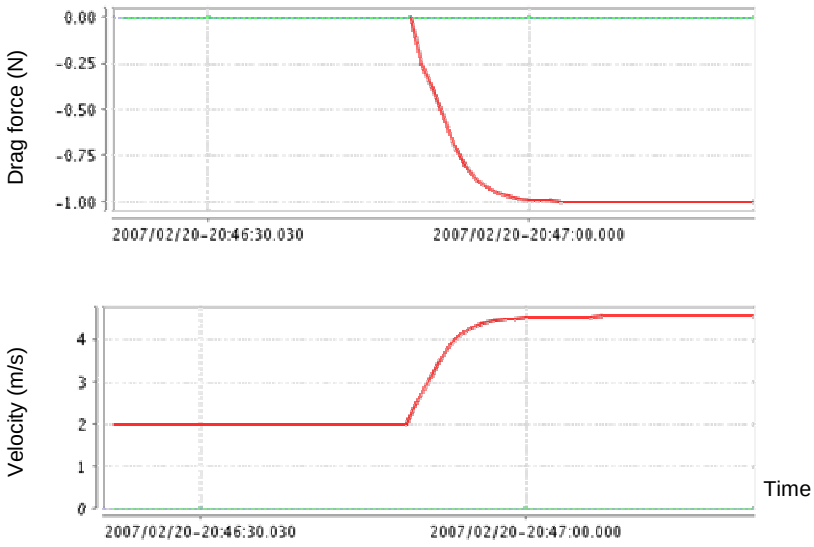

Figure 26: Test 6 – MMI screenshot showing drag force and velocity at equilibrium

5.2.7 Test 7 – Terrain: Generation of Reaction Force

This test verifies the generation of ground reaction force by the Terrain model. Using the Dynamics, Gravitation, and Terrain model (all others disabled), the simulation was initialized with the data in Table 6. The simulated body fell to the ground under the influence of gravity and, upon touching the ground, the Terrain model immediately began generating a reaction force. The damping quickly absorbed the energy and the body remained stationary at a height of 1.513 m (see Figure 27). Because of their symmetrical placement, all landing gears are subjected to the same force and deformation. Since each gear is fixed 1 m below the centre of mass of the body, and the nominal length of each spring is 1 m, then the compression of each spring is 0.487 m. This value can be confirmed with the expected theoretical value by performing a balance between spring reactions and weight

$$4\,k\,\Delta l = m\,g \tag{5.5}$$

where g = 9.7471 m/s^2 as in Section 5.2.4. Replacing with the values on Table 6 gives Δl = 0.487, which matches the simulated value to the present degree of precision.

Parameter	Value
Mass	1 kg
Inertia matrix (Kg.m^2)	$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$
Height above ground	2.2 m
Number of landing gears	4
Nominal length of the springs	1 m
Spring constant	5 N/m
Damping coefficient	1 N/(m/s)
Position of the landing gears, in the ABC (body) frame (m)	Gear 1: (1, 1, 1) Gear 2: (1, -1, 1) Gear 3: (-1, 1, 1) Gear 4: (-1, -1, 1)
Direction of the landing gears, in the ABC (body) frame (m)	All vertical, i.e. (0, 0, 1)

Table 6: Test 7 – Simulation start-up data

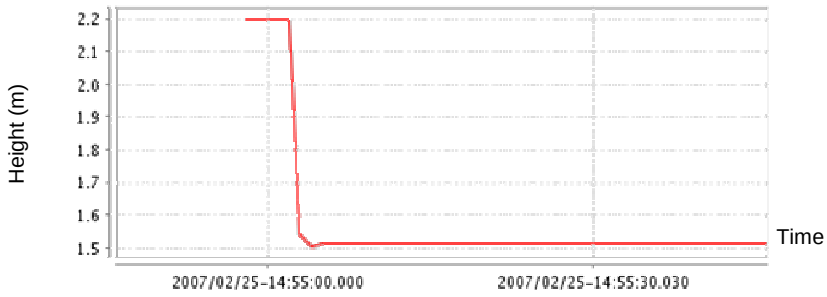

Figure 27: Test 7 – MMI screenshot showing height of centre of mass at equilibrium

5.2.8 Test 8 – FlightGear Interface: Visualization of UAV Position and Attitude

This test verifies the capability of the FlightGear interface to display UAV position and attitude. Using the Dynamics model and the FlightGear interface, a body was positioned in a specific location. Figure 28, left, displays a screenshot of FlightGear showing the position of the body. A 3D visual model of an arbitrary aircraft was used in FlightGear. Google Earth interface was used for visualization and for validating the positioning in FlightGear (see Figure 28, right). The two visual representations showed very similar positions. Differences in the UAV position in Figure 28 are mostly due to the separate ways in which the two programs draw the image, to the different camera perspective (which is defined by hand), and to inaccuracies in the positioning of the maps used by the two programs.

Additionally, the body was initialized with different values for the Euler angles defined in XML, which were correctly processed by the Dynamics model and sent to FlightGear. FlightGear allows for the user to examine its internal values for the Euler angles, and these were found to match the values defined in XML. Additionally, visual examination showed that the 3D aircraft model orientation was also coherent with the values defined in XML.

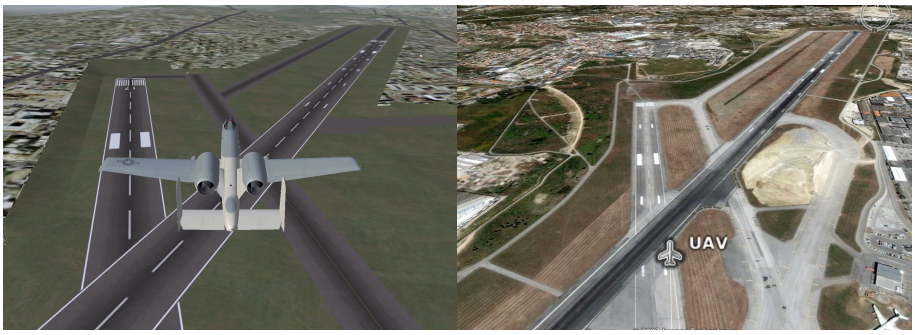

Figure 28: Test 8 – Comparison between positioning in FlightGear (left) and Google Earth (right)

5.2.9 Test 9 – Fire Propagation and Google Earth Fire Visualization: Observing the Evolution of a Fire

This test verifies the propagation of a fire (as simulated by the Fire Propagation model) and its visualization in Google Earth. A small rectangular area of cells was initialized as "burning" (see Figure 29, top left) and the simulation was run for several time steps, obtaining the evolution depicted in Figure 29. The fire spreads in a rectangular pattern in the first steps, due to the fact that the initial "burning" cells form a rectangle. Over several steps it grows into a more circular shape.

Figure 29: Test 9 – Evolution of a simulated fire

5.2.10 Test 10 – Google Earth Flight Plan: Defining and Loading a Flight Plan

This test verifies the correct loading of a flight plan defined using Google Earth. Several placemarks were set in Google Earth, each representing a waypoint, and were saved as a KML file, under the filename corresponding to the default flight plan. SimUAV loaded that file automatically upon simulation start-up, and the Google Earth interface model correctly provided the waypoint data (latitude, longitude, and altitude) to the AFCS model.

This test differs from the test described in Section 5.2.1. Although both tests read numeric values of latitude, longitude, and altitude from a file, the processing executed on those parameters is completely different.

5.2.11 Test 11 – Fire Front Following: Monitoring the Evolution of a Fire

Due its nature, this model is quite complicated to test individually, requiring its division into smaller components. A proper test of the model as a whole requires the use of the complete simulation. For that reason, this model is tested in the mission described in Section 5.3.4.

5.3 System Testing

System testing verifies whether the software, as a whole, meets the requirements laid out in the requirements engineering phase. System tests involve SimUAV as a whole, although each particular test may not verify the operation of all models. For these purposes, a model of a specific UAV platform was required. The model would have to include all data required by each simulation component. The choice fell upon the RQ-2 Pioneer UAV, which is used by the USAF (see Figure 30).

Figure 30: RQ-2 Pioneer UAV [40]

Pioneer was chosen because it is a typical mid-size UAV used in monitoring and surveillance scenarios and it was possible to find a complete set of data characterizing it. The data for the UAV was collected from various sources [40] [41] [42] [43] [44] [45]. The main source was [45], a Master's thesis which consisted in a wind tunnel study of the Pioneer UAV. The wingspan of the vehicle is 5.15 m and its length is 4.27 m. Cruise speed is 65 knots (120 km; 33.4 m/s) and range is approximately 100 nautical miles (185 km).

5.3.1 Test 12 – Telemetry Generation and Sending to UAV Console

In this test, SimUAV was connected to the UAV console developed in the scope of the same Critical Software project [19]. The command and control features of the console were tested. Several parameters of the simulation were monitored and the simulated values reported by the SIMSAT MMI matched the values reported by the UAV console. Flight plans were defined using the console and were uploaded to the simulation, resulting in the UAV following those plans correctly and as

expected. This test demonstrated that SimUAV can operate with a STANAG 4586 compliant console, providing the console with telemetry and receiving its telecommands.

5.3.2 Test 13 – Following Discrete Commands

In this test, the UAV was set in manual control mode, which means that it is possible to specify e.g. a value of altitude for the UAV to fly to. The AFCS model controls the actuators and the engine in order to accomplish the commands. The UAV was initialized at 200 m and at a heading of 0º and was simultaneously commanded to climb to 250 m and to follow a heading of 30º (0.52 rad). The values of altitude and heading as the UAV during the simulation are presented in Figure 31. The UAV correctly followed the commands, reaching the flight conditions specified.

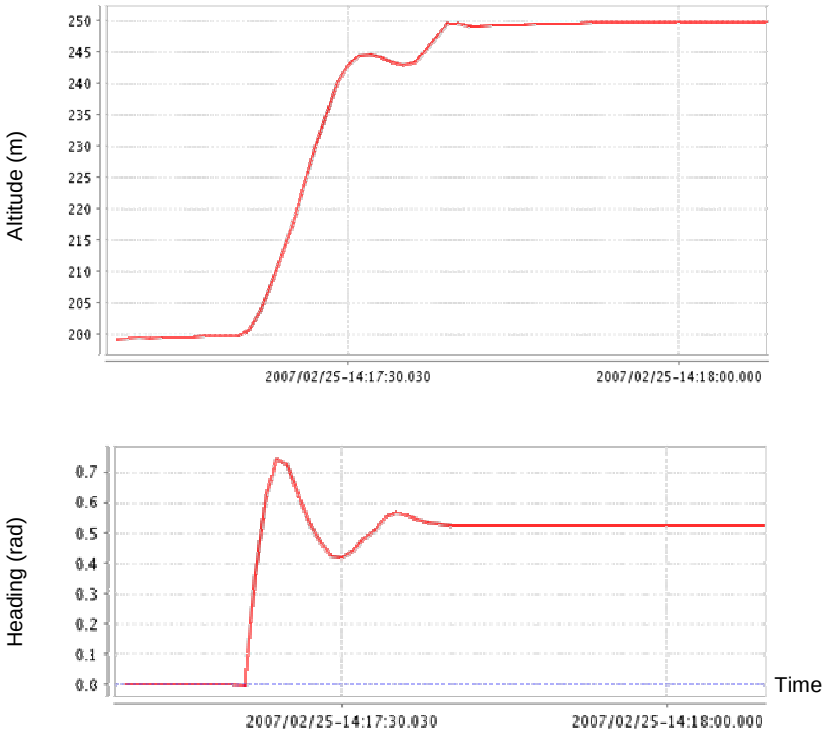

Figure 31: Test 13 – Altitude and heading of UAV while following the issued commands

5.3.3 Test 14 – Flight Plan Following and Loiter

In this test, a flight plan was defined using placemarks in Google Earth and was uploaded to the UAV (as in Section 5.2.10). The UAV was commanded to follow the flight plan, flying to each of the waypoints in the specified order. Upon reaching the final waypoint, the UAV entered loiter mode over the waypoint, as expected (see Figure 32).

Figure 32: Test 14 – Flight plan following and loiter mode

5.3.4 Test 15 – Fire Monitoring

In this test, the application of SimUAV to a fire monitoring scenario is demonstrated. A region was set up for the fire propagation model, and a flight plan crossing that region was uploaded to the UAV. The UAV autonomously began following the front of the fire, starting at the instant that it detected its presence.

The evolution of the fire and the resulting path described by the UAV are depicted in Figure 33. A close-up of the last image in Figure 33 is given in Figure 34, where an enlargement in the “radius” of the flight path is visible on the second time the UAV encircles the fire. This enlargement of the “radius” is due to the growth of the fire.

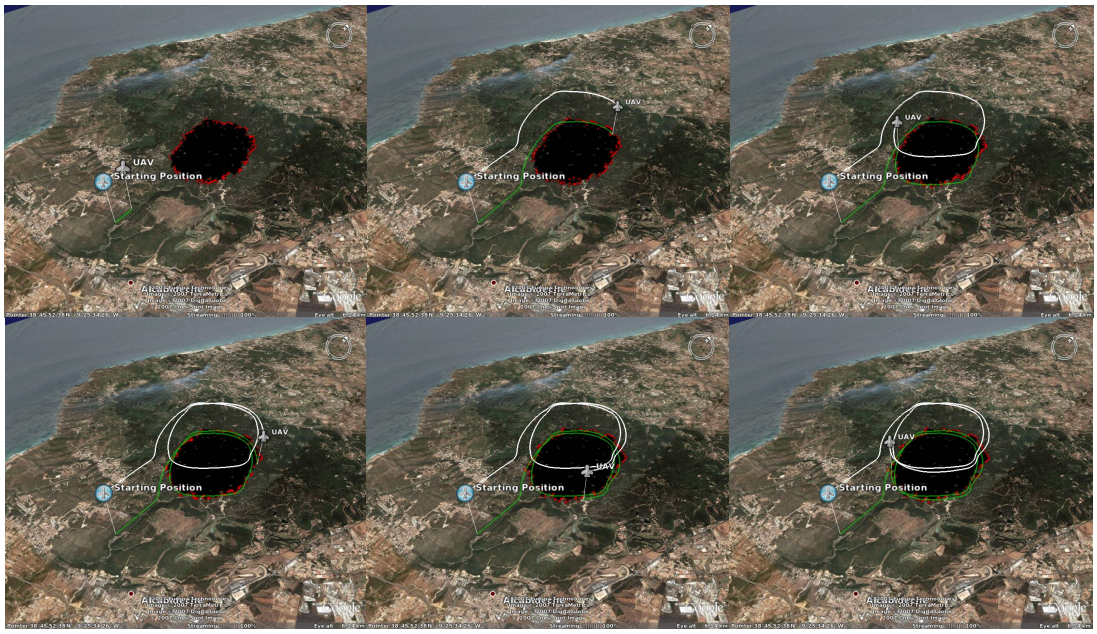

Figure 33: Test 15 – Evolution in time of the trajectory of the UAV and its ground projection in the scenario of fire monitoring

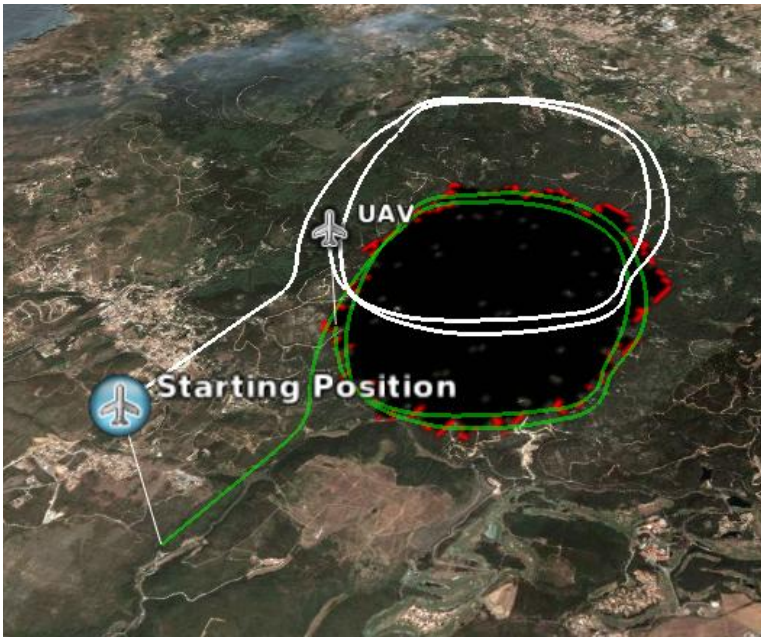

Figure 34: Test 15 – Application of SimUAV to fire monitoring scenario

5.4 Synopsis

The tests performed on SimUAV were presented. A description was given for each test, as well as the results obtained, which were as expected for all situations. The tests that focused on the behaviour of small groups of components were presented first, followed by the system tests which demonstrated the functionality of the simulator as a whole.

Chapter 6

Conclusions

This work has resulted in the implementation of an SMP compliant UAV flight simulator, SimUAV. The development of the simulator focused on modularity and versatility, and followed the software development life cycle as implemented by Critical Software, which promoted final product quality.

6.1 Contributions to the State of the Art

There are two major contributions in this thesis:

- First, the implementation of SimUAV resulted in a platform of fundamental simulation models based on a modular architecture. This platform is generic and highly versatile since it allows implementation of new models and changes to existing ones, and can therefore easily be customized or improved for application to specific UAVs and scenarios.

- The second contribution is that the ESA SMP standard has been applied to the field of aeronautics for the first time. This standard was developed for use in space simulation but has now been shown to be applicable to UAV simulation. SMP compliance is ensured by the common model specification, which formed the basis for the development of the simulation models. This specification also ensures that other simulation standards – such as the upcoming version 2 of SMP – will be easy to implement in the future.

6.2 Conclusions

A Simulation Model Portability (SMP) compliant UAV fight simulator, SimUAV, was implemented, promoting modularity and versatility while following the software development life cycle as implemented by Critical Software. Each simulation model was individually tested, and the simulator as a whole was validated in a scenario of fire monitoring.

The advantages of following the software life cycle were evident in all stages of this work. The benefit of a clear definition of the objectives of the software in the requirements engineering phase became evident during the implementation of SimUAV. The added value of a formal approach to the specification of the software architecture was also observed during the implementation phase. Finally, the need for a thorough testing procedure was self-evident as implementation errors surfaced during initial tests.

The technology requirement of SMP compatibility resulted in the establishing of a common model specification which was the basis for the development of all simulation models. The specification promotes modularity and versatility by separating each model from the rest of the simulation as much as it is possible. It also ensures that SimUAV can be made compatible with simulation standards other than SMP by performing a minimum amount of changes. Since the SMP standard was developed and is widely used by ESA, compliance with it promotes interoperability with existing space simulators.

The SIMSAT platform used for running the simulation was found to be very reliable and powerful, despite offering rather limited data visualization functionality. SIMSAT allowed focusing on developing the simulation models instead of worrying about issues such as scheduling of operations, data transfers, and management of models.

The implementation of each model answered the requirements laid out in the requirements engineering phase. Generic, fundamental models for UAV simulation were constructed in a mission and vehicle independent way. The models for the demonstration scenario of fire monitoring were also implemented. Promoting the independence of each model greatly aided in the implementation phase since development could focus on one model at a time. While some of the models developed offer a very high level of functionality, some are not as advanced. However, they can easily be upgraded or replaced because of the modular architecture. The use of XML configuration files allows for easy definition of vehicle and mission parameters.

Tests were executed on individual models, as well as on groups of models and on the simulation as a whole. The models behaved as expected, allowing for the simulation of waypoint following, loiter over waypoint, and following discrete commands. The applicability of SimUAV to the fire monitoring scenario was demonstrated through the simulation of a fire and the implementation of an algorithm for following its front and monitoring its evolution.

The two interfaces developed, with Google Earth and FlightGear, were found to be very valuable tools for visualizing the simulation output. These tools were an invaluable aid to the development process, since the data visualization of the MMI of SIMSAT is extremely limited. SimUAV was also shown to operate correctly when connected to the STANAG 4586 compliant console developed in [19].

6.3 Further Work

Further work should focus on the following aspects:

- The simulation platform should be upgraded for compliance with the upcoming SMP 2 standard. SMP 2 offers many improvements over the present version of SMP, especially in terms of architecture. The upgrade will ensure that SimUAV will continue to be up-to-date with ESA simulation standards. The common model specification that was established will greatly assist in this upgrade.

- A set of essential models for UAV flight simulation was established. These models should be improved upon. The most pertinent upgrades at this time are simulation of engine and propeller performance and modelling of complex wind effects, as well as retrieval of wind data from the Internet. Ground and runway models should be implemented for the Terrain model. The Fire Front Following model could be upgraded to operate cooperatively with other UAVs in the fire monitoring scenario.

- New models should be implemented, such as specific payload models (e.g. cameras and other sensors) and a model for situational awareness. This model would be responsible for reading altitude data from Geographic Information Systems (GIS) files and commanding the UAV to follow a constant altitude relative to the terrain (currently, a constant absolute altitude is followed). Additionally, sense and avoid algorithms should be implemented in this model.

- A standard should be established that would formally define what models should be present in an SMP compatible UAV simulator, and what data they should exchange. This would greatly aid the development of new simulations and models by promoting compatibility between them.

- A scenario of interoperability with an existing simulator in the area of space should be considered. A possible scenario would be the simulation of communications with an orbiting satellite. This could be part of a greater scenario of application of SimUAV to planetary exploration. This would require changes to some of the existing models and development of mission-specific payload models.

References

[1] M. Okrent (2005), *European Civil Unmanned Air Vehicle Roadmap Volume 3 – Strategic Research Agenda*
http://www.uavnet.com
Retrieved on February 25, 2007

[2] J. Blacklock and L. Zalcman (2006), The ADGESIM, Global Hawk UAV, Testbed Simulator. *Proceedings, SimTecT 2006*

[3] J. Eggleston and D. Donald (2005), *SIMSAT Simulator Designer's User Manual*

[4] W. J. Bezdek, D. J. Mays, and R. R Powell (2004), The History and Future of Military Flight Simulators. *Proceedings, AIAA Modelling and Simulation Technologies Conference and Exhibit, 16 - 19 August 2004, Providence, Rhode Island*

[5] Wikipedia (2007), *Flight simulator*
http://en.wikipedia.org/wiki/Flight_simulator
Retrieved on February 25, 2007

[6] R. David (2005), *Advanced Concepts Flight Simulator*
http://ffc.arc.nasa.gov/cvsrf/acfs.html
Retrieved on February 25, 2007

[7] Microsoft Corporation (2007), *Flight Simulator X*
http://www.microsoft.com/games/pc/flightsimulatorx.aspx
Retrieved on February 25, 2007

[8] P. Chisholm (2007), *Learning to Fly UAVs*
http://www.military-training-technology.com/article.cfm?DocID=1256
Retrieved on February 12, 2007

[9] S. J. Rasmussen and P. R. Chandler (2002), MultiUAV: A multiple UAV simulation for investigation of cooperative control. *Proceedings, Winter Simulation Conference, 2002, 869-877*

[10] D. W. Casbeer, R. W. Beard, T. W. McLain, S. Li, R. K. Mehra (2005), Forest Fire Monitoring With Multiple Small UAVs. *Proceedings, American Control Conference, 2005, 3530-3535*

[11] Ness Technologies (2007), *UAV Simulators*
http://www.ness.com/GlobalNess/Solutions+and+Services/Command+And+Control+and+Real-time+systems/UAV+Simulators.htm
Retrieved on February 20, 2007

[12] J. T. Ball and K. A. Gluck, The Predator UAV Synthetic Task Environment

[13] Australian Government Department of Defence (2006), *Simulation – UAV Simulator*
http://www.defence.gov.au/teamaustralia/simulation_UAV_simulator.htm
Retrieved on February 25, 2007

[14] Opal-RT Technologies (2007), *RT-LAB UAV Engineering Simulator*
http://www.opal-rt.com/productsservices/simulations_testequipment/
/rtlab_uavsimulator/index.html
Retireved on February 25, 2007

[15] T. Grabowski (2003), *Simulation techniques for mission planning of UAVs*
http://www.uavnet.com/DL/Document_Library/Massy_Meeting/Simulation_tech_Grabowski
.pdf
Retrieved on February 25, 2007

[16] A. Goossens, G. Koeners, J. Tadema, and E. Theunissen (2004), Using Simulation to Refine UAV Operator Station Functional Requirements. *Proceedings, AIAA Modelling and Simulation Technologies Conference and Exhibit, 16 - 19 August 2004, Providence, Rhode Island*

[17] Google (2007), *Google Earth*
http://earth.google.com/
Retrieved on February 25, 2007

[18] C. L. Olson (2007), *FlightGear Flight Simulator*
http://earth.google.com/
Retrieved on February 25, 2007

[19] J. Mendes (2007), *UAV Flight Simulator based on ESA Infrastructure: Generic graphical user interface for UAV Command and Control,* Master thesis, Instituto Superior Técnico da Universidade Técnica de Lisboa

[20] Mercury Computer Systems (2007), *Mercury Computer Systems Introduces Complete UAV Solution for Cost-Effective Remote Sensing Application*
http://www.mc.com/mediacenter/pr/news_details.cfm?press_id=2007_01_17_0900_101537_
556815pr.cfm
Retrieved on February 25, 2007

[21] *Software Development Process* (2006), Critical Software internal document

[22] D. Schultz, J. Bachman, L. Landis, M. Stark, S. Godfrey, M. Morisio (2002), *Space Engineering Software ECSS-E-40b Draft 1*

[23] *Requirements Analysis* (2006), Critical Software internal document

[24] A. Relvas, A. Almeida, and J. Mendes (2006), *UAV Simulator System Requirements,* Critical Software internal document

[25] A. Relvas, A. Almeida, and J. Mendes (2007), *UAV Simulator Software Requirements Specification,* Critical Software internal document

[26] *Programming Conventions for C++ Software Projects* (2007), Critical Software internal document

[27] *Joint Strike Fighter Air Vehicle C++ Coding Standards* (2005)
http://www.research.att.com/~bs/JSF-AV-rules.pdf
Retrieved on February 25, 2007

[28] A. Relvas, A. Almeida, and J. Mendes (2007), *UAV Simulator Software Detailed Design*, Critical Software internal document

[29] *Software Testing Process* (2006), Critical Software internal document

[30] A. Relvas, A. Almeida, and J. Mendes (2007), *Test Case Specification*, Critical Software internal document

[31] North Atlantic Treaty Organization (2004), *STANAG 4586*

[32] European Space Agency (2003), *Simulation Model Portability Handbook*

[33] B. L. Stevens and F. L. Lewis (2003), *Aircraft Control and Simulation*. Wiley

[34] Free Software Foundation (2006), *GNU General Public License*
http://www.gnu.org/copyleft/gpl.html
Retrieved on February 25, 2007

[35] J. S. Berndt (2005), *JSBSim: the open source flight dynamics model in C++*
http://jsbsim.sourceforge.net/
Retrieved on February 25, 2007

[36] Free Software Foundation (2006), *GNU Lesser General Public License*
http://www.gnu.org/licenses/lgpl.html
Retrieved on February 25, 2007

[37] W. W. Hargrove, R. H. Gardner, M. G. Turner, W. H. Romme, D. G. Despain (2000), Simulating fire patterns in heterogeneous landscapes. *Ecological Modelling 135 (2000) 243-263*

[38] E. W. Weisstein (2007), *Runge-Kutta Method*
http://mathworld.wolfram.com/Runge-KuttaMethod.html
Retrieved on February 25, 2007

[39] J. Roskam and C. Lan (1997), *Airplane Aerodynamics and Performance.* DAR Corporation

[40] Wikipedia (2007), *RQ-2 Pioneer*
http://en.wikipedia.org/wiki/RQ-2_Pioneer
Retrieved on February 25, 2007

[41] Pioneer UAV, Inc. (2002), *Pioneer: the legend continues*
http://www.puav.com/
Retrieved on February 25, 2007

[42] M. Selig, R. Deters, and G. Dimock (2002), *Aircraft Dynamics Models for use with FlightGear*
http://www.ae.uiuc.edu/m-selig/apasim/Aircraft-uiuc.html
Retrieved on February 25, 2007

[43] J. Pike and S. Aftergood (2000), *Pioneer Short Range (SR) UAV*
http://www.fas.org/irp/program/collect/pioneer.htm
Retrieved on February 25, 2007

[44] A. Parsch (2004), *IAI/AAI RQ-2 Pioneer*
http://www.designation-systems.net/dusrm/app2/q-2.html
Retrieved on February 25, 2007

[45] R. M. Bray (1991), *A Wind Tunnel Study of the Pioneer Remotely Piloted Vehicle*, Master thesis, Naval Postgraduate School, Monterey, California