INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa

UAV Flight Simulator based on ESA Infrastructure

Generic graphical user interface for UAV Command and Control

José Domingos Pereira Mendes

Dissertação para obtenção do Grau de Mestre em

Engenharia Aeroespacial

Júri

Presidente: Prof. Agostinho Rui Alves da Fonseca

Orientador: Prof. Paulo Jorge Soares Gil

Co-Orientador: Doutora Ana Filipa Caetano Relvas

Vogal: Prof. Bertinho Manuel D'Andrade da Costa

Abril de 2007

Acknowledgments

I would like to express my deepest thanks to my supervisor at Critical Software Ana Relvas, who made it all possible; for providing me with the opportunity to undertake this internship and execute this thesis as well as the possibility to give a higher meaning to my last Academic Project. Her personality made everyday at Critical different and more enjoyable.

I would also like to thank professor and supervisor Paulo Gil from the IST Aerospace Department for his patience and thorough review of this thesis.

To my work colleague and friend, António Almeida, I would like to send my deepest regards. He made it possible to extend the complexity and usefulness of the project in a way that could never be feasible otherwise. Together, we made SimUAV.

To the entire team at Critical Lisbon offices who provided me a hand for those long and arduous tasks of compiling free software, a big thank you. I will never forget the great soccer games we had together and gave me another reason to look forward for those Wednesday nights. I would also like to thank Critical Software as a whole for the family spirit they sponsor and making me feel welcome.

Lastly, I would like to send my love and friendship to Débora for her patience and understanding during these times. She gave me the strength and guidance to finish this thesis.

Abstract

Uninhabited Aerial Vehicles (UAVs), like other highly complex systems, require a thorough integration and validation before they can be considered fit for active duty. Highly specialized operators need several hours of flight training with such aircrafts to achieve acceptance. UAV Flight Simulators play an important role towards this goal. Not only they allow testing of all of the integrated onboard software and hardware without compromising a real aircraft, they also provide safe and affordable means to train operators and plan missions.

This thesis defines an UAV Flight Simulator software architecture based on the European Space Agency (ESA) Software Infrastructure for Modelling SATellites (SIMSAT). This architecture defines how models should be developed to simulate the flight of an UAV. This thesis also implements an Automatic Flight Control System model, Sensors and Actuators models, and a Vehicle Specific Model based on STANAG 4586 standard to handle a generic interface for the simulator.

Additionally, this thesis defines the development of a generic UAV Command and Control console. This console is based on the STANAG 4586 standard for compatibility with the developed simulator and other compliant aircrafts. Its development is heavily based on Geographical Information Systems (GIS) technology. In order to assist UAV operators in mission planning, an informed algorithm to plan courses is implemented within the console.

An example of a simulator run will be presented, along with the discussion of some of the results obtained with the integration of the entire work.

Keywords: Uninhabited Aerial Vehicle, Flight Simulation, Simulation Model Portability, SIMSAT, STANAG 4586, Command and Control Console.

Resumo

As Aeronaves Não Tripuladas (ANTs), bem como todos os sistemas complexos necessitam de ser integrados e submetidos a uma validação intensiva antes de poderem ser considerados aptos ao serviço. Operadores altamente especializados necessitam de largas horas de voo nestes aparelhos de forma a obterem a certificação necessária. Os Simuladores de ANTs têm portanto um papel decisivo neste âmbito. Não só permitem testar todo o equipamento e software a bordo do aparelho com segurança, como ainda providenciam uma forma económica para treinar operadores e planear missões.

No âmbito desta tese será definida a arquitectura de um simulador de ANTs baseado na infra-estrutura de simulação de satélites da Agência Espacial Europeia (SIMSAT). Esta arquitectura define como os modelos deverão ser desenvolvidos. Nesta tese serão também desenvolvidos modelos do Sistema de Controlo de Voo, Actuadores e Sensores, bem como um modelo que implementa uma interface genérica e compatível com o standard STANAG 4586.

Adicionalmente, será apresentado o desenvolvimento de uma consola genérica de Commando e Controlo de ANTs implementando o standard STANAG 4586 e capaz de interagir com o Simulador e outras aeronaves. O desenvolvimento é baseado em Sistemas de Informação Geográfica. Para assistir os operadores, um algoritmo informado para planear rotas é implementado.

Será apresentada uma missão de exemplo produzida pelo simulador, bem como a discussão de alguns resultados considerados relevantes.

Palavras-chave: Aeronave Não Tripulada, Simulação de Voo, Simulation Model Portability, SIMSAT, STANAG 4586, Consola de Comando e Controlo.

Contents

ACKNOWLEDGMENTS ..... **I**

ABSTRACT ..... **III**

RESUMO ..... **V**

CONTENTS ..... **VII**

LIST OF FIGURES ..... **IX**

LIST OF TABLES ..... **XI**

ACRONYMS ..... **XIII**

CHAPTER 1 INTRODUCTION **1**

1.1 BACKGROUND AND MOTIVATION 1
1.1.1 UAV simulation *1*
1.1.2 UAV command and control *3*
1.2 STATE OF THE ART 4
1.3 SCOPE OF THE THESIS 7
1.4 THESIS LAYOUT 9

CHAPTER 2 SOFTWARE DEVELOPMENT LIFE CYCLE **11**

2.1 INTRODUCTION 11
2.2 OVERVIEW 11
2.3 PHASE 1: REQUIREMENTS ANALYSIS 13
2.4 PHASE 2: ARCHITECTURE DESIGN 13
2.5 PHASE 3: IMPLEMENTATION 14
2.6 PHASE 4: VALIDATION 14
2.7 SYNOPSIS 14

CHAPTER 3 REQUIREMENTS ANALYSIS **17**

3.1 INTRODUCTION 17
3.2 UAV SIMULATOR 17
3.2.1 Technology Requirements *17*
3.2.2 Simulator Operation Requirements *25*
3.2.3 Models Requirements Overview *28*
3.3 UAV CONSOLE 32
3.3.1 Overview *32*
3.3.2 Trade-off Analysis of GIS Tool *34*
3.3.3 Technology Requirements *35*
3.4 SYNOPSIS 38

CHAPTER 4 ARCHITECTURE DESIGN AND SPECIFICATION **39**

4.1 INTRODUCTION 39
4.2 UAV SIMULATOR 39
4.2.1 System Logical Breakdown *39*
4.2.2 Generic Model Architecture Breakdown *42*
4.2.3 Automatic Flight Control System *43*
4.2.4 Sensors *47*
4.2.5 Actuators *47*

4.2.6 Vehicle Specific Model ... *48*
4.3 UAV CONSOLE ... 49
4.3.1 System Logical Breakdown ... *49*
4.3.2 Workspace Manager ... *50*
4.3.3 GeoViewer ... *51*
4.3.4 Flight Planning ... *52*
4.3.5 UAV Manager ... *53*
4.3.6 LogViewer ... *53*
4.3.7 Risk Area Route Planner ... *54*
4.3.8 UAV Instance ... *58*
4.4 SYNOPSIS ... 62

CHAPTER 5 VALIDATION RESULTS ... 63

5.1 INTRODUCTION ... 63
5.2 RISK AREA ROUTE PLANNER PROTOTYPE VALIDATION ... 63
5.3 SYSTEM VALIDATION ... 65
5.4 SYNOPSIS ... 67

CHAPTER 6 SYNTHESIS ... 69

6.1 CONCLUSIONS ... 69
6.2 FURTHER WORK ... 70

REFERENCES ... 73

APPENDIX A DEMONSTRATION PLATFORM ... 77

List of Figures

FIGURE 1 - SIMULATION OF UAVS SYSTEMS. 2
FIGURE 2 - THE LINK TRAINER. 4
FIGURE 3 - ADVANCED CONCEPTS FLIGHT SIMULATOR (ACFS). 5
FIGURE 4 - PIONEER UAV GROUND CONTROL STATION. 6
FIGURE 5 - RAYTHEON UNIVERSAL CONTROL SYSTEM COCKPIT. 7
FIGURE 6 - CDL SYSTEMS VEHICLE CONTROL STATION. 7
FIGURE 7 - LIFE CYCLE PHASES - GENERIC APPROACH. 12
FIGURE 8 - SYSTEM'S FRAMEWORK. 18
FIGURE 9 - SIMSAT RELEASE TIME TABLE. 18
FIGURE 10 - OVERVIEW OF SIMSAT ARCHITECTURE. 19
FIGURE 11 - UCS FUNCTIONAL ARCHITECTURE. 21
FIGURE 12 - ROLE OF THE VEHICLE SPECIFIC MODULE. 22
FIGURE 13 - MESSAGE WRAPPER STRUCTURE. 23
FIGURE 14 - SIMULATOR HIGH LEVEL LOGICAL BREAKDOWN. 29
FIGURE 15 - UAV CONSOLE LOGICAL BREAKDOWN. 32
FIGURE 16 - OPENEV SCREENSHOT. 37
FIGURE 17 - UAV SIMULATOR ARCHITECTURAL BREAKDOWN. 40
FIGURE 18 - UAV SIMULATOR LOGICAL BREAKDOWN DETAILED INTERFACES. 41
FIGURE 19 - GENERIC SIMSAT ARCHITECTURE. 42
FIGURE 20 - SIMULINK BLOCK DIAGRAM. 44
FIGURE 21 - AUTOMATIC FLIGHT CONTROL SYSTEM PROCESSING. 46
FIGURE 22 - VSM TERMINAL PROCESSING. 49
FIGURE 23 - UAV CONSOLE ARCHITECTURAL BREAKDOWN. 50
FIGURE 24 - WORKSPACE MANAGER WINDOW. 51
FIGURE 25 - GEOVIEWER WINDOW. 52
FIGURE 26 - FLIGHT PLANNING WINDOW. 52
FIGURE 27 - UAV MANAGER WINDOW. 53
FIGURE 28 - LOG VIEWER WINDOW. 54
FIGURE 29 - NODE GENERATION PROCESSING. 55
FIGURE 30 - ROUTE PLANNING PROCESSING. 57
FIGURE 31 - RISK AREA ROUTE PLANNER WINDOW. 58
FIGURE 32 - COMMANDING WINDOW. 59
FIGURE 33 - BASIC T MONITORING WINDOW. 61
FIGURE 34 - AIR & GROUND STATES MONITORING WINDOW. 61
FIGURE 35 - INERTIAL STATES MONITORING WINDOW. 61
FIGURE 36 - BODY RELATIVE STATES MONITORING WINDOW. 62
FIGURE 37 - ROUTE PLANNING ALGORITHM OUTPUT ROUTE. 64
FIGURE 38 - EXAMPLE GRAPH TREE. 64
FIGURE 39 - FULL MISSION GOOGLE EARTH SCREENSHOT. 65
FIGURE 40 - FULL MISSION CONSOLE SCREENSHOT. 65
FIGURE 41 - LANDING SCREENSHOT WITH CHASE CAMERA. 66
FIGURE 42 - AIRCRAFT PARAMETERS DURING LANDING AND TAKE-OFF. 66
FIGURE 43 - CIRCULAR LOITER CONSOLE SCREENSHOT. 67
FIGURE 44 - RQ-2 PIONEER UAV. 77

List of Tables

TABLE 1 - MESSAGE SUMMARY AND PROPERTIES. 24
TABLE 2 - COMMANDS AVAILABLE FOR THE SIMULATION OPERATOR. 26
TABLE 3 - COMMANDS AVAILABLE FOR THE TRAINING/CONFIGURATION OPERATOR. 27
TABLE 4 - COMMANDS AVAILABLE FOR SIMULATION SETUP. 27
TABLE 5 - TRADE-OFF ANALYSIS OF INITIALLY SELECTED GIS TOOLS. 35
TABLE 6 - GENERAL CHARACTERISTICS OF THE RQ-2 PIONEER UAV. 79

Acronyms

AFCS	Automatic Flight Control System
API	Application Programming Interface
AV	Air Vehicle
C4I	Command Control Communication Computers and Intelligence
CCI	Command and Control Interface
CCISM	Command and Control Interface Specific Module
CUCS	Core UAV Control System
DLI	Data Link Interface
DoF	Degrees of Freedom
ESA	European Space Agency
GIS	Geographic Information Systems
GUI	Graphical User Interface
HCI	Human Computer Interface
HCISM	Human Computer Interface Specific Module
MMI	Man Machine Interface
SIMSAT	Simulation Infrastructure for Modelling SATellites
SMI	Simulation Model Interface
SMP	Simulation Model Portability
SO	Simulation Operator
TCO	Training/Configuration Operator
UAV	Uninhabited Aerial Vehicle
UCS	UAV Control System
VSM	Vehicle Specific Model
XML	eXtensible Markup Language

Chapter 1

Introduction

1.1 Background and Motivation

1.1.1 UAV simulation

Uninhabited Aerial Vehicles (UAVs) are used by men from even before the age of manned flight. After first appearing during the United States Civil War in the form of balloons filled with explosives they have evolved significantly. Nowadays UAVs come in several sizes and shapes and are of particular interest for missions known as "dull, dirty or dangerous". Most UAVs have been designed for military purposes to cope with the risk associated with losing human pilots or for situations where the human pilot is a limitative factor, such as long repetitive reconnaissance flights. High risk patrol missions or tactical incursions over enemy territory are two examples of missions where UAVs can offer advantages.

While the usage of UAVs spurred in the military context, the civilian side has lagged behind largely due to unresolved issues of operating in commercial airspace. However, common applications will usually involve wide area coverage activities. Example applications [RD-1] 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, environmental observation, mail and package delivery and numerous other uses for governments, industry, academia and science.

The rapid growth of both platforms and applications also has spurred airspace regulators to begin drafting the rules under which unmanned aircraft will be able to operate in the same airspace with passenger flights. Europe has taken the lead in the civilian arena, through efforts of the Join Aviation Authorities and the European Organization for the Safety of Air Navigation (EUROCONTROL), an agency responsible for the advancement of air traffic management. These vehicles are nevertheless a highly complex system, made of several complex sub-systems, thus making the integration and validation of these aircrafts a serious issue.

The level of automation of these vehicles demands intensive and thorough tests to validate the integration of the various aircraft systems, as well as the integration with support systems such as ground control. The validation of the

interface to the ground control systems is a critical requirement before the aircraft can be deemed operational and, at an initial stage of this process, this validation is supported by the simulation of the aircraft behaviour. The simulator allows the verification and validation of ground systems and equipments and analysis of new operational concepts, such as integration with command and control systems. In this particular scenario, the simulator receives the commands generated from the ground systems, simulates the execution of these commands, and generates an output coherent with the output a real aircraft would provide, as illustrated in Figure 1, where SIMSAT (which will be presented within this thesis) provides the simulated segment of the system. The validation of these systems based on simulation tools allows for a fast, low-cost and safe solution. Another example of application of simulation to support validation activities is the validation of the automatic on-board systems to handle equipment faults (e.g. Datalink, Propulsion, Control, etc) as is common procedure among space simulation products. A simulator can also assist in the development of the actual hardware that will incorporate the aircraft, in the way that the simulator can generate realistic stimulus to the hardware and the response can be recorded and analysed. This concept is usually known as "hardware in the loop" within the simulator. This is an important asset as it allows for more debug and testing of each unit separately without the need to fully integrate the aircraft before the assembly tests begin.

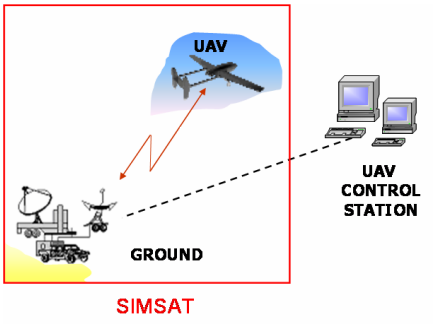

Figure 1 - Simulation of UAVs systems.

Another very important application of simulation is the training of ground crew to operate the vehicle and especially to handle emergency situations. The use of simulation as a training tool reduces the learning costs and increases safety and availability of the real aircrafts. At the same time, it exposes pilots to a large set of difficult and dangerous situations, which increases awareness and responsiveness in a real crisis, without endangering any life or equipment.

The use of simulation as a training and validation tool is already a common practice in space projects where simulation supports each step of the development of a space vehicle and mission. Simulators support the development and integration phases of all the systems that compose the space vehicle; support the testing and validation of its interfaces and functionalities; support the mission planning phase and assist in the training of the ground crew.

Each space mission has its particularities and goals; therefore, they must have specific simulators. However, it is obvious that there is a set of common features between the different simulators, which are independent of the mission and consequently exist within several simulator tools. One important common feature to all the simulators is the

infrastructure responsible for managing and running the models[1] execution, or in other words, for running the simulation. In the interest of reutilization, this infrastructure or framework might be developed in order to be simulator independent and to provide generic and common functionalities to simulator developers. This enhances the focus on model development. Requirements such as how the models are executed in real time, treatment and displaying of information, displaying messages and providing interfaces to the user are supplied by the infrastructure.

Due to the nature of the critical requirements for space simulations, the infrastructures available (e.g. SIMSAT [RD-2] and EuroSim [RD-3]) are very mature and generic. Mature in the sense that these simulators are stable and well established in the space industry, being commonly used to support in various mission phases, and generic as they provide a level of abstraction that is clear enough to separate the simulation services from the services the models provide, therefore allowing for the simulation of anything within the models. This provides the major motivation for the work of this thesis: bridge the space technology and all the advantages in the fields it excels in, to the aeronautical scope of interest in order to create a UAV Simulator.

1.1.2 UAV command and control

UAVs, Command and Control consoles and operators all in conjunction make a system of its own. The UAV is essentially a tool, the operator assumes the responsibility of making decisions and the console provides the interface between both. The console itself is composed by the hardware and software necessary to provide this interface. Even tough the major element of this trio is the aircraft, the advantage brought by them could never be fully appreciated if the console cannot properly interact with the UAV, or the operator cannot interact with the console. This leads to the second scope of this thesis: the interaction between the console to the UAV and thus the operation of the console.

Until recently there were no generic UAV consoles on the market mainly as a consequence of the inexistence of a common and well-defined interface between the consoles and the UAVs. This has lead to the development of consoles exclusively for a single aircraft. The publication of the STANAG 4586 [RD-4] standard for UAV interoperability in 2004 brought new advantages for the development of the UAV industry. Consoles compliant with this standard can be developed almost completely independent of the target vehicle. In practice this leads to a dramatic reduction of development and validation costs for the console. As a consequence of a single user interface to several aircrafts, costs of operator training and certification are also reduced. Some tailoring in the consoles always has to be done in order to handle unique features of a particular UAV model, but the bulk of the work remains the same as a consequence of this standard.

STANAG 4586 defines the architectures, interfaces, communication protocols, data elements, message formats and identifies related standards to which compliance is required to operate and manage multiple legacy and future UAVs. These standards provide interoperability within the data links (STANAG 7085), digital sensor data transmission between the payload and the air vehicle element of the data link (STANAG 7023, 4545), and for on board recording

[1] A model is a software representation of the behaviour of a real-word system or phenomenon.

device(s) (STANAG 7024, 4575). STANAG 4586, despite being issued by the North Atlantic Treaty Organization (NATO) and clearly oriented to military purposes, also provides a common interface to civil vehicles as all the necessary features for commanding and control of a UAV are included within.

One motivation for this thesis within the scope of the generic and STANAG compliant UAV command and control segment is to develop a prototype that will in the future lead to a commercial solution. Additionally, as a proof-of-concept of simulation as a support tool, the console development will be supported by the UAV simulator previously mentioned.

1.2 State of the Art

Actual flight simulators have been used since the dawn of aviation as a tool for training pilots without endangering them – or the aircraft. The first systems were very simple non-powered aircraft replicas with restricted movements and a limited range of simulated features. World Wars I and II eventually brought about the need for better and cheaper pilot training, to which simulation was an effective answer. Simulators began to include electrical actuators for cockpit motion, such as the famous Link Trainer in Figure 2 [RD-5]. More features were progressively added, such as flight instrumentation. In 1940, the use of computers to solve the equations of flight resulted in the first electronic simulators and, in 1969, the first Six Degrees of Freedom (DoF) simulators were developed.

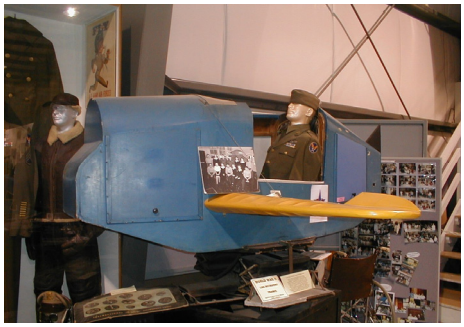

Figure 2 - The Link Trainer.

Nowadays, modern flight simulators can range from simple video games to full-size computer-controlled cockpit replicas used for aircrew training. All revolve around the concept of simulating the flight of an aircraft to a higher or lower degree depending on the purpose and budget of the product. Figure 3 [RD-5] shows NASA's Advanced Concepts Flight Simulator (ACFS), a Six DoF simulator of generic commercial transport aircraft for pilot training purposes.

Figure 3 - Advanced Concepts Flight Simulator (ACFS).

Due to the growing popularity of UAVs in the past few years, there has been an increasing demand of UAV flight simulators for the purpose of operator training. A set of projects supported by the European Commission [RD-6] identify the need to develop technologies and actively participate in the UAV industry, including the development of communication systems and standardization of interfaces. These new technologies should support overcoming difficulties associated with air traffic management and certification. All of these areas can be supported by simulation platforms. Currently there are very few simulators exclusively designed for UAVs, and most are developed within Military projects and specific to a given vehicle. Some were developed in order to support the development of new technologies and others to train operators or to assist in the implementation of new concepts.

The Civilian European UAV Industry is in development [RD-7]. This makes the development of an aircraft and mission independent UAV Simulator very appealing. The following UAV simulators exemplifying the state of the art on this field could be found:

- MultiUAV [RD-8] [RD-9]: Developed by the American Air Force and the Institute for Scientific Research. It is based on MATLAB/Simulink with the purpose of being a research tool to develop cooperative control strategies. The simulation is capable of simulating multiple unmanned aerospace vehicles which cooperate to accomplish a predefined mission.

- UAVRTB [RD-10] [RD-11]: Developed by the Canadian Armed Forces. It is an open and modular system developed to evaluate platforms and specific sensors, consisting of a ground control station and a synthetic environment that includes simulations of the UAV airframe, sensors as well as additional computer-generated forces and weather effects. STANAG 4586 compliant.

- UAV Simulator [RD-12]: Developed by Ness Tech and Israel Aircraft Industry. The system may be appended to a UAV shelter or operated as a stand-alone classroom trainer. It consists of a trainee station, which is equipped with all the relevant devices; an instructor station, which controls and records a complete after mission review and analysis; and a rapid, user friendly, scenario generation tool, correlated with the visual terrain data base.

- Virtual UAV [RD-13]: UAV Training Simulator developed by 5DT. It is powered by a virtual reality simulator to generate scenarios. This system may be adapted for a specific UAV or Remotely Piloted Vehicle.

The MultiUAV Simulator is a research tool based on Matlab/Simulink with the goal of simulating specific missions and evaluating algorithms. It is not a generic and aircraft independent simulator. The UAV Simulator is a fully integrated system with an incorporated console. It is focused mainly on the console and operator training. Unfortunately it was not possible to find any more relevant information regarding the simulator of the aircraft itself and its capabilities. Virtual UAV appears to be a simple simulator to provide operator training in payload (camera) manipulation. None of the simulators provide the capability to be generic enough and act as a baseline to simulate any aircraft and mission. They focus and are restrained to a specific application.

Following in parallel with the development of UAV simulators are the UAV consoles. Before the publication of the STANAG 4586 standard, all consoles were developed exclusively to a specific aircraft. Figure 4 shows the ground control station specifically produced for the Pioneer UAV [RD-14]. Currently, a higher abstraction can be achieved by implementing the aircraft/simulator and the console in a STANAG 4586 compliant manner. This allows for the development of a superior multi-aircraft and modular console.

Figure 4 - Pioneer UAV Ground Control Station.

Some companies have devoted their exclusive attention to the ground segment of this duo, like Raytheon for example with a new Advanced Multi-Unmanned Aerial System's Cockpit [RD-15] as illustrated in Figure 5. This new cockpit, called Universal Control System is designed to simplify control of multiple unmanned aerial systems, by improving situational awareness and ability to control multiple unmanned platforms.

Figure 5 - Raytheon Universal Control System Cockpit.

Figure 6 presents the Vehicle Control Station developed by CDL Systems to control and monitor Unmanned Vehicles [RD-16], which is now used to control two military UAVs. One operator can control multiple vehicles using only one control station. On the other hand, vehicle and payload information and control can be shared between multiple consoles and operators to ease up the task of managing the system. This console is fully STANAG 4586 compliant.

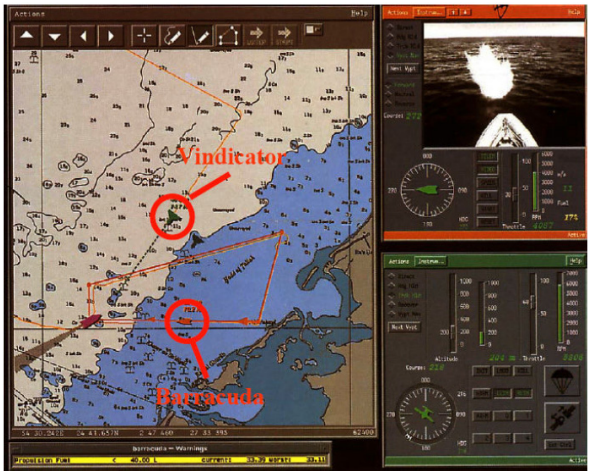

Figure 6 - CDL Systems Vehicle Control Station.

1.3 Scope of the Thesis

This thesis was done within an internal research project at Critical Software. This work was originated from the Critical Software need for a UAV Simulator in order to test and validate their future Command and Control platform, as well as to be among the first European companies to produce a UAV Simulator. It covers part of the work developed to achieve this goal, and presents an overview of the UAV simulator as well as some developed modules.

It was required that the simulator would follow a similar architecture and infrastructure as used in space simulators currently in development by ESA to increase modularity, reusability and scalability. This was set to be one of the main goals and challenges due to the fact that no other UAV simulator could be found following such a strategy. It was also required that the simulator could provide a standard interface to the most recent consoles and systems within the industry. This has lead to the choice of STANAG 4586 as the standard to be implemented within the simulator external interface. Therefore, this thesis will focus on the architecture of the simulator itself and how modularity, reusability and scalability were achieved. Modularity reflects itself in the fact that models can be easily removed and replaced within the simulation. This brings an advantage in the sense that models can be developed concurrently for one or various simulators, and simplifies the upgrading of outdated models tremendously.

The models developed for the simulator will also be presented, namely the Automatic Flight Control System model, a Sensors model, Actuators model and Vehicle Specific Model to interface with the external STANAG 4586 compliant systems. The Automatic Flight Control System acts as a demonstrator flight control system for the simulator. It was not intended to be a state of the art model or perform the functions of a ready to fly system as it was not designed with considerations of aircraft performance. A fully capable and approved flight control system must be designed for a single aircraft, and would be another thesis of its own, which was not the point of this project. The Sensors model provides the bridge between the simulated environmental conditions and states of the aircraft, to the UAV systems. This is a simplified model as the work is based on the assumption that any UAV can be used. It is a fact that different aircrafts carry different sensors payload, and this project was meant to be as much independent of the aircraft as possible, therefore not investing much effort into this model. It also simulates the signal received from an Instrumented Landing System to provide guidance for the Automatic Flight Control System to auto land the UAV. The Actuators model follows the idea of the Sensors model. It simulates the behaviour of the fundamental aircrafts control surfaces: ailerons, elevator and rudder. The Vehicle Specific Model intends to bridge the simulator as a whole to external monitoring and commanding systems. It does not simulate the radio data link between the UAV and ground systems. It simply provides the common interface as specified by the Data Link Interface of STANAG 4586. All the current and most advanced UAV simulators and consoles provide a STANAG 4586 interface because of its generality. Its purpose in the scope of this project will be to connect the UAV Simulator to the developed Console.

An initial prototype of a Command and Control console will also be developed within this thesis. This console also adopts the STANAG 4586 standard for compliance with the previously mentioned simulator and other commercial systems.

Additionally, to support the scenario of fire combat support a special algorithm was developed for this project to plan a route based on a risk area map. This algorithm is independent of the vehicle and is applicable to a myriad of situations as long as there is quantifiable information regarding the zone to be monitored. Within this simulator, this will be used to plan the UAV mission in order to maximize a scouted area for fire prevention, while still respecting range constraints due to fuel.

1.4 Thesis Layout

In the second chapter, the Software Development Life Cycle presents a short description of the Critical Software quality policy for the development of software which defines a set of phases and goals that software development should follow. This allows for the standardization of software development and enables other people to continue the work with the purpose of maintaining or improving it. The work produced follows these standard set of rules to break down the complexity of the problem, thus providing a set of stable baselines after each step to use in the following. The phases of the software development lifecycle provide the guideline for most of the chapters of this thesis: each phase is covered in a particular chapter.

In Chapter three, Requirements Analysis, a set of requirements is presented for the UAV Simulator as a whole, as well as for the models developed within the scope of this work. It also presents the requirements for the UAV Console.

The fourth chapter, the Architecture Design and Specification introduces the high level architecture for the simulator. Its purpose is to guide the reader on how the blending of all the different models was achieved to obtain a working product. It does not intend to go into coding details for every module. The architecture and detailed design for the UAV Console is also presented.

Fifth chapter, Validation Results will present the results obtained using a specific platform (UAV) to validate the UAV Simulator. The Risk Area Route Planner algorithm results will be analyzed and commented here. Finally, the demonstration of the integration of the entire system will be provided.

A set of conclusions of the work conducted and a baseline for further improvements will be presented in the last chapter, Synthesis.

Appendix A provides information regarding the Pioneer UAV used to validate the Simulator and design the Automatic Flight Control System.

Chapter 2

Software Development Life Cycle

2.1 Introduction

The work presented in this thesis was produced in the scope of an internship at Critical Software which influenced the approach to solve the problem. The work follows a well-defined software development process. The purpose of this chapter is to outline Critical Software project life cycle policy regarding software development. The Critical Software Quality Management System [RD-17] is designed to encompass the requirements of international reference models (e.g. ISO 9001, TickIT, ISO 15504 (SPICE), ISO 12207, AQAP 2110 and 150, EN/AS 9100 and 9006, and ESA standards). These set of procedures define and characterize the different project phases required to produce software and were followed in this project defining the approach to solve the problem.

In this chapter, a short overview of the entire software production process life cycle will be presented. A section will also be dedicated to each of the phases of the software life cycle: Requirements Analysis; Architecture Design; Implementation and Validation. These sections will provide a more in depth description of the entire process and provide guidance to the work produced, since each of the following chapters is closely related to each of the phases described herein.

2.2 Overview

Life cycle models vary accordingly with the nature, purpose and use of the project. Despite a necessary and apparently endless variety of project life cycle models, there is an underlying, essential set of characteristics in the life cycle of any project.

To assure effective phasing and planning, the life cycle is broken into phases, each having its associated milestones. Phases are an indicator of the project focus in an instant of time. They provide a framework within which organisation management has high-level visibility and control of the project and technical processes. The life cycle milestones are used by project stakeholders to measure the progress and risks associated with costs, schedule and functionality.

A project phase is a sequenced set of distinct activities carried out in a project that together constitutes the project life cycle. Each phase has a clear distinct purpose and contribution to the whole. It describes the major progresses and achievements of the project through its life cycle and they are completed by a milestone. The number and purpose of each phase depends on the project characteristics; in small projects one phase may have more than one single purpose (may be two or more phases merged) because it is more efficient to achieve the project goals.

This abstract and generic approach outlines the basic principles that may be applied to any kind of project type. This generic approach is presented in Figure 7.

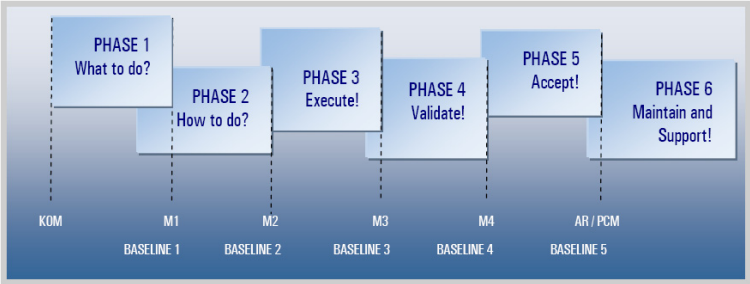

Figure 7 - Life cycle phases - generic approach.

The names and purpose of life cycle phases and milestones are tailored and defined by each project. The KOM (Kick-Off Meeting) and PCM (Project Closedown Meeting) milestones are mandatory in all projects and identify the formal start and end of the project. When applicable, the AR (Acceptance Review) and PCM milestone can be performed at the same time. In the particular case of software development, each phase has the following nomenclature:

- Phase 1 is the Requirements Analysis, where the criteria of what the software shall and should do is defined.
- Phase 2 is the Architecture Design, where the high level proposal to approach the problem is defined.
- Phase 3 is the Implementation, where the coding takes place, along with its respective documentation.
- Phase 4 is the Validation, where a formal verification of the software is performed for acceptance.

The project has not undergone Phases 5 and 6. Due to the academic nature of the work there is no specific customer to accept the product. The following sections will present what is done on each phase, including detailed descriptions of the expected output.

2.3 Phase 1: Requirements Analysis

The purpose of this phase is to produce a clear, complete, consistent, and ascertain (testable) specification of the problem identified and described by the customer. It produces the input information to be used by the project team. It establishes the knowledge of the work to be done into the first baseline. In practice, this phase produces a document specifying what the final product should do. Typical results from this phase are:

- Specification of "what to do". A set of statements (requirements) from the customer (or agreed with the customer) specifying what the customer wants as a result of the project.

- Specification of "what to validate". A set of statements and criteria (might be in the form of acceptance test cases) from the customer (or agreed with the customer) that will be used to validate the result of the project (to verify in the end that one is delivering what is specified).

The specific output of this phase within this project is a System Requirements document [RD-18] focusing essentially on the "what to do". This document is the baseline for the following phases.

2.4 Phase 2: Architecture Design

The purpose of this phase is to produce a technical response to "what to do" as specified on the previous phase. It produces the input information to be used by the project team during the Implementation (next phase). It establishes the technical details containing a precise and coherent definition of activities/functions, performances, cost, schedule and plans for all activities to be undertaken during the project. Typical results from this phase are:

- Specification of "how to do". A set of statements (detailing the specification from previous phase), diagrams, architecture, data flows, templates, etc, specifying what the project has to do in order to produce the result of the project.

- Specification of "how to validate". A set of statements (criteria) that will be used to validate the result generated by the project team. In some cases this output may be only the identification of a methodology, standard or other mechanisms that will be used by the project team to validate the result within the project.

- Definition of methodology to apply during the Implementation phase.

In practice, this phase produces a document specifying how the product goals defined in the previous phase will be accomplished and validated. This is a high level and logical breakdown of the software to be developed.

The output of this phase within this project is a Software Requirements document [RD-19] focusing essentially on the high level architecture of the simulator. This document was the baseline for the following phases.

2.5 Phase 3: Implementation

The purpose of this phase is to create and integrate the final product to deliver to the customer. It follows the how to do specification defined in previous phase. This is the most important phase, but not necessarily the longest. The output of this phase is detailed design, the code and the software produced. The detailed design provides a detailed description of how the software was implemented.

The output of this phase within this project is a Software Detailed Design document [RD-20] focusing essentially on the details of the implementation, and the code/program produced for the project.

2.6 Phase 4: Validation

The purpose of this phase is to verify and validate that the "how to do" specification is correctly implemented. It produces the end-to-end validation before delivering the product to the customer. Effectively, this phase takes on the code produced during the previous phase and applies to it the battery of tests that were defined during the Architecture Design phase. During this time, everything must be carefully documented in order to trace all tests.

Typical results from this phase are:

- A set of validated work products ready to be delivered to the customer.
- A set of records proving that the validation was performed.

As there was no commercial costumer, little formal validation was performed at the level of the components defined in the previous phase. The focus lies on producing and delivering the maximum amount of functionalities, and obtain a prototype capable of demonstrating concepts and technology. However, the system as a whole was thoroughly tested in normal conditions. This phase formally produced a Test Case Specification document [RD-21] containing the environment in which the tests where conducted, the results and all necessary information in order to reproduce these tests.

2.7 Synopsis

This chapter provided a global overview of Critical Software generic project life cycle, and consequently, the structure of the work produced within this thesis:

- The Requirements Analysis section presented the first phase of the project where the "what to do" is defined.

- The Architecture Design section presented the second phase of the project, where the “how to do” is defined.
- The Implementation section presented the third phase of the project, where the work was implemented.
- The Validation section presented the fourth and last phase of the project, where results are demonstrated.

Chapter 3

Requirements Analysis

3.1 Introduction

This is the first phase of the software development project life cycle, as defined in Section 2.3. More detailed information regarding the work produced can be found at [RD-18] and [RD-19]. The purpose of this chapter is to provide the analysis of general requirements for the Simulator as a whole (Section 3.2), and in particular for the models produced and presented in this thesis: the Automatic Flight Control System; Sensors; Actuators; and Vehicle Specific Model. General requirements are common to all models contained within the Simulator, either a product of this thesis, or developed within [RD-22]. A set of general and detailed requirements for the Console and all of its modules will also be provided in section 3.3.

Technology requirements and challenges expected within the work will be presented in the appropriate section. This shall introduce the specific technology needs that must be known before presenting an overview of the Simulator and the solution to the proposed problem. Operation capabilities will detail the expected user interaction with the final product which needs to be accounted for. Regarding the UAV Console, an overview of the problem will also be provided, as well as the expected technological particularities inherent to the solution itself.

3.2 UAV Simulator

3.2.1 Technology Requirements

SIMSAT

The UAV flight simulator will be developed on top of SIMSAT. This infrastructure is free for use among ascertained companies and projects within the European Space Agency (ESA) member states. This is the compelling reason that led to support a research project based on this infrastructure. Additionally, it is also currently used in other space projects where it has a clean record of reliability while still promoting modularity.

SIMSAT is a soft real-time simulation infrastructure/environment developed by ESA. The design concept is based on the principle that every simulator can easily be broken down into an invariant tool part (infrastructure) and a part that is specific to the subject being simulated (model). By means of careful design of the tool component, this can be used for both small and large simulators. Among other features, SIMSAT synchronizes different models and provides visualization of simulation data, possibility to interact with the simulation (e.g. injecting failures), and save/restore the simulation state. The more usual scenario is simulating the satellite and the ground segment with SIMSAT as illustrated in Figure 8.

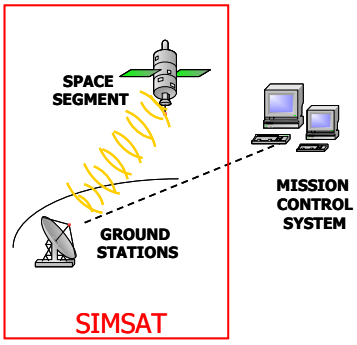

Figure 8 - System's framework.

The SIMSAT infrastructure is being developed based on 'milestone' deliveries as illustrated in Figure 9. The SIMSAT project was announced in the beginning of 1997. The SIMSAT Release 2 (2004) was based on Windows. The SIMSAT Release 3.0 (2005) was mainly a porting activity to migrate the SIMSAT infrastructure from a MS Windows dependant system to the use of "portable" technologies and a validation on the Linux operating system. SIMSAT has always been tightly coupled with the Simulation Model Portability (SMP) standard (see next section), which defines how the models shall be developed to be run by SIMSAT. Both these releases are compliant with SMP1 [RD-23]. The SIMSAT Release 4.0 (expected within 2007) will focus on evolving the functional aspects of the SIMSAT infrastructure including the support for SMP2 compliant simulators. Due to the unavailability of SIMSAT 4.0 at the time of implementation, the UAV simulator is built on top of SIMSAT 3 and is therefore SMP1 compliant.

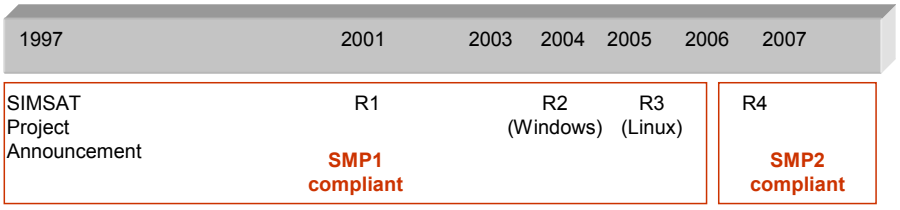

Figure 9 - SIMSAT release time table.

SIMSAT is composed of the Man-Machine Interface (MMI) and the Kernel (see Figure 10). The MMI is the graphical user interface. The SIMSAT Kernel is the framework for running the simulations and providing the facilities for command and control of the simulation, model scheduling, time keeping, data logging and recording.

SIMSAT however does not provide any models. SMP provides a set of rules on "how to" implement the models to be able to interact with SIMSAT. The functionality within these models is the complete responsibility of the developer, and anything can be implemented, as SIMSAT does not impose any restrictions to its content.

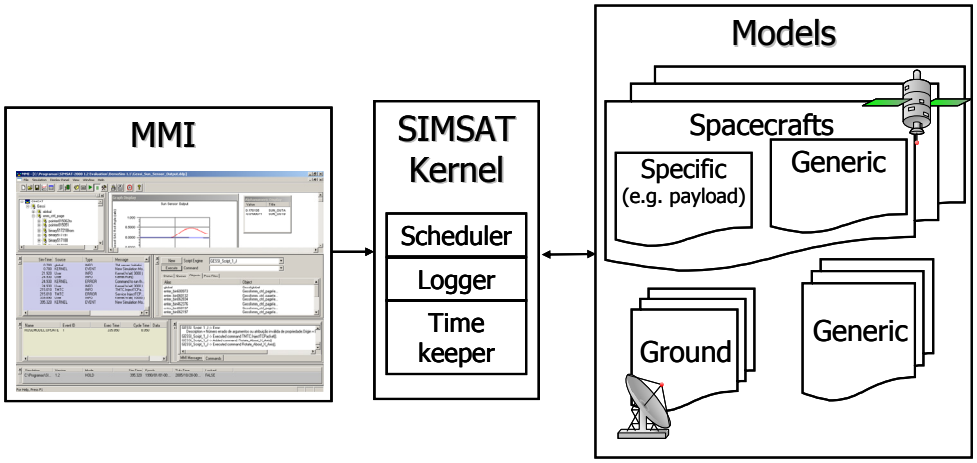

Figure 10 - Overview of SIMSAT architecture.

Simulation Model Portability (SMP)

The simulator is compliant with the Simulation Model Portability standard [RD-23] [RD-24]. The SMP standard has been developed by ESA in order to provide a "plug-and-play" approach for the development of simulators. In practice, the SMP standard provides the developer with rules on how models should be implemented to be SMP compliant, and therefore compliant with SIMSAT or any other simulation infrastructure supporting SMP. Additionally, and by adopting a consistent architecture for model development, it is also possible to easily port specific models between simulators.

The first version of this standard was SMP1 [RD-23]. The Simulation Model Interface (SMI) is a software implementation of the SMP1. SMI allows to publish models, services, and data, and to interact with the simulation environment. Some limitations of SMP1 are being overcome by the recent version SMP2 [RD-24]. Compliance of the simulation models with SMP2 promotes portability and reuse of models by minimising their interaction with the execution environment, standardizing the interface used by models, making the models own interface simpler, and making the model understandable for other developers. The major goal of these standards is to enhance the reuse of models within the European Space Industry and the portability between simulation environments, such as SIMSAT and EUROSIM.

The models are developed in C++, which is one of the standard languages supported by SMP. The SMP handbooks [RD-23] introduce the concepts and present them with C++ examples. Additionally, and delivered together with SIMSAT is a SMP wizard, which facilitates the integration of the models. The models implementation follows the rules and recommendations presented in the Programming conventions for C++ [RD-25] of Critical Software. This project makes use of SMP1 due to the unavailability of SIMSAT Release 4.0 as discussed previously.

STANAG

The inclusion of a STANAG 4586 [RD-4] standard interface within the Simulator provides the possibility to connect to external systems at a later time. This is driven by the absence of other open and common standards for UAV Interoperability, either civilian or military. This is a NATO standard that specifies the interfaces required to achieve the operational interoperability according to the respective UAV theatre of operations. This is accomplished through implementing standard interfaces in the UAV Control System (UCS) to communicate with different UAVs and their payloads, as well as with different external systems. The implementation of standard interfaces also simplifies the integration of components from different sources as well as the interoperability of legacy systems.

Even tough the simulator is not restricted to military applications, following the STANAG 4586 provides a consistent baseline to understand the system interfaces and the operation of UAVs providing an initial baseline for the simulator operational requirements. Additionally, if the design of the UAV Simulator and UAV Console is compliant with this standard from the beginning, the task of integration of this software with other systems will be easier later on.

The UCS Functional Architecture required to support interoperability among UAV systems is illustrated in Figure 11. It is important to support interoperability among legacy UAV systems (i.e. UAV systems that already exist at this moment) and future UAV systems. This architecture establishes the following functional elements and interfaces:

- Air Vehicle (AV) – The AV is the core platform consisting of all the flight relevant subsystems but without payload and data link.

- Vehicle Specific Module (VSM) – A function that resides between the DLI and the air vehicle subsystem. The VSM provides the compliance with STANAG by acting as a bridge between standard DLI data formats, protocols and a specific air vehicle.

- Core UCS (CUCS) – The CUCS provides the UAV operator with the functionality to conduct all phases of a UAV mission. It must support the requirements of the DLI, CCI and HCI which are described next.

- Command Control Communication Computers and Intelligence System (C4I) – External systems that must interact with the UCS.

- Command and Control Interface Specific Module (CCISM) – Conversion software and/or hardware between the CCI and incompatible C4I systems. The CCISM can range in complexity from a simple format or protocol translator to a user-specific application to adapt the type of information to C4I requirements.

- Human Computer Interface (HCI) – Definitions of the requirements of the functions and interactions that the UCS should allow the operator to perform

- Human Computer Interface Specific Module (HCISM) – The HCISM can be considered the physical realisation of the HCI (e.g., the set of controls and displays available to the operator(s)).

- Data Link Interface (DLI) – The interface between the VSM and the UCS core element. It provides for standard messages and formats to enable communication between a variety of air vehicles and standardised ground control stations.

- Command and Control Interface (CCI) – The CCI is an interface between the UCS Core and the external C4I systems. It specifies the data requirements that shall be adopted for communication between the UCS Core and all C4I end users through a common, standard interface.

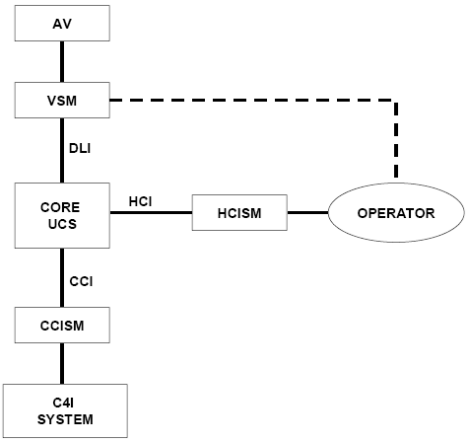

Figure 11 - UCS Functional Architecture.

STANAG 4586 contains the standard messages and protocols required at the DLI that enable the CUCS to communicate with and exploit different UAVs and payloads and to support the required UAV System operator(s) interface as specified in the HCI. This standard message set and accompanying protocols have been developed to be air vehicle and payload class independent. In addition, the DLI specifies the mechanism for the processing and display of vehicle specific messages. Within the Simulator, only the Data Link Interface (DLI) recommendations are followed, as they deal directly with the onboard representation of data, and transmission to the ground station and vice-versa. Not all STANAG 4586 messages are implemented since they are not required for basic functionality. Modularity allows for the easy implementation of support for new/updated messages. The messages to be implemented at this stage are the ones that carry the most important information generated within the simulation. Further details regarding the message structure are defined within STANAG 4586 document [RD-4].

Figure 12 presents the role of the Vehicle Specific Model and the Core UAV Control System. Data transmission through antennas takes place before the VSM module, but that is not always the case. If one were to control a different UAV system with the CUCS, one could have to replace every item in the figure except for the CUCS. For example, a different UAV system could use a different communication system requiring different antennas.

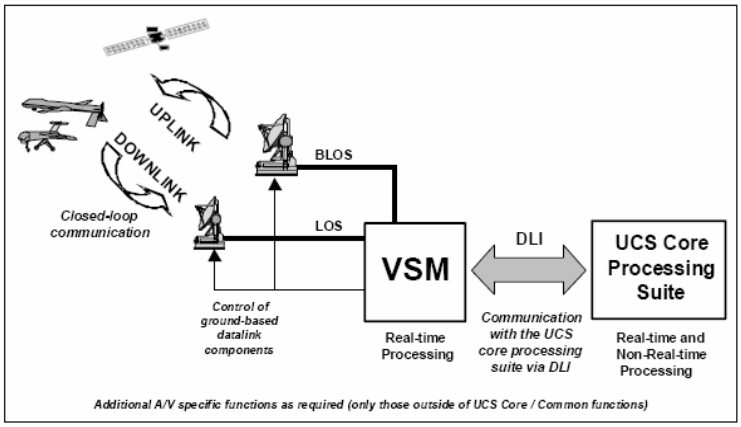

Figure 12 - Role of the Vehicle Specific Module.

The Vehicle Specific Module provides the unique/proprietary communication protocols, interface timing, and data formats that the respective air vehicles require. The VSM also provides any necessary "translation" of the DLI protocols and message formats to the unique air vehicle requirements. Since the VSM is unique to each air vehicle, the air vehicle manufacturer would generally provide it. The VSM function can be hosted on the air vehicle or on the ground. A ground based VSM function can reside on the same, different, or even remote hardware with respect to the core UCS, as long as sufficient bandwidth is provided for the message interface.

The CUCS Processing Suite provides the UAV operator with the functionality to conduct all phases of a UAV mission. Ideally, it must support the requirements of the DLI, CCI, and HCI. The CUCS shall provide a high resolution, computer generated, graphical user interface that enables the UAV operator to control different types of UAVs and payloads with minimal additional training. Depending on the appropriate level of interoperability, the CUCS Processing Suite should provide:

- The functionality and capability to receive, process, and disseminate payload data from the aircraft; perform mission planning; monitor and control the payload; monitor and control the vehicle; and monitor and control the data links.

- An open software architecture to support additional future air vehicles and payload capabilities.

- The UAV operator with the necessary tools for computer related communications, mission tasking, mission planning, mission execution and monitoring, data receipt, data processing, and data dissemination.

STANAG 4586 DLI only defines the format and content of the messages being transmitted between the VSM and CUCS. There is a set of common and generic messages that must be implemented in order for the system to obtain a certain level of interoperability. Within this work, only a few relevant messages for the models being simulated are implemented. As not all the systems aboard the aircraft are being simulated, or are not being thoroughly simulated, such as payload or the propulsion. This provides a baseline for further development and customization tough. Further

details regarding the message structure are defined within the standard itself at [RD-4]. Following is a brief description of STANAG 4586 protocol.

The philosophy for developing the message types in the DLI is to use metric (SI, ISO 1000:1992) units wherever possible. Since this relates only to internal system representation, any conversions required for human readability/familiarity or message generation (STANAG 4586 mandates the use of non-SI units in certain situations) can be performed at the appropriate interface. All Earth-fixed position references are expressed in the latitude-longitude system with respect to the WGS-84 ellipsoid in units of degrees. All time related variables are represented in Universal Coordinated Time (UCT) in seconds since Jan 1, 1979. Angles are referenced in radians although, for convenience, control surface deflections that are typically measured in degrees are expressed in that way. Bearings are measured clockwise from true north. Elevation is referenced from local horizontal, positive towards the zenith.

Each message has a wrapper around the message body consisting of a header and a trailer, as depicted in Figure 13. The header contains information that enables the message handling software to manage transmission and distribution of the messages to the appropriate entities. The footer contains the checksum information that assists identifying transmission errors. Next, a brief description of the data items in the wrapper and its role in the message handling system are presented. The extensive definition of each message format can be found at Appendix B1, Section 3.3.1.3 of STANAG 4586 [RD-4].

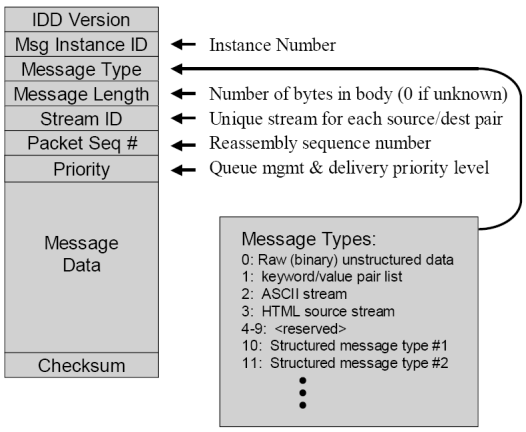

Figure 13 - Message Wrapper Structure.

In this thesis, only the STANAG messages defined as Structured Binary Data Sets are implemented. Structured binary data are data represented as binary numbers or symbols that are aligned in a pre-specified way to permit parsing algorithms to be defined and developed. The message is divided into a number of fields, each of which has a fixed length representation of some variable. Parsing software knows how to find a certain data element because it always appears in the same location in the message. Data labels and typing information are unnecessary since they are implicit in the organization of the binary data. The structure is generally defined using a tabular format identifying fields, lengths per field, and variable type.

STANAG 4586 defines several Common Message Formats in Section 4.1 of Annex B [RD-4]. The goal of the common message set is to provide a standard information group required by the CUCS for displays that are common to compliant implementations. Provisions are also made for vehicle-specific message types. Manufacturers may provide any amount of information, as required by their particular design. They may transmit information already contained in mandatory STANAG-compliant messages. However, the common message types guarantee interoperability with CUCS functionality, though not every data element is needed in every application. Table 1 summarizes the implemented message types implemented within the simulator. The same numeration used in STANAG 4586 is retained for common messages. Each message type is identified with several properties, indicated in the rightmost six columns.

Msg Type	Description	Criticality Level	Type	Source	Ack. Req'd	Guar. Delivery	LOI
5	Inertial States (Lat/Lon	Flight	Push	VSM	No	No	2
6	Air and Ground Relative	Flight	Push	VSM	No	No	2
7	Body-Relative Sensed States	Flight	Push	VSM	No	No	2
8	Vehicle Operating States	Flight	Pull	VSM	-	Yes	3
9	Engine Operating States	Mission	Pull	VSM	-	Yes	4
10	Vehicle Operating Mode	Mission	Push	CUCS	Yes	Yes	4
11	Vehicle Steering Command	Flight	Push	CUCS	Yes	Yes	4
15	Mission Plan Upload	Mission	Push	CUCS	Yes	Yes	4
40	Message Acknowledgement	None	Pull	Any	-	Yes	2

Table 1 - Message Summary and Properties.

The six properties are defined for a single message:

- Criticality Level: this refers to how the message affects system performance. In general, messages that are flight critical could, if lost, result in a chain of events that might result in loss of control. Mission critical messages are those that, if lost, would affect mission performance but not necessarily result in loss of control. Non-critical message types do not directly or immediately affect system performance.

- Type: messages are labelled as either "push" or "pull" types. Pull messages are messages that are generated in response to a request. This mechanism is used to assure that data link bandwidth is not unnecessarily consumed by unneeded data. Push messages are broadcast either periodically or based on some event, but do not require a request to result in sending a message.

- Source: identifies the entity from which the message is issued.

- Acknowledgement Required: specifies whether the receiving function must acknowledge receipt of the message type. Acknowledgements are only required for "Push" type messages, since "Pull" type messages are themselves a response to a request. If the request is not granted in a timely fashion, the requestor must generate a fresh request. Acknowledgement is accomplished by sending message #40.

- Guaranteed Delivery: identifies whether the given message requires guaranteed delivery. In general, guaranteed delivery is required in the case of pull type messages (response to a specific request) or push type messages that are commands. Messages requiring guaranteed delivery must be communicated through a port using TCP/IP protocols. Messages not requiring guaranteed delivery (such as periodic state information broadcasts for which failure to receive and process the data is not a critical event) may be transmitted using UDP datagrams.

- Level of Interoperability: lists the minimum level of UAV interoperability applicable to each of the common message types. The levels of control refer to the authority level with which the user may provide commands that alter air vehicle state and are defined in Annex B [RD-4]. The LOI specifies the lowest level and all levels above (e.g., level 2 means that the message is required to support level 2, 3, 4 and 5). In the scope of the implemented messages, this property has no functionality although to maintain coherence and future reuse the implemented messages follow the same convention.

3.2.2 Simulator Operation Requirements

This section describes the desired simulator operation, the expected users, and the functionalities available to them. To understand the functionalities a software tool should provide it is necessary to identify who is going to use that tool and how it is going to be used, while retaining that the objective of this UAV Flight Simulator is to provide affordable means to test/validate ground stations and train operators. Two user profiles are foreseeable:

- Simulation Operator (SO) – this user is going to interact with the simulator as a UAV operator would interact with the UAV. This user is the operator responsible for the management of the ground station, and UAV commanding.

- Training/Configuration Operator (TCO) – this user is responsible by the simulation itself. Any training exercise requires appropriate supervision intervention, which in this case is provided by this user. This user is also responsible for the setup of the simulations.

The TCO is usually located in a control console, separated from the operational console, where he can access the entire simulation domain. In this simulator, this scenario can easily be managed from within SIMSAT Man Machine Interface (MMI), providing direct control and access to simulation parameters and variables. The SO is stationed in a console independently from the simulator, with only a standard set of operations available to him, as would happen if commanding a real UAV. In practice, both user roles can be performed by the same person.

After defining the user for the Simulator, an assessment of what they can do is required. The Simulator is composed by a set of models that within the system can be controlled directly in a detailed manner that would not be physically feasible otherwise. This provides the TCO with the possibility to insert failures and anomalies into the simulator and systems of the UAV for testing and training purposes. The full list of commands available for each of the operator is presented in the following sections. As a consequence, there are two modes of controlling the UAV:

- Direct control mode – In this mode, the possible set of commands include the deflection of the UAV control surfaces, and issuing actuator commands (e.g., engine throttle and ailerons). This can only be performed within certain conditions that should usually not be met during an usual simulation run. Other possibilities are to change internal model parameters. Essentially, nearly any change to the simulation state can be done as the majority of the parameters should be available through SIMSAT. Only the TCO can perform these operations.

- Normal mode – In this mode, the user is only able to steer the aircraft with high level commands as defined in the following section, thus, not having direct access to the UAV systems. The definition of these commands is based on the list of possible parameters and operating modes to be passed to the UAV, as defined by STANAG 4586, and essential to guarantee interoperability This is the usual case on a simulator run and resembles a real aircraft operation scenario.

Simulation Operator Commands

This operator has at his disposal commands to control the UAV in Normal mode only. The commands specified in Table 2 will be followed by a brief description. These commands are defined according to the STANAG 4586 list of possible parameter values for command messages.

Command	Description
Commanded Airspeed	This command issues the desired airspeed for the aircraft.
Commanded Altitude	This command issues the desired flight altitude.
Commanded Altitude Rate	This command issues the desired flight altitude rate.
Commanded Rate-Limited Altitude	This command issues the desired flight altitude, but never exceeding the indicated altitude rate.
Commanded Heading	This command issues the desired heading.
Commanded Heading Rate	This command issues the desired heading rate.
Commanded Rate-limited Heading	This command issues the desired heading, but never exceeding the specified heading rate.
Flight Plan Upload	This command uploads a preset Waypoint list to the flight plan.
Loiter	This command puts the aircraft in a circular loiter flight pattern at its current location.
Autoland	This command puts the aircraft in autoland mode. This will bring the vehicle into a controlled descent following an Instrumented Landing System glide signal and land in the runway.
Take-Off	This command puts the vehicle in Take-Off mode. The aircraft will start its roll on the runway and lift off automatically.
Fire Following	This command activates a mode which will cause the aircraft to attempt to follow fire fronts, if any is detected.

Table 2 - Commands available for the Simulation Operator.

Training/Configuration Operator Commands

This operator has at his disposal commands to control the UAV systems internally. Note that almost any parameters can be changed within the simulation. Only the most common will be presented here. Table 3 specifies the commands available for the TCO in Direct mode. The commands are specified and followed by a brief description.

Command	Description
Elevator	This command issues the desired elevator position.
Aileron	This command issues the desired aileron position.
Rudder	This command issues the desired rudder position.
Engine Throttle	This command issues the desired engine throttle.
Landing Gears	This command issues the desired landing gears position.

Table 3 - Commands available for the Training/Configuration Operator.

Table 4 specifies the possibilities available to the TCO during simulation for setup and management purposes. The commands are specified and a brief description provided.

Command	Description
Load Simulation	This command setups the simulation from the specified XML file. This is provided by SIMSAT.
Save Simulation	This command saves the simulation to the specified XML file. This is provided by SIMSAT.
Load Aerodynamics	This loads the initial aircraft aerodynamics parameters from the specified XML file.
Load Dynamics	This loads the initial dynamics parameters from the specified XML file. This includes states such as positions and velocities.
Load Atmosphere	This loads the initial atmospheric parameters from the specified XML file.
Load Sensors	This loads the initial sensors parameters from the specified XML file.
Load Actuators	This loads the initial actuators parameters from the specified XML file.
Load AFCS	This loads the Automatic Flight Control System parameters from the specified XML file.
Start Simulation	This command starts/resumes the simulation. This is provided by SIMSAT.
Pause Simulation	This command pauses the simulation. This is provided by SIMSAT.
Shutdown Simulation	This commands pauses and unloads the simulation. This is provided by SIMSAT.
Start Telemetry Generation	This command starts the telemetry generation in the Vehicle Specific Model.
Stop Telemetry Generation	This command stops the telemetry generation in the Vehicle Specific Model.
Define Simulation Rate	This command specifies the speed of the simulation. It is possible to run the simulation in real time (default) and batch mode (faster than real time). This is provided by SIMSAT.

Table 4 - Commands available for simulation setup.

3.2.3 Models Requirements Overview

Each model represents a reality to the simulator. This can be an aircraft system or a phenomenon of nature for example. Models can also provide functionality to the simulator even tough it may not exist in the real system, in order to enhance it. Within the simulator these models have the possibility of mutual interaction to simulate a reality as complex as desired.

The set of models to be implemented provide a baseline for further development and customization. Further developments can be performed at any time in the future to enhance accuracy, performance, etc. The idea is to set up an architecture that would provide guidance for further work and customization to achieve specific goals defined by users. The models are developed using a modular approach and run over SIMSAT. SIMSAT drives the execution of simulation models in time and provides the ability to interact with the simulation and the visualisation of data. This is one of the main advantages of using an existent simulation infrastructure: as these fundamental functionalities are already provided. As a consequence, the simulator developer can focus on the development and improvement of specific models. Further details regarding SIMSAT infrastructure are presented in Section 3.2.1.

In the scope of this work, special attention is given to modularity and portability to not restrict the simulator to specific UAV platforms. It allows for the simulation of generic missions as long as they are feasible with the set of commands available and defined in the previous section and to the particular UAV being simulated. Additionally, and to support demonstration purposes, extra functionality is implemented within [RD-22] in order to simulate the specific scenario of fire detection as demonstration of the simulator for a purpose documented in section 1.1.1. The implementation of a generic interface provided by STANAG 4586 allows the simulator to easily connect to a commercial ground station. This is provided by the Vehicle Specific Model produced within this thesis, thus allowing the simulator to interact with the console software described in Section 3.3. The simulator also interacts with external visualization applications (Google Earth and FlightGear), which are described later but implemented within [RD-22]. The need for these visualization applications became evident during the initial implementation of the Simulator as it was not feasible to completely understand and test the response of the simulation without visual and real-time results.

During the requirements gathering, one of the goals is to identify and characterize each of the simulator models. At this point, it is not necessary to characterize the detailed design of each model neither to identify dependences between models. Figure 14 provides the high level logical breakdown for the Simulator.

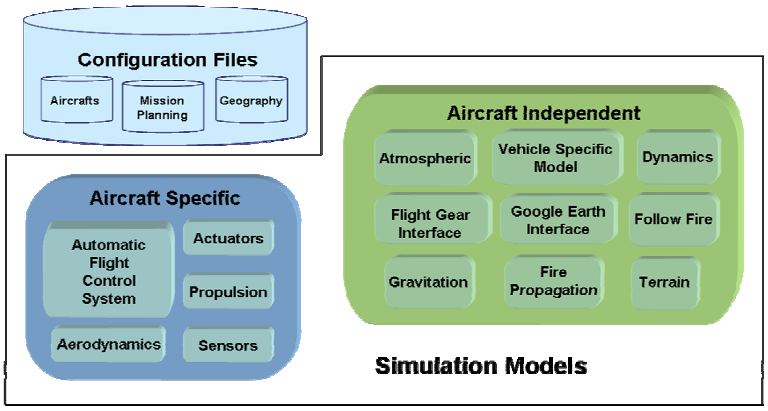

Figure 14 - Simulator high level logical breakdown.

To better organise and structure the simulator logical breakdown, it was divided into two different major branches:

- Aircraft Specific Models: These set of models are naturally dependent on the aircraft properties and systems they intend to simulate. It is possible to implement these models with a layer of abstraction and generality in order to represent the majority of the aircrafts, increasing reuse. Special functions such as unique payloads, actuators and sensors to be simulated within a particular aircraft can be added at a later phase.

- Aircraft Independent Models: These set of models are independent of the aircraft by concept. They are completely generic, and contribute only to the accuracy of the simulation as a whole. The models described therein can also provide additional functionality to the simulator and not necessarily simulate a reality. This is the case of the FlightGear and Google Earth Interfaces. Other models may also add logic functionality specific to a certain mission, but does not change when the aircraft changes as long as it supports this functionality as well. This happens in the Follow Fire model case for example.

The configurations required by each model are provided by a set of Configuration Files, mainly written in eXtensible Markup Language (XML) to be easily editable without additional software, but still be accessible by the models.

The design, implementation, and validation of each model mentioned is a very demanding task. The work presented in this thesis covers part of these tasks, in particular the design, implementation and validation of the automatic flight control system, sensors, actuators and vehicle specific model which will be presented shortly in the appropriate sections. For the sake of completeness, the remaining models (developed within [RD-22]) are presented next, together with a short description of their basic functionalities:

- Dynamics Model – This model deals with the dynamic simulation of the UAV as a rigid body. It is the core of the simulator since it determines the UAV status at each time step therefore it is indispensable. This supports a six DoF mathematical formulation.

- Propulsion Model – This model simulates the propulsion force and the behaviour of the engine system and eventual failures. The determination of the propulsion force is indispensable to the simulator. However, additional requirements that deal with engine dynamical response and fault insertion are optional.

- Aerodynamics Model – The purpose of this model is to calculate aerodynamic forces applied to the aircraft on every step. It is based on a non-linear aerodynamic coefficient build-up method sponsoring accuracy, aircraft portability and customization.

- Atmospheric Model – This model simulates basic atmospheric parameters, such as pressure, density and temperature. As a minimum it supports the International Standard Atmosphere (ISA) model. Additional, it can support wind or more complex atmospheric models.

- Gravitation Model – The purpose of this model is to calculate the gravitational forces applied to the aircraft on every step. It is designed to be independent of the planet.

- Terrain Model – This model simulates the ground reaction forces when the aircraft is touching the runway with the landing gears.

- Fire Propagation Model – The purpose of this model is to simulate the propagation of a large scale fire. This is used to implement and validate the Follow Fire Model.

- Follow Fire Model – This model commands the Automatic Flight Control System so that the aircraft follows the front of a large scale fire continuously.

- Google Earth Interface – This model provides an interface between the simulator and the Google Earth. It transmits data to Google Earth for displaying the location of the aircraft in near real-time and also its path. Additionally, it enables mission planning, in particular the definition of waypoints, in Google Earth to be transposed to the Simulator.

- Flight Gear Interface – This model interfaces the simulator with FlightGear for visualizing the position and attitude in 3D of the aircraft within FlightGear.

Automatic Flight Control System (AFCS)

The Automatic Flight Control System is part of the aircraft specific group of models. Any aircraft flight system is a complex piece of hardware and software tightly coupled together, which must be thought and thoroughly prepared for one specific target vehicle and set of missions. The design, implementation and validation of one complete flight system for one aircraft would be out of scope for this project. However, some degree of autonomy is necessary in the UAV for demonstrations purposes, leading to the development of this model. The Automatic Flight Control System is a very simplified control system developed exclusively for demonstration purposes. Little effort is done to formally

validate the system and implement complicated or truly generic features for all flight modes. It supports the set of actions already defined in section 3.2.2 for the Simulation Operator.

Effort should be invested in making the AFCS as generic and independent of the aircraft as possible; however this is not a priority. The AFCS should be modular enough to separate the mathematical formulation involving control laws from the logical and flight plan modes implementation. This model takes as inputs aircraft states, mission planning commands, and output actuators orders.

Sensors

The Sensors model is part of the aircraft specific group of models. Ideally, each sensor type should correspond to a model itself. This would increase the portability and modularity of the simulator as a whole. However it also increases the complexity tremendously due to the number and different type of sensors that can possibly be fit in the same aircraft. This number can vary very reasonably for each aircraft, leading to numerous problems when integrating models in different simulators for different aircrafts. In order to present a baseline of models to work with and further customize, the sensors model is done in such a way that all the states of the aircraft are visible by the sensors and can be an output of them as well. Additionally, other environment parameters may also be an output of this model, depending on the needs of other aircraft internal system models.

When applicable, sensed parameters should also be affected by measure lag and noise, to increase the realism of the simulation. It is also the responsibility of this model to generate an Instrumented Landing System error signal based on the location of the aircraft in relation to the ideal landing glide slope. This is used to assist the AFCS in landing the aircraft.

Actuators

The Actuators model is part of the aircraft specific group of models. Much in line with the justification on the Sensors model case, every actuator on the aircraft should provide a model of its own. For simplicity, the most important actuators are grouped within this model. When applicable, control surface positions should also be affected by a lag, to increase the realism of the simulation.

Vehicle Specific Model (VSM)

As mentioned previously, the UAV simulator interfaces to external systems are defined by the STANAG 4586 Data Link Interface specification. This specification provides a great deal of generality to connect to a real commercial ground system that supports this standard, as well as the UAV Console to be developed within this project. Due to this reason, no distinction is made between the two cases. More information relating to STANAG 4586 can be found at section 3.2.1. This model implements the Vehicle side of the Data Link Interface only to some extent. The full protocol is not implemented as it would be beyond the scope of this project due to its complexity. Enough functionality is implemented to allow the Simulator to be demonstrated, and at the same time provide a baseline for further work.

3.3 UAV Console

3.3.1 Overview

This section describes the set of requirements for the Console developed within this project. This console should provide a consistent baseline of modules to allow an operator to monitor and control a set of UAVs with the assistance of Geographical Information Systems (GIS). Special attention was given to the module development to easily allow further customization and the addition of new functionalities.

The Console allows the operator to control a baseline of functionalities of any STANAG 4586 [RD-4] compliant UAV, for any given mission, as required by the STANAG interoperability rule set. Therefore, it is built over a simplified Core UAV Control System, compliant with the Data Link section of STANAG 4586 (see Section 3.2.1). The list of operations that the Console allows the Simulation Operator to perform is specified in Section 3.2.2. The set of operations is the same for both products to enable the console to fully appreciate the capabilities within the Simulator. The Console is composed of core functionality and a Graphical User Interface (GUI). This architecture is necessary to promote modularity and scalability and greatly reduce the effort for future developers working on this baseline. The developed was done in C/C++ technology and the product runs on a Linux workstation, as the usage of free software for both the development and implementation of the final product reduces costs and maintenance. Special attention was given to the layout of the console and provision of information to the user. The operator should be able to choose to see only the information relevant for the mission in progress, as too much information is as bad as too few.

To promote modularity and scalability, several modules distinguished by functionality and hierarchy have been defined and are presented in Figure 15.

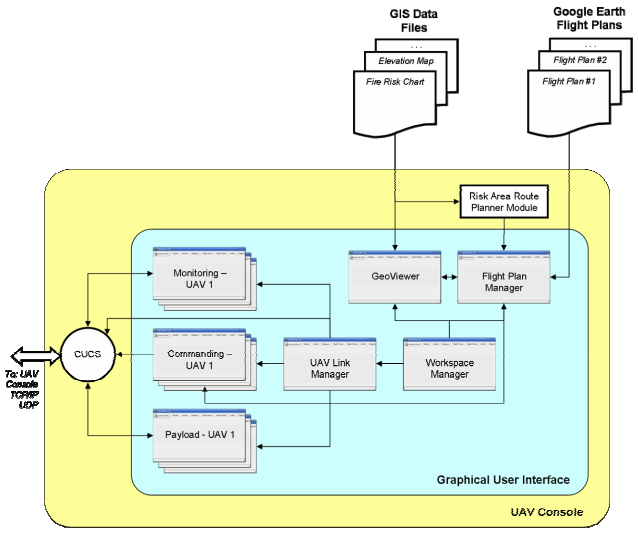

Figure 15 - UAV Console Logical Breakdown.

The Workspace Manager is a window that allows the SO to save and load the layout of the other windows, as well as define if the three directly dependent windows are visible, namely: the UAV Link Manager; the GeoViewer and the Flight Plan Manager. This window is always visible, and be as small as possible to minimize the loss of useful view area that could otherwise be used by a more relevant module to provide information. By controlling the visibility of the other windows, the operator can remove unimportant information from the screen and more easily focus on what is relevant.

The GeoViewer displays all associated geographical information to the SO. It will display maps, both in raster and vector format, and will also display other objects, such as flight plans or UAVs. This window allows for the geographical visualization of the theatre of operations, including for example a ground picture, the location of the UAVs and their flight path and plans. This is an essential feature for any console.

The Flight Plan Manager allows the SO to manipulate the currently loaded Flight Plans, upload them to the UAV or create new ones. The Flight plans are editable from the GeoViewer. The high level planning and execution of the mission is deemed a requirement in section 3.2.2. This window in conjunction with the GeoViewer allows the SO to plan the mission before issuing it to the aircraft.

The UAV Link Manager provides the SO with a list of the currently monitored UAVs, the ability to connect or disconnect to and from a VSM, and to control the visibility of the directly dependant windows for each UAV, such as Monitoring Windows, Commanding Windows or Payload Windows. This is necessary to maintain the requirement of allowing the SO to manage the UAVs being monitored, and to satisfy the requirements of allowing the SO to hide unimportant information.

The CUCS provides the Console communication endpoint for the VSM. It internally handles all the encoding and decoding of messages, and provides a simple interface for the remaining modules. It implements the STANAG 4586 standard.

The Monitoring, Commanding and Payload modules are UAV specific windows that provide the SO with UAV monitoring and commanding capabilities, for both the flight generic equipment and the on-board mission specific payload. A set of windows is defined to provide the SO with the generated telemetry within the Simulator, as required by the messages transmitted from the VSM to the CUCS, defined in section 3.2.1.

The Risk Area Route Planner is a module that assists the SO in the definition of surveillance Flight Plans by automatically generating one based on geographical and risk information. This answers to the need for an automatic system to assist the SO in mission planning by providing a mathematical optimal solution.

The functionality associated with GIS and used within the GeoViewer is provided by an external toolkit application to reduce implementation cost and risk. To properly choose the toolkit to assist with the GIS functionality a trade off analysis is performed and documented in [RD-26]. A short description of the main goals while performing the trade-off is described next.

3.3.2 Trade-off Analysis of GIS Tool

The Console provides the SO with the ability to visualize flight plans and paths, and plan a mission graphically. A tool can be developed to comply with these requirements but it is more fruitful to reuse an already existent and validated tool. This leads to the necessity to perform a trade-off analysis to already existent GIS packages and choose one that is most appropriate. However, GIS software can fork into very different programs at the functional level that may or may not suit the particular needs of this project and greatly vary in complexity. In order to keep the integration with GIS simple, the following requirements have been identified:

- The ability to load GIS data files and to provide the contained information to the underlying application.
- The ability to display the loaded data.
- The ability to draw on the display.

The research is conducted based on web searches but mostly on the software lists available on [RD-27] and [RD-28]. These two websites are dedicated to promoting open source GIS software. Taking into account the needs described above, several factors were taken into consideration when choosing the software package to be used:

- Purpose: Preference is given to toolkit-oriented GIS packages. Standalone projects increase the risk and complexity of the project. Preferably, these toolkit-oriented packages should have example applications.
- License: The Lesser General Public License (or a similar one) is desirable, in order to simplify the process of reutilization and implementation.
- Technology: The software should be implemented either in C/C++ (popular and fast programming language that usually all core software developers are familiar with) or in Java (since it provides advantages in fast GUI building).
- Web Map Service[2] (WMS) support: The software should support modern standards for GIS information interchange services.
- Level of activity: the software should be an active project, i.e. have recent and regular releases.
- Documentation: a good documentation (user manual, API, etc.) is fundamental.
- Community: an active community (mailing lists, forums, etc.) is critical for obtaining support during the development stage.

From the extensive list of products found (several dozens of very different tools), five were identified as most adequate to the objective:

- Demeter [RD-29]

[2] A WMS produces maps of spatially referenced data dynamically from geographic information. This international standard defines a "map" to be a portrayal of geographic information as a digital image file suitable for display on a computer screen. A map is not the data itself. WMS-produced maps are generally rendered in a pictorial format such as PNG, GIF or JPEG.

- Mapnik [RD-30]
- deegree [RD-31]
- OpenMap [RD-32]
- OpenEV [RD-33]

The analysis of these tools, with the above criteria in mind led to the results presented in Table 5. For each criteria (i.e. column), entries with a shaded cell indicate that the respective tool fulfils the desirability requirements of that criteria. Other aspects are important in picking the right application, such as look & feel and compatibility with many GIS file formats, but the items listed above are deemed the most important. As a result, the OpenEV toolkit [RD-33] was chosen mainly because it demonstrated much better runtime performance compared to the other equivalent toolkits. Also, it was developed in C which at the time was an advantage as it eliminated the risk and time requirements of learning a new technology. This resulted in the development of the GUI under GTK+ [RD-34]. OpenEV will be presented is section 3.3.3.

	Type	License	Language	WMS	Active	Docs	Community
Demeter	Toolkit	LGPL	C++	No	Not much	Good	Dead
Mapnik	Toolkit	LGPL	C++, Python bindings	Yes	Yes, but seems rather immature	Poor	Nearly dead mailing lists
deegree	Toolkit + example	LGPL	Java	Yes	Yes	Very good	Very active mailing list
OpenMap	Toolkit + example	Appears similar to LGPL	Java	Yes	Seems to be	Excelent!	Active mailing list
OpenEV	Toolkit + example	LGPL	C, Python bindings	Yes	More or less	Very good	Active mailing list

Table 5 - Trade-off analysis of initially selected GIS tools.

3.3.3 Technology Requirements

Geographical Information Systems (GIS)

GIS concept and software is explored within the Console to provide geo-localization of objects. Therefore a small introduction to GIS will be provided within this section. According to [RD-35], "GIS is a collection of computer hardware, software, and geographic data for capturing, managing, analyzing, and displaying all forms of geographically referenced information". In other words, GIS is an emerging concept that intends to deal with data that only makes sense if geographically localized. These data can be elevation maps, salinity maps, border lines, road maps, hospital locations, population distribution, etc. The applications are varied and can fall into the concepts of analysing and displaying information in real-time or creating a database for future processing. For example, it is already common practice to integrate road maps with Global Positioning Satellite receivers displaying the location on

the map, as well as assisting the user in reaching his destination by providing the shortest path. This is all done in real-time, over a map that was previously created.

Even tough the concept of GIS is abstract and allows for a great deal of diversity and imagination on how storing the necessary information to process, it has matured into two different types of data that are used within the console, and mentioned through this thesis: Raster and Vector.

- Raster data are image files that contain the information associated with each pixel. The pixels themselves are in a coordinate system which is specified within the file and will allow the GIS software programs to read and correctly display them. The most common format is the GeoTIFF, which is an extension to the ordinary TIFF format for regular images.

- Vector data is based on a different approach. The fundamental primitive is the point which is the entity that is located in space. Objects are usually created by connecting points with straight lines (some systems allow for connections with arcs of circles). Areas, or sometimes called polygons, are defined by sets of lines. The information on the geo-localization of the points is stored differently from format to format. The most common format is the ESRI Shapefile.

OpenEV

As defined previously (see section 3.3.2), OpenEV is used to provide the Console with the ability to display map images, and geo-localized objects. This section is reserved to explore this tool as technological requirement, but mostly challenge. OpenEV [RD-33] is a software library and application for viewing and analysing raster and vector geospatial data. It is used by private companies, universities, governments and non-profit organizations around the world. It is both:

- An application for displaying and analysing geospatial data.

- A developer library from creating new applications, including a simple example program.

OpenEV is released under the GNU Lesser General Public License (LGPL). Essentially, this means that an application based on this library can be proprietary without prohibitive restrictions. It is multiplatform, available for Windows, Linux, Solaris, and Irix operating systems. Additional advertised features of OpenEV are:

- Handle raster and vector data.

- Support 2D and 3D display.

- Gracefully handle very large (gigabyte) raster datasets.

- Support multi-channel, and complex raster datasets.

- Provide view manipulation functions (pan, zoom, rotate) at interactive frame rates.
- Uses OpenGL for high speed raster and vector rendering.
- WMS database support (This is handled in Python however).

The Graphical User Interface (GUI) is implemented in GTK+, providing portability and sophisticated GUI components. The project OpenEV is based on GTK+ 1, however, the second version, usually called OpenEV2 is based on GTK+ 2. The core of OpenEV is implemented in C and detailed Python bindings are provided for applications that are built on top of OpenEV. OpenEV2 does not support Python bindings yet which is of no relevance for this project as the development is mandatory to be performed in C and C++. OpenEV provides an Application Programming Interface (API) for the Python bindings, which in many cases are an extension of the C functions. This provides most of the documentation available for OpenEV. Unfortunately, there is no document to directly use the C library, and the code available is not documented. OpenEV incorporates two additional libraries noteworthy of mention, GDAL [RD-36] and OGR [RD-37] that provides the console with flexibility to open and display a myriad of file types for maps:

- Geospatial Data Abstraction Library (GDAL) – This is a general library used to access data in raster file format. It supports over 50 file formats, providing OpenEV with a great deal of flexibility in accessing raster files.
- The OGR Simple Feature Library is a library that provides read and write access to a variety of vector file formats (around 25). However, for ESRI Shapefiles, OpenEV uses its own library.

This software package provides the necessary GIS functionalities for this work to speed the development of the Console. However, caution should be taken when using an external open license library to produce software as there is no guarantee that the software is validated and properly working. Figure 16 presents a screenshot of OpenEV.

Figure 16 - OpenEV screenshot.

3.4 Synopsis

This chapter defined the requirements analysis for both the UAV Simulator and the UAV Console to be used in the following phases of the project. A Generic overview and logical breakdown of both systems was also defined, with relevance to the models developed within this project for the Simulator, namely: the Automatic Flight Control System; Sensors; Actuators and Vehicle Specific Model. The different modules that compose the Console were also defined. Technology requirements and challenges were presented including a short overview of the GIS concepts and the OpenEV package.

Chapter 4

Architecture Design and Specification

4.1 Introduction

This chapter corresponds to the second phase on the software development project life cycle, as defined in Section 2.4. More complete information regarding the work produced can be found at [RD-19] and [RD-20]. The software architecture for the UAV Simulator and the UAV Console, will be defined as well as implementation details for the two products. Within the UAV Simulator section, a detailed description of the Automatic Flight Control System, Sensors, Actuators and Vehicle Specific Model will be provided. The formulation and the processing of these models will also be defined. Within the UAV Console, a detailed description of the composing modules will be defined. This will cover the user interaction and the processing of the Workspace Manager, GeoViewer, Flight Planning, UAV Manager, Risk Area Route Planner, and UAV Instance.

4.2 UAV Simulator

4.2.1 System Logical Breakdown

The Requirements for the UAV Simulator are described in Section 3.2 and served as a starting point for this phase. As mentioned, a baseline of models will be implemented so that basic functionality is achieved. The Logical Breakdown is shown in Figure 17, and can be found at [RD-19] together with other documentation regarding this phase.

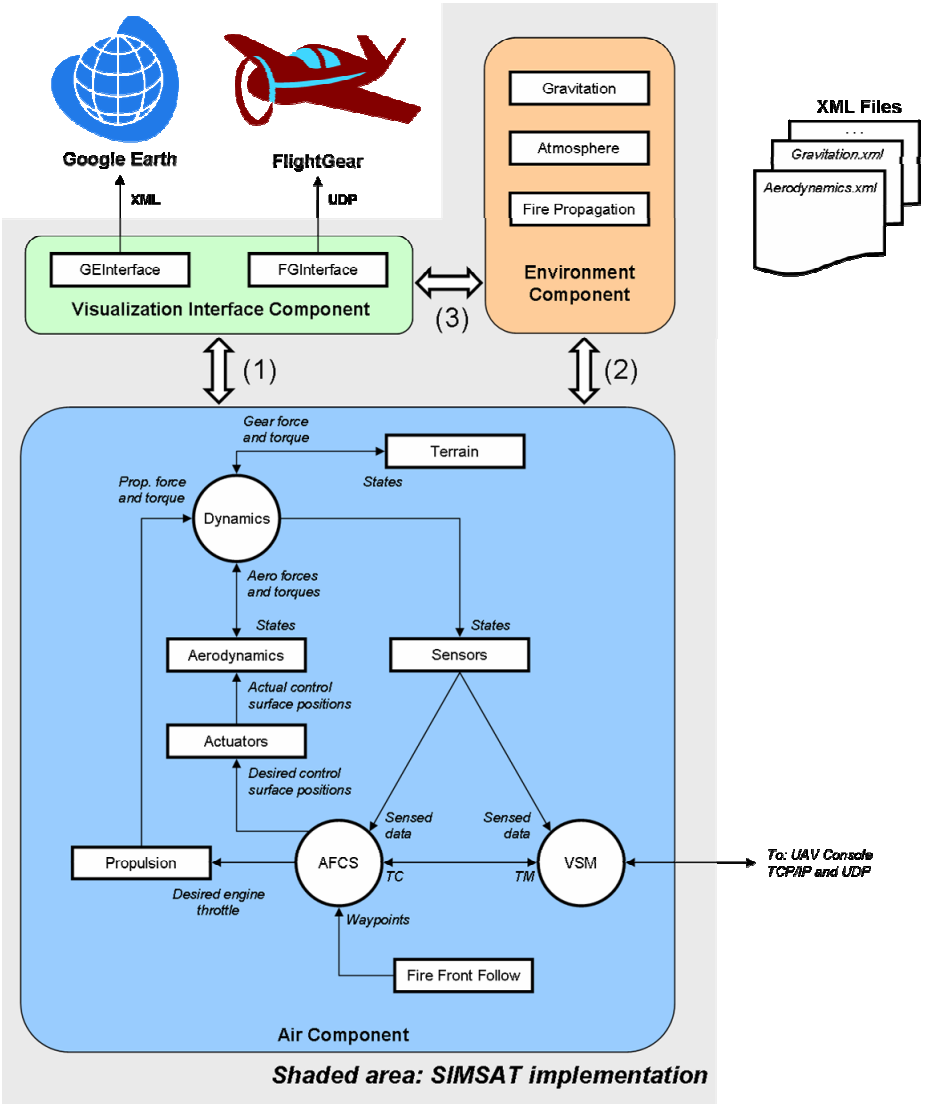

Figure 17 - UAV Simulator Architectural Breakdown.

Visible in the figure is the separation of the simulator in three components: Air Component, Environment Component, and Visualization Interface Component. The Air Component features the models that are implemented in order to simulate flight. The environment component is responsible for simulating the physical reality around the UAV. The Visualization Interface Component is comprised of two components that perform the interaction with Google Earth and FlightGear for real-time visualization of the simulation output. This separation is different than was done within Section 3.2.3 due to the fact that now it is necessary to clarify dependencies between models. Despite that apparently both breakdowns are different they complement each other. Where this breakdown pretends to demonstrate the relations between models, the breakdown at Section 3.2.3 intends to provide a global overview of what models are necessary, and how they relate to the modularity and portability of the simulation as a whole and may contribute to other different simulators. The interfaces between the major components are not shown for clarity and are labelled (1), (2), and (3). They are presented in detail in Figure 18.

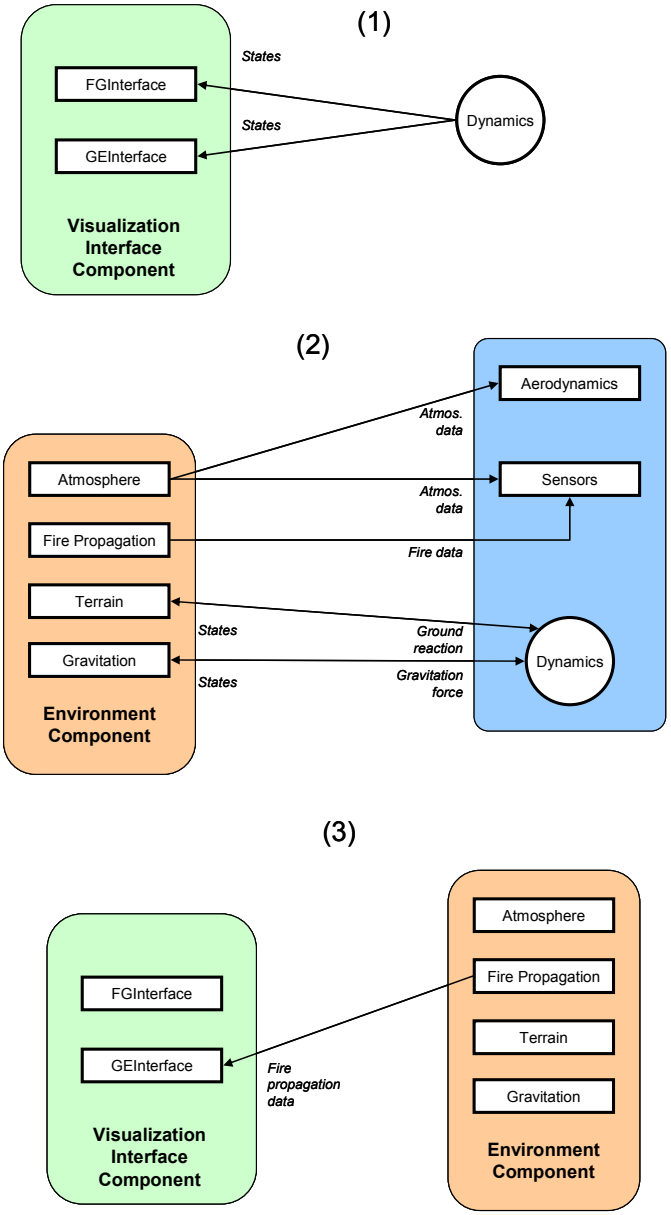

Figure 18 - UAV Simulator Logical Breakdown Detailed Interfaces.

The requirements of each specific model are described in Section 3.2.3. A more thorough description of the models not implemented in this thesis will be provided within this chapter. Further detail and information concerning the implementation of these models can be found within [RD-22]. The Automatic Flight Control System (AFCS), Sensors, Actuators and Vehicle Specific Model (VSM) will be defined in the appropriate sections.

The Dynamics model will receive force and moment data from the Propulsion, Aerodynamics, Gravity, and Terrain models and will integrate the equations of motion in time, providing the trajectory of the air vehicle. The Propulsion and Aerodynamics models are responsible for determining the forces and moments generated by the engine and the aerodynamic surfaces, respectively. The Atmospheric model is responsible for providing the simulation with

atmospheric parameters and wind data. Note that if e.g. the AFCS model requires the current pressure level, it must read this value from the Sensors model since in the real world the considered value is affected by sensor performance. However, if the Aerodynamics model requires the current pressure level for e.g. calculating a lifting force, it must read this value from the Atmospheric model directly, since that value will be the real value. The Gravitation model is responsible for reading the current UAV state and determining the gravitation force at its position. The Terrain model must provide the Dynamics model with the ground reaction force that exits when the UAV is in contact with the ground. The objective of the Fire Propagation model is to simulate the existence of a fire so that it is possible to demonstrate that the UAV is able to follow the fire front.

The visualization Interface Component is composed of the Google Earth and Flight Gear. The interaction with Google Earth is performed through Google Earth KML files (which are actually XML files) and involves the reading of flight plans defined using Google Earth into the simulation and the writing of the UAV position to be read and displayed by Google Earth. The interaction with FlightGear is performed through a network connection, and involves the sending of current aircraft states into FlightGear, for visualization purposes. A possible functionality that may be of interest is the visualization in Google Earth of the fire front as simulated by the fire propagation module.

4.2.2 Generic Model Architecture Breakdown

To promote modularity, the implementation of each model is done inside a C++ class that does not make use of any simulation standard. This class therefore requires an adapter that will instantiate it and make use of its public methods, associating its functionalities with code compatible with a simulation standard (e.g. SMP1) to allow it to run on a simulation platform (e.g. SIMSAT 3). In this work, each implementation class has associated SMP1 code that handles aspects such as publishing and scheduling, so the class will be able to communicate with SIMSAT and other models. This concept is illustrated in Figure 19. The Implementation class is responsible for the implementation of each model at the functional level. This component holds all the data and methods necessary to execute the model, almost independently of the Interface Layer.

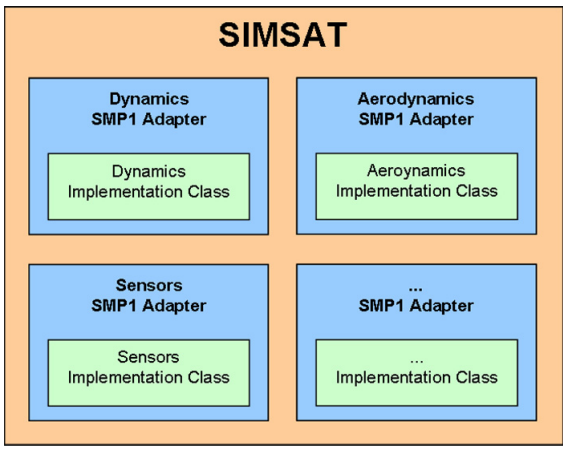

Figure 19 - Generic SIMSAT Architecture.

The purpose of this architecture is to separate the Implementation from the Adapter to a specific standard. In this way, it is possible to easily reuse and adapt the Implementation code in various simulation environments and standards, namely SMP2 for SIMSAT version 4. Therefore, the simulator is assured to have higher reuse capability and longevity by supporting the most modern simulation standards and environments.

4.2.3 Automatic Flight Control System

Overview

The purpose of the Automatic Flight Control System (AFCS) is to engineer a simple state feed-back controller and high level logical functionalities to control the aircraft. This model aims to be a demonstrator for the UAV Simulator, and not in any way a replacement for a real controller, as this is clearly outside of the scope of this thesis, and a complete aircraft controller would be one project of its own. This component retrieves the sensed aircraft states and performs the necessary autopilot calculations to provide control inputs to the actuators. The control law used is based on modern control (e.g Linear-Quadratic Regulator with output weighting). This is specified next within the Formulation. Several additional layers exist to support the Flight Control Mode requirements, also defined next within the Processing.

Formulation

The Controller was designed and tested on Matlab/Simulink, based on the wind tunnel Aerodynamic data available for the pioneer UAV from [RD-38]. The controller state feed-back algorithm was implemented as C/C++ code in the AFCS model.

For this project, the controller was based on a modern control technique, namely Linear-Quadratic state-feedback with output weighting (lqry). This is a widely known design technique, and references can be found on [RD-39] and [RD-40]. The details of the design have been omitted for simplicity, thus, only the procedure and results will be shown. The following requirements have been identified for the controller:

- The controller is designed to operate for a steady flight condition of 66 Knots (approximately 34 meters per second), with a steady pitch angle and angle of attack of 6 degrees at sea level altitude. These are the steady state conditions for levelled flight in the Pioneer case.

- The controller is able to follow desired Air Speeds, Altitudes, Altitude Rates, Headings and Heading Rates around the steady state condition, as well as Altitude and Heading Commands, to comply with the operator commands previously defined in section 3.2.2.

- In addition to the command inputs stated above, the controller is also expected to receive the following aircraft states: Angle of Attack, Pitch Rate, True Airspeed, Pitch Angle, Altitude, Side Slip Angle, Roll Angle, Roll Rate, Yaw Rate and Heading Angle.

The Control method is based on the plant

$$\begin{aligned}\dot{\mathbf{x}} &= \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \\ \mathbf{y} &= \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u}\end{aligned} \tag{1}$$

where $\mathbf{x}$ is the perturbed State Vector, and $\mathbf{u}$ the perturbed Inputs Vector. $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, $\mathbf{D}$ are coefficient matrices. To satisfy the requirements, the following aircraft State Vector has been defined:

$$\mathbf{x} = \begin{bmatrix} \alpha & q & vt & \theta & h & \beta & \phi & p & r & \psi \end{bmatrix}^T \tag{2}$$

where: α is the angle of attack; q is the pitch rate; vt , is the true airspeed; θ is the pitch; h is the altitude; β is the sideslip angle; ϕ is the roll angle; p is the roll rate; r is the yaw rate; ψ is the yaw angle. The Inputs Vector is:

$$\mathbf{u} = \begin{bmatrix} \delta_e & \delta_t & \delta_a & \delta_r \end{bmatrix}^T \tag{3}$$

where: δ_e is the elevator deflection; δ_t is the propulsion actuation; δ_a is the ailerons deflection; δ_r is the rudder deflection. The $\mathbf{A}$ and $\mathbf{B}$ matrices depend on the stability coefficients of the aircraft. Following the considerations in [RD-39], the matrices $\mathbf{A}$ and $\mathbf{B}$ can be constructed with the data available for the Pioneer on [RD-38]. The $\mathbf{C}$ matrix is an identity matrix since all aircraft states are observable. The $\mathbf{D}$ matrix is null.

For the purpose of automating the process a MATLAB script is used. It provides a function that automatically calculates the controller gains, and supplies the Simulink model with the necessary variables to run. Figure 20 presents the Simulink block diagram used as a prototype to the controller.

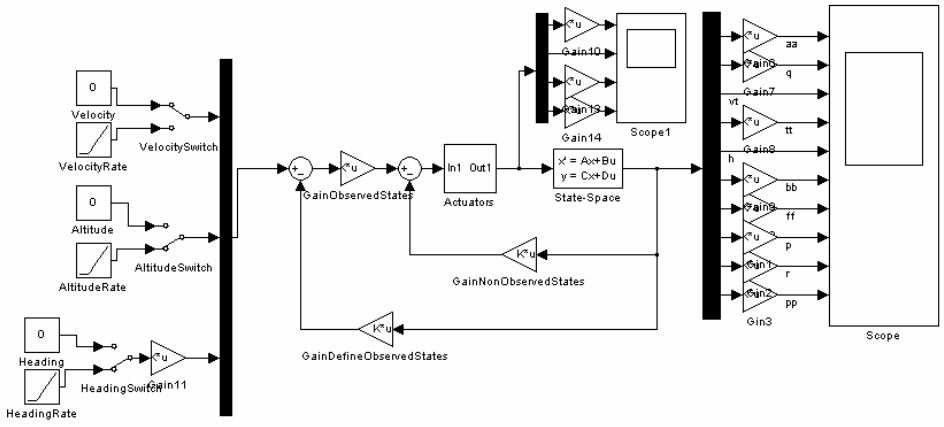

Figure 20 - Simulink block diagram.

After some iteration on the lqry tuning parameters, a good controller can be defined, and the implementation of the algorithm on the AFCS model can be performed. A complete validation and assertion of the controller responses and flight levels is not formally performed due to the demonstration nature of this model. Only the mathematical part of the problem has been solved so far. Above this implementation, a set of logical functions must be set to allow the aircraft to fly safely and provide the SO with a simple commanding interface.

Processing

This section defines how the model operates internally. This model is divided in several layers relating to the functionalities each of them provide:

- ImplementationController
- ImplementationControllerAdapter
- FlightPathControlModes

The ImplementationController layer is the state feedback layer presented in the Formulation Section. This is where the states are multiplied by gains to produce actuators commands. The ImplementationControllerAdapter is created to provide the ImplementationController layer with the states and references and execute it. This is necessary to control the states and references that are sent to the ImplementationController (e.g. The ImplementationController, receives altitude, however the command given is climb at constant altitude rate. The necessary adaptation is made at the ImplementationControllerAdapter level). This layer directly receives the following commanding parameters:

- SetAirspeed
- SetAltitudeCommand
- SetAltitude
- SetAltitudeRate
- SetHeadingCommand
- SetHeading
- SetHeadingRate

The FlightPathControlModes layer is the highest level. This layer is divided in multiple functions, one for each Flight Path Control Mode. The respective function is called directly by a periodic update function that is responsible for executing the model. These are responsible for setting up the references for the ImplementationControllerAdapter layer and executing it. In this work there are six possible modes:

- Manual Flight
- Waypoint Following Flight
- Circular Loiter Flight
- Autoland
- Take-Off
- Follow Fire Front

This flow of information is describing on how the layers interact with each other is depicted in Figure 21.

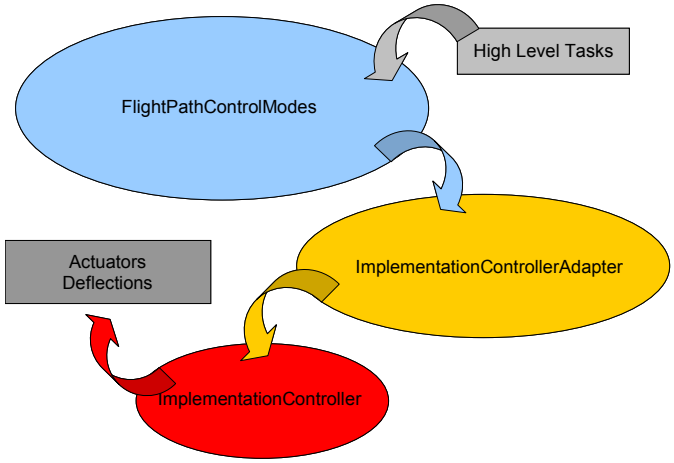

Figure 21 - Automatic Flight Control System processing.

In Manual Flight mode, the controller directly follows the commanding parameters as they are specified by the user. In Waypoint Following mode, the controller follows a specified list of waypoints. This list can be loaded using GoogleEarth Interface, or can come from the VSM Model. When within this mode, the controller will automatically follow the Flight Plan, starting on the first waypoint. When the last waypoint is achieved, the controller will automatically engage the Circular Loiter mode. In the Circular Loiter mode, the controller attempts to follow a circular pattern over the point where it was first engaged. The algorithm is simple: when the mode is first engaged plot a Flight Plan consisting of eight waypoints defined over a circle around the loiter location, and then follow the waypoints in order. In Autoland mode, the aircraft must first be within the Instrumented Landing System glide slope. If this is not verified, the aircraft will maintain its current altitude and heading. In normal conditions, the aircraft will align itself with the runway alone, and adopt the descent rate given by the glide slope until it touches the runway and comes to a full stop. In Take-Off mode, the aircraft should first be stopped and fully aligned with the runway. After engaged, the aircraft will start its roll, gaining speed. When the airspeed is sufficient, the aircraft will leave the runway and start its climb. In Follow Fire Front mode the aircraft will retrieve one waypoint from the Follow Fire

Front Model and follow it if a fire has been detected. If no fire was detected, then the aircraft will maintain its heading and altitude. The responsibility for generating this waypoint belongs to the Fire Front Following Model, as defined in [RD-22].

4.2.4 Sensors

Overview

The Sensors model retrieves the actual simulated aircraft states (e.g. position, orientation), and provides this information to other models in the simulator as sensed states, including a small delay when applicable (to simulate sensor lag). This model is also responsible for generating an Instrumented Landing System vertical and horizontal glide slope to assist in the aircraft landing.

Processing

This component reads published values from the Dynamics and Atmospheric models, and makes them available for the aircraft systems, namely the AFCS and VSM. Whenever necessary, vector decomposition and unit conversion will be done. When applicable, a first order lag filter is implemented in order to more accurately simulate a real sensors dynamic response. The first order lag filter is implemented using the discrete equation:

$$y_k = \alpha \cdot y_{k-1} + (1-\alpha) \cdot u_k , \tag{4}$$

where y_k is the output of the filter for the step k, y_{k-1} is the previous output, u_k is the input of the filter for the step k, and α is the discrete time pole.

The generation of the Instrumented Landing System error signal is also simple. The location of the beginning and end of the runway, in conjunction with the location of the aircraft are used to determine the error in relation to the ideal glide slope that would be passed to the aircraft.

4.2.5 Actuators

Overview

The Actuators model deals with the simulated aircraft actuator surfaces (ailerons, elevator and rudder). It retrieves the desired actuator surfaces positions from the AFCS model, apply a first order lag filter (as defined for the Sensors model) over these values and make them available for the Aerodynamics model to calculate aerodynamic forces.

4.2.6 Vehicle Specific Model

Overview

The VSM model implements the Vehicle Specific Module for the Simulator. This model acts as a standalone network server, waiting connections from a ground station. Communication is performed by a set of messages as specified in STANAG 4586. This component gathers all the necessary data from the Sensors, Actuators and AFCS models and wraps it according to the structure specified by STANAG 4586 (see Section 3.2.1), acting as a telemetry generation and telecommand sink module for the CUCS.

Processing

This model acts as a network TCP[3] server, capable of supporting multiple CUCS clients. For visual guidance of the process, Figure 22 has been provided.

This model contains a shared memory that is defined by two structures, namely the shared input memory and the shared output memory. Parallel to these two shared memories, there are regular memories. Publication to the SMI is performed with the data available on the regular memories due to inter-process synchronization issues. Input to message encoding and output from message decoding are always based on the shared memory sets. To synchronize the access to these memory structures, a set of two semaphores is used. Periodically, bound to the model update function, the shared memories are synchronized with the regular memories by copying the entire memory blocks.

When started, the TCPServer creates a thread that will wait for inbound connections on a given port (specifiable by the user). When there is a successful connection from a new client, the server spawns a new thread to handle that particular connection and returns to its previous state of waiting for new inbound connections. The newly spawned thread instances an UDP[4] client to be able to send periodic UDP telemetry messages. This thread will hold for a specifiable time, waiting for messages. If no message is received meanwhile, the thread will timeout and close that particular connection. This procedure is necessary to drop client connections that are no longer present, but still consuming resources. Whenever a telecommand message is received, the VSM decodes it and publishes its values to the shared output memory. A set of periodic functions have been published and scheduled in SIMSAT to trigger the sending of the periodic telemetry messages. The algorithm behind the sending of these messages is simple. If a given TCP connection is active, then send an UDP message to that client. This model architecture guarantees that there is no memory corruption due to the possibility of multiple accesses to the same memory address by different processes,

[3] The Transmission Control Protocol (TCP) is one of the core protocols of the Internet protocol suite, often simply referred to as TCP/IP. Using TCP, applications on networked hosts can create connections to one another, over which they can exchange data. The protocol guarantees reliable and in-order delivery of data from sender to receiver.

[4] The User Datagram Protocol (UDP) is one of the core protocols of the Internet protocol suite. Using UDP, programs on networked computers can send short messages sometimes known as datagrams. UDP does not provide the reliability and ordering while TCP does. Datagrams may arrive out of order, appear duplicated, or go missing without notice. Without the overhead of checking if every packet actually arrived, UDP is faster and more efficient for many lightweight or time-sensitive purposes. Also, its stateless nature is useful for servers that answer small queries from huge numbers of clients.

and the reception of messages is asynchronous. At any time, the entire process can be shut down, and this command will terminate all threads.

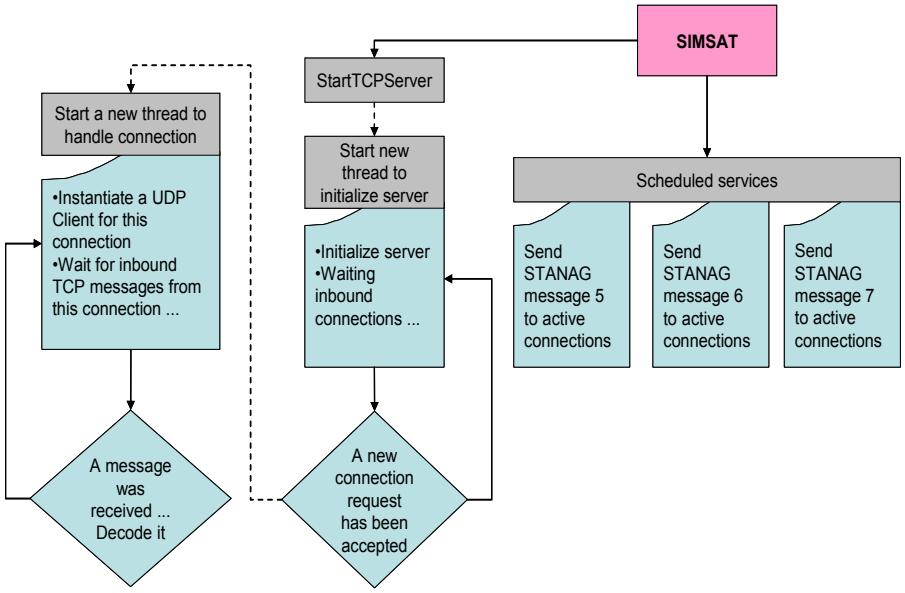

Figure 22 - VSM Terminal processing.

4.3 UAV Console

4.3.1 System Logical Breakdown

The requirements for the UAV Console are defined in Section 3.3 and served as a starting point into this phase. As mentioned, the console follows a modular and a service oriented structure to increase scalability of the console. The Logical Breakdown is shown in Figure 23. The Console is implemented in a way that easily allows the addition of new functionality. The modules implemented compose a baseline for further work. Each of these modules will be analysed in the appropriate sections.

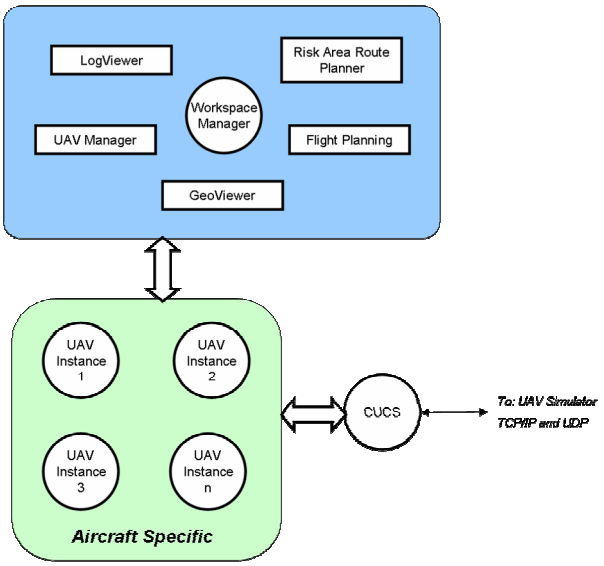

Figure 23 - UAV Console Architectural Breakdown.

There is a separation between Aircraft Specific modules from the rest of the Console. The simulator was produced in an open architecture regarding the addition of new windows to monitor and control specific UAVs. A specific UAV type can have functionalities that others do not, and this must be reflected in the interaction with the operator. An unlimited number of UAVs can be handled at the same time by the console. Therefore, each UAV Instance is no more than a set of windows for monitoring and controlling one particular UAV of a given type and logic to support the generality of having multiple UAVs being handled concurrently.

4.3.2 Workspace Manager

Overview

The Workspace Manager is a module that allows the user to control the window visibility of the main modules: GeoViewer, LogViewer, UAV Manager and Flight Planning. It also serves the purpose of saving and loading the other windows layouts on screen to provide the SO with the ability to easily organise the work environment.

User Interaction

Figure 24 provides the window for this module. Buttons to save and load the layouts of the major five windows are provided (Workspace Manager, GeoViewer, Flight Planning, UAV Management, LogViewer).

A “Lock Display” button is provided to avoid accidentally hiding or moving a window out of place.

Check boxes near the name of other four windows are provided to allow the SO to hide or show any of these windows. This removes unnecessary information from the window of the SO, allowing him to focus on what is relevant.

Figure 24 - Workspace Manager window.

4.3.3 GeoViewer

Overview

The GeoViewer is where the hearth of the Geographical Information Systems is located. In order to simplify the integration of further functionality at a later time, the interaction with OpenEV is done within this module. This allows further improvements to be done for the console without having the need to interact with this external library, which is not trivial. In practice, this module is responsible for showing maps, aircraft locations, flight plans and any information to be displayed on top of a map.

User Interaction

Figure 25 presents the window for this module. The SO can visualize flight plans, UAVs and flight paths within this window. Functionality for controlling the view is also provided. Zooming and Panning can be done with the mouse or the keyboard. Raster or Vector map for an extensive number of formats (see section 3.3.3) can be loaded from a file. The SO can also choose to hide or show different layers within the view:

- Raster (map): This provides a map as an image.
- Vector (line map): This provides a map as a set of contourns.
- Regions (polygons): This provides an overlay with special regions of interest (e.g. no fly zones)
- Flight plans (lines): This provides a line for each Flight Plan.
- Flight paths (lines): This provides a line for each UAV Flight Path.
- UAVs (points): This provides the current location of each UAV.
- Objects (points): This provides the last known location of a given object.

Figure 25 - GeoViewer window.

4.3.4 Flight Planning

Overview

The purpose of the Flight Planning is to allow the Simulation Operator to deal with Flight Plans, manage them internally and make this information available to the other modules. There is no limitation to the number of flight plans, or waypoints this module can manage.

User Interaction

Figure 26 presents the window for this module. This window presents the user with the ability to load and save flight plans. Flight plans are no more than a sequence of waypoints with coordinates (latitude, longitude and altitude) connected by lines. The SO can also create new flight plans selecting this mode and clicking on the GeoViewer the location of the new waypoints. The SO can choose as many waypoints as necessary. An Edit function is also available, to allow the editing of the waypoints from the GeoViewer. This module also provides a button to call the Risk Area Route Planner window.

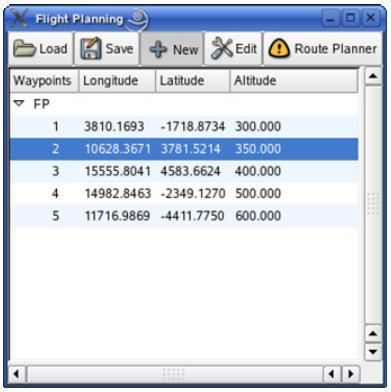

Figure 26 - Flight Planning window.

4.3.5 UAV Manager

Overview

The UAV Manager module allows the SO to initiate new sessions in which he can monitor and control UAVs. It is important to note that an unlimited number of UAVs can be monitored, and that any or all windows for all the UAVs can be visible or hidden. Whenever a new session is initiated, a UAV Instance is instantiated to allow the user to monitor and control that particular UAV. This module also provides some internal logic to handle the different UAV Instances.

User Interaction

Figure 27 presents the window to this module. Two buttons allow the SO to initiate or terminate connections. A list of all the currently opened sessions is also provided. For each UAV there is also a list of all the UAV specific windows available to it, as well as the option to hide or show them.

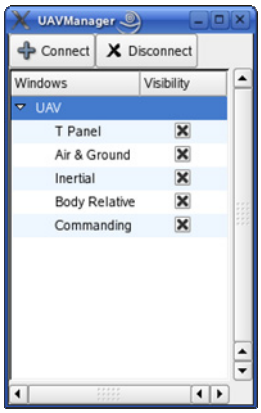

Figure 27 - UAV Manager window.

4.3.6 LogViewer

Overview

The LogViewer is a simple module to allow messages generated from other modules to be recorded and presented to the user. This module allows three types of messages: Information, Error and Debug.

User Interaction

Figure 28 presents the window to this module. Three check boxes allow the SO to choose whether each message type is visible or not.

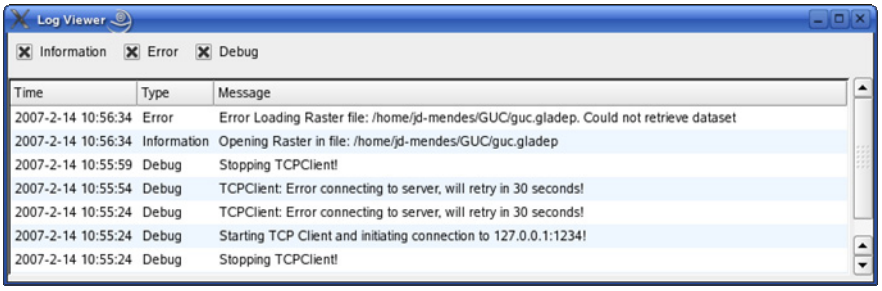

Figure 28 - Log Viewer window.

4.3.7 Risk Area Route Planner

Overview

The Risk Area Route Planner provides the SO with the ability to automatically generate flight plans based on quantifiable information regarding risk areas. Within the simulator, the purpose of this algorithm is to automatically plan a mission to optimize the fuel and maximize the surveillance of the higher risk areas by using a fire risk map. The module can be divided in two sections, the window that provides the interface to the user and the algorithm that provided the functionality mentioned above. The following sections will present the interface to the SO and the internal processing of the algorithm developed for this thesis. The development of this algorithm is necessary due to the fact that no suitable algorithm could be found after an extensive research of publications and papers on the subject.

Processing

This section describes the algorithm developed in this thesis, in order to generate the best route based on geo-localized raster risk information by the application of a search algorithm, while being restricted by a maximum allowable range, which is usually the case on vehicles. In order to structure the problem, the algorithm is divided in two steps:

- Node Generation - An algorithm that generates and applies a mesh to the raster containing the risk information, and provides a list of best explorable nodes, within the given geo-localized bounds.

- Route Planning - An algorithm that determines the best route between a given number of nodes, respecting a maximum possible range. If necessary, not all nodes are visited, but no node is visited twice.

In either case, the solution is independent of the problem. Each of the two phases will be documented in the following sections. However, it must respect the following conditions/restrictions:

- The Risk Information must be provided in a Raster format map, where each pixel corresponds to a risk value, or a gain depending on the context of the problem.

- The user must provide the area where to perform the search on the map, as well as the maximum allowable range for the vehicle.
- The user must provide the size of the cell that will compose each node.
- The user must provide a start and a finishing location on the map.
- The user must provide the number of nodes that will be generated.

Node Generation

In order to better understand the algorithm's processing, Figure 29 provides a visual guidance. The processing of this algorithm is:

- Isolate the area where the search is conducted based on the user provided bounds.
- Divide this area in smaller rectangles corresponding to the size of cell provided by the user.
- Perform an average of all the pixel values in each of the cells and store this information.
- Generate a list of possible explorable nodes with the N (provided by the user) highest value nodes. This list contains the location of the nodes as well as its gain, and is the input to the second phase.

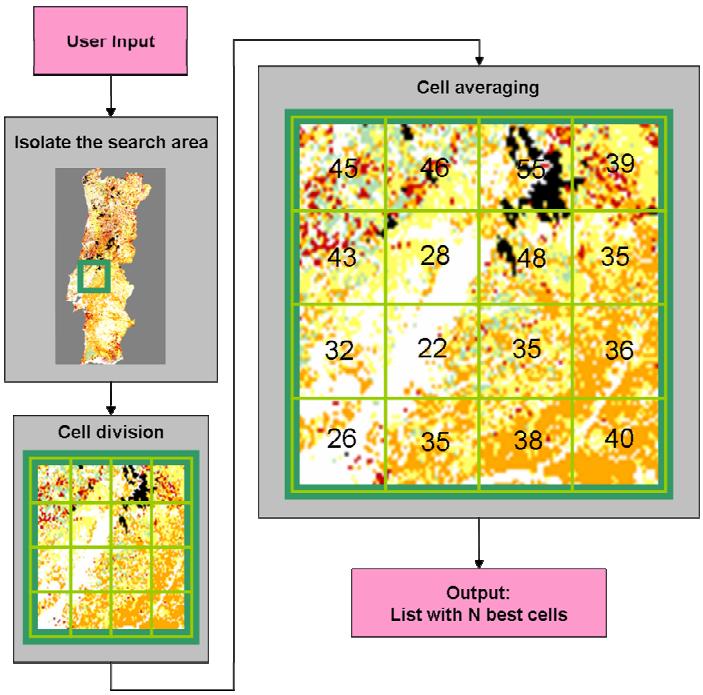

Figure 29 - Node Generation processing.

Route Planning

The developed algorithm for this phase is heavily based on the A* (A star) search algorithm [RD-41]. This is a widely know algorithm to solve route planning scenarios. However it is not directly applicable to the current problem and its restrictions. The A* is a graph search algorithm that finds a path from a given initial node to a given goal node. It employs a heuristic estimate that ranks each node by an estimate of the best route that goes through that node and ends at the goal node. It visits the nodes in order of this heuristic, making it a best-first search type algorithm. The follow characteristics can be enumerated:

- It is complete in the sense that it is guaranteed to find a solution if one exists.
- If the heuristic estimate is admissible (never overestimates the cost to the goal node), then the algorithm is optimum if a closed set is used. This guarantees that it finds the best possible solution.
- A* is optimally efficient for any given heuristic, meaning that no other algorithm employing the same heuristic will expand fewer nodes, except for some specific cases where the heuristic predicts the exact cost of the optimal path.

The nature of the problem, more specifically the geo-localization and attributes of each node, makes it not possible to directly use the A* algorithm, and forces the development of an appropriate heuristic estimate applicable to the problem. The problem at hand requires that the algorithm finds a route from an initial node to a goal node by optimizing the gain, without ever going over the vehicle range limit. The gain can be understood as the symmetric of the cost in the conventional A* formulation. The gain of a node is obtained by averaging the pixel cells (this happens in phase one). The gain of a route can be found by summing the gains of each visited node:

$$g(n) = \sum_{i=1,n} g_i \ , \tag{5}$$

where g_i is the gain for the i-th node. A route is a sequence of visited nodes, where the order is relevant and must be stored. Therefore, the following equation is valid to assign a priority to a path, or also knows as the overall gain function:

$$f\left(n\right) = g\left(n\right) + h\left(n\right) \tag{6}$$

In the context of this problem, a heuristic estimate is admissible if it never sub-estimates the gain to the goal node. A heuristic for the problem that includes both the gains and the range characteristics of the problem is:

$$h\left(n\right) = r(n) \cdot K \ , \tag{7}$$

where

$$K = \frac{G^{+}}{D^{-}} \tag{8}$$

and $r(n)$ is the range still available to the vehicle in this particular route, G^{+} is the highest gain of all nodes, and D^{-} is the minimum distance between any two nodes.

As a consequence of the defined formulation, every route must maintain a list of all visited nodes including the order they were visited. Preferably, it should also maintain a variable holding the remaining distance the vehicle can travel in the context of the problem, as well as the accumulated gain so far. The last two variables are not mandatory as they can be calculated by knowing the list of visited nodes; however it considerably speeds up the process at cost of a small amount of memory. The processing of the algorithm is depicted in Figure 30. There are three lists to maintain the routes on every step:

- Open - This list maintains all the valid routes that have yet to be explored.
- Best - This maintains only one and the best route so far that reaches the goal node.
- Current – This maintains only one and the currently being analysed route.

The initial node supplied by the user is used to create a route that has only one node. This route is placed in Current and only happens at the start. Note that valid routes are routes that have an overall gain function higher than the current best route.

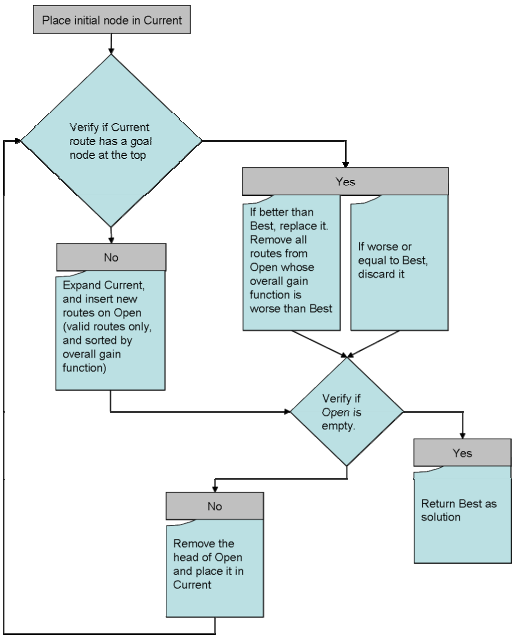

Figure 30 - Route Planning processing.

User Interaction

Figure 31 presents the window to this module. This window allows the SO to choose the risk map that will be used to plan the route. Two buttons allow running each step of the algorithm: Node Generation and the Route Planning. By using this separation, the user will be able to edit the location of the nodes Generated before applying the Route Planning algorithm. The SO has a number of fields to fill with the information that will be used by the algorithm:

- The location of where the route generation is centred, as well as the radius to which it is extended.
- The size of each cell as defined in the Node Generation phase. It also allows the SO to choose the output number of cells from this phase. This has an extremely high impact on the time the algorithm takes to run and find a solution.
- The band of the raster where the information regarding risk area is located. This is necessary because one image may contain various sets of data, and this must be identified.
- The location of the First and Last node in the Route Planning phase. This determines where the aircraft starts, and where it ends.

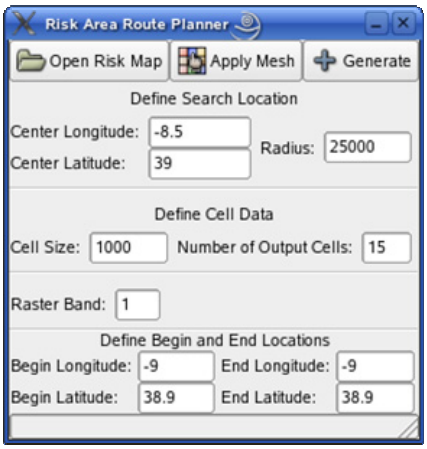

Figure 31 - Risk Area Route Planner window.

4.3.8 UAV Instance

Overview

The UAV Instance module provides the SO with the interface to the UAVs themselves. This is composed by a set of windows that are specific to a given UAV type. Different UAVs could require different functionalities, and this must be taken into consideration when displaying the information to the SO. Therefore this module was developed so that it is easy to create new windows in future Console development. This module also provides the servers to receive and transmit network data, and the encoding and decoding facilities which implement the STANAG 4586 protocol. Within this thesis, this module is composed by the following windows that will be present in the appropriate sections:

- Basic T – This window displays basic information regarding the main aircraft parameters.
- Air & Ground Relative States – This window provides parameters relative to air and ground states.
- Inertial states – This window provides parameters relative to inertial states.
- Body relative states – This window provides parameters relative to body relative states.
- Commanding window – This window enables the SO to command the UAV.

UAV Commanding

As already mentioned, there are several windows associated with each specific UAV being simulated. One of these windows is the Commanding window, as depicted in Figure 32.

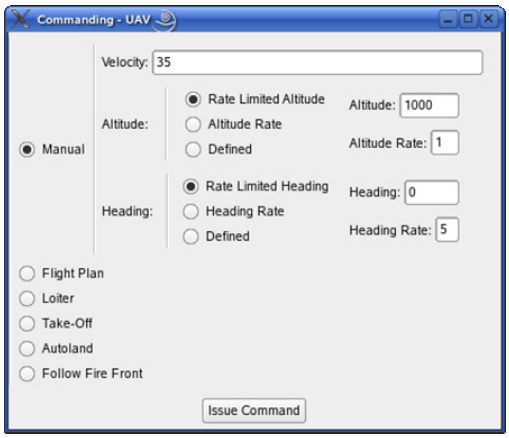

Figure 32 - Commanding window.

This window allows the user to command the UAV. The list of commands available is in concordance to the list of required commands specified in Section 3.2.2. Several options are available:

- Manual Mode
- Waypoint Following Mode
- Loiter Mode
- Take-Off
- Autoland
- Follow Fire Front

The following section presents the functionality of each option: A common feature to all options is, after the desired option is configured, the Command button should be pressed in order to issue the command to the UAV..

Manual Mode

This mode allows the user to command the UAV manually. A set of parameters can be chosen with radio buttons and text entries:

- Velocity: Desired velocity for the UAV
- Altitude:
    - Rate Limited Altitude: Go to the specified Altitude with the provided Altitude Rate.
    - Altitude Rate: Assume the specified Altitude Rate.
    - Defined: Go immediately to this altitude.
- Heading:
    - Rate Limited Heading: Go to the specified Heading with the provided Heading Rate.
    - Heading Rate: Assume the specified Heading Rate.
    - Defined: Go immediately to this heading.

Waypoint Following Mode

This mode uploads a waypoint to the UAV. To do so, one must first select the Flight Plan to upload in the Flight Planning window. By pressing the Issue Command button, the UAV will automatically start following the first waypoint of the new flight plan.

Loiter Mode

When this mode is selected, the UAV will start a circular loiter over the current location. The AFCS is responsible for the Loiter characteristics.

Take-Off Mode

The aircraft must be correctly placed in a runway before issuing this command. When the command is issued, the aircraft will start its roll on the runway, gaining airspeed, and lift off when enough airspeed is attained.

Autoland Mode

This mode should only be engaged when the aircraft is in the Instrumented Landing System glide path cone generated from the SimUAV Sensors Model. If this condition is not verified, the UAV will simply maintain its current altitude and heading until a new order is issued or the UAV enters the glide cone. If the UAV is in the glide

path, the Automatic Flight Control System will make a controlled descent along the glide path, land smoothly and come to a full stop on the runway.

Follow Fire Front Mode

This mode activates the Follow Fire Front model. This mode attempts to follow a fire front if one is detected. In the case that it is not detected, the UAV will simply maintain its current Altitude and Heading.

UAV Monitoring

The several windows associated with Monitoring the UAV progress are depicted within this section.

Basic T – This provides the six most important parameters for an aircraft, following the standard location in an instruments panel (hence the name "basic T"). It contains airspeed, pitch, altitude, roll angle, heading, and vertical speed information as depicted in Figure 33.

Figure 33 - Basic T monitoring window.

Air & Ground relative states – This provides a variety of parameters that are measured relatively to the airflow outside of the aircraft, and the ground beneath it, as depicted in Figure 34.

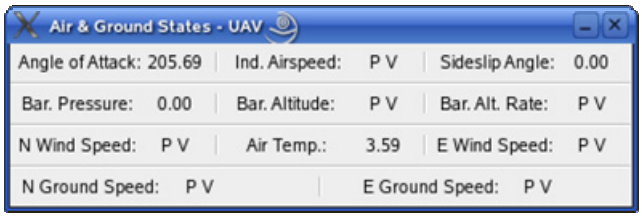

Figure 34 - Air & Ground states monitoring window.

Inertial states – This provides a set of parameters that are measured relatively to the (assumed) inertial frame, including Latitude and Longitude, as depicted in Figure 35.

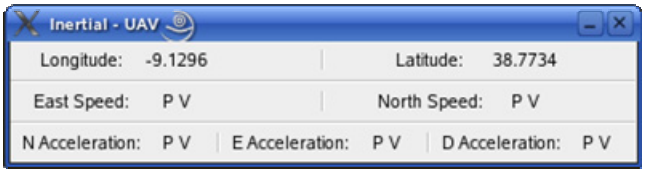

Figure 35 - Inertial states monitoring window.

Body Relative states – This provides a set of parameters that are measured in the aircraft reference frame. This includes aircraft body accelerations and angular rates, as depicted in Figure 36.

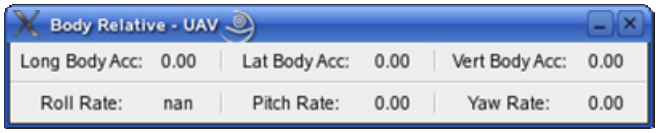

Figure 36 - Body Relative states monitoring window.

4.4 Synopsis

This chapter defined the work executed during the second and third phases of the software development life cycle, Sections 2.4 and 2.5. The software architecture for the UAV Simulator and the Console were also provided. Details regarding the implementation of the Automatic Flight Control System, Sensors, Actuators and Vehicle Specific Model were presented on the Simulator part. Within the UAV Console, a detailed description of the Workspace Manager, GeoViewer, Flight Planning, UAV Manager, Risk Area Route Planner, and UAV Instance was also provided along with figures of the console windows and details regarding the interaction with the user.

Chapter 5

Validation Results

5.1 Introduction

This chapter presents the fourth phase on the software development project life cycle, as defined in Section 2.6. More information regarding the work produced can be found at [RD-21]. Details regarding the UAV used to demonstrate the Simulator can be found in Appendix A. Results obtained from the prototyping phase of the Risk Area Route Planner will be presented. Additionally, a demonstration of a small mission obtained with the Simulator completely integrated will also be shown. The Console will be used to command the simulation run.

5.2 Risk Area Route Planner prototype validation

The implementation of the Risk Area Route Planner (see Section 4.3.7) was performed in steps. The first step aimed at the proof of concept of the algorithm to generate the flight plan from a list of possible nodes (defined as Route Planning phase). The algorithm in question was developed exclusively for this thesis. It was not supported by any previous work other than being based upon another well known search algorithm. However, in order to demonstrate that the algorithm is feasible, it was implemented in a prototype just for demonstration and validation purposes. One run from the prototype will be presented here.

This algorithm accepts a list of possible explorable nodes along with their location and gain to produce results, in addition to the location of the route starting and ending node. To support the prototype testing, a total of 15 nodes were automatically generated with a random function, both for the gain of these nodes, as for the location. The nodes are restricted in a Cartesian plane with coordinates between 0 and 100. The same method was used to generate the value of the nodes. The route stating node is located at (0,0), and the end node at (100,100). Figure 37 presents the location of the randomly generated nodes in the plain together with an idea of their risk value (the bigger the circle, the higher the risk is in the context of the problem). The algorithm was restricted to a maximum of 200 map units. The output of the algorithm is drawn in blue.

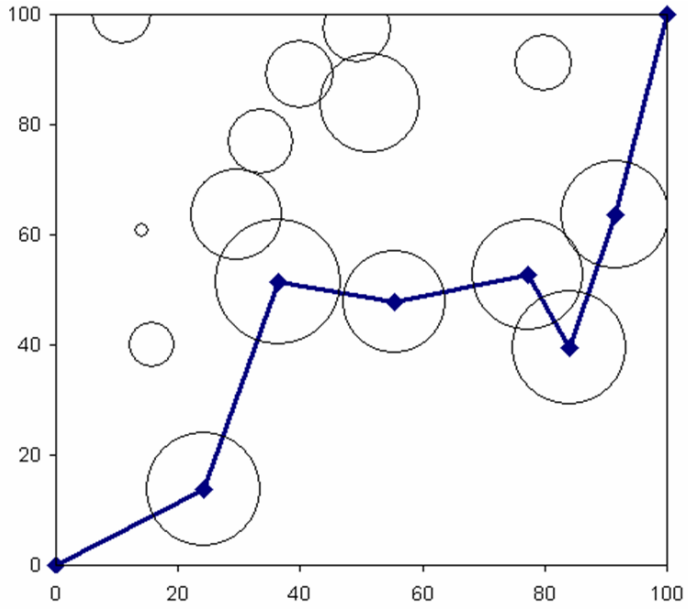

Figure 37 - Route Planning algorithm output route.

The total distance of the generated route is 186.6 map units, lower than the imposed restriction. It is visible that the algorithm successfully completed its purpose: To generate the best route between the initial and final node within the range restriction. However, it is not possible to compare or prove these results analytically. One solution would be to use another already proven algorithm that can solve this particular problem and provide a similar result. Unfortunately, one is not available. Alternatively, the graph tree can be generated to prove that the result is the best possible. A tree graph is a graphical representation of all the paths the algorithm can take, therefore showing a tree with branches that show all the results possible. Figure 38 presents an example of a small graph tree. For 15 nodes, the problem is in the order of 15!, which comes to roughly 1,307,674,368,000 full routes. These routes however do not respect the range restriction of the problem. A more correct approach would be to calculate the permutations of 6 elements from the 15 possible nodes (6 nodes is the number of nodes actually visited by the route in this case). This would still results in a graph with roughly 3,603,600 possible results, which is still not feasible.

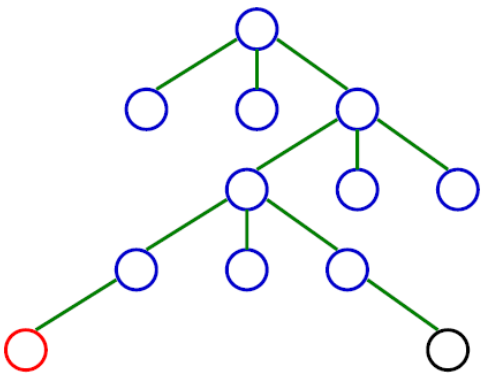

Figure 38 - Example graph tree.

5.3 System Validation

This section presents a simple mission performed by the Simulator. The Console is used to control and monitor the aircraft during all the mission segments: Take-Off, Cruising, Loitering and Lading. Figure 39 presents the last leg of the simulated mission as visible from Google Earth. This image was provided by the Google Earth Interface model.

Figure 39 - Full mission Google Earth screenshot.

Figure 40 presents the fully simulated mission as visible from the console. The red line is the flight path of the aircraft. The blue lines are the flight plans.

Figure 40 - Full mission Console screenshot.

The aircraft started on the air, and assumed a course to the left in the picture. This was done while following a small flight plan that brought it within the Instrumented Landing System glide slope. At that moment (leftmost on the picture), the aircraft was re-planned, and the Autoland mode was engaged (hence the visible blue line meaning that the original flight plan was abandoned before it had the time to be completed) and a controlled descent all the way to the touchdown where it finished the landing procedure after coming to a full stop on the runway. Figure 41 presents a screenshot taken from FlightGear moments before the touchdown.

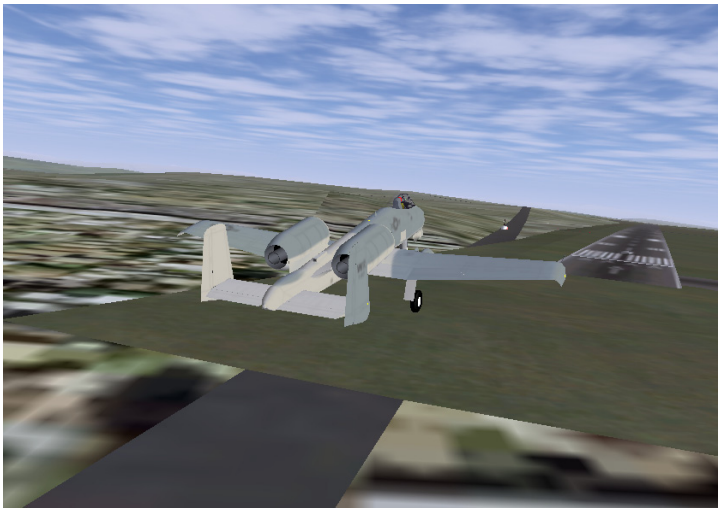

Figure 41 - Landing screenshot with chase camera.

Figure 42 presents the Airspeed, Angle of Attack, Pitch and Altitude during the Landing and Take-off procedures for the aircraft. It is important note that both of these procedures are within the full responsibility of the Automatic Flight Control System.

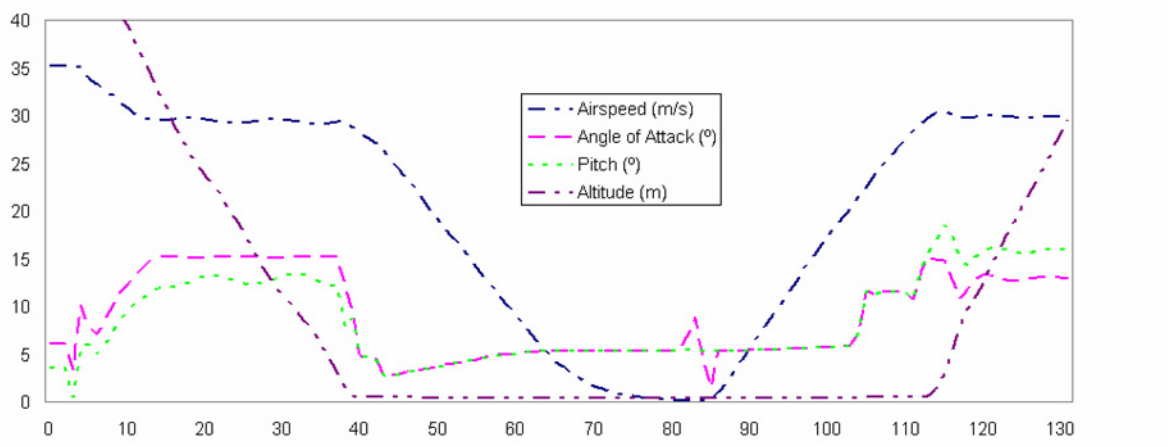

Figure 42 - Aircraft parameters during landing and take-off.

This is the final leg of the descent, moments before landing. The touchdown happens at approximately 40 seconds. The aircraft reduces the airspeed to approximately 30 meters per second just before touchdown. Note that during this period, the angle of attack is about 15 degrees, which is almost closing to stall for the pioneer. Since flaps were not modelled, the aircraft cannot land any slower. After touching down, the angle of attack and pitch are below 6 degrees and then raise. This is due to the torque created by the wheel brakes. The take-off starts at approximately 85 seconds. The small perturbation in the angle of attack is caused by the raising of the throttle to full with the wheel brakes on. When the wheel brakes are released, the aircraft starts to accelerate. At about 105 seconds, it lifts the nose wheel of the runway. This is a standard procedure to avoid damage and unnecessary stress on the front wheel. At about 30 meters per second, the Automatic Flight Control System increases the pitch, and the aircraft lifts off naturally from the runway and starts its climb at constant speed. Additionally, the aircraft also loiter over certain locations. The loiter is circular and maintains the speed and altitude. Figure 43 presents the result of the loiter manoeuvre as seen from the Console. The aircraft came from the top of the image, started loitering with a right turn, and stayed there until orders were issued to carry on the mission to the bottom left of the image.

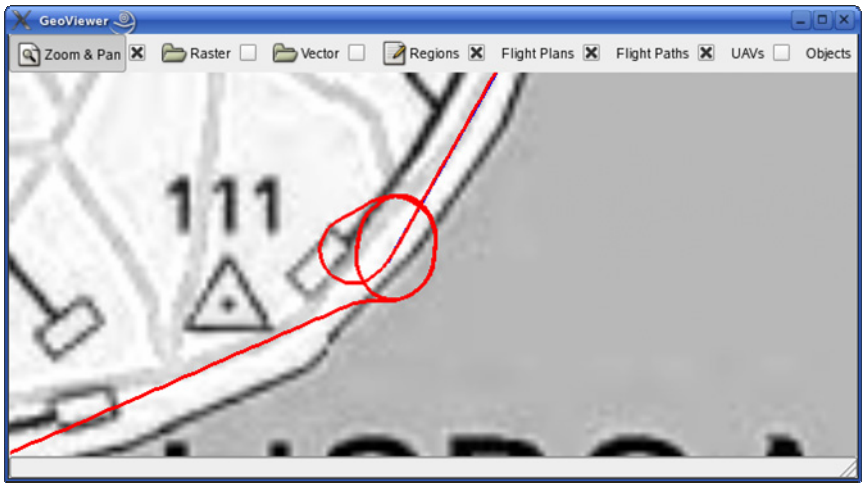

Figure 43 - Circular loiter Console screenshot.

5.4 Synopsis

The Validation phase of the software development life cycle was presented in this chapter. The Pioneer UAV was defined as the platform to validate the simulator. Results from the prototype of the Risk Area Route Planner were also presented. A fully integrator simulator run was also exemplified for a simple mission: follow a mission plan, loiter to monitor an area, land and take-off again.

Chapter 6

Synthesis

6.1 Conclusions

Uninhabited Aerial Vehicles (UAVs) are scaling up in use, and rapidly starting to replace the conventional aircraft for dangerous and monotonous tasks. However, these vehicles require a thorough integration testing and validation before considered fit for active duty and highly specialized operators need several hours of flight training to achieve acceptance. UAV Flight Simulators play an important role towards this goal and the most sophisticated simulators have undoubtedly been developed for space applications where every contingency must be studied to the hearths content of hundreds of engineers. This thesis demonstrated how a space technology was successfully used to create an aeronautical application, from grasping the concepts and the requirements of the project, to providing a UAV Simulator architecture and models.

A functional UAV Simulator was successfully implemented using ESA space simulation infrastructure (SIMSAT). This simulator, developed in a modular and scalable fashion is effectively a baseline to support scientific and industrial work in a cost effective manner. SIMSAT proved to be a stable and straightforward infrastructure to build the simulator upon. This tool enhanced the focus on model development and simplified simulator implementation, reducing time and effort and avoiding errors related with the models execution in real time. The usage of Simulation Model Portability standard allows the models developed to be ported to different simulations and simulation infrastructures that also support this standard. This opens a set of possibilities for reuse and interoperability. The implementation of a simplified Vehicle Specific Model as defined in the STANAG 4586 standard for UAV interoperability provides the Simulator with the state of the art standard interface among current and future UAVs. This exposes a standard and open interface to external systems, such as commercial consoles, effectively placing the simulator among the few that are compliant with this standard. The development of a generic set of Sensor and Actuator models guarantees compatibility for a myriad of mission and air vehicles. The implementation of an Automatic Flight Control System capable of automated cruising, flight plan following, loitering, take-off and landing enables this Simulator to present itself as a technology demonstrator.

A UAV Console was developed to answer the need to easily command and control the aircraft within the Simulator. This Console, while not aiming to be a state of the art product among its kind, is an effective solution to control a

STANAG 4586 compliant simulator and real UAVs. The modular and contained design of the Console also provides a stable baseline for future improvements. The Risk Area Route Planner algorithm developed especially for this work provides an optimal and complete solution to mission planning needs, leveraging down Artificial Intelligence solutions to assist in ground operations.

6.2 Further work

Further work should focus on the further development of the Simulator models and capabilities. The design of the architecture of the simulator is decisive on how the models were implemented. This detail, even tough not presenting itself as of great value at a first glance, defines how future models should be implemented, and current ones upgraded to more precise and specific versions. In practice this is the foundation of all the further work that will be developed. Due to the modularity of the Simulator any functionality may be easily added at a later time. Upgrading current models is also a possibility. Each aircraft has its particular hardware set, and in order to correctly simulate it, specific models must be developed and implemented. This dictates that all the aircraft specific models can be tailored for one specific vehicle. The aircraft independent models remain the same, which proves to be a major advantage of the modularity within the simulator architecture. Also, one could use the Simulator as a research platform, and test models that would later be incorporated in a real aircraft, such as interferometers, ground penetrating radars, radiological and bio-hazard sensors, etc. Due to SIMSAT real time scheduler, interaction with hardware is also possible, thus providing a baseline for hardware-in-the-loop testing and validation. This can directly assists in the development phase of the real UAVs for example.

By achieving compliance with SMP2 and SIMSAT R4.0, the portability and reuse of models would be greatly enhanced by minimising their interactions with the execution environment and standardizing the model's interface, thus increasing the value of the simulator. This can easily be done due to the separation of the core of the models from its interface to SIMSAT, which also allows for the portability of the functionality to infrastructures other than SIMSAT, not necessarily supporting the SMP standard.

The focus on further work for the Console should be driven by the need to develop a particular module to support a new functionality specific to a target UAV. New functionalities must first be implemented in the particular aircraft being monitored and the respective payload, or otherwise risk the development of something that will not fit its purpose. However, this does not remove the generality of the Console to monitor several different UAVs, as the design was fully oriented towards such purpose. The idea is to have a target UAV and mission to sponsor the creation of new modules and windows that are necessary, and later reuse these modules within another platform, in a "plug-in" style. Little work must be done to achieve this goal in the sense that as long as the model is implemented within the Console, any aircraft can automatically be loaded and make use of it. For example, a given forest fire patrol UAV would most likely have an Infrared camera on-board, and therefore a module to support this special payload must be developed for the Console. A completely different UAV with a possible different role could still reuse this module, and therefore be compliant with the Console.

Many improvements can and should be made to the Console on how information is presented to the user, as this greatly influences his situational awareness and responsiveness. Many issues have arisen regarding continued operator exposure to a stressful and complex working environment, leading to physical and mental fatigue which eventually ends in accidents. Not much attention was given to this issue other than that of the architecture of the console itself that allows full customization of windows locations, sizes (when applicable) and visibilities to the user's choice. For example, improvements should be made to replace text labels with parameters by graphical gauges and appliances to make a more comfortable working environment.

References

[RD-1] Z. Sarris, *Survey of UAV Applications in Civil Market*s, 2001.

[RD-2] SIMSAT homepage, http://www.egos.esa.int/portal/egos-web/products/Simulators/ /SIMSAT, 2006.

[RD-3] EuroSim homepage, http://www.eurosim.nl/, 2006.

[RD-4] *STANAG 4586: Standard Interfaces of UAV Control System (UCS) for NATO UAV Interoperability*, North Atlantic Treaty Organization, 2004.

[RD-5] *Flight Simulator*, Wikipedia article, http://en.wikipedia.org/wiki/ /Flight_simulator, 2006.

[RD-6] *European Civil Unmanned Air Vehicle Roadmap*, Volume 3, Strategic Research Agenda, 2005, http://www.uavnet.com/.

[RD-7] *European Civil Unmanned Air Vehicle Roadmap*, Volume 1, Overview, 2005, http://www.uavnet.com/.

[RD-8] W. Niland, B. Stolarik, S. Rasmussen, K. Allen, K. Finley, *Enhancing a Collaborative UAV Mission Simulation Using JIMM and the HLA*, Proceedings of the 2005 SISO Spring Simulation Interoperability Workshop, San Diego CA, 2005.

[RD-9] S. Rasmussen, P. Chandler, *MultiUAV: A Multiple UAV Simulation for Investigation of Cooperative Control,* Proceedings of the 2002 Winter Simulation Conference, 2002.

[RD-10] B. Kim, B. Johnson, R. Youssef, A..Vallerand, C. Herdman, M. Gamble, R. Lavoie, D. Kurts, K. Gladstone, *JSMARTS Initiative: Advanced Distributed Simulation across the Government of Canada, Academia and Indestry – Technical Description*, Defense R&D Canada - Ottawa, 2005.

[RD-11] *Unmanned Aerial Vehicle (UAV) Research Test Bed*, Defence Research and Development Canada, http://www.ottawa.drdc-rddc.gc.ca/html/FFSE-207-uav_e.html, 2006.

[RD-12] *UAV Simulator*, Ness Technologies, http://www.ness.com/GlobalNess/Solutions+and++Services/Command+And+Control+and+Real-time+systems/UAV+Simulators.htm, 2006.

[RD-13] *Unmanned Aerial Vehicle (UAV) Training Simulator*, Fifth Dimension Technologies, http://www.5dt.com/products/puav.html, 2006.

[RD-14] *Pioneer Short Range UAV*, Federation of American Scientists, http://www.fas.org/irp/program/collect/pioneer.html, 2006.

[RD-15] *Advanced Multi-Unmanned Aerial System's Cockpit*, Raytheon Company, http://www.defense-update.com/products/u/UCS.htm, 2006.

[RD-16] O. Lorrain Jr., J. Fiset, *State of the Art: Human-Machine Interfaces for Unmanned Vehicle Systems*, Version 1, 2006.

[RD-17] *Project Life Cycles Management*, Critical Software internal document CSW-QMS-2003-PCS-2296-project-life-cycles, Version 3, 2005.

[RD-18] A. Almeida, A. Relvas, J. Mendes, *UAV Simulator System Requirements*, Critical Software internal document CSW-DPSIM-2006-YRS-1715, All Versions, 2006-2007.

[RD-19] A. Almeida, A. Relvas, J. Mendes, *UAV Simulator Software Requirements Specification*, Critical Software internal document CSW-DPSIM-2006-SRS-1751, All Versions, 2006-2007.

[RD-20] A. Almeida, A. Relvas, J. Mendes, *UAV Simulator Software Detailed Design*, Critical Software internal document CSW-DPSIM-2006-DDS-2771, All Versions, 2006-2007.

[RD-21] A. Almeida, A. Relvas, J. Mendes, *UAV Simulator Test Case Specification*, Critical Software internal document CSW-DPSIM-2006-TCS-4453, All Versions, 2006-2007.

[RD-22] A. Almeida, *UAV Flight Simulator Based on ESA Infrastructure*, Master thesis, 2007.

[RD-23] *Simulation Model Portability Handbook*, Issue 1, Revision 4, European Space Agency internal document EWP-2080, January 2003.

[RD-24] *SMP2 Handbook*, Version 1, Issue 1, European Space Agency internal document EGOS-SIM-GEN-TN-0099, 2005.

[RD-25] *Programming conventions for C++ Software Projects*, Version 3, Critical Software internal document Critical-1998-GBK-0001-04, 2006.

[RD-26] A. Almeida, A. Relvas, J. Mendes, *Analysis of GIS tools*, Issue 1, Critical Software internal document CSW-DPSIM-2006-RPT-5446, 2006.

[RD-27] Open Source GIS homepage, http://opensourcegis.org/, 2006.

[RD-28] Free GIS homepage, http://freegis.org/, 2006.

[RD-29] Demeter homepage, http://www.tbgsoftware.com/, 2006.

[RD-30] Mapnik homepage, http://mapnik.org/, 2006.

[RD-31] deegree homepage, http://www.deegree.org/, 2006.

[RD-32] OpenMap homepage, http://openmap.bbn.com/, 2006.

[RD-33] OpenEV homepage, http://openev.sourceforge.net/, 2006.

[RD-34] GTK homepage, http://www.gtk.org/, 2006.

[RD-35] The Guide to Geographic Information Systems homepage, http://www.gis.com/, 2006.

[RD-36] Geospatial Data Abstraction Library homepage, http://www.gdal.org/, 2006.

[RD-37] OGR Simple Feature Library homepage, http://www.gdal.org/ogr/, 2006.

[RD-38] R. Bray, *A Wind Tunnel Investigation of the Pioneer Remotely Piloted Vehicle*, Master thesis, Naval Postgraduate School, Monterey, CA, 1991.

[RD-39] B. Stevens, F. Lewis, *Aircraft Control and Simulation*, Second Edition, John Wiley & Sons Inc, 2003.

[RD-40] J. Azinheira, *Sumário da disciplina de Controlo de Voo*, 2004/2005.

[RD-41] R. Dechter, J. Pearl, *Generalized best-first search strategies and the optimality of A**, Journal of the ACM (JACM), Volume 32, Issue 3 (July 1985), Pages: 505 – 536, 1985.

[RD-42] Pioneer UAV Inc. homepage, http://www.puav.com/, 2006.

[RD-43] *RQ2-Pioneer*, Wikipedia article, http://en.wikipedia.org/wiki/RQ-2_Pioneer, 2006.

[RD-44] *UIUC Applied Aerodynamics Group: Aircraft Dynamics Models for Use with FlightGear,* M. Selig, R Deters, G Dimock, http://www.ae.uiuc.edu/m-selig/apasim/Aircraft-uiuc.html, 2006.

[RD-45] Federation of American Scientists: article on Pioneer Short Range UAV, http://www.fas.org/irp/program/collect/pioneer.htm, 2006.

[RD-46] Directory of U.S. Military Rockets and Missiles, http://www.designation-systems.net/dusrm/app2/q-2.html, 2006.

Appendix A

Demonstration Platform

A model of a specific UAV platform is necessary for validation purposes. It must include data such as aerodynamic coefficients, inertias, and engine performance. The choice fell upon a model of the RQ-2 Pioneer UAV [RD-42] because detailed non-linear aerodynamic data was available at [RD-38]. Unfortunately there was no engine performance, but the necessary data could be extrapolated from aerodynamic and performance charts. The remaining data could also be found throughout several publications and documents, [RD-42], [RD-43], [RD-44], [RD-45] and [RD-46]. The RQ-2 Pioneer UAV (Figure 44) was developed jointly by AAI Corporation and Israel Aircraft Industries [RD-43]. It is the USA's first deployed aircraft of its kind [RD-42]. The Pioneer has served with United States Navy, Marine, and Army units, deploying aboard ship and ashore since 1986. Initially deployed aboard battleships to provide gunnery spotting, its mission evolved into reconnaissance and surveillance, primarily for amphibious forces. Launched by rocket assist, by catapult, or from a runway, it recovers into a net or with arresting gear after flying up to 5 hours with a 75-pound payload. It relays video in real time via a line-of-sight data link. Since 1991, Pioneer has flown reconnaissance missions during the Persian Gulf, Bosnia, Kosovo and Iraq conflicts.

Figure 44 - RQ-2 Pioneer UAV.

The primary functions of the RQ-2 Pioneer UAV are [RD-43]: Artillery Targeting and Acquisition, Control of Close Air Support, Reconnaissance and Surveillance, Battle Damage Assessment, Search and Rescue and Psychological Operations. General characteristics of the RQ-2 Pioneer UAV are summarized in Table 6.

Generic	
Contractor	Pioneer UAV, Inc.
Manufacturer	AAI
Propulsion & Performance	
Power Plant (for Pioneer RQ-2A)	Sachs & Fichtel SF2-350 piston engine; 19.4 kW (26 hp)
Range	185 km (100 nautical miles)
Fuel Capacity	40-47 liters
Endurance	Approximately 4 hours
Speeds	Stall: 52 knots (96 km/h) Cruise: 65 knots (120 km/h) Maximum: 110 knots (204 km/h)
Service Ceiling	15.000 ft (4600 m)
Geometry	
Length	4.27 m
Height	1.01 m
Wingspan	5.15 m
Aerodynamic Chord	0.5486 m
Wing surface	2.826 m^2
Inertia	
Maximum Weight	205 kg
Empty Weight	178 kg
I_{xx}	47.23 $kg.m^2$
I_{yy}	90.95 $kg.m^2$
I_{zz}	111.5 $kg.m^2$
I_{xz}	-6.646 $kg.m^2$

Data Link	
Frequency	C-band/UHF
Line-of-sight	Yes
Satellite	No
GPS	Yes

Table 6 - General Characteristics of the RQ-2 Pioneer UAV.