

**DEVELOPMENT, ANALYSIS, AND IMPLICATIONS OF OPEN-SOURCE  
SIMULATIONS OF REMOTELY PILOTED AIRCRAFT**

by

© Oihane Cereceda Cantarelo, B.Eng., M.Eng.

A Doctoral Thesis submitted to the  
School of Graduate Studies

In partial fulfillment of the requirements for the degree of

Doctor of Philosophy

**Faculty of Engineering and Applied Science**

Memorial University of Newfoundland

**February 2020**

St. John's

Newfoundland and Labrador



# Abstract

In recent years, the use of Remotely Piloted Aircraft (RPAs) for diverse purposes has increased exponentially. As a consequence, the uncertainty created by situations turning into a threat for civilians has led to more restrictive regulations from national administrations such as Transport Canada. Their purpose is to safely integrate RPAs in the current airspace used for piloted aviation by evaluating Sense and Avoid (SAA) strategies and close encounters. The difficulty falls on having to rely on simulated environments because of the risk to the human pilot in the piloted aircraft.

In the first part of this research, the technical difficulties associated with the development and study of RPA computer models are discussed. It explores the rationale behind using Open-Source Software (OSS) platforms for simulating RPAs as well as the challenges associated with interacting with OSS at graduate student level. A set of recommendations is proposed as the solution to improve the graduate student experience with OSS.

In the second part, particular challenges related to the design of OSS computer models are addressed. Based on: (1) the differences and similarities between piloted and RPA flight simulators and (2) existing Verification and Validation (V&V) approaches, a validation method is presented as a solution to the subject of developing fixed-wing RPAs in OSS environments.

This method is used to design two flight dynamics models with SAA applications. The first computer model is presented in tutorial format as a case study for the validation procedure whereas the second computer model is specific for testing SAA strategies. In the last part, one of the designed RPAs is integrated into a computer environment with a representative general

aircraft. From the simulated encounters, a diving avoidance manoeuvre on the RPA is developed. This performance is observed to analyze the consequences to the airspace.

The implications of this research are seen from three perspectives: (1) the OSS challenges in graduate school are wide-spread across disciplines, (2) the proposed validation procedure is adaptable to fit any computer model and simulation scenario, and (3) the simulated OSS framework with an RPA computer model has served for testing preliminary SAA methods with close encounters with manned aircraft.

# Acknowledgements

This thesis is dedicated to my parents, Luis Cereceda and Adelina Cantarelo, who always believed in my potential and made from me the person that I am today. To my family and dear friends, who encouraged me to stay strong during the most challenging times.

This research would not have been possible without the financial support of the InnovateNL of Tourism, Culture, Industry and Innovation, Newfoundland and Labrador as part of the ArcticTECH research program. I would like to express my sincere gratitude to my supervisors, Dr. Siu O'Young and Dr. Luc Rolland, for their continuous support and assistance during my PhD program. Their motivation and knowledge were vital. I would also like to thank the rest of my thesis committee for their insightful comments and encouragement. I would like to express my very great appreciation to Dr. Itziar Cabanes and her team at the University of the Basque Country (UPV/EHU) during my research stay in the Faculty of Engineering in Bilbao. She is a true inspiration and her support and help were invaluable.

During my program, I have worked with many brilliant graduate students who have been crucial in defining and developing this thesis. I am especially grateful to Danielle Quinn, with whom I shared countless discussions that contributed to shaping one of the chapters of this thesis.

I thank my fellow colleagues Iryna Borshchova, Bruno Artacho, and Robert MacIsaac for the stimulating discussions and guidance they provided in the periods when I had lost all confidence.

The various groups I have been involved with at Memorial University have become my second family. I want to thank each and every single one of the people I met through the MUN Mentors

Program and Women in Science and Engineering Graduate Student Society for their encouragement aside of my research work, without which I could not have succeeded.

Finally, this thesis would have not been possible without the unconditional support, patience and dedication of my partner, Dave Noseworthy, who was always there to cheer me up and encourage me to be positive.

# Table of Contents

<b>Abstract.....</b>	<b>i</b>
<b>Acknowledgements .....</b>	<b>iii</b>
<b>Table of Contents .....</b>	<b>v</b>
<b>List of Figures.....</b>	<b>ix</b>
<b>List of Tables .....</b>	<b>xiii</b>
<b>List of Abbreviations .....</b>	<b>xv</b>
<b>List of Appendices.....</b>	<b>xviii</b>
<b>Chapter 1. Introduction .....</b>	<b>1</b>
1.1. Context of the Remotely Piloted Aircraft Systems Problem.....	2
1.2. The Relevance of Simulated Frameworks for Testing SAA Methods.....	5
1.3. Objectives of this Research .....	6
1.3.1. Application Scope .....	8
1.3.2. Out of Scope .....	9
1.4. Significance.....	10
1.5. Thesis Outline .....	11
<b>Chapter 2. Background .....</b>	<b>13</b>
2.1. Remotely Piloted Aircraft: Definition, Context, and Requirements .....	14
2.2. Simulation Platforms: JSBSim and FlightGear.....	19
2.2.1. Existing Open-source FDM.....	20
2.2.2. JSBSim Open-source FDM .....	23
2.2.3. FlightGear Flight Simulator .....	26
2.3. Theoretical Background: JSBSim FDM .....	26
2.3.1. Frames of Reference .....	27

2.3.2.	Forces and Moments .....	28
2.3.3.	Equations of Motion .....	31
2.3.4.	Propulsion .....	33
2.3.5.	Flight Control.....	34
2.4.	Computer Model Development.....	35
2.4.1.	Validation methods for RPAs.....	40
2.5.	Sense and Avoid Basis .....	41
2.5.1.	Scope of the SAA application .....	42
2.6.	Summary .....	43
 <b>Chapter 3. Open-Source Software in Aerospace with RPA Applications: Challenges and Solutions.....</b>		<b>44</b>
3.1.	Basic Flight Simulator Framework .....	45
3.2.	Technical challenges of working on/with OSS in Academia.....	48
3.2.1.	Software in the Scientific World.....	49
3.2.2.	Benefits of Using OSS in Graduate School .....	50
3.2.3.	Challenges of Developing OSS in Graduate School.....	52
3.2.4.	Challenges of Working with OSS in Graduate School.....	54
3.2.5.	Overall Perspective on Best Practices for a Positive OSS Experience in Graduate School	55
3.2.6.	The Future of OSS in Academia.....	59
3.3.	Technical challenges of JSBSim .....	60
3.4.	Summary .....	61
 <b>Chapter 4. High-level Validation Approach for OSS FDMs: JSBSim Application .....</b>		<b>63</b>
4.1.	High-level validation approach for OSS FDMs.....	64

4.1.1.	Validation of Aircraft Flight Simulators: Piloted vs. Remotely piloted Aircraft ...	65
4.1.2.	Initial Procedure for a Correct Validation .....	69
4.1.3.	Validation Methodology for the FDM Development .....	72
4.2.	Simulation Assumptions for the Case Studies .....	75
4.3.	EPP FPV Case Study: JSBSim Tutorial.....	76
4.3.1.	EPP FPV Simulation Context .....	76
4.3.2.	EPP FPV Computer Modelling and Development .....	77
4.3.3.	EPP FPV Computer Model Validation .....	80
4.1.3.	Simulation of SAA Manoeuvres.....	96
4.3.	Summary .....	97
 <b>Chapter 5. Giant Big Stik Computer Model Development with Sense and Avoid Applications.....</b>		<b>99</b>
5.1.	Giant Big Stik Computer Model Development.....	100
5.1.1.	Giant Big Stik Simulation Context (Phase 1) .....	101
5.1.2.	Giant Big Stik Computer Modelling and Development (Phase 1) .....	102
5.1.3.	Giant Big Stik Computer Model Validation (Phases 2 and 3).....	104
5.2.	Giant Big Stik Application: Vertical Collision Avoidance.....	113
5.2.1.	Encounter Geometry and Intruder's Trajectory.....	114
5.2.2.	Collision Avoidance Conditions and Methodology for Solving the SAA problem for RPAs .....	116
5.2.3.	Implications.....	120
5.3.	Summary .....	124
 <b>Chapter 6. Discussion, Recommendations, and Conclusion .....</b>		<b>126</b>
6.1.	Impact and Contributions .....	126

6.1.1.	Open-Source Software Contribution.....	128
6.1.2.	JSBSim Contribution .....	129
6.1.3.	Development and Validation of RPA Computer Models .....	130
6.1.4.	Collision Avoidance Application.....	132
6.2.	Future Research.....	133
6.2.1.	Proposed Improvements to Open-Source Software in Academia.....	133
6.2.2.	Proposed RPAS Flight Simulator Requirements Improvements .....	134
6.2.3.	Proposed Computer Model Validation Improvements .....	134
6.2.4.	Proposed Improvements to CA in RPAs .....	137
6.3.	Conclusion.....	138
6.4.	List of publications.....	139
<b>References .....</b>		<b>142</b>
<b>Appendix A. A Simplified Manual of the JSBSim Open-source FDM for Fixed-wing RPA Applications.....</b>		<b>162</b>
<b>Appendix B. Coefficients calculation for the EPP FPV R/C.....</b>		<b>184</b>
<b>Appendix C. Extended Sense and Avoid Basis and Existing Work .....</b>		<b>192</b>
<b>Appendix D. Encounter Geometries .....</b>		<b>207</b>
<b>Appendix E. Code and Configuration Files.....</b>		<b>210</b>

# List of Figures

Figure 1.1. Thesis significance .....	11
Figure 2.1. Reported RPA incidents in Canada in the last 5 years .....	16
Figure 2.2. JSBSim block structure .....	24
Figure 2.3. Aircraft frames.....	28
Figure 2.4. Giant Big Stik. Forces and moments .....	34
Figure 2.5. Model and simulation within the system development cycle (Diston, 2010) .....	37
Figure 2.6. Modelling of a dynamic system (Buchholz et al., 1996).....	37
Figure 2.7. The correlated inspection approach (Law, 2014) .....	40
Figure 3.1. Basic aircraft simulation framework .....	46
Figure 3.2. FDM blocks .....	46
Figure 4.1. Levels of an aircraft flight simulation model validation .....	65
Figure 4.2. Stages of a correct validation .....	70
Figure 4.3. Validation methodology for RPA FDM.....	72
Figure 4.4. Mini SGS-126 Glider (SimplePlanes, 2019) .....	78
Figure 4.5. Open-loop AeroSim layout in MATLAB/Simulink.....	81
Figure 4.6. EPP FPV static test. Airspeed ( $V_a$ ) .....	82
Figure 4.7. Airspeed error .....	82
Figure 4.8. EPP FPV elevator test. Pitch angle.....	83
Figure 4.9. EPP FPV elevator test. Airspeed .....	83
Figure 4.10. EPP FPV aileron test. Roll angle.....	84
Figure 4.11. EPP FPV aileron test. Airspeed.....	84
Figure 4.12. 2-Sample standard deviation test for roll. Aileron test in the interval [5, 7).....	86
Figure 4.13. EPP FPV aileron test. Sideslip .....	87
Figure 4.14. 2-Sample Standard Deviation Test for $\beta$ . Aileron test in the interval [5, 7).....	88
Figure 4.15. EPP FPV rudder test. Roll angle .....	89

Figure 4.16. EPP FPV rudder test. Sideslip .....	90
Figure 4.17. EPP FPV rudder test. Yaw rate .....	90
Figure 4.18. 2-Sample standard deviation test for $\beta$ . Rudder test in the interval [5, 8).....	90
Figure 4.19. 2-Sample standard deviation test for yaw rate. Rudder test in the interval [5, 8) ....	90
Figure 4.20. Aileron and elevator deflection for small and medium signals in Phase 3B.....	93
Figure 4.21. Computer model vs. real model. Roll output .....	93
Figure 4.22. Computer model vs. real model. Pitch output .....	93
Figure 4.23. Rudder horn and carbon strip in the EPP FPV (HobbyKing.com, n.d.).....	95
Figure 4.24. EPP FPV avoidance manoeuvre. Pitch angle .....	97
Figure 4.25. EPP FPV avoidance manoeuvre. Trajectory .....	97
Figure 5.1. The Giant Big Stik aircraft in FlightGear.....	103
Figure 5.2. Giant Big Stik static test. Steady-state airspeed .....	105
Figure 5.3. Giant Big Stik elevator test. Pitch angle.....	106
Figure 5.4. Giant Big Stik elevator test. Airspeed .....	106
Figure 5.5. Giant Big Stik elevator test for small signal (-0.015rad, -1°). Pitch angle.....	107
Figure 5.6. Giant Big Stik elevator test for a medium signal (-0.10rad, -6°). Pitch angle .....	107
Figure 5.7. Giant Big Stik aileron test. Roll angle.....	108
Figure 5.8. Giant Big Stik aileron test. Airspeed.....	108
Figure 5.9. Giant Big Stik rudder test. Yaw rate .....	109
Figure 5.10. Giant Big Stik rudder test. Sideslip angle ( $\beta$ ).....	109
Figure 5.11. Aileron and elevator deflections in Section 1.....	110
Figure 5.12. Aileron and elevator deflections in Section 2.....	110
Figure 5.13. Section 1. Computer model vs. real model. Roll angle .....	111
Figure 5.14. Section 1. Computer model vs. real model. Pitch angle.....	111
Figure 5.15. Section 2. Computer model vs. real model. Pitch angle.....	111
Figure 5.16. $\Phi$ encounter geometry and its waypoints .....	115
Figure 5.17. $\Phi$ manoeuvre: Giant Big Stik and Cessna's trajectories with conflict points .....	115
Figure 5.18. CA flowchart procedure .....	118
Figure 5.19. CA Scenarios and solving procedure based on Figure 5.20.....	118
Figure 5.20. Elevator deflection vs. $\tau$ .....	122

Figure 6.1. Thesis contributions (updated from Figure 1.1) .....	127
Figure A.1. JSBSim standalone mode structure .....	164
Figure A.2. JSBSim program command line and options .....	166
Figure A.3. JSBSim program command line in batch mode .....	166
Figure A.4. FlightGear interface. Advanced options. Input/output properties .....	169
Figure A.5. FlightGear block structure .....	170
Figure A.6. FlightGear interface. Advanced options. Flight Model .....	170
Figure A.7. FGFDMExec and JSBSim Initialization process (Berndt & JSBSim Development Team, 2011) .....	175
Figure A.8. Multiplayer mode in FlightGear with a Cessna 172 as seen from the cockpit of another aircraft .....	178
Figure A.9. Giant Big Stik on the field before tests.....	179
Figure A.10. RPA roll and pitch angles .....	181
Figure A.11. Giant Big Stik in FlightGear v2.0.0 performing an aerobatic manoeuvre in JSBSim standalone mode.....	182
Figure A.12. Giant Big Stik manual flight in FlightGear v2.0.0 .....	183
Figure B.1. Top view of the RPA .....	185
Figure B.2. Side view of the RPA.....	185
Figure B.3. Mini SGS-126 Glider (SimplePlanes, 2019) .....	190
Figure C.1. How TCAS Works (1/2) (Kochenderfer, Holland, & Chryssanthacopoulos, 2012)	194
Figure C.2. How TCAS Works (2/2) (Kochenderfer et al., 2012).....	194
Figure C.3. Taxonomy of SAA systems (Fasano et al., 2016) .....	196
Figure C.4. Thresholds (Federal Aviation Administration, 2013b). SST: Self-separation threshold. WCV: Well-clear violation. CAT: Collision avoidance threshold. NMAC: Near mid-air collision .....	197
Figure C.5. Aircraft recognition and reaction time (Unmanned Systems Canada, 2017) .....	198

Figure C.6. Air occurrences in 2017. Accidents involving Canadian-registered aircraft, by operation type, 2017 (Transportation Safety Board of Canada, 2017) .....	201
Figure C.7. SAA encounter timeline (Federal Aviation Administration, 2013b).....	204
Figure D.1. 4D Opposing circuits. Ideal case (Cereceda & Stevenson, 2014) .....	208
Figure D.2. Waypoint adjustments to synchronize the time of arrival (Fang, 2014) .....	208
Figure D.3. The Phi ( $\Phi$ ) manoeuvre (Cereceda & Stevenson, 2014) .....	209

# List of Tables

Table 2.1. RPAS categories and requirements. Regulatory project.....	18
Table 2.2. Overview of available software .....	22
Table 4.1. EPP FPV Parameters .....	79
Table 4.2. EPP FPV aerodynamic coefficients.....	79
Table 4.3. Control surface deflections range .....	80
Table 4.4. Pearson correlation coefficients on results in Figure 4.21 and Figure 4.22.....	95
Table 4.5. Collision avoidance events for testing the EPP FPV in SAA (3D) manoeuvres.....	96
Table 5.1. Giant Big Stik Parameters.....	103
Table 5.2. Giant Big Stik aerodynamic coefficients .....	103
Table 5.3. Control surface deflections range .....	104
Table 5.4. Pearson correlation coefficients on results in Figure 5.13,Figure 5.14 and Figure 5.15 .....	112
Table 5.5. Simulation runs for estimating $\tau$ .....	121
Table 5.6. CA stages for the avoidance study.....	123
Table B.1. List of simple geometries in the EPP FPV .....	185
Table B.2. Moment of inertia around X axis .....	186
Table B.3. Moment of inertia around Y axis .....	187
Table B.4. Moment of inertia around Z axis .....	187
Table B.5. Product moment of inertia in XZ.....	189
Table B.6. Mini SGS-126 drag force aerodynamic coefficients.....	190
Table B.7. Mini SGS-126 side force aerodynamic coefficients .....	190
Table B.8. Mini SGS-126 lift force aerodynamic coefficients .....	190
Table B.9. Mini SGS-126 roll moment aerodynamic coefficients .....	191
Table B.10. Mini SGS-126 pitch moment aerodynamic coefficients.....	191
Table B.11. Mini SGS-126 yaw moment aerodynamic coefficients .....	191

Table C.1. Most significant sensing technology for SAA ..... 199

# List of Abbreviations

ABSAA	Airborne-Based Sense and Avoid
ADS-B	Automatic Dependent Surveillance-Broadcast
ATC	Air Traffic Control
ATM	Air Traffic Management
BVLOS	Beyond Visual Line-of-Sight
CA	Collision Avoidance
CAA	Civil Aviation Authority
CAT	Collision Avoidance Threshold
DAA	Detect and Avoid
DoF	Degrees of Freedom
EPP	Expanded Polypropylene
FAA	Federal Aviation Administration
FDM	Flight Dynamics Model
FPV	First Person View
GA	General Aviation
GBSAA	Ground-Based Sense and Avoid
GCS	Ground Control Station
GNC	Guidance, Navigation and Control
GS	Ground Station

GUI	Graphical User Interface
ICAO	International Civil Aviation Organization
JOSS	Journal of Open Source Software
M&S	Modelling and Simulation
MAC	Mid-Air Collision
MDP	Markov Decision Process
NMAC	Near Mid-Air Collision
NSP	National Simulator Program
OSS	Open-source Software
POH	Pilot's Operating Handbook
R/C	Remote Control
RAs	Resolution Advisories
RPA	Remotely Piloted Aircraft
RPAS	Remotely Piloted Aircraft System
RPV	Remotely Piloted Vehicle
SAA	Sense and Avoid
SDA	Sense, Detect and Avoid
SFOC	Special Flight Operations Certificate
SS	Self-Separation
SST	Self-Separation Threshold
SWaP	Size, Weight and Power

TAs	Traffic Advisories
TAS	True airspeed
TCAS	Traffic Collision Avoidance System
UAV	Unmanned Aerial Vehicle
V&V	Verification and Validation
VRAC	Vertical Resolution Advisory Complement

# List of Appendices

Appendix A. A Simplified Manual of the JSBSim Open-source Software for Fixed-wing RPA Applications

Appendix B. Coefficients Calculation for the EPP FPV R/C

Appendix C. Sense and Avoid Basis and Existing Work

Appendix D. Encounter Geometries

Appendix E. Code and Configuration Files

# Chapter 1

## *Introduction*

Along with the development of aviation, the remotely piloted aircraft system industry has rapidly advanced over the last 20 years. The current congested airspace is now shared by a wide range of aircraft classes and sizes, creating a challenge for administrations globally. The main consequence is that encounters between piloted and remotely piloted aviation are happening more often. With more encounters, there are more chances of airborne collisions, and thus, there is a need to study and minimize the negative effect (e.g. cost and injuries) of those encounters. International administrations are currently addressing this problem with more restrictive regulations that limit the flight tests for recreational and work/research purposes (Transport Canada, 2018a, 2018b). As a result, flight simulators have become the main frameworks for testing close encounters between aircraft.

For this research and application, flight simulators must be able to integrate piloted and remotely piloted aircraft into the same context when testing encounters. On one hand, proprietary software provides a specific library, which is limited by the available aircraft models when the license was purchased (e.g. X-plane - (“X-Plane 11 Flight Simulator,” 2018)). This requires a constant

upgrade of their scenarios and capabilities to stay up-to-date. On the other hand, open-source packages allow for the design and integration of new computer models, being the flexible setting required for testing encounters between piloted and remotely piloted aircraft (e.g. FlightGear – (“FlightGear Flight Simulator,” 2019)).

However, Open-Source Software (OSS) platforms have particular technical difficulties (e.g., limited user’s programming skills, lack of support and resources, and incomplete documentation), which are singularly wide-spread in graduate school.

The research documented in this thesis addresses these issues and provides an OSS framework for testing encounters between piloted and remotely piloted aircraft.

This chapter describes the motivation behind this study, explaining its objectives and specific goals. The interest of the topic and its relevance are briefly introduced as well but are further explained in Chapter 6.

## **1.1. Context of the Remotely Piloted Aircraft Systems**

### **Problem**

In aviation, simulating aircraft computer models requires computer models as the result of lengthy studies. The recent integration of Remotely Piloted Aircraft Systems (RPAS) into the airspace represents a challenge that current international administrations are addressing by evaluating the effect and risk of close encounters between RPAS and piloted aircraft. The uncertainty associated with their integration is especially critical in urban areas and near airports, where their flights are limited (in case of flying with recreational purposes in urban areas) or

restricted (near airports) (Transport Canada, 2018a). The objective is for the current airspace to incorporate a wide range of sizes and types of vehicles while ensuring a safe environment.

When the allowed flying areas are limited, the chances of encountering other aircraft increase, and therefore, measures must be taken in order to avoid possible collisions. Large piloted aircraft (e.g. passenger jets) carry onboard a Traffic Collision Avoidance System (TCAS) (Federal Aviation Administration. U.S. Department of Transportation, 2011) (see also Appendix C – Figure C.1 and Figure C.2), which is able to identify other aircraft with a transponder and issues advisories to the pilot. However, General Aviation (GA) aircraft are limited to the capability of the pilot to visually detect hazards and perform an avoidance manoeuvre on time. This setting introduces two issues: (1) the inability of the TCAS to detect RPAS and (2) the human pilot's identification capabilities depend on the size and the airspeed of the intruder's aircraft. Sense and Avoid (SAA) systems provide the aircraft with the capability of detecting and avoiding other aircraft in the vicinity and represent one of the main elements for the integration of RPAS into the airspace.

The current regulations limit flight tests for testing close encounters, in real environments due to their hazardous nature and, as a consequence, the simulated environment must provide a high level of certainty in the absence of flight tests.

Flight simulators usually include the aircraft system and the models that interact with it: propulsion system, weather and atmosphere, scenery, etc. The aircraft is represented by the Flight Dynamics Model (FDM), which is a mathematical representation of the aircraft. During the development of these systems and the FDM in particular, the designer must have a set of reports on the aircraft performance and behaviour that can use as a reference. However, close

encounter scenarios require extreme manoeuvres, and therefore, the modelling signal range is reduced to large signals. In general terms, the model does not need to provide a comparative performance over all ranges, just those where the requirements of the application are defined.

### ***1.1.1. Technical Difficulties***

The computer model development approach presented will address the issue of relying on computer software to test SAA strategies (application). Even though flight tests are a determining element in the aircraft validation process, close encounters introduce undesirable risks in the airspace. From the flight tests perspective, current regulations from Transport Canada prevent the flight of RPAS in scenarios with piloted aircraft (for example, near airports) (Transport Canada, 2018a). This means that during the computer validation process (further explained in Chapter 4), the flight tests that serve as a reference are limited to regular missions.

In order for the model to be competitive and used by a wide group of developers in future applications, the software must be open-source and easy to use. Proprietary software is eliminated for this research since it does not allow for the integration of other computer models not included in their corresponding libraries.

However, challenges were faced during the development of this thesis when working with OSS. General challenges associated with the lack of programming skills, resources or academic support are wide-spread in academia. Related to the aerospace application of this work, particular challenges also arose with the OSS FDM: JSBSim. For example, their documentation lacked guidelines on how to create and validate new models. As a result of this and other technical difficulties encountered with the software (Chapter 3 – Section 3.3), a JSBSim guide

for the development of Remotely Piloted Aircraft (RPA) computer models was also created and uploaded to the online community (Appendix A) (Cereceda, 2019).

## **1.2. The Relevance of Simulated Frameworks for Testing**

### **SAA Methods**

Modelling and Simulation (M&S) are important fields of study for the assessment of real models in critical situations. Flight simulators and OSS packages in academia are commonly used as a platform for simulated environments with piloted and RPA. First, flight simulators allow for testing possible scenarios prior to take them to the field and second, OSS packages allow for the integration of new models into existing frameworks.

JSBSim, the OSS used in this thesis, is a valuable tool for the integration of aircraft computer models into versatile environments. However, the literature on JSBSim lacks information and reports on RPA computer models. This thesis aims to fill the aforementioned gap with the development of two RPA computer models for assessing close encounters with piloted aircraft. Both models will be added to the JSBSim online platform for any designer to download and use.

The computer development process faced the issue of a lack of validation standards for RPA computer models in the current literature. Therefore, the proposed validation methodology should initialize discussion on this topic, encouraging future RPA model designs. The aim is to establish standardized practices similar to the criteria used for piloted aircraft by the administrations.

SAA methods are necessary for the correct functioning of the airspace. In piloted aviation, the human has the capacity to identify and avoid any element surrounding the aircraft. However, an

additional challenge is faced when the pilot is not able to identify an RPAS due to its size or travelling airspeed. For that reason, SAA methods are not only relevant for the RPAS but also for pilots who lack the capability of detection.

Considering SAA as the application, it is important that the simulated environment is a true representation of the main elements that interact in the airspace: piloted aircraft and RPAS. Stable and validated models are required for establishing a framework able to test and evaluate close encounters. Then, validation procedures are fundamental for developing computer models with SAA applications.

### **1.3. Objectives of this Research**

The purpose of this research is to provide a framework for the assessment of SAA methods based on the RPA computer model design. This thesis seeks to address the following questions:

- What are the challenges that graduate students face while interacting with OSS? How do graduate students fit in the OSS loop?
- Are RPA and piloted flight simulators equivalent? How reliable are RPA computer models?
- Is JSBSim reliable for testing RPAs and encounters with piloted aircraft?

The first objective is to evaluate the educational and technical barriers of interacting with OSS and the means to improve the graduate student experience. The second objective is to define the particular challenges associated with the development of FDMs in open-source simulators. The most relevant element of this objective is to define a methodology for the validation of FDMs particular to SAA approaches that could be extendable to other RPAS and applications. Finally,

the third objective is to develop a simulated environment including a fit-for-purpose FDM whose performance is proved to be equivalent to the real RPAS under certain conditions. This includes contributing to the JSBSim sustainability by improving the documentation and adding UAV aircraft models to their library.

Drawing from these primary objectives, the specific goals of this study are:

- 1) Describe the challenges of working with open-source platforms in academia and provide an overview of the means to overcome these challenges.
- 2) Specify a validation methodology based on existing Verification and Validation (V&V) techniques, proving that once a designed RPA FDM is validated, it is appropriate to run the aircraft in simulations and obtain results without flight tests.
- 3) Develop a 6-DoF RPA FDM in JSBSim that expresses the actual performance of the aircraft and serves for testing SAA strategies in simulations.
- 4) Validate the FDM designed according to the method described in (2).
- 5) Address the issues encountered when working with OSS packages such as JSBSim and provide an alternative document to serve as a user's guide for the development of RPA computer models.
- 6) Assess the implications of the integration of the designed FDM in (3) into a computer simulation in JSBSim for testing SAA applications.

While the first listed goal has a wide application to any scientific discipline, the remaining goals have aerospace applications. In particular, goals 2, 4, and 6 focus on RPAS whereas goals in 3 and 6 directly affect the JSBSim package.

Although testing SAA methods is one of the motivations for this research, it is not the focus of this thesis. Stable and reliable simulated frameworks are needed for assessing encounters and SAA applications. Then, the issue addressed in this thesis is the technical difficulties associated with the development and study of RPAs in simulated environments.

### ***1.3.1. Application Scope***

RPAS are under strict control by administrations that limit their flights depending on their size and application. Additionally, close encounters are not permitted due to the hazard they present to the airspace. As a consequence, the framework for the study of avoidance tasks is reduced to simulated structures. The core of this study is related to the design and definition of the FDM in a computer environment that is precise and well defined. Since the real flight tests are limited to regular flight and usual manoeuvres, this work only presents the results of real tests when adjusting and designing the FDM. The model is precise enough to be tested in a computer environment but additional flight tests with extreme manoeuvres must be conducted in order to increase the model reliability (Chapter 6).

The simulated environment represents a Class G airspace in the surrounding areas of Fogo Island in Newfoundland and Labrador (NL), Canada. The representative RPAs for this research are the EPP FPV and the Giant Big Stik, whereas the Cessna 172 is the representative general aircraft. Likewise, the real flight tests for validation took place in the surrounding areas of St. John's, NL, Canada following the Transport Canada regulations in the allowed areas in Class G airspace.

### ***1.3.2. Out of Scope***

Some factors have been neglected or not included in the following research either because they have been addressed before by other researchers or have been limited for the sake of simplicity.

An example of a factor not being taken into account is the wind. Its effect on the aircraft performance and control presents a particular challenge since the wind is unpredictable. Simulated environments allow for its modelling based on previously collected data of the area or using constant winds with a pre-defined angle of incidence. Extensive work on this topic has been carried out by other researchers belonging to the same team (Artacho, 2018; Fang, 2018).

By disregarding the wind component on an aircraft, its performance depends on three other components: trajectory geometry, control, and airspeed. The control surfaces have the greatest impact on the performance and, therefore, the other two components (trajectory and airspeed) have been eliminated from the study. The geometry has been selected based on how often an encounter takes place, whereas the airspeed is kept at optimum performance. Excluding factors with particular behaviours not only simplifies the modelling problem, but it also helps investigate how the control component affects the aircraft performance.

Remote SAA systems have also been eliminated from this study since it requires a communication link between the aircraft and the SAA, adding extra challenges. For example, a delay in communication could mean an inability of the aircraft to conduct an avoidance manoeuvre on time, leading to a collision.

In the scenario described in Chapter 5, the SAA problem is not addressed as a whole; more information about SAA and the rationale behind focusing on a particular component is further explained in Appendix C.

Any simulation configurations other than JSBSim that are used as a reference (e.g. AeroSim in MATLAB/Simulink) are not discussed in detail; they have been used and described in previous work (Stevenson, 2015), and are solely used as reference systems.

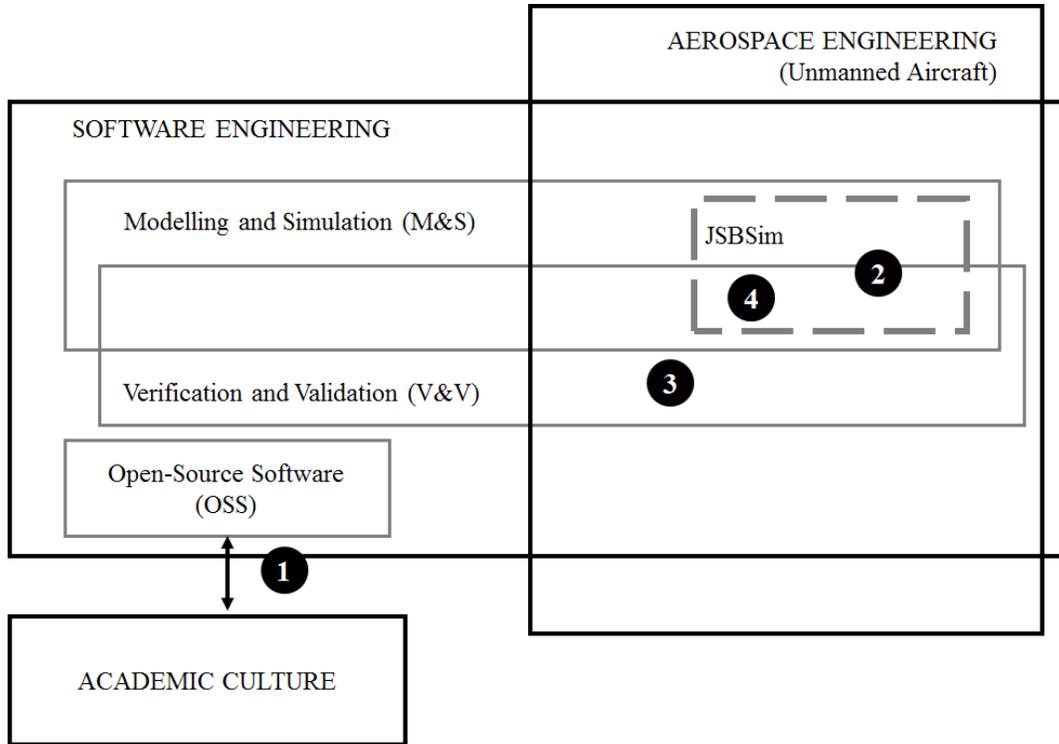
Complementary recommendations on the conducted research, and its limitations and highlights are further explained in Chapter 6.

## **1.4. Significance**

The significance of this thesis is expressed through its contributions to the fields of academic culture, aviation, and software engineering (belonging to M&S, V&V, and OSS). In the following chapters, the research conducted is explained and documented, with the contributions further discussed in Chapter 6.

Overall, the significance of this research is observed from four perspectives:

- A discussion on the process of working with OSS and overcoming difficulties from a graduate student perspective, and how this affects the academic culture and the software engineering field (#1 – Figure 1.1).
- The RPA model development and a set of guidelines for designing RPA models in JSBSim (#2 – Figure 1.1).
- Introduction of a flexible methodology that uses straightforward approaches to validate RPA computer models for particular purposes (#3 – Figure 1.1).
- Development of RPA FDMs for testing SAA methods and evaluating close encounters between a piloted and RPA. RPAS applications (#4 – Figure 1.1).



**Figure 1.1.** Thesis significance

## 1.5. Thesis Outline

This first chapter has provided an introduction to the purpose and motivation of this thesis along with the challenges and significance to the scientific community.

Chapter 2 gives a general overview of all the fields of engineering this thesis is based on: RPA and RPAS concepts and definitions, the simulation platform (JSBSim) along with its theoretical background, classic V&V techniques and model development approaches, and encounter geometries.

Chapter 3 discusses and addresses, in detail, the technical challenges of modelling RPAs in open-source platforms. The chapter starts with an introduction of the minimal elements for designing RPA computer models. It continues with an overview of the current situation of OSS in

academia from a graduate student perspective, and justifies the use of these platforms in the aerospace field. Then, the chapter continues with the particular technical challenges presented by JSBSim.

Chapter 4 starts with the evaluation of the difference between piloted and remotely piloted flight simulators. As a solution to the challenges in Chapter 3 and the lack of defined validation methods for RPA computer models in flight simulators, a high-level validation procedure for designing RPA FDMs is presented. An example of how to implement the validation procedure is shown with the EPP FPV validation in a tutorial format.

Chapter 5 introduces another RPA computer model: the Giant Big Stik aircraft. The procedure for their development is based on existing V&V techniques, established by the methodology described in Chapter 4 and is fit for the purpose of testing SAA strategies. The simulation environment for a close encounter is built where a diving manoeuvre is the consequence of an encounter between two representative aircraft (the Cessna 172 for the piloted aircraft and the Giant Big Stik as the RPA). The goal of this chapter is to study the performance of RPAS with SAA applications and evaluate their implications.

Finally, Chapter 6 summarizes this thesis, defines its limitations, highlights its contributions and adds recommendations for possible future work.

This thesis also contains three appendices that add extra information: (A) a simplified guide for the development of small fixed-wing aircraft in JSBSim; (B) the inertia and aerodynamic coefficients calculation for the representative RPA modelled in Chapter 4 (the EPP FPV); (C) the SAA basis, regulations, and recommendations; and (D) the aircraft configuration files created for this research.

# Chapter 2

## *Background*

The Sense and Avoid (SAA) issue has become a very popular research topic in the field of Remotely Piloted Aircraft (RPAs) since it allows for evaluating their integration into airspace. A direct consequence is that RPAs must have an SAA capability implemented to reduce the risk of Mid-Air Collisions (MAC). Simulators are now essential for the development of SAA strategies. In particular, the Flight Dynamic Model (FDM) expresses the dynamics of the aircraft in a simulator, including all forces and moments involved in the performance. In other words, it is the computer-generated expression of the aircraft's performance in a simulated context.

Considering the importance of flight simulators for RPA applications, the goal of this chapter is to provide an overview of the existing work and background related to the different fields mentioned in this thesis.

This chapter begins by discussing the concepts related to RPAs and the current Transport Canada regulations. An overview of existing open-source platforms is presented next with a particular analysis of the software tools used in this thesis. This frames the FDM, which is explained from

a physical perspective. This chapter concludes with a summary of general computer model development techniques and a description of the encounter geometries for the RPA application.

## **2.1. Remotely Piloted Aircraft: Definition, Context, and Requirements**

In recent years, the use of RPAs, commonly referred to as “drones”, for recreational purposes has increased exponentially. However, these devices should not be considered harmless; records and studies have proved that their incorrect use and underestimation has led to an increase in risk situations (Boivin, 2017; Dunn, 2018; Kesteloo, 2018; Starmetro Staff, 2018).

These uncertain situations can turn into a threat, and this has led to new regulations from various international administrations, such as Federal Aviation Administration (FAA) and Transport Canada, to include RPAs in the current airspace. The purpose of this section is to define RPAs and summarize their current status in Canada along with the regulations in effect.

### ***2.1.1. Remotely Piloted Aircraft vs. Remotely Piloted Aircraft System***

In the literature, RPAs are known by different names, such as Unmanned Aerial Vehicles (UAV) and Remotely Piloted Vehicle (RPV). RPA and RPAS are the term used and defined within the Canadian Aviation Regulations SOR/96-433 (“Canadian Aviation Regulations,” 2018). According to those, an RPA is “a navigable aircraft, other than a balloon, rocket or kite, that is operated by a pilot who is not on board” and an RPAS is “a set of configurable elements consisting of a remotely piloted aircraft, its control station, the command and control links and any other system elements required during flight operation”.

Since the initial development of RPAs in World War I with the experimental Kettering Bug project (National Museum of the US Air Force<sup>TM</sup>, 2015), significant achievements have been accomplished in remotely piloted aviation (Newcome, 2005). At the beginning of the 20<sup>th</sup> century, technology was limited in terms of automatic stabilization, remote control, and autonomous navigation. Once those barriers were overcome, the development of RPAs accelerated, not yet reaching its peak. A promising future shows piloted flight complemented by remotely piloted flight.

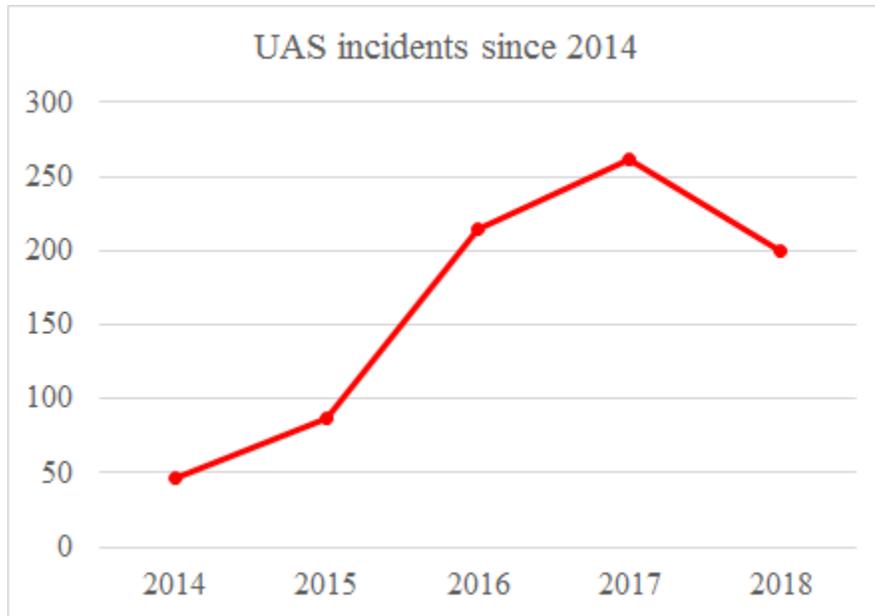
A Remotely Piloted Aircraft System (RPAS) is a system that includes an aircraft or vehicle, a Ground Station (GS) that the pilot uses to operate the aircraft, and a communication link between the two. It is important to distinguish between RPA and RPAS since the vehicle itself (RPA) is part of a larger system needed for its operation (RPAS).

### ***2.1.2. RPAS in Canada***

RPAS development in Canada presents a great opportunity for innovation and technology. However, international and national regulators, in particular Transport Canada, are facing demands from the aviation industry to adopt existing regulations to these newly developed technologies.

#### ***2.1.2.1. The Growth of RPAS in Canada***

Over the last few years, the number of reported incidents between piloted and RPA in Canada has grown (Transport Canada, 2019a). Those reported incidents include RPAS encountered near airports and by piloted aircraft. Since the reports started to be collected in 2014, the incidents where RPA were involved increased over 200% until 2017 (Figure 2.1).



**Figure 2.1.** Reported RPA incidents in Canada in the last 5 years

Over the last year (2018), the concerns associated with those incidents and the more restrictive measures taken by international administrations have resulted in a reduction of the number of incidents. However, a shared airspace with piloted aircraft has been questioned over the last few years. The concerns arising from these events have led to the establishment of regulations for the safe integration of RPAs into a shared airspace with piloted aircraft. Two main issues have been addressed in the latest regulatory projects. The first issue is directly related to the fact that there is no pilot on board, meaning the control of the vehicle is remote and dependent on the communication link between the pilot and vehicle. This discussion is out of the scope of this thesis but it has been addressed in the literature (Gupta, Jain, & Vaszkun, 2016; Heppe, 2015; Zeng, Zhang, & Lim, 2016).

The second issue is the fact that the RPA is not aware of its surroundings in the way that a pilot is when operating a piloted aircraft. In order to operate correctly and independently, the RPA

must carry a system onboard capable of identifying and avoiding all kinds of air-to-air surrounding threats. This system is called “Sense and Avoid” and it will be discussed in later sections in this chapter.

*2.1.2.2. Flying Safely: Canadian Aviation Regulations SOR/96-433 Part IX (Government of Canada, 2019)*

RPAs range in size and weight from small to large aircraft and therefore, the requirements and regulations vary depending on the type of vehicle, the application of its flight, and the environment it operates in. As of November 2019, the current Canadian Aviation regulations make a distinction depending on whether the drone is operated within the pilot’s visual line-of-sight. Whereas (1) beyond-line-of-sight operations and (2) drones over 25kg need a Special Flight Operations Certificate (SFOC), (3) drones under 25kg require a pilot certificate (Transport Canada, 2019c). For example, for the flight tests carried out in Chapter 4, two representative RPAs have been flown as part of a project for which an SFOC was issued.

Current regulations have recently been implemented on June 1, 2019. The main changes did not affect the need for an SFOC for flying RPA Beyond Visual-Line-of-Sight (BVLOS), whereas work/research flights with less than 25kg operated within Visual-Line-of-Sight (VLOS) are now exempt from the permission certificate. These updated regulations, which are summarized in Table 2.1, introduce three categories depending on the size, pilot, and environment (rural – small limited and urban – small complex). For more information about the requirements for flying RPAs in Canada, visit (Transport Canada, 2019b).

**Table 2.1.** RPAS categories and requirements. Regulatory project

RPAS category	General operating and flight rules		
<p><i>Very Small (250 g – 1 kg)</i></p> 	<p><b>Pilot</b> </p> <ul style="list-style-type: none"> <li>• Minimum Age of 14</li> <li>• Successfully complete written knowledge exam (5 year renewal)</li> <li>• Have liability insurance of at least \$100,000</li> </ul>	<p><b>Product</b> </p> <ul style="list-style-type: none"> <li>• Identification: Name, address and telephone number clearly visible on the UAS</li> <li>• UAS is properly maintained to manufacturer's instructions</li> </ul>	<p><b>Operating Rules</b> </p> <ul style="list-style-type: none"> <li>• Must be fit to fly – not suffering from fatigue, under the influence of alcohol or drugs</li> <li>• Stop operating if there is an accident, damage, injury, collision</li> <li>• Do not operate in a reckless or negligent manner</li> </ul>
<p><b>Safe Distances</b></p> <ul style="list-style-type: none"> <li>• Maximum Altitude of 300ft</li> <li>• Max visual distance of 0.25nm</li> <li>• No flights within 3nm of an aerodrome / 1nm of a heliport</li> <li>• 100ft minimum (lateral/height) from people, vehicles and vessels</li> <li>• No night operations</li> </ul>			
<p><i>Small Limited (1 kg – 25 kg)</i></p> 	<p><b>Pilot</b> </p> <ul style="list-style-type: none"> <li>• Minimum Age of 16</li> <li>• Successfully complete written knowledge exam (5 year renewal)</li> <li>• Have liability insurance of at least \$100,000</li> </ul>	<p><b>Product</b> </p> <ul style="list-style-type: none"> <li>• Identification: Name, address and telephone number clearly visible on the UAS</li> <li>• UAS is properly maintained to manufacturer's instructions</li> <li>• Maintain technical records: air time of flight, maintenance actions, modifications, repairs, etc.</li> </ul>	<p><b>Operating Rules</b> </p> <p><i>(In addition to Very Small rules)</i></p> <ul style="list-style-type: none"> <li>• Conduct a physical site survey prior to operations</li> <li>• Procedures for system assembly, pre-flight checks and tests, take-off and landing</li> <li>• Emergency procedures are in place: pilot, engine, structural, equipment, controls failures, fly-aways</li> </ul>
<p><b>Safe Distances</b></p> <ul style="list-style-type: none"> <li>• Maximum Altitude of 300ft</li> <li>• Visual distance of 0.50 NM</li> <li>• No flights within 3nm of an aerodrome / 1nm of a heliport</li> <li>• 250 ft. minimum (lateral/height) from people, vehicles and vessels</li> <li>• 500 ft. from assemblies of people</li> <li>• No night operations</li> </ul>			
<p><i>Small Complex (1 kg – 25 kg)</i></p> 	<p><b>Pilot</b> </p> <ul style="list-style-type: none"> <li>• Minimum Age of 16</li> <li>• Complete written knowledge exam</li> <li>• Flight review by UAS pilot permit holder</li> <li>• Obtain TC Pilot Permit</li> <li>• Complete a recurrent training program in last 2 years</li> <li>• Liability insurance of at least \$100,000</li> </ul>	<p><b>Product</b> </p> <ul style="list-style-type: none"> <li>• UAS registered with TC and marked with TC marks</li> <li>• Complies with design standard w/ declaration of compliance and a statement of conformity</li> <li>• UAS maintained to manufacturer instructions</li> <li>• Position and anti-collision lights for night operations</li> </ul>	<p><b>Operating Rules</b> </p> <p><i>(In addition to General and Limited rules)</i></p> <ul style="list-style-type: none"> <li>• Prior coordination with Air Traffic Control (ATC) for operations in controlled airspace.</li> <li>• Maintain two-way communication with ATC.</li> <li>• Comply with ATC instructions and clearances</li> </ul>
<p><b>Safe Distances</b></p> <ul style="list-style-type: none"> <li>• Maximum Altitude of 400ft</li> <li>• Maintain visual distance of 1 NM</li> <li>• Flight permitted within: 3nm of an aerodrome/1nm of a heliport, subject to notifying ATC</li> <li>• 100 ft. minimum (lateral/height) from people, vehicles and vessels</li> <li>• 500 ft. from open-air assemblies of people (unless at a minimum height of 300ft)</li> <li>• Night operations permitted with lighting</li> </ul>			

## **2.2. Simulation Platforms: JSBSim and FlightGear**

Simulators are a crucial tool of the aviation industry. The importance of simulators is allowing pilots to train without increasing flight hours and allowing to minimize associated costs. This allows the pilot to lead the system into unsafe situations where its limits are tested. This fact also allows designers to test all possible circumstances that could lead to avoiding eventual accidents with significant damage.

In the application addressed in this thesis, a computer environment is crucial for the correct implementation of dynamic simulation models in aviation, whereby small and large RPAs pose a threat to other aircraft.

Computer models must provide sufficiently high levels of fidelity with a performance proven to be sufficiently precise to the real RPA under certain conditions. In particular, its dynamic performance is given by the FDM, which is the combination of physical models based on mathematical equations that express the aircraft dynamics in a simulator; including all forces and moments involved during the flight.

It is obvious to think that the first approach to a reliable model and simulator can be found in aircraft manufacturers; the aircraft is designed, improved and widely tested before going to market. This information and software are only used for internal purposes, making it proprietary. This makes it challenging for a researcher to find a good platform to test applied algorithms where the purpose is not to discuss or test the aircraft itself; it is a tool to develop, verify and test new methods and algorithms. Open-source packages provide the flexibility for designing and validating new computer models that can be later integrated into existing simulations. Therefore, in this section, open-source FDMs are presented and discussed.

### ***2.2.1. Existing Open-source FDM***

LaRCSim was an FDM developed by NASA in the early 90s (Jackson, 1995) and renamed UIUC after some modifications by the University of Illinois in the early 2000s (Selig, Deters, & Dimock, 2002). It was also implemented with FlightGear (Section 2.2.3) as its default FDM. LaRCSim was one of the first 6-DoF FDM to use subroutines for the description of aerodynamics, atmosphere and other elements involved in the flight. This practice created shorter processing times and enabled real-time simulation. LaRCSim and UIUC models are currently in disuse and both projects are inactive.

Another FDM, YaSim (“YASim - FlightGear,” 2018) is currently one of the most used open-source FDM. However, there is little documentation available, and the documentation that does exist has been made difficult to follow by developers. Its modelling approach is very simple: it uses the geometry of the aircraft to generate base flight characteristics such as aerodynamic coefficients. YaSim is the appropriate model to work with if the final application does not require an accurate solution, or when the aerodynamic coefficients of the aircraft are unknown, since it is an approximation to the real flight. However, if the model needs to match the real flight, either using another FDM or adjusting the YaSim model are better alternatives.

JSBSim is a 6-DoF FDM that has been used in aeronautics for over 20 years (Berndt & JSBSim Development Team, 2011). It is an open-source software with a large library that is in constant development, which makes the dynamic model versatile and easy to use for any designer. The programming source code is in C++, the configuration files are implemented in XML-format and it runs under most of the operating systems. Although it has been used in previous projects for RPA modelling (Vogeltanz & Jašek, 2015; Wong et al., 2008), JSBSim was initially developed

for general and commercial aircraft. As a result, there was a lack of reliability analysis for RPA models, raising concerns associated with its quality and limitations for small fixed-wing aircraft.

JSBSim requires a knowledge of aerodynamics and the fundamentals of flight that makes working with it an arduous task. However, the time invested in understanding the software is made worthwhile by the results provided by the model.

In order to avoid incompatibilities and future complications (e.g. unsupported system updates), only Open-Source Software (OSS) is going to be examined in this thesis. The tools and packages presented in this section have come as a result of limiting the search to software used in robotics, specifically in RPAs or aviation.

To start with, ODE (Smith, 2001) is a widely used physics simulator developed in 2001 and currently used for modelling any type of robotic system. It is used for simulating rigid bodies and is considered very useful for analyzing the collisions of dynamic bodies and their interactions in mobile robots. It has also been used in several applications and games. ODE is part of other robotics simulation software such as Gazebo and V-REP. Since the objective of this thesis is not the study of collisions, ODE is eliminated as a potential software.

Gazebo (“Gazebo,” 2014) is a visual software commonly used with ROS (“ROS,” 2008). The structure of the system is based on nodes that send messages and communicate with each other through a publish/subscribe system. During 2015, our research team, including this author, tried to implement JSBSim FDM into the ROS environment. The task was more arduous than expected due to the software’s framework and a large number of properties to be shared and the project was cancelled after a few months. The use of ROS would have been useful since it allows

the integration of different types of robotic models into the same simulation. The development of fixed-wing aircraft computer models in ROS remains as future work.

For the last few years, the research team has used the MATLAB/Simulink AeroSim toolkit (Unmanned Dynamics, 2006) to model RPAs and run simulations (Cereceda & Stevenson, 2014) combined with FlightGear. AeroSim was widely used in different projects (Yun, Li, & Zheng, 2013) and known as a reliable tool for creating RPA simulations. MATLAB is a licensed software largely used for academic purposes, but its need for frequent updates results in incompatibilities. Even though AeroSim is easy to use due to its graphic environment, it is currently out-of-date and the simulations are only limited to versions up to MATLAB R2010a. This led this author to seek out another FDM that would substitute the AeroSim FDM for RPA simulations, although this software will have a relevant role in the validation process and design of the optimal FDM in Chapter 3.

**Table 2.2.** Overview of available software

Software (FDM)	Visual Environment	Highlights	Status	Comments	
ODE + V-REP or ROS	Included in the software	<ul style="list-style-type: none"> <li>Models collisions between dynamic bodies</li> </ul>	Active	Rejected	
ROS + Gazebo	Gazebo	<ul style="list-style-type: none"> <li>Does not require significant computation</li> <li>Publish/subscribe system</li> </ul>	Active	Dismissed in 2015	
MATLAB Simulink	AeroSim JSBSim	Simulink or FlightGear	<ul style="list-style-type: none"> <li>Graphical environment and setup</li> <li>Well-known software</li> </ul>	Out-of-date In current development	Not viable Insufficient resources
JSBSim + scripts	FlightGear	<ul style="list-style-type: none"> <li>Open to design with an online library</li> <li>Flexible programming</li> <li>No incompatibilities with OS or older versions</li> </ul>	Up-to-date	Best solution	

For a more extensive survey on existing OSS platforms for the modelling and simulation of an RPA, visit the following reference: (Vogeltanz, 2015). The described software and applications are not only limited to the design and simulation of the FDM; it also includes tools for the estimation of aerodynamic coefficients and the design of propulsion systems, among others.

### ***2.2.2. JSBSim Open-source FDM***

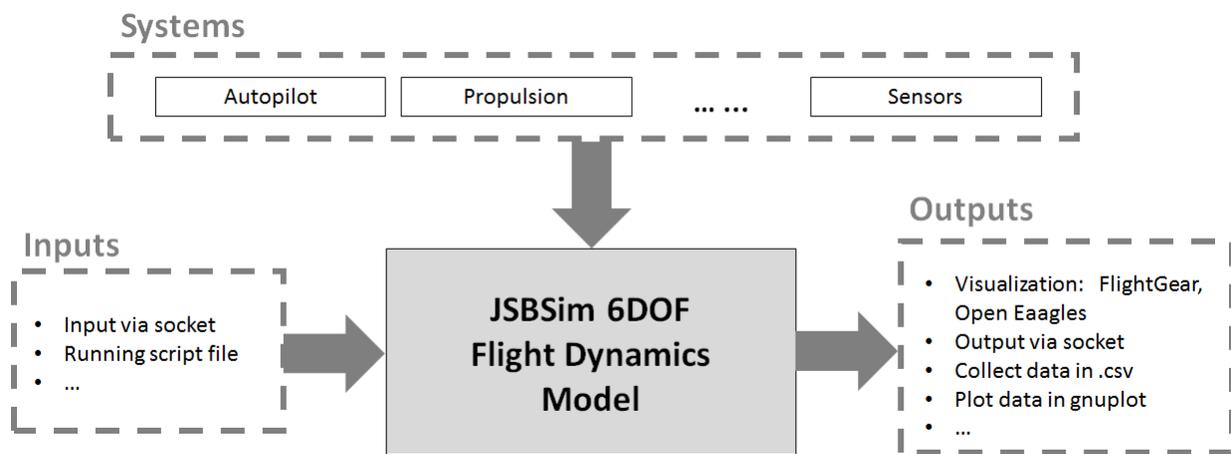
The versatility of JSBSim is one of its strongest attributes. This software can be downloaded from its website (JSBSim Development Team, 2005b) or directly from SourceForge (JSBSim Development Team, 2018). In this section, two different options for its implementation are described. The software analysis included in this paper, as well as the work shown, has been done on a Windows-based computer. Therefore, the software commands in this and the following sections are run under the Windows operating system. However, in the past, the full package has also been tested under the Linux operating system.

#### ***2.2.2.1. Standalone Mode: Scripting***

In the simplest case scenario, JSBSim can be run through scripts entirely implemented through code with FlightGear as the visualization software. In the diagram in Figure 2.2, a simple simulation with JSBSim is shown formed by four main blocks.

- The inputs block defines the initialization of the system as well as future commands or tasks to do by the model.
- The core of the FDM includes the dynamics of the system as well as other physical entities like the atmosphere.

- The FDM also needs extra information from other systems such as the propulsion. Other specifications like actuator and sensors could be added as well.
- The outputs are defined at the end of the FDM configuration file where the type of output will also define its use; for example, an output to FlightGear allows for a visualization of performance, while the generation of an excel file permits the evaluation of certain parameters.



**Figure 2.2.** JSBSim block structure

For more information on how to set up the package and become familiar with its operation, please follow the referenced tutorial (Galbraith, 2010).

#### 2.2.2.2. *Integration into FlightGear*

In this simulation mode, FlightGear hosts the execution of the simulation. The current version is 2019.1.1 (as of November 2019) but the user can download the most suitable version for a specific computer, since the latest version is not always the most appropriate.

In earlier FlightGear developments, JSBSim was its representative aircraft FDM. Currently, this option is still available but also shared with other popular FDM such as YaSim. This flight

simulator only needs the user to select an aircraft, a scenario, and particular conditions in the launching interface. Unlike the standalone version, there is no need to download the entire JSBSim package as only the aircraft configuration and the propulsion files along with additional systems are needed. FlightGear also requires the files associated with the visual model to each aircraft. The base package only includes basic aircraft but a complete list of available models to download can be found online (“FlightGear Flight Simulator,” 2019). Unfortunately, the current online aircraft library does not include any RPA, thus the graphic models used in this thesis were designed from scratch and later imported.

### *2.2.2.3. JSBSim for RPA Applications*

The main reason for developing RPA computer models using the JSBSim package is its capabilities. JSBSim allows the developer to create new aircraft based on the basic package and, simultaneously, its code is also configurable for adding new features. This thesis wants to initiate a discussion on the development of fixed-wing RPA computer models in JSBSim. Considering that an RPA offers a different scenario than jets or other aircraft, slow airspeed and low altitude flying conditions simplify the modelling effort in gravity and air density calculations.

Unlike other flight simulators, JSBSim does not incorporate any graphics and as a result, the processing time required decreases. Certain simulations for this research, such as the computer tests in Sections 4.3.3.1 and 5.1.3.1, do not need any visual display, therefore, the simulation becomes more efficient.

As with any OSS, JSBSim is license-free, capable of working on any platform, and has a large online community. As of November 2019, it has 57 aircraft/aerodynamic models and a large selection of additional systems.

The integration of JSBSim into FlightGear also provides the ability to run two aircraft in the same scenario when in multiplayer mode. This allows for the analysis of their interactions, and is a helpful tool in the development of SAA trajectories for fixed-wing RPAs.

Appendix A includes a full discussion of this topic and a short version of the aforementioned document was also presented at the Newfoundland Electrical and Computer Engineering Conference (NECEC2017) in 2017.

### ***2.2.3. FlightGear Flight Simulator***

Flight Gear (“FlightGear Flight Simulator,” 2019) is an open-source flight simulator which has been in constant development since 1997 (at time of publication, the last version was released on March 14<sup>th</sup>, 2019). The aircraft visual models and environmental components including scenery and airports from different parts of the world are available for download from its online library. It also allows the developer to design new aircraft in SketchUp (“SketchUp,” 2019) and import them later. It is usually run alongside JSBSim as their roles are complementary; JSBSim provides the dynamics of the system and FlightGear the visual performance.

For more information about how to run FlightGear with JSBSim see Appendix A.

## **2.3. Theoretical Background: JSBSim FDM**

The mathematical model used in JSBSim is derived from the most widely used method, Newton’s second law ( $F = m a$ ), where the position, velocities, and accelerations are solved from the total forces and moments, assuming that the aircraft is a 6-DoF rigid body (Cook, 2007). Other methods include the blade element theory (Chen, 1989), which is used in the X-Plane Flight Simulator.

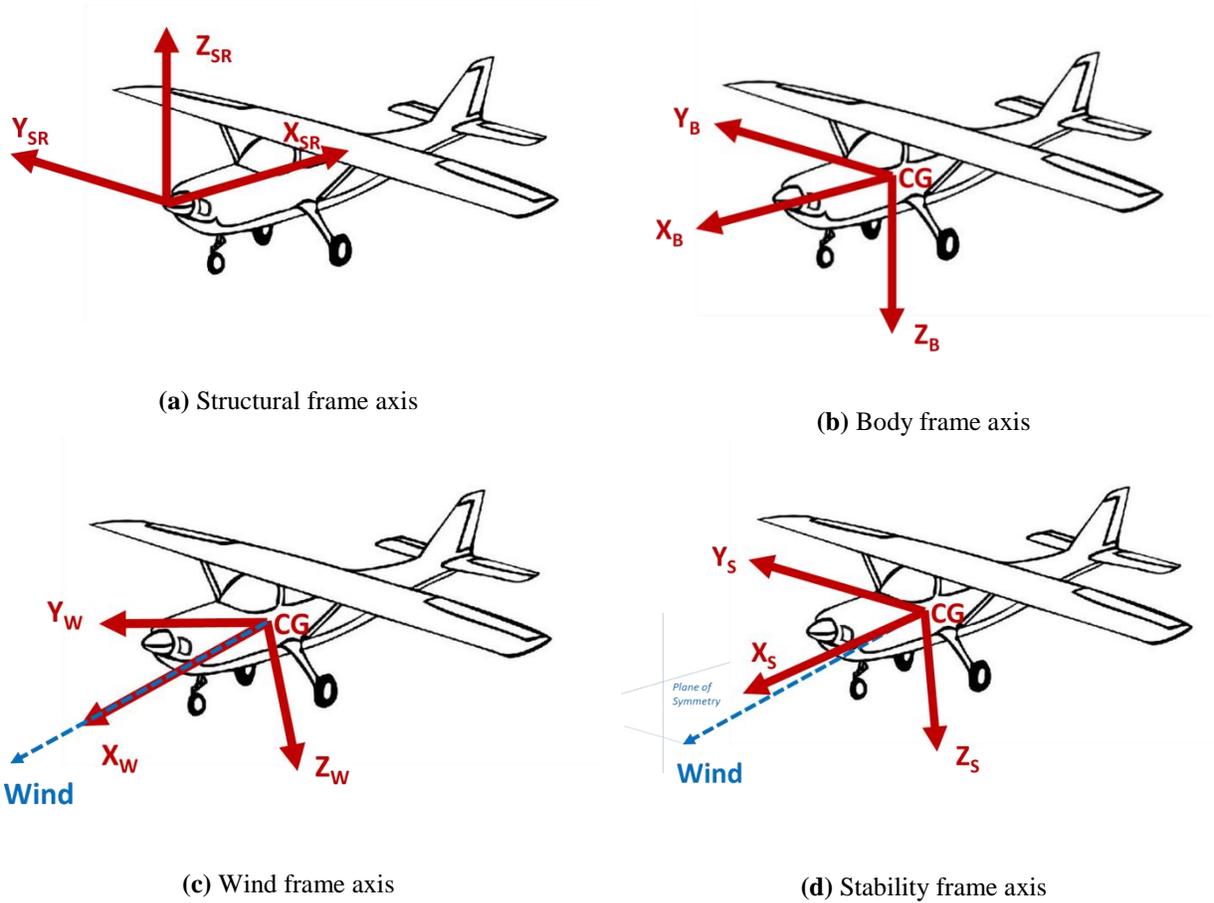
### ***2.3.1. Frames of Reference***

The frames of reference define the structure where the dynamics of flight are calculated. Thus, in order to avoid certain design mistakes that lead to wrong calculations of forces and moments, those frames need to be stated beforehand. There are general standards, but those presented here are particular of the JSBSim package.

- A. Structural frame (Figure 2.3(a)):** The frame relative to the dimensions of the aircraft such as the locations of the masses and the Center of Gravity (CG). The X-axis increases from the nose towards the tail, the Y-axis from the fuselage to the right side and the Z-axis is positive upwards following a right-hand coordinate system. The origin of this frame is located in front of the nose; generally at the tip.
  
- B. Body frame (Figure 2.3(b)):** Where the forces and moments are summed and calculated. It is important that the location of this frame be perfectly referenced from the other frames in order to have correct calculations in the equations of motion (Section 2.3.3). The X-axis points from the tail towards the nose, the Y-axis is placed similarly to the structural frame, pointing right and the Z-axis points downwards. The origin is located in the CG of the aircraft.
  
- C. Wind frame (Figure 2.3(c)):** The frame where the airspeed is calculated, where the X-axis points directly to the relative wind, the Y-axis points to the right and the Z-axis points downwards or upwards depending on the velocity vector, but always in the plane of symmetry and according to the right-hand coordinate system.

**D. Stability frame (Figure 2.3(d)):** The X-axis points to the projection of the relative wind in the plane of symmetry, the Y-axis points towards the right and the Z-axis downwards.

This frame is fixed despite the changes in the direction of the relative wind.



**Figure 2.3.** Aircraft frames

Besides those listed above, there are other frames of reference relative to the Earth that are useful from a navigation point of view but are not important for the present case. Therefore, they are dismissed from this study.

### 2.3.2. Forces and Moments

JSBSim obtains the aerodynamic forces and moments using the coefficient buildup method, meaning that all the contributions to the generation of a specific force or moment are calculated

and summed in order to obtain that total force/moment. The coefficients are taken from flight tests, calculated by hand or using software. JSBSim, through its tool Aeromatic v2.0 found on its website (JSBSim Development Team, 2005a), also provides an approximate set of coefficients calculated from the aircraft dimensions.

The dynamic pressure expresses the approximate relationship between pressure and speed for low flow speeds (e.g. RPA case). By definition, the dynamic pressure represents the kinetic energy by a unit of volume of air used to calculate the force (e.g. lift:  $L$ ) by multiplying a surface area and the aerodynamic coefficient.

$$q = \frac{1}{2} \rho V^2 \quad (2.1)$$

$$L = \frac{1}{2} \rho V^2 S C_L = q S C_L \quad (2.2)$$

Where  $q$  is the dynamic pressure,  $\rho$  is the density and  $V$  is the airspeed –expressed in the wind frame,  $S$  is the wing area and  $C_L$  is the non-dimensional lift coefficient.

The other two forces (drag ( $D$ ) and side force ( $Y$ )) are calculated by following the same concept:

$$D = q S C_D \quad (2.3)$$

$$Y = q S C_Y \quad (2.4)$$

$C_D$  represents the drag coefficient and  $C_Y$  is the side force coefficient. Note that all the aerodynamic coefficients are non-dimensional.

The expressions for the moments (roll ( $l$ ), pitch ( $m$ ), and yaw ( $n$ )) are similar:

$$l = q S b C_l \quad (2.5)$$

$$m = q S c C_m \quad (2.6)$$

$$n = q S b C_n \quad (2.7)$$

Where  $b$  is the wingspan,  $c$  is the wing chord and  $C_l$ ,  $C_m$  and  $C_n$  are the coefficients for roll, pitch and yaw coefficients respectively.

As mentioned earlier, each coefficient is calculated from all the contributions to the force. As an example, the lift dependencies are the following:

$$C_L = C_L(\alpha, \dot{\alpha}, q, \delta) \quad (2.8)$$

Where  $\alpha$  represents the angle of attack,  $\dot{\alpha}$  the rate of angle of attack,  $q$  the pitch rate and  $\delta$  the flight deflections given by the control surfaces. For the slow airspeed and low altitude of the RPA case, the Mach number and altitude elements are neglected because their dependencies in all the coefficients are not significant.

Therefore, the lift coefficient, including the dependencies and its coefficients, is as follows:

$$C_L = C_{L0} + C_L^\alpha \alpha + C_L^{\dot{\alpha}} \dot{\alpha} + C_L^q q + C_L^\delta \delta \quad (2.9)$$

$C_{L0}$  represents the lift force at zero angle of attack and each of the lift coefficients ( $C_L^\alpha$ ,  $C_L^{\dot{\alpha}}$ ,  $C_L^q$ ,  $C_L^\delta$ ) is consequence of the corresponding  $\alpha$ ,  $\dot{\alpha}$ ,  $q$ ,  $\delta$  component (superscript).

Following the same approach as expressed for the lift coefficient, the drag coefficient  $C_D$ , the side force coefficient  $C_Y$ , the roll coefficient  $C_l$ , the pitch coefficient  $C_m$  and the yaw coefficient  $C_n$  for an RPA case in JSBSim are expressed as:

$$C_D = C_{D0} + K C_L^2 + C_D^\alpha \alpha + C_D^\delta \delta \quad (2.10)$$

$$C_Y = C_Y^\beta \beta + C_Y^p p + C_Y^r r + C_Y^\delta \delta \quad (2.11)$$

$$C_l = C_l^\beta \beta + C_l^p p + C_l^r r + C_l^\delta \delta \quad (2.12)$$

$$C_m = C_{m0} + C_m^\alpha \alpha + C_m^{\dot{\alpha}} \dot{\alpha} + C_m^q q + C_m^\delta \delta \quad (2.13)$$

$$C_n = C_n^\beta \beta + C_n^p p + C_n^r r + C_n^\delta \delta \quad (2.14)$$

Where  $\beta$  is the sideslip angle,  $p$  is the roll rate,  $r$  is the yaw rate and each of the corresponding coefficient dependencies is indicated by their subscript.

The drag coefficient includes a special term  $K C_L^2$ , which indicates the induced drag consequence of the lift. Equations (2.9)-(2.14) are general statements, but they must be modified according to the aircraft. JSBSim allows modifications in the aircraft configuration file in case the design carries other dependencies.

Each contribution is expressed as a function of the corresponding “LIFT”, “DRAG”, “SIDE”, “ROLL”, “PITCH”, “YAW” axes in the JSBSim setup. Each of the coefficients may be expressed either as a constant coefficient or as a value dependent on a certain property taken from a lookup table.

For further information about the physics involved in the flight, the following references are recommended: (Barnard & Philpott, 2010; Shevell, 1989; Stevens, Lewis, & Johnson, 1992)

### ***2.3.3. Equations of Motion***

The performance of the aircraft is defined by its motion variables as expressed in the body frame by:

$$\{v\}_B = \begin{bmatrix} u \\ v \\ w \end{bmatrix}, \quad \{\dot{v}\}_B = \begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix} \quad (2.15)$$

Where  $[u, v, w]$  are the linear velocities and  $[\dot{u}, \dot{v}, \dot{w}]$  their corresponding derivatives. According to the Newton’s second law, where  $F = m a$ , the forces equation expressed in the body frame is:

$$\{F\}_B = \begin{bmatrix} F_X \\ F_Y \\ F_Z \end{bmatrix} = m \{\dot{v}\}_B + m \{\Omega\}_B \{v\}_B, \quad \text{where } \{\Omega\}_B = \begin{bmatrix} 0 & -r & q \\ r & 0 & -p \\ -q & p & 0 \end{bmatrix} \quad (2.16)$$

The total force calculated in the body frame  $\{F\}_B$ ,  $[F_X, F_Y, F_Z]$  is calculated from the mass,  $m$ , the acceleration,  $\{\dot{v}\}_B$ , and the cross-product equivalent matrix consequence of the derivation of the rotating frame,  $\{\Omega\}_B$ . Thus, the linear velocities are directly calculated by:

$$\begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix} = \frac{1}{M} \begin{bmatrix} F_X \\ F_Y \\ F_Z \end{bmatrix} - \begin{bmatrix} 0 & -r & q \\ r & 0 & -p \\ -q & p & 0 \end{bmatrix} \begin{bmatrix} u \\ v \\ w \end{bmatrix} \quad (2.17)$$

Additionally, the motion rate is given as the angular velocities  $[p, q, r]$  and their corresponding derivatives  $[\dot{p}, \dot{q}, \dot{r}]$ , can be calculated following the next equations according to the rotating moment of inertia definition:

$$\{w\}_B = \begin{bmatrix} p \\ q \\ r \end{bmatrix}, \quad \{\dot{w}\}_B = \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} \quad (2.18)$$

$$\{M\}_B = \begin{bmatrix} M_X \\ M_Y \\ M_Z \end{bmatrix} = I_B \{\dot{w}\}_B + \{\Omega\}_B I_B \{w\}_B \quad (2.19)$$

$$\begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = I_B^{-1} \begin{bmatrix} M_X \\ M_Y \\ M_Z \end{bmatrix} - I_B^{-1} \begin{bmatrix} 0 & -r & q \\ r & 0 & -p \\ -q & p & 0 \end{bmatrix} I_B \begin{bmatrix} p \\ q \\ r \end{bmatrix} \quad (2.20)$$

Where  $\{M\}_B$  is the angular momentum of the aircraft in each of the axis  $[M_X, M_Y, M_Z]$  and  $I_B$  is the inertia matrix.

The Euler angles  $[\phi, \theta, \psi]$ , which specify the orientation of the aircraft, are obtained from the transformation stated by the matrix  $T_{H,B}$ :

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = T_{H,B} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \quad (2.21)$$

The Euler angles are expressed in the local-horizontal reference frame used for navigation. Its origin is the CG of the aircraft with the X-axis pointing north, the Y-axis east, and the Z-axis to the centre of the Earth. Although the Euler angles are important parameters during flight, this transformation is usually carried out using quaternions due to singularities generated by general rotation matrices.

In summary, when all the forces and moments involved in the flight due to aerodynamics, thrust, weight, external forces, etc. are given, it is possible to compute and derive the motion variables expressed by the linear/angular velocities and the Euler angles.

JSBSim executes the equations stated in this section in the file called FGAccelerations under the FGModel hierarchy. For more information about the different classes used in the core of the software, see Appendix A or visit (JSBSim Development Team, 2017).

#### ***2.3.4. Propulsion***

JSBSim includes different types of propulsion systems depending on the engines used to generate the thrust: a piston engine model, a jet turbine engine model, a turboprop engine model, a rocket engine model, and an electric engine model. In case of RPAs, only the piston and electric models will be used. The thrust generation presents the same setting where among all the options found in JSBSim –direct, nozzle, propeller and rotor- only the propeller is used in the fixed-wing RPA case.

The propulsion system, including the engine and the origin of the thrust generation, are called from the aircraft configuration file. The thrust generated in this system has its own reference frame relative to the structural frame, which is defined in the aircraft configuration file and later taken to the body frame with the rest of the forces during flight.

The thrust generated in the body frame is given by:

$$Thrust = C_T q S, \quad \text{where } C_T = C_T(Velocity, Controls) \quad (2.22)$$

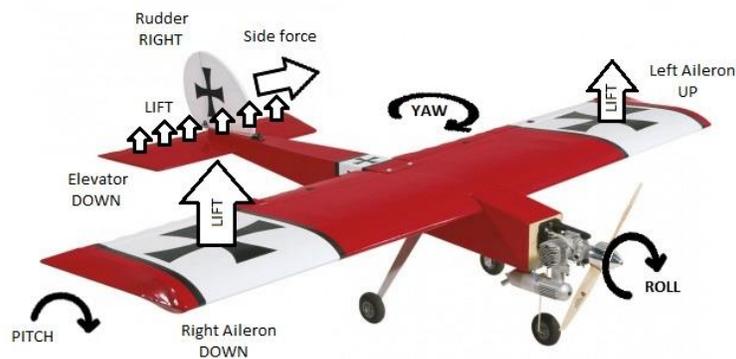
Where  $q$  is the dynamic pressure,  $S$  is the wing area and  $C_T$  is the non-dimensional thrust coefficient.

Two types of propulsion systems in JSBSim with a piston engine in the Giant Big Stik case and an electric engine in the case of the EPP FPV are shown later in Sections 3.3.2 and 3.3.1 respectively; the other systems are irrelevant.

### 2.3.5. Flight Control

In a simple simulation, the primary controls for the model are the surface command elements of the aircraft: ailerons, elevator, rudder, and flaps (if existed).

Each of the control surfaces has a specific range according to the characteristics of the physical model. The convention for positive or negative deflections according to (Durham, 2013) is as follows (Figure 2.4):



**Figure 2.4.** Giant Big Stik. Forces and moments

- The aileron deflection is positive when the right aileron is trailing-edge down and the left aileron is trailing-edge up. Under this condition, there is a difference in the lift being greater on the right wing, creating a positive roll and causing the aircraft to turn left.
- The elevator deflection is positive when positioned downward, lifting the tail. Thus, the nose points down and the angle of attack decreases.
- The rudder deflection is positive when the rudder is trailing-edge right, heading right.

Staging these requirements for a true flight will allow the designer to implement a correct validation and future model modifications.

The main focus of this thesis in terms of validation (Chapter 3) is on the manual or open-loop control where the model is directly evaluated from changes to the control inputs. Note that automatic flight involves an autopilot, requiring a closed-loop test. This control structure may lead to wrong conclusions; the adjustments of the model can be done by the control, but that does not mean that the model expresses the dynamics of the system.

The source of the surface deflections for the purpose of this work comes from a pilot's manual/open-loop control, whereas closed-loop control with autopilot will be used mainly in Chapter 5.

## **2.4. Computer Model Development**

By definition, a model is a representation of a system for a particular purpose and application, meaning that the model is not required to be an exact representation as long as its limitations are clearly defined.

Computational modelling plays an important role in the aerospace industry for the development of the aircraft and its environment. Modelling and Simulation (M&S) methods are used to replicate aircraft performance with a wide range of applications. The aircraft is considered a system composed of subsystems that need to be modelled such as the aerodynamics, atmosphere and control systems.

In software development, the system lifecycle is generally expressed by the classic V-diagram (Forsberg & Mooz, 1991). As expressed in Figure 2.5, time runs from left to right, the tasks related to the early stages of the computer development are located on the left side, and the integration of the model into its application is on the right side. In the early stages of model development, the design is validated by M&S, whereas physical tests are used as a validation tool later in its development.

In (Diston, 2010) – Figure 2.5, each stage of the lifecycle is divided into mini “V” lifecycles that allows the systems to be tested in each stage. The system must satisfy the requirements before continuing with the next step, preventing problems to be extended to other development stages. For example, the first part of Chapter 4 is based on the “V-diagram” where the computer model development and its mini-lifecycles are the focus. The first part of Chapter 4 relates to the simulation viability and Chapter 5 integrates a model into the final simulation and application.

Verification and Validation (V&V) procedures are critical aspects of model development. As defined in (Murray-Smith, 2012), the verification concept certifies the correctness of the specifications while the validation confirms the behaviour of the system according to the requirements. The validation stage is important to demonstrate that the model can reproduce real scenarios.

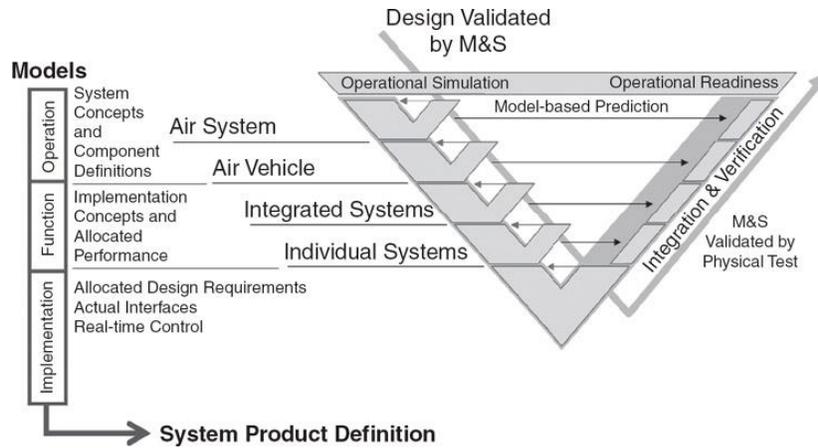


Figure 2.5. Model and simulation within the system development cycle (Diston, 2010)

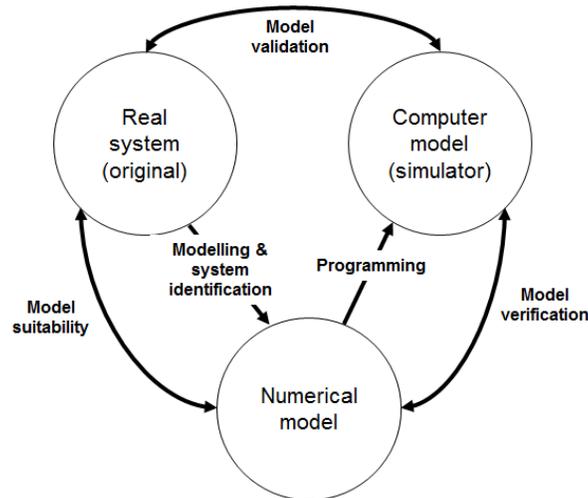


Figure 2.6. Modelling of a dynamic system (Buchholz et al., 1996)

Figure 2.6 in (Buchholz, Bauschat, Hahn, & Pausder, 1996) expresses the basic structure of the modelling of a dynamic system (aircraft). In this configuration, the *real system* is what is to be modelled, the *numerical model* contains the mathematical description of the real system (*M&S*), and the *computer model* is the algorithm of the numerical model in a particular programming language (*programming*). The main requirement for any computer system is to be the

representation of the real system, which is evaluated by *model validation* techniques. *Model verification* is defined as the correctness of the computer model compared to the mathematical expression of the real system.

For this thesis, the focus is on the *programming* and *model validation* tasks, assuming the relationship between the real system and the numerical model is approximate and the errors in programming depend on the coding approach (or FDM tool) of choice.

In Aerospace, the methods for computer model validation in general aircraft are assorted (McGovern, 2007) and dependent on the company/country regulations:

- 1- Military services criteria:** Most military entities have their own requirements that are either agreed to with other entities or followed according to the regulatory agencies' standards; the requirements are specific to the final purpose of the project.
- 2- Regulatory agencies simulation qualification:** Agencies such as the FAA, Transport Canada, International Civil Aviation Organization (ICAO), and Civil Aviation Authority (CAA) include special standards for the development of aircraft simulators. They mostly cover flight simulators and their requirements are very specific. For example, simulators for aircraft under the jurisdiction of the FAA are validated by the corresponding FAA Advisory Circular under the National Simulator Program (NSP) (Federal Aviation Administration, 2019). In Canada, similar qualifications are included in the Aeroplane and Rotorcraft Simulator Manual (TP9685E) (Transport Canada, 1998) under the National Simulator Evaluator Program (Transport Canada, 2010a).
- 3- Experimental flight testing:** This method collects data from experimental flights to analyze the limitations and accuracy of the computer models.

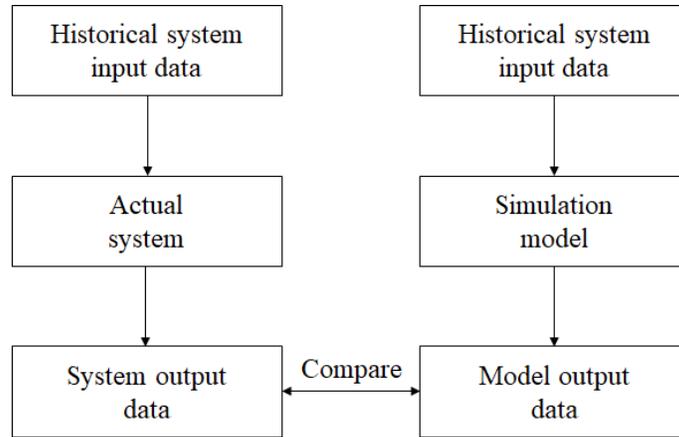
- 4- **Pilot validation:** The Pilot's Operating Handbook (POH) regularly contains general data and recommended practices useful for the validation of the model by a pilot.
- 5- **Operational flight procedures:** Similar to the Cooper-Harper handling qualities for aircraft flight (Cooper & Harper, 1969), a computer model can be evaluated by following the same scale for its intended purpose.
- 6- **Observation/inspection of aircraft model performance:** This includes traditional software testing: inspection, results verification by exercising the real model, and observation of the model's behaviour. These techniques are used to observe and determine the viability of a model in a more accessible way when the model design is highly limited.

In a general modelling simulation, traditional methods for validating include (Law, 2014):

- Consulting with experts on the real model performance.
- Conducting sensitivity analysis to determine important factors.
- Using statistical procedures to evaluate the similarities between the real model and the simulation output.

These techniques, the latter in particular, present a resourceful tool since both models (real and simulated) receive exactly the same observations from input variables. This procedure is called the *correlated inspection approach* (Figure 2.7) and it has been frequently used in V&V since it provides comparable output data to evaluate whether the simulation model correlates to the real model and follows the assumptions.

In most simulation model scenarios in this research, there is not a high interest in expressing all the real characteristics of the model. The historical data must represent the simulation context and define the level of model detail in the simulation.



**Figure 2.7.** The correlated inspection approach (Law, 2014)

A successful model validation must also include detailed documentation, including the model requirements and limitations. Additionally, the problem formulation is essential in the first stage of the validation procedure whereas the model credibility discussion is essential in the closing section.

### ***2.4.1. Validation methods for RPAs***

To the best of the author’s knowledge and as of November 2019, there are no official recommendations for validating RPAs computer models. The most common practice found in the literature is to integrate the physical and aerodynamic models into the simulation framework and compare the performance against flight test data. This informal procedure has been followed by many researchers (Ke, Wang, & Chen, 2018; Liu, Egan, & Santoso, 2015; Zekry, Nabil

Mobarez, Ouda, & Zekry, 2016). However, this practice lacks standardization and it becomes hard to compare computer models at the same level.

Other means of validation include the application. These particular approaches focus on the task to be carried out by the system rather than the computer model itself. In these cases, the aircraft performance is not considered; the mission scope determines the relevant stages for the validation (M. Mueller, Smith, & Ghanem, 2016; Villa, Salimi, Morton, Morawska, & Gonzalez, 2016).

The design of flight simulations also lacks guidelines and researchers have validated their simulators by testing the final application of the simulator (e.g. multi-RPA missions in Rodriguez-Fernandez, Menendez, & Camacho, 2015) instead of the computer aircraft performance.

As shown, this gap of standards in the literature is one of the main motivators of this research.

## **2.5. Sense and Avoid Basis**

In (Government of Canada, 2019), the concept of SAA, is defined as “the capability to see, sense or detect conflicting traffic or other hazards and take the appropriate action to comply with the applicable rules of flight”. Although there are comparable terms used in the literature such as Detect and Avoid (DAA) and Sense, Detect and Avoid (SDA), they are often misused. Considering the application of this thesis, the term SAA is preferred.

The term *sense* describes the ability of the system to identify the hazard either through a cooperative system (e.g. TCAS transponder ((Federal Aviation Administration. U.S. Department of Transportation, 2011) and Appendix C) or Automatic Dependent Surveillance-Broadcast

(ADS-B)(Ramasamy & Sabatini, 2015)) or using a non-cooperative approach (e.g. RADAR or a vision-based system); whereas the second term, *avoid*, refers to the automated control required to avoid a collision that has been detected in the first stage. Both elements have equal importance and offer a challenge in order to integrate the UAVs into the shared airspace.

The minimum requirement for intruder detection and the SAA task is that there is enough time for the aircraft to perform a manoeuvre and remain safe. The functional boundaries and thresholds define the risk of an airborne collision. The two major components of the SAA task are (1) Self-Separation (SS) and (2) Collision Avoidance (CA) (Federal Aviation Administration, 2013b).

### ***2.5.1. Scope of the SAA application***

The SAA application on this thesis focuses on the avoidance stage. This means that the concept of detecting (Mcfadyen & Mejias, 2016), tracking (Bharati, Wu, Sui, Padgett, & Wang, 2018), and estimating the Closest Point of Approach (CPA) (Fasano, Accado, Moccia, & Moroney, 2016) are out of the scope of this work.

Regarding the avoidance task, the SS component is also out of scope since the goal of SAA is to avoid a close encounter. CA represents the last stage prior to a collision and the focus of this particular application. The objective of the CA task is to perform a manoeuvre when all previous avoidance measures have been failed and the intruder is close to the ownship's collision volume (Federal Aviation Administration, 2013b).

For more detailed information and related concepts to SAA, please see Appendix C.

## **2.6. Summary**

The author is aware of the diversity of the fields covered in this document and for that reason, the main goal of this chapter was to provide a strong context for the understanding of the research conducted and justify the methodologies and applications presented.

The application of the work that follows is the safe integration of RPAS into the airspace and considering their interaction with piloted aircraft. As part of that, basic concepts, definitions and an overview of the RPAS problem in Canada were introduced. The new regulatory project (recently in effect since June 2019), deserved a special mention to present the future steps of this technology. Even though this chapter has provided a brief overview of the SAA issue, more specific context and background are included in Appendix C.

Overall, this chapter can be divided into three main topics: (1) the RPA and the SAA challenge, (2) the simulation platform and (3) the tools used to calculate the results, which includes M&S concepts and computer model development methods.

# Chapter 3

## *Open-Source Software in Aerospace with RPA Applications: Challenges and Solutions*

The technical difficulties associated with the development and study of Remotely Piloted Aircraft (RPA) computer models in flight simulators were the main motivation for the creation of this chapter. It aims to join concepts from the fields of Modelling and Simulation (M&S), Verification and Validation (V&V) methods, engineering education and computer science.

Even though the application in this research is oriented towards RPA M&S, certain sections are expandable to other disciplines within and outside engineering. This chapter is organized with three main objectives: (1) to discuss why open-source platforms are the most suitable framework for the design, development and evaluation of RPA computer models, (2) to analyze the technical difficulties that graduate students face when using Open-source Software (OSS), and (3) to list the main challenges of in JSBSim, which are not only limited to designing small fixed-wing aircraft.

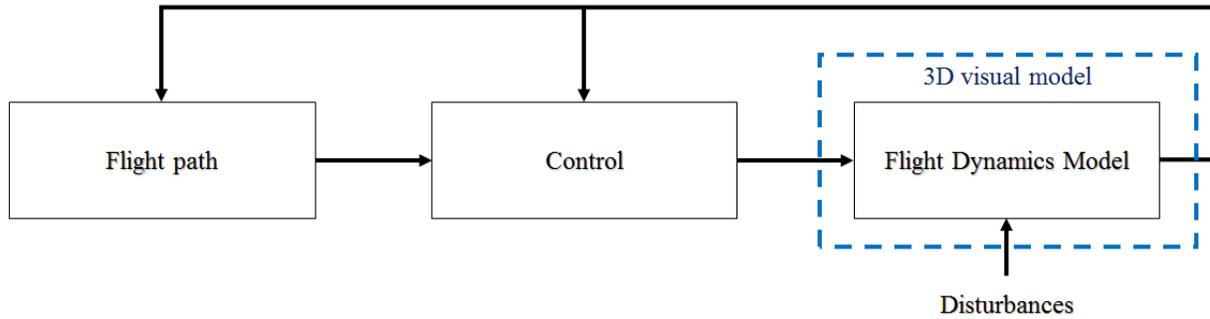
### **3.1. Basic Flight Simulator Framework**

Chapter 2 – Section 2.1. observed and evaluated the current situation of RPAS in Canada and the regulations as a consequence of their integration into the airspace. However, the challenges of an increasing RPAS industry are not only limited to this integration. Every year, a large variety of vehicles with different designs and capabilities (“Taking flight,” 2017) as well as sensors and other related devices are created and validated.

Performance and visual simulators are a crucial element in the RPAS industry development. Real models must be tested and analyzed in a simulated context before implementing them in a real situation. Special cases, like Sense and Avoid (SAA) and Collision Avoidance (CA) studies require simulators in each of their implementations due to the hazard that they represent in the airspace.

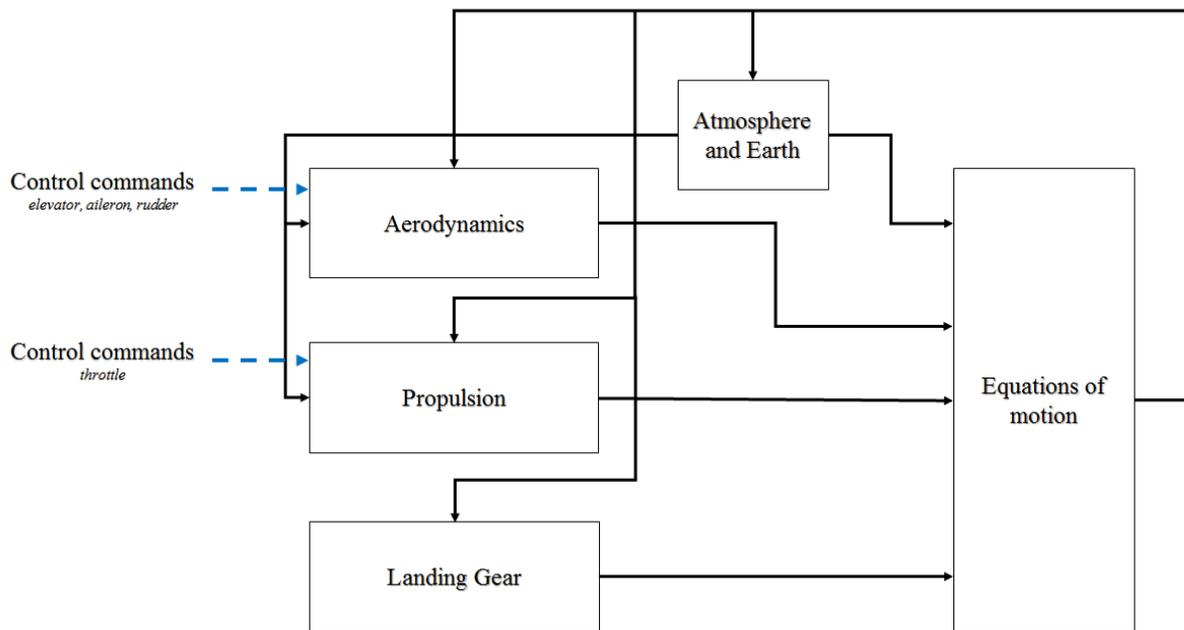
In general terms, a flight simulator must be divided into the following blocks (Figure 3.1), which are not only limited to the vehicle model (Allerton, 2009). These include:

- The Flight Dynamics Model (FDM) expressed by the mathematical equations that represent the forces and moments acting on the aircraft.
- The 3D graphical model able to display the aircraft performance.
- The control system and/or autonomous flight control.
- The flight path expressing the route of the aircraft.



**Figure 3.1.** Basic aircraft simulation framework

In particular, the minimum representation of an FDM must include the following blocks:



**Figure 3.2.** FDM blocks

Whereas the Atmosphere and Earth blocks are unchanging models for any vehicle size, the Aerodynamics, Propulsion and Landing Gear blocks are particular of an aircraft configuration. For the low altitude and slow airspeed case of the RPAs, the atmosphere and Earth models are not relevant. Considering the application of SAA and the study of in-flight manoeuvres, the landing gear is neither relevant. Similarly, the propulsion system can be simplified by indicating

the power of the engine and the propeller configuration (see Appendix D). Finally, the mathematical foundation and solving method (Section 2.3.3) is based on known approaches that are common to any vehicle.

Therefore, the FDM, excluding the Equations of motion block, represents the computer expression of the real aircraft performance that is designed in the modelling stage in an RPA simulation (Diston, 2010). The Control and Flight path blocks (Figure 3.1), belonging to the control stage in the aircraft design, are defined and tuned after the computer model is verified.

Overall, the challenge of simulating RPAs is to find a platform able to integrate new models while maintaining the context of the simulation (e.g. 3D graphical model). Ideally, the simulation platform would be expandable to other models and allow for the interaction of the model with other elements in the airspace, such as piloted and other RPA.

Most of the available flight simulators used for training are proprietary software (“Flight Simulator : Plane Pilot - Microsoft,” 2018; “X-Plane 11 Flight Simulator,” 2018), limiting the aircraft library to what was available when the license was purchased. A possible solution to this problem is to model the blocks in Figure 3.1 and Figure 3.2 from scratch, which adds unnecessary extra workload and cost.

OSS offers a framework able to integrate computer models into existing simulated environments. With the possibility of accessing the code, the simulation becomes flexible to modifications depending on the application and expandable to other and even external models. Formally, OSS refers to any software that is available to the public, including the concept and its development. It also allows for collaborative work across disciplines by integrating computer theory into solving real cases.

Designing an RPA is not an easy task; existing OSS packages (Section 2.2) cover a wide range of applications that help the user with the aircraft modelling and control task. As an example of its complexity, the aerodynamic coefficients are particular of a vehicle and are usually estimated from its geometry; an arduous task that could be simplified by using software.

While the OSS applications are widely varied, its quality is usually questioned (Pandey & Tiwari, 2011). An open-source FDM must represent the aircraft performance with a certain confidence level. V&V procedures are required to not only rate the quality of the software but also the authenticity of the FDM. The first issue associated with the validation of the OSS simulated platform is further discussed in Section 3.2, whereas validation procedures for RPA FDMs present a particular challenge that is introduced and addressed in Chapter 4.

## **3.2. Technical challenges of working on/with OSS in Academia**

Overall, OSS is an increasingly important tool used for producing scientific research. It has received the attention of both academia and the private sector over the years. Its adaptability to a specific set of requirements as determined by a project is what has resulted in the common application of OSS in academia. In fact, most academics agree that software is a significant part of their work and, without said software, their research would be ineffective (Nangia & Katz, 2017).

In most of the scientific fields outside of technology, however, programming skills are not taught (Bowlick, Goldberg, & Bednarz, 2017). This lack of skillset in a student's curricula may be translated into practical setbacks at the graduate level (van de Schoot, Yerkes, Mouw, &

Sonneveld, 2013). While experiencing difficulties with understanding and working with somebody else's code is expected (Vanhanen, Lehtinen, & Lassenius, 2012), this journey could be eased by following a simple set of practices from the software developer.

Other barriers faced by graduate students and researchers include the lack of standards for publishing and citing code. The amount of time required to learn the basics of programming, and the traditional view of software as “just a tool” and not a scientific output, has resulted in many students to avoid OSS entirely (A. M. Smith et al., 2018).

This section discusses the issues associated with OSS from the software developer and user's perspectives and aims to create a debate about its correct use in academia. A session on this topic was presented at The ACM Canadian Celebration of Women in Computing 2018 in Halifax, Canada and the Teaching and Learning Conference 2019 at Memorial University, St. John's, Canada, in May 2019. A more extensive work including more specific barriers and recommendations is currently pending publication for the open-access Facets journal (Canadian Science Publishing, 2019).

### ***3.2.1. Software in the Scientific World***

According to a survey by the US National Postdoctoral Association, 95% of the scientists used software as the main tool for their work and 63% said that their work would be meaningless without it (Nangia & Katz, 2017). Scientists spend 30% of their time developing software, and for 90% of them, the ability to use software is self-taught (Prabhu et al., 2011).

Nowadays, it is incongruous to think of doing science as a discipline without the contribution of software developers and the input of computer science. Software and its development represent

an important research output and must be understood as a significant part of science (Carey & Papin, 2018).

Discussion around the correct development of OSS has been taking place since the 90s, with the book *The Cathedral and the Bazaar* and other related publications (Nikolai, 1999; Raymond, 2001). However, there are still no clear solutions, standards, or recommendations established worldwide for the development of OSS. The scientific community is greatly in need of guidelines to organize and administer the best practices for the development and use of OSS.

The reliability of OSS has also been questioned in the scientific community. Much of this discussion comes from closed source defenders, who claim that OSS is more vulnerable to security flaws and attacks due to its accessibility. On the contrary, it has been demonstrated that publishing source code allows for peer review and ensures that the code executes what it should do. Studies have shown that OSS has the same or better level of reliability compared to proprietary software (Pandey & Tiwari, 2011; Wheeler, 2015).

### ***3.2.2. Benefits of Using OSS in Graduate School***

There is a high interest in integrating OSS in academia since it allows for modifications, the addition of extra features, and it is distributed without cost. However, according to the Open Source Initiative (OSI), OSS must not only be viewed as free software. It also needs to meet certain criteria in terms of its source code, integrity and license (“The Open Source Definition,” 2007).

With the digitalization of education through online courses, virtual universities, and education portals, academic institutions want to make code more accessible to everyone (Shaheen E.

Lakhan, 2008). Through the use of OSS, these organizations are capable of having a source code base that the students can access and work with for a specific course.

The main advantages of using OSS in general, and in academia in particular include:

- 1) OSS is license-free. Academic institutions currently pay a large amount of money for using licensed software when it is used by many students. Specialized software is missing from academia due to a high cost per student; with no license cost associated, OSS is accessible to every student.
- 2) OSS is constructed from the fruitful collaboration of a group of developers. It allows for continuous maintenance and expansion, depending on the user's needs and the progress of the technology in the field of application. Every OSS has a community behind it and a large network that is maintained by volunteers; bugs are fixed in hours due to accessible and transparent code. The growth of the internet worldwide has also contributed to the remote communication between developers allowing the production of hundreds of thousands of new projects.
- 3) OSS provides a large opportunity for research. Research teams develop software that will be useful to other students/researchers in the future. Similarly, if source code is divulged, another team in the same or a similar field can integrate the software and save time in favour to research work.
- 4) OSS is interdisciplinary. As with any other software, OSS provides an apparatus for solving problems, with applications varying from business to biology and education. The flexibility of OSS means that the same source code can be adjusted from one discipline to another.

- 5) OSS communities create quality software. Software projects are validated and tested before they are released. Recognized OSS projects include Linux, Mozilla and Open Office. Current OSS trends in the industry show that OSS drives the expansion and maintenance of web browsers, artificial intelligence, operating systems, financial services, machine learning, etc (Burning Glass Technologies, 2016).

From the academic instructors' perspective, the use of OSS in their courses allows the students to develop their ideas, motivate a collaborative environment and serve as a link with industry in hackathons (Shaheen E. Lakhani, 2008). From the students' perspective, OSS presents an attractive tool to build their programming skills and other marketable features like teamwork and interdisciplinary research.

Unfortunately, developing software demands time, maintenance, and an overall commitment that not everyone in the community is willing to deliver.

### ***3.2.3. Challenges of Developing OSS in Graduate School***

The main problem faced by graduate students in most fields is getting approval from the scientific community by publishing their work. This allows them to be considered an accepted researcher and to build their curricula. Publishing research software is a complicated obstacle for graduate students: there are no formulated citation standards, not enough journals that accept OSS for submission, and other obstacles associated with its publication and review. This debate has begun to be addressed, and within the last few years, some journals have been founded in order to fill that gap ("Journal of Open Research Software," 2018; "SoftwareX," 2019; "The Journal of Open Source Software," 2018). As an example, in the first year and a half since its formation, the Journal of Open Source Software (JOSS) published nearly 200 articles (Smith et

al., 2018). These journals have shown that they are clearly in great demand based on the increasing number of submissions every month.

The creation of any OSS depends on the project's program and the developers' motivation. Most of the OSS contributors get involved with a project because they have an interest and they maintain the platform voluntarily. In larger research projects, OSS relies on awarded funding, similar to the applied sciences. The problem associated with this scenario is that the funding can run out and as a result, the project might be left incomplete.

If supported voluntarily, the main creator of the software might be the only person that keeps the software alive. Under these circumstances, there is a high risk of technical support being discontinued due to the burnout of the only developer. Significant projects could die because of a lack of funding, active contributors, and interest from the community.

In the absence of funding, the main developers need cooperation from other contributors to keep the software alive. However, graduate students do not appreciate the importance of collaborating with OSS projects, since traditional science does not consider developing software as a scientific contribution (A. M. Smith et al., 2018). The student questions whether the effort of producing code will be useful towards completing their graduate degree.

This situation can be avoided if the code is adaptable to other scientific fields, according to the criteria listed by OSI. The source code should be constructed for a group of people who will use it in the future, and not because the original developers want to create an application that only interests themselves. There is a high risk that this code will disappear unless the software is one of interest. From a graduate student's perspective, there is no incentive to develop software if it is not going to be useful to others. Therefore, the code must be adaptable, as listed by OSI in

their criteria for OSS. These and other practices are suggested in (Wilson et al., 2017). These recommendations were designed based on the experience of both software developers and users in order to maintain communication between the two groups and make the code more accessible and easy to understand.

The importance of keeping sufficient documentation deserves a special mention, since manuals help users understand the code. In the same way the research is documented, code must be described and referenced. Furthermore, the reference manual should also be reviewed by users, taking into account that the creator of the source code will likely take certain key elements for granted. Weak and incomplete documentation leads to unnecessary delays on the user's end.

Another issue faced by researchers is that most of them do not know how reliable their software is because of the lack of validation by external reviewers (Bacharakis, 2018; Barnes, 2010). This might result in errors in other projects and published research where that software has been used. There is a demand for revision and validation of OSS from other developers that can also contribute towards its publication.

#### ***3.2.4. Challenges of Working with OSS in Graduate School***

As pointed out in Section 3.2.2, OSS provides a powerful source for graduate students to build their programming skills and develop a tool that they will use in their research. However, the main problem encountered by students who work with OSS and are not part of the developer's team are the difficulties associated with working with an unfamiliar code (Vanhanen et al., 2012). If a minimum set of documentation is not provided along with the software, using the tool becomes an arduous task and adds unnecessary delays and extra barriers that are not related to the research. Complications with the learning process, system incompatibilities (List, Ebert, &

Albrecht, 2017), and lack of guidance are the most common barriers that students experience during their program.

In addition to the current lack of journals and publications mentioned in Section 3.2.3, there is also a lack of citation standards. Little has been commented on this topic (Smith, Katz, & Niemeyer, 2016) and it remains one of interest for both software developers and users. Graduate students need to cite their code as part of their research work; a wrong citation could cause misleading arguments.

Lack of cooperation and community support is another big issue when working with OSS (Geiger, Varoquaux, Mazel-Cabasse, & Holdgraf, 2018). The technical support for OSS is inferior and more inconsistent in comparison to commercial software, especially when the platform is maintained by a single person.

### ***3.2.5. Overall Perspective on Best Practices for a Positive OSS Experience in Graduate School***

Many challenges often discourage students from integrating these computing skills into their research and interacting with the OSS community. The following series of recommendations address these barriers from the perspectives of the students, supervisors, administrators, and OSS members and present solutions that if conducted simultaneously, can significantly improve the graduate student experience. This will motivate the students to conduct effective research with the support of OSS mentors and peers while the OSS community benefits from the students' involvement.

*a) Recommendations for graduate students: be active and contribute back*

Students are usually aware of their limitations but are unaware of what falls beyond their knowledge. Keeping the research application as the framework, students could learn computing skills without considering it a wasted effort. Learning takes time and patience but ultimately, it benefits the student and it helps to advance their research. Eventually, these newly acquired computer skills are added to the students' curricula, making an impact and opening it for new opportunities.

Some students do not recognize the strong environment that academia offers: in addition to attending workshops to improve their skills and learn new ones, students are encouraged to talk about their research and the barriers they face. Specific to OSS and coding, graduate students often avoid speaking about the challenges they encounter because they do not want to show their own limitations. However, they will likely find support and guidance from other students and/or researchers who are familiar with the issue, and who might be interested in getting involved with the OSS project. An example of a direct local resource is senior graduate students: they have the knowledge and experience of dealing with similar difficulties. More opportunities can arise outside academia: for example, Twitter and other social media platforms provide an incredible setting for the student to involve themselves with the OSS community. The more visible and active the student is, the more support they will have when integrating OSS into their research.

Students should not also limit themselves to only generating a piece of code that facilitates the analysis of their results. They should ask for feedback, document, and publish their code. This is not only beneficial to the student from an academic perspective, but it also contributes to sustaining the OSS loop. Additionally, it is important that the student remains active even after they have contributed to the OSS community. Future students may encounter the same issues,

and the expertise of former students will help others avoid unnecessary delays and support the whole community.

***b) Recommendations for supervisors: be a resource and a role model***

The support and involvement of a student's supervisor is crucial throughout the research process. As a mentor, they are encouraged to defend the creation of software as a research product, and motivate students to code, develop, and contribute back to the OSS community.

Supervisors must also acknowledge their own limitations and direct students to other members of the academic community who can offer additional support. They should offer students opportunities to complete an introductory course to programming before the skills are required in their research. They must make their expectations of the students clear, and provide constant feedback on goals and timelines. This will ensure that the students follow the academic procedures and the research plan without missing the perspective of the application.

Introducing the full perspective of a research problem and showing a student the importance of interdisciplinary research showcases how broad research is and the opportunities it offers. Supervisors should connect peers in an interdisciplinary setting; knowledge is naturally shared between the members of a team and therefore, a diverse group promotes faster learning.

***c) Recommendations for administrators: influence the academic culture***

Current undergraduate and graduate programs often miss the fundamental coding skills that research, technology, and the job market currently demand. Administrators are encouraged to evaluate these knowledge gaps and integrate basic programming courses into the current curriculum across all disciplines. However, this process is lengthy, and so, a series of actions can be taken in the meantime to ease the graduate experience. For instance, institutions can offer

seminars/workshops with the purpose of providing a computing skillset to those students who require additional training to carry out their research work.

The main challenge of short coding seminars is that instructors are not familiar with a student's background and, as a result, neglect to provide applicable resources for the integration of these skills. Thus, the students may fail at changing those computing skills from the workshops into a useful tool that can help them during their research. Follow-up sessions and additional advisory support are encouraged. Administrations should organize and operate a centralized framework to connect individuals and share code within their institutions.

*d) Recommendations for OSS community members: share the expertise on the software*

The OSS community represents the main resource that students access to when facing specific technical issues. As seen in Section 3.2.4, insufficient or incomplete documentation is a common barrier that students face, which may dissuade students from using these resources in the future. Aside from maintaining the code, the OSS community should work to improve the existing documentation, while making it comprehensive for all levels of expertise.

The main communication platform for any OSS is online, which introduces particular challenges. For example, most software users regularly consult online discussions that are disorganized and hard to understand. If an inexperienced user is looking for help online on a particular issue, and they are not able to fully understand the responses/solutions or even relate to their initial issue, there is a high chance that they will become discouraged and quit. OSS platforms are encouraged to classify the topics from low to high complexity in order for students or other novice users to answer their questions without feeling overwhelmed. Instructive solutions, as opposed to definite ones, create dialogue, transforming forums into a learning resource.

Additionally, it is important that the OSS community members review the graduate students' work in order to ensure that no bugs have been extended to other parts of the research, and the application has been correctly addressed. While the students benefit from the community's feedback, the community benefits by gaining new contributors to the software.

From the OSS community's perspective, creating a supportive community around graduate students is exceptionally significant, since they will likely remain contributors if their experience is positive.

### ***3.2.6. The Future of OSS in Academia***

This section has introduced an overview of the technical challenges that graduate students usually face when working with OSS. It has also provided a series of best practices for promoting a positive graduate student experience with OSS.

Over the last few years, the software community has started to identify another growing issue with its development: as OSS spreads, there is a concern associated with its sustainability (*Roads and Bridges: History and Background of Digital Infrastructure*, n.d.). Many software ecosystems are currently based on OSS. The community does not want these programs to disappear, but at the same time, there is a lack of contributors to these projects due to insufficient technical expertise and economic support.

The cost involved in the development of OSS is not generally appreciated, and only a few patrons finance certain projects (Crichton, 2018). The sustainability of OSS remains a problem and the software community must look for other alternatives, more contributors, and encourage graduate students to keep their code alive even after they graduate.

The education system is gradually adopting OSS for online learning initiatives and courses with the idea of initiating collaborations and building a community around software (Lakhan & Jhunjhunwala, 2008). In research, GitHub has become a popular tool for sharing free data. It uses version-control software to keep track of the changes that the code and data have had since its first appearance on the application (Perkel, 2016).

Overall, there is a lack of interest from the scientific community in this topic where little research has been done and the formalization of standards/guidelines remains uncertain after decades. OSS will be the leading platform for the new generation of but as of now, its framework is still undefined.

### **3.3. Technical challenges of JSBSim**

As an OSS FDM, JSBSim also presents particular challenges that will be addressed throughout this thesis. Over the years, JSBSim has been mostly used by enthusiasts who wanted to fly large aircraft in a simulated environment. When flying commercial aircraft models, the user does not need to have a deep understanding of software development methodologies, since they can access a large online aircraft library and fly any aircraft manually (JSBSim Development Team, 2018). The problem associated with this running mode is that JSBSim does not include a visualization environment and FlightGear must be used to provide the graphics of the simulation.

In situations where the user wants to run scripts for autonomous flight, the JSBSim manual is impractical (Berndt & JSBSim Development Team, 2011) since it does not have any tutorials that explain how to create and run the software in standalone mode. However, they provide a user-created tutorial on their website (Galbraith, 2010; JSBSim Development Team, 2017). This

document focuses on how to run simulations, but it lacks the guidelines on how to create the scripts for the simulations.

Since it is an open-source FDM, the package allows for modifications and new models to be integrated into simulations in standalone mode, or simply as part of FlightGear in manual mode. If the user wishes to visualize the model, graphics must go through FlightGear, since JSBSim does not support it. At this stage, users with aerospace knowledge are also required to understand software development basics in order to adjust and develop the required code for their particular cases.

Currently, there are no guidelines on how to create, develop and fine-tune new computer models. Furthermore, there are no recommendations on how to validate computer models in JSBSim.

Additionally, the creators claim that JSBSim can be run under MATLAB/Simulink (JSBSim Development Team, 2005b). Even though there is an FDM block that can be downloaded and integrated into Simulink (Gong, De Marco, & Berndt, n.d.), it is still under development and does not include the same features as the standalone running mode.

In order to overcome some of the challenges described above, this author has created a short user guide for the development of RPA computer models in JSBSim that is included in Appendix A and it has been shared with the JSBSim community as a technical report (Cereceda, 2019).

### **3.4. Summary**

The importance of simulators is well-known across disciplines not only limited to engineering. The FDM is the combination of physical and mathematical equations that express the dynamics of the aircraft in a simulator. For the RPA with SAA applications case, the focus of the

modelling and design of computer models is the aerodynamic and propulsion forces that are solved based on the general equations of motion.

With a larger variety of RPAs in the market nowadays, open simulators and OSS are merely the platform that allows for the implementation of computer models into existing software, since proprietary flight simulators do not enable the addition of new models. RPA integration also permits to evaluate their performance, as well as their interaction with other elements in the airspace.

Additionally, OSS presents particular challenges especially in academia and common to graduate students conducting research in all fields of science. Nowadays, research would not be possible without software. However, the lack of computing knowledge or community resources can easily discourage students from working with tools that require coding or OSS. To encourage a positive experience, a series of practices want to establish a framework for all levels of academia and OSS from which the graduate students and the OSS community can benefit. Even though these recommendations focus on the graduate student, they can also be applied to researchers in general.

As part of the solutions proposed in this thesis to overcome barriers associated with OSS, specific challenges related to JSBSim were evaluated and a simplified manual for the development and use of fixed-wing RPA computer models in JSBSim was created. This short manual aims to help any user who does not have a deep understanding of computer programming with the simulation of RPAs in JSBSim.

# Chapter 4

## *High-level Validation Approach for OSS*

### *FDMs: JSBSim Application*

Verification and Validation (V&V) techniques have wide use in engineering and other scientific fields, being the common goal to establish the credibility of a certain model or system in order to reduce risks for a specific task. Discrepancies might happen depending on the organization so it is important to define what is relevant to evaluate the feasibility of the simulation/model.

Finding a V&V method that could be applied to a dynamic model is a big part of its success. In aviation, regulations and recommended guides for commercial and military aircraft model validation are well known (Chapter 2 – Section 2.4) making the validation stage as relevant as the design of the model itself. Smaller vehicles such as Remotely Piloted Aircraft (RPAs) present a different case but the same scenario, and some of the steps in general aircraft validation could be adapted.

Based on the technical challenges of working with Open-Source Software (OSS) and with JSBSim in particular, this chapter: (1) evaluates the differences between piloted and remotely

piloted flight simulators, (2) lists the minimal requirements for RPA simulators, and (3) introduces a validation method based on existing V&V approaches for the development of fixed-wing RPA computer models.

The application of the validation method is presented with the modelling and development of the EPP FPV JSBSim FDM in a tutorial format. This case study was used as an application example of the extended validation method in paper currently submitted for review to a scientific journal whose initial procedure is included in (Cereceda, Rolland, & O'Young, 2019). In the following chapter, the Giant Big Stik computer model will be created and improved with the purpose of implementing it in scenarios with piloted aircraft for the study of avoidance manoeuvres.

For more details on JSBSim and how to run the package and integrate it into FlightGear, see Appendix A. This chapter aims to serve as a tutorial and, therefore, relevant pieces of code are added into this chapter. The complete code and configuration files can be found in Appendix D.

## **4.1. High-level validation approach for OSS FDMs**

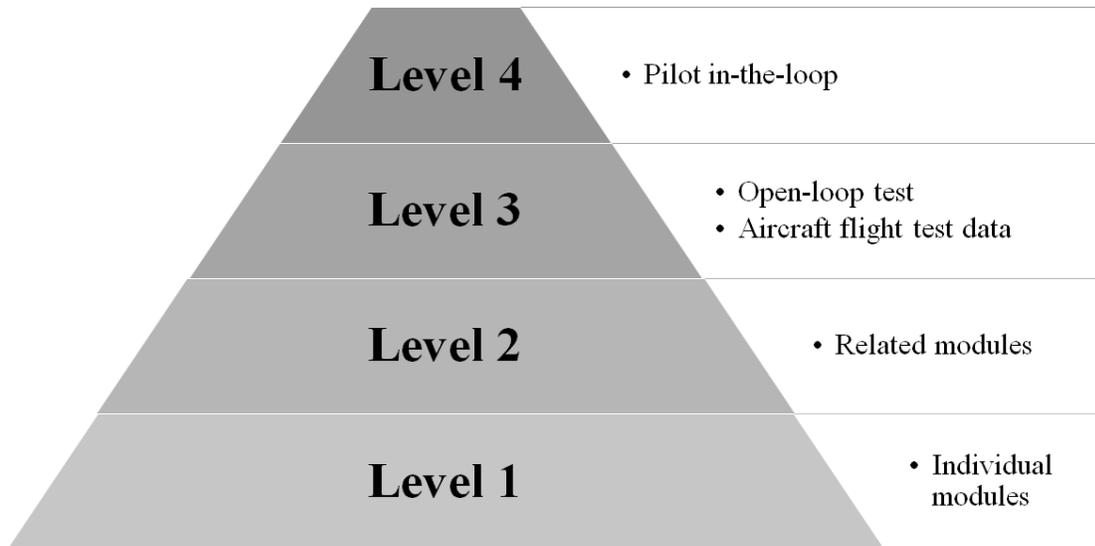
The technical difficulties associated with the incomplete and inactive JSBSim GUI (JSBSim Commander (Gong et al., n.d.)), led to a review of how RPA computer models are developed and validated. As pointed out in Chapter 3 (Section 3.1), when a computer model is designed, a V&V methodology is required in order to define the limitations and accuracy. This represents the real system in a simulated environment.

The purpose of this section is to define a validation process, composed by direct steps, with the intent to guide any developer who does not have a deep understanding of software development and validation methods in the design and improvement of an RPA computer model.

The work presented is an extended version of a previously presented work (Cereceda, Rolland, & O’Young, 2016) and is the initial stage for the study of RPA in simulated frameworks (Cereceda et al., 2019). A second case study with more specific SAA applications is shown in the following chapter where the model is integrated into NMAC scenarios.

**4.1.1. Validation of Aircraft Flight Simulators: Piloted vs. Remotely piloted Aircraft**

Traditionally, the validation task for a piloted aircraft flight simulator has been conducted by a pilot from the manufacturing company, a pilot from the corresponding regulating agency, or a military officer. Training has been the main application of flight simulators and the input from pilots has been a crucial element in the development, validation and documentation of these systems over the decades.



**Figure 4.1.** Levels of an aircraft flight simulation model validation

The AC 120-45A advisory circular, created by the FAA, defines the procedure for developing and validating flight simulators (Federal Aviation Administration, 2019). The equivalent in

Canada is regulated by the TP9685E (Transport Canada, 2010a). In those, flight simulators are validated on four levels (Figure 4.1). In this section, each of these levels are evaluated from the RPA's perspective and a series of minimal requirements for developing and validating RPA FDM as part of a flight simulator are listed.

In the first level, elements in the flight simulator are tested individually at a low level. Each element is evaluated on whether it satisfies the minimal requirements for that particular module (e.g. if the mathematical method is correctly solved). The second level evaluates the interaction of modules or packages. For example, the propulsion system with a propeller and an engine is evaluated individually in level 1, and on its interaction in input/output with the rest of the elements in the FDM in level 2.

At the third level, the aircraft is considered as a system of several systems as expressed in Chapter 3 (Section 3.1). Two test tools are used at this level: an open-loop test and aircraft flight test data. In the first test, the FDM is assessed in an open-loop for a series of generated signals (step, ramp and sine wave). Since the inputs are simulated and steady, the tests are reproducible and the dynamic response of the aircraft is easily monitored. In the second test, an additional program simulates aircraft flight data as an input to the mathematical aircraft model, so the FDM is driven by the same controls as the real aircraft. The flight control system can be validated by examining each of the controls in the aircraft, one by one. If the FDM has the same inputs as the aircraft flight controls, the aero surface deflection is the same, and the response of the simulator is equivalent.

The final at level 4 adds a pilot-in-the-loop. Although the mathematical model has been evaluated in previous levels, a pilot must approve a realistic simulated environment, including different visual elements and human limitations during manoeuvres.

In general terms, piloted flight simulators differ from remotely piloted simulators in three ways: (1) the role of the human pilot, (2) the application and (3) the instrumentation.

In piloted aviation, human pilots ensure the separation from other aircraft and the ground during the tasks of take-off and landing with no instrumentation, whereas their role is to supervise the control during level flight. However, R/C pilots can only operate the aircraft from the ground, limiting the flights to visual line-of-sight missions. Therefore, the simulation context where the human pilot is located onboard and controlling the aircraft is not extendable to RPAs. The onboard cameras that the pilot remotely monitors from the ground do not provide the same viewpoint, since there is no sense of depth and the perception is inaccurate. This means that the cockpit controllers (e.g. stick, throttles, rudder pedal, and brakes) are translated into the manual joystick that the pilot has on the ground and it is independent of a particular FDM. The cockpit instrumentation is not applicable to the RPA case. The aural system (used to generate engine sounds and wind noise) is not relevant since there is no human on board. With an RPA, the visual and motion systems are controlled or supervised by the ground control station.

Although the main purpose of flight simulators in both the piloted and the RPA cases, is training, there are also differences in the way modules are tested in flight simulators. While the pilot defines the fidelity of an improved element in the aircraft, quantitative test procedures are the relevant elements of the RPAs computer model validation. For example, Beyond Visual Line-of-Sight (BVLOS) missions with RPAs depend on the aircraft data collected in the ground station.

In this circumstance, the actual performance of the aircraft from a human's perspective is not relevant.

Considering the differences mentioned above and that RPA flight simulators can be considered as a subset of piloted aircraft flight simulators, the minimal requirements for RPA flight simulators are:

- 1 - Main modules: flight control, FDM, and visualization system.
- 2 - Main FDM modules: aerodynamics, propulsion, and equations of motion (Chapter 3 – Figure 3.2).
- 3 - Each of the modules must be tested individually.
- 4 - Additional modules must be flexible and must not affect the aircraft capabilities.
- 5 - Open-loop tests are required to evaluate the effect of the flight controls on the aircraft response.
- 6 - Aircraft flight test data as the primary validation procedure.

Concepts described in points 1, 3, 4, and 6 are also shared with piloted aircraft flight simulators, whereas number 5 is particular of RPAs and number 2 might change depending on the aircraft system and the class airspace.

The main difference between a piloted and an RPA simulator is the role of the pilot. Instead of acting as the approving figure, the pilot in an RPA flight simulator acts as a support to the validation quantitative procedures. This means that levels 3 and 4, as expressed in Figure 4.1, switch their positions in an RPA flight simulator. However, the expertise of the human pilot is recommended in each of the levels to assure that the computer model effectively represents the real system.

Even though a brief evaluation of the differences between piloted and remotely piloted flight simulators has been presented, along with a list of minimal requirements for RPA simulators, this topic has more aspects to consider and it remains as future work.

In the following subsections, a methodology is introduced based on the minimal requirements listed above. Supplementary elements, such as the manufacturer's flying recommendations and the comparison with a trusted FDM, are added to support the absence of the pilot-in-the-loop during the validation.

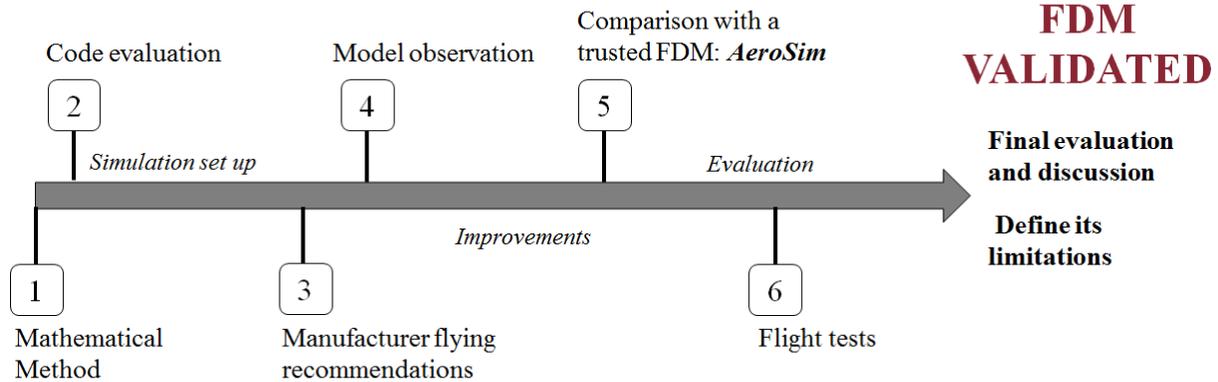
#### ***4.1.2. Initial Procedure for a Correct Validation***

Along with the computer model design, evaluation and validation procedures are required to define the limitations and accuracy in order to represent the original aircraft in a simulated environment. The following steps (Figure 4.2) specify the different aspects the designer must consider when the validation stage is reached. This section defines the minimum categories from which the proposed methodology is derived.

##### ***4.1.2.1. Code/software Evaluation***

This step reviews the computer platform where the mathematical model is expressed. There are many examples of available software for implementing the code like those previously mentioned in Chapter 2: MATLAB/Simulink, JSBSim or LaRCSim. At this point, the visual software should be considered as well since incompatibilities might happen between the code and the simulator. If needed, there is often the chance of sketching a new aircraft and implementing an FDM linked to it (only possible in OSS frameworks). The designer has to assess the

requirements of the task and the final purpose of the model in order to choose the suitable software.



**Figure 4.2.** Stages of a correct validation

#### 4.1.2.2. *Flying Recommendations*

Manufacturers provide indications for first users and test flights. In general and commercial aircraft, the Pilot's Operating Handbook (POH) is a guide for the pilot that contains limitations, procedures, performance and other useful information related to the aircraft. Similar to this document, small fixed-wing manufacturers sometimes provide an instruction manual that includes flying recommendations. Although it is actually a rough set of suggestions for the tasks of take-off, landing and straight flight, following these instructions gives a first representation of the computer model performance and similarity to the real aircraft.

#### 4.1.2.3. *Model Observation*

Not every aircraft belonging to the same series reacts equally. Even though the model performs in a similar manner to the manufacturer recommendations, specific procedures might produce a better performance; this non-official concept comes from the experience of a pilot. Given a flight

controller and a visual simulator linked to the FDM, the system is tested by a pilot. The tasks are mainly to lead the aircraft to its control limits, and based on its execution, the pilot gives useful feedback about its performance and possible improvements.

#### *4.1.2.4. Comparison with a Trusted FDM Model*

When working with simulators, there is a noticeable difference depending on the software used; in this document, the JSBSim FDM is validated against another reliable FDM (MATLAB/Simulink - AeroSim toolkit). Two identical computer simulations, one for each FDM, are configured where both models are exposed to the same conditions/inputs and compared. When the dynamics are evaluated, the airspeed and the Euler angles, which express orientation, are essential to acquiring valuable conclusions in this particular RPA scenario.

Note that even though the model to be validated and the reference model of choice derive from the same physic principles, differences are expected based on the simulation platform and additional protection systems that could be reflected in the performance. This step prior to validating the model against flight data is recommended since it minimizes the error towards an acceptable level in the following test. However, in cases where there is no possibility of accessing a reference computer model, this step could be skipped.

#### *4.1.2.5. Experimental Test*

In a real flight, data are collected from a planned mission and compared to the data under the same conditions for the FDM designed. The inputs to the real model must be recorded as well in order to simulate them in the computer simulation.

### 4.1.3. Validation Methodology for the FDM Development

The previous section framed the important elements to address during the validation. The validation methodology used in this work is presented below in two main stages: the first stage being related to the modelling and requirements definition and the second to the validation itself. The main difference between this validation methodology and the existing methods is that the final computer model is particular to a pre-defined task.

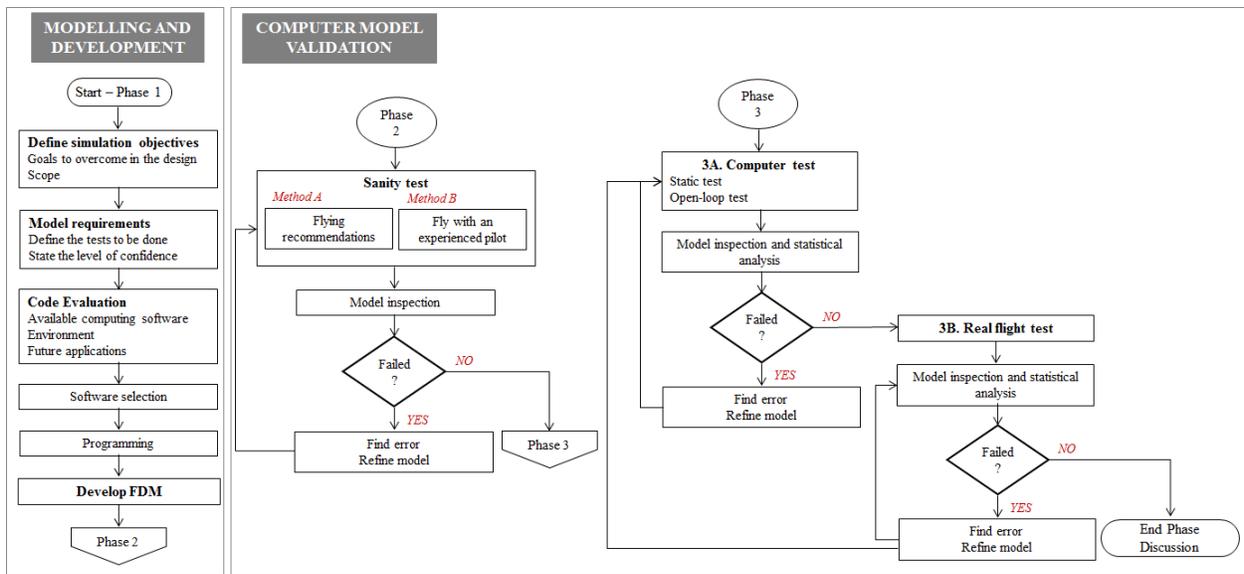


Figure 4.3. Validation methodology for RPA FDM

#### 4.1.3.1. Modelling and Development

First, the simulation objectives are defined and listed as part of the final goals that the entire system needs to achieve. The baseline of the model development is the formulation of its requirements from the specification, which will define the decisions to take in future steps during the design process. According to the simulation goals, at this stage (Phase 1 -Figure 4.3), the tests in later stages should be defined and the level of confidence determined.

The computer platform where the mathematical model is expressed must be reviewed as well. There are many examples of existing software suites such as open-source visualization software and build FDM packages that help with the RPA design process. Open source visualization software exists and can be linked to the FDM software. This initial phase is open to the model performance criteria and the future application of the simulation. By the end of this evaluation, the designer is expected to have assessed the task requirements, the final purpose of the model, and have designated the most suitable software for the desired application.

#### *4.1.3.2. Computer Model Validation*

The second phase provides the sanity test as the first approach to the computer validation. Two methods are suggested:

- A. Flying recommendations
- B. Experienced pilot's test

Small fixed-wing manufacturers provide an instruction manual that includes flying recommendations similar to the POH for piloted aircraft (Method A). Although it is in fact a rough set of suggestions for the tasks of take-off, landing, and straight flight, following these instructions provides an estimated approach to the computer model performance. Additional flight procedures might improve performance, and this comes from the test pilot's experience. In Method B, given a remote control (R/C) controller and a visual simulator linked to the FDM, a pilot tests the system and gives feedback about its performance and possible improvements.

Although this second stage is valuable, no quantitative data are present; the verification test in both methods is done by user inspection. The qualitative results in Phase 3 should be able to confirm whether the model fulfils the requirements. In case the computer model fails the

inspection test, the sanity test in Phase 2 must be done iteratively until the model passes the sanity test.

Thus far, the RPA simulation model has been fine-tuned as far as possible. In order to closely evaluate its performance, a standard model is used as a reference in two scenarios: a reference computer model (Phase 3A) and the real RPA (Phase 3B). Note that Phase 3 directly relies on the specifications defined in Phase 1.

In the computer test (Phase 3A), two identical computer simulations, one for each FDM, are configured so both models are exposed to the same conditions/inputs and compared. These simulations should be divided into two main categories: a static test and an open-loop dynamic test. Starting with a static test, the aircraft is left to glide without any command. This test is useful when analyzing how the model performs in a situation where there are no perturbations to the system. In the second test, the aircraft characteristics and performance are evaluated for simple primary control inputs. The isolation of the inputs one at a time is crucial to determine the stability of the aircraft in the corresponding axis and towards understanding the effects of disturbances. The open-loop dynamic procedure is divided into three tests corresponding to the surface deflections: an elevator test, an aileron test, and a rudder test. By the end of each test in Phase 3A, the similarities of both computer models are evaluated following inspection and statistical analysis tools (Goldberg et al., 1994).

In the real flight test (Phase 3B), data generated by multiple inputs are collected from a task mission and compared to the simulation data from the FDM under the same conditions. The computer model vs the real model response is evaluated by inspection and statistical analysis similarly to the computer test in Phase 3A.

Particular observations in Phases 2 or 3 might have more relevance than others in order to define the accuracy and reliability of the computer model. The end phase evaluates the conclusions and the validation procedure all together in order to define the limitations of the computer model

This methodology expands, step by step, the different aspects of the design of the computer model. It evolves from the simplest approach in Phase 2 to the final evaluation using statistical and analytical tools. This allows the designer to identify the error and refine the model at any time during the design process.

The unpredictable atmospheric conditions and disturbances must be taken into account in order to select the valuable data from the open-loop flight test and the level of accuracy in the computer model inspection.

## **4.2. Simulation Assumptions for the Case Studies**

This thesis focuses on the implications of OSS on RPAs with aviation applications. Therefore, particular assumptions related to the aerodynamics model were considered in order to frame the research. The author is aware of the limitations that this brings and hopes that it encourages future work from the fluid mechanics and system dynamics' perspective. The following assumptions were considered:

1. The aerodynamics model in JSBSim was not questioned and it was assumed that it follows the physical principles defined in Section 2.3 – Chapter 2.
2. The deflection input signals for testing the models in Phase 3A were pulses/steps; this scenario is only possible in simulated frameworks.

3. Particular cases consequence of extreme manoeuvres were not taken into account.
4. Diving manoeuvres are particular to aerobatic and military aircraft.
5. The airflow is assumed to be steady, incompressible, and the friction by viscosity is assumed to be negligible in order to apply Bernoulli's question.

### **4.3. EPP FPV Case Study: JSBSim Tutorial**

The EPP FPV vehicle is an Expanded Polypropylene (EPP) foam RPA manufactured and sold by HobbyKing.com (HobbyKing.com, n.d.). It is primarily designed for First Person View (FPV) camera flight. The expanded polypropylene foam makes it light (perfect for gliding) and robust during landings and crashes, making it perfect for beginners. The thrust is provided by a propeller-driven by a DC electric motor. The propulsion system is located in the rear of the fuselage to allow for electronic equipment and batteries installation in the front of the aircraft.

#### ***4.3.1. EPP FPV Simulation Context***

The model designed and validated in this section serves as the example for the validation procedure with a focus on Collision Avoidance (CA) in Near Mid-air Collisions (NMAC). The application is related to the context of this research although this procedure is adaptable to any RPA application. Following the diagram showed in Figure 4.3, the context of the task is defined as Phase 1 and can be summarized as:

- **Simulation objectives:** The model must be able to perform critical manoeuvres. Extreme performance is only applicable for a computer test and in order to validate the model using real flight data, they are changed to small and medium signals due to the concern

associated to flying large signals on the field. Thus, the scope of the tests covers the entire range of the deflections.

- **Model requirements:** In Phase 3A, aside from the static test, large signals are simulated in the elevator, ailerons, and rudder for the open-loop computer test whereas, in Phase 3B, small/medium signals are tested. The system is evaluated by inspection and statistical analysis with a 95% confidence level.
- **Code evaluation, programming, and development of the Flight Dynamics Model (FDM):** FlightGear provides the visualization platform in Phases 2 and 3. It supports JSBSim and instead of running the model as an external system, the aircraft is integrated into FlightGear for Phase 2. The AeroSim FDM (Unmanned Dynamics, 2006) run in MATLAB/Simulink is the reference model used to validate EPP FPV JSBSim FDM in Phase 3A.

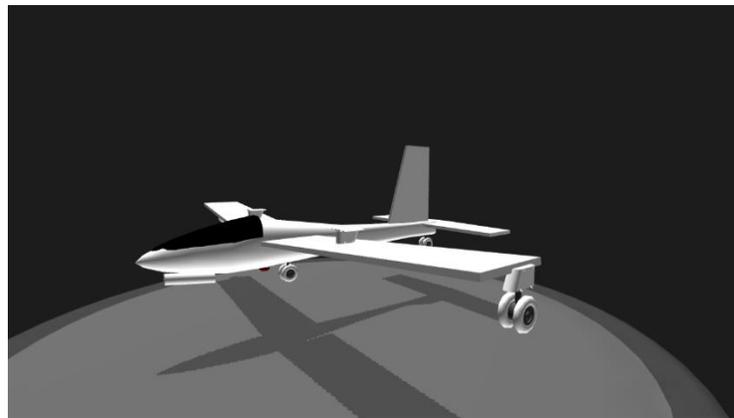
The wind is also a disturbance that must be taken into consideration as its influence in small aircraft is more significant than in larger aircraft. One alternative when addressing this issue is to model the steady wind and wind gusts and add them to the computer model as disturbances. However, the wind does not affect how the validation is carried out or the results of it, and therefore, it is not evaluated in this chapter or thesis.

#### ***4.3.2. EPP FPV Computer Modelling and Development***

The modelling process starts with the parameters, coefficients and metrics identification for the design of the RPA (Table 4.1 and Table 4.2). The manufacturer often provides information on their manual and in case certain metrics are missing, they can be measured from the real model

(Appendix B). However, the aerodynamic coefficients remain unavailable and with that in consideration, JSBSim offers an online tool (Aeromatic v2.0 (JSBSim Development Team, 2005a)) that computes the aerodynamic coefficients based on metrics and other features.

The initial aerodynamic coefficients for the EPP FPV model were taken from a similar aircraft called “mini sgs glider” (Figure 4.4), found in the online JSBSim aircraft library (“JSBSim Flight Dynamics Model - Code aircraft/minisgs,” 2016).



**Figure 4.4.** Mini SGS-126 Glider (SimplePlanes, 2019)

During the development of the model, some of the coefficients have been modified from the initial version as part of its tuning. During the validation process, the model was adjusted according to (1) the suggested coefficients given by Aeromatic v2.0 based on the aircraft metrics, and (2) the differences and similitudes found between the computer and the reference models in Phase 3A and between the computer and real models in Phase 3B. The most common identified differences were offsets due to previous no neutral states, gains consequential of the inputs range, and delays due to real elements in the equipment. The tuning was carried out accordingly by adding gains in the control commands and delays in the aero surfaces.

**Table 4.1.** EPP FPV Parameters

<metrics>		<mass_balance>	
Wing area (ft2)	4.24	Ixx (Slug*ft2)	0.05034
Wing span (ft)	5.90551	Iyy (Slug*ft2)	0.01701
Chrod (ft)	0.7217848	Izz (Slug*ft2)	0.12940
H tail area (ft2)	0.7	Ixy (Slug*ft2)	0
H tail arm (ft)	2.723097	Ixz (Slug*ft2)	0.00911
V tail area (ft2)	0.23	Iyz (Slug*ft2)	0
V tail arm (ft)	2.559055	Empty weight (lbs)	4.4
AERORP (in)	[13.9764, 0, 3.937]	CG (in)	[16.54, 0, 0]

**Table 4.2.** EPP FPV aerodynamic coefficients

Lift Coefficient			Drag Coefficient			Side Coefficient	Roll Coefficient	Pitch Coefficient	Yaw Coefficient
$C_{L0}$	0.0		$C_{D0}$	0.0007				$C_{m0}$	0.102
$C_L^\alpha$	$\alpha$	$C_L^\alpha$	$C_D^\alpha$	$\alpha$	$C_D^\alpha$			$C_m^\alpha$	-1.573
	-0.1571	0.0		-0.0175	0.01				
	-0.1369	0.06		0.0	0.015				
	...	...		...	...				
	1.3963	0.26		1.3963	1.5				
	1.5708	0.03		1.5708	1.46				
								$C_m^{\dot{\alpha}}$	-5.2
						$C_Y^\beta$	-0.83	$C_l^\beta$	-0.0313
								$C_l^p$	-0.47
								$C_m^q$	-9.0
								$C_l^r$	0.15
$C_L^\delta$	Elevator	-0.3420	$C_D^\delta$	$\delta_e$	$C_D^{\delta_e}$	$C_Y^\delta$	0.0	$C_l^\delta$	0.0
				-1.0	0.114			$C_m^\delta$	-1.261
				0.0	0.0				
				1.0	0.114				
	Aileron	0.0		0.0			-0.0456		0.0
	Rudder	0.0		0.0			0.1880		0.0
									0.0115
									-0.037

The information provided by the manufacturer about the range of the control surfaces for the EPP FPV is approximate. Our pilots carried out flight trials to identify the operational limits and have confirmed that in real flight, the aircraft is unlikely to fly under extreme conditions. Thus, the deflection ranges have been corrected as recommended (Table 4.3).

Taking the information given by the tables into account, the aircraft configuration file is built along with the engine and propeller files (Appendix D in sections D.2.1., D.3.1. and D.4.1. respectively)

**Table 4.3.** Control surface deflections range

Control surface	Manufacturer		Pilot Recommendations	
	$\delta_{min}$	$\delta_{max}$	$\delta_{min}$	$\delta_{max}$
Elevator	-20° (-0.3491rad)	20° (0.3491rad)	-15° (-0.2618rad)	15° (0.2618rad)
Aileron	-25° (-0.4363rad)	25° (0.4363rad)	-20° (-0.3491rad)	20° (0.3491rad)
Rudder	-20° (-0.3491rad)	20° (0.3491rad)	-15° (-0.2618rad)	15° (0.2618rad)

### 4.3.3. *EPP FPV Computer Model Validation*

The validation stage of the fixed-wing aircraft computer model starts with Phase 2 (Figure 4.3). In this phase, the two suggested evaluation methods are carried out simultaneously. First, the manufacturer's manual was consulted (HobbyKing.com, n.d.) and unfortunately, it was observed that the flying recommendations did not give relevant information. However, as the EPP FPV is a widely used R/C aircraft, different forums provided reviews from R/C pilots (RCGroups, 2017) that were useful to obtain general information on the aircraft's performance. According to their comments, the rudder should be used in order to compensate the turns during roll manoeuvres.

At the beginning of the manual simulation in FlightGear, the EPP FPV was hard to operate due to a shift in control that affected its stability. Therefore, establishing those initial conditions is crucial for the pilot to have enough time to control the aircraft in the simulation. After these initial moments, the model was quite sensitive to the change of the commands as expected.

As described in Phase 3 and specified in the context, the FDM in this phase is run in two scenarios: (1) extreme manoeuvre scenario where large input signals are used to test the SAA capabilities and (2) small input signals under the same conditions as in a real flight mission.

### 4.3.3.1. EPP FPV Phase 3A: Computer Test

The reference model needs to be addressed at this point. The EPP FPV fixed-wing RPA implemented in MATLAB/Simulink using the AeroSim toolkit is used as the reference model in this research (Figure 4.5)<sup>1</sup>. In this case study, a simple simulation in AeroSim includes a set of simulated control inputs and an FDM block expressing the RPA dynamics. With a FlightGear interface block, the visualization of the performance is optional but recommended.

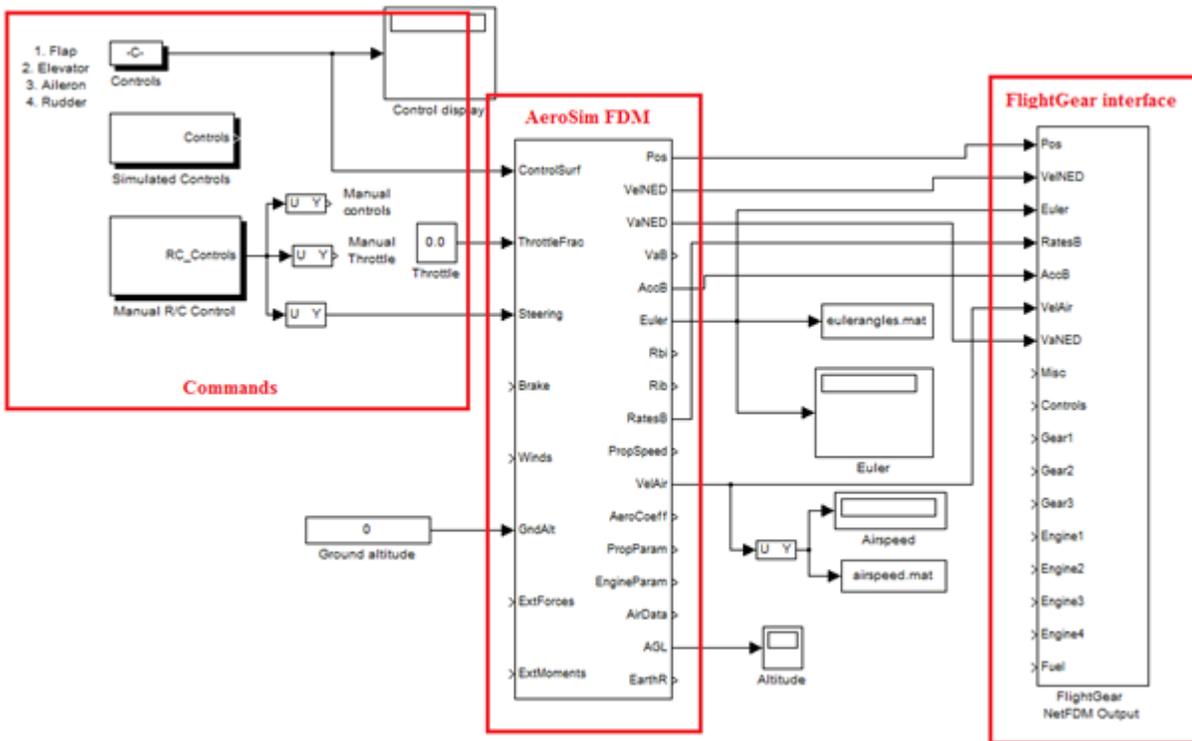


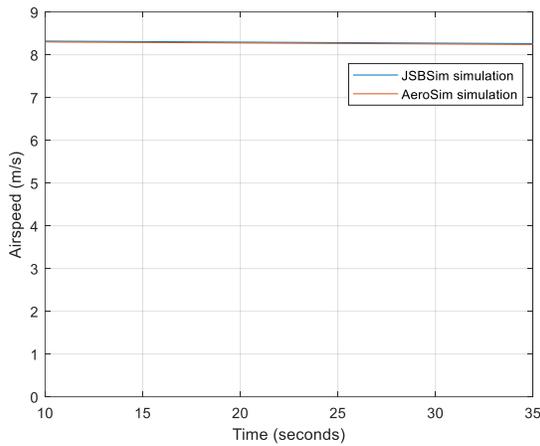
Figure 4.5. Open-loop AeroSim layout in MATLAB/Simulink

Assuming that the FDM in JSBSim has already been designed following Section 4.3.2, and with the help of Appendix A, the analysis starts with the **static test**.

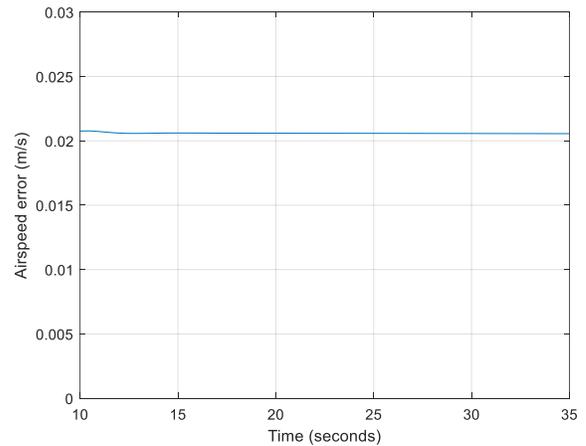
<sup>1</sup>The reader may consider another reference computer model.

Note that the initial conditions –location and airspeed- must be the same on both platforms and all simulations last 35s. The performance evaluation is given by the Euler angles  $(\phi, \theta, \psi)$ , airspeed ( $V_a$ ) and other relevant properties (depending on the final application).

In the static test, with no inputs, the airspeed is evaluated for both computer models.



**Figure 4.6.** EPP FPV static test. Airspeed ( $V_a$ )



**Figure 4.7.** Airspeed error

As shown in Figure 4.6, both models perform likewise in the static test. The difference in airspeed between the models (Figure 4.7) shows a constant value of  $\sim 0.02$  m/s (0.25% error) during the full simulation<sup>2</sup>. This error is nearly non-existent and consequently, both models are considered equivalent under the static test.

In the open-loop test, the system, represented by the FDM, is assessed for input combinations of the flight control surfaces: elevator, ailerons, and rudder with no closed-loop feedback. Similar to the static test, the first seconds of the simulation correspond to the transient response of the system, which were removed from the static test.

---

<sup>2</sup>The beginning of the simulation is neglected since the results show the transient response from the initial conditions to the simulation system steady state and are not relevant.

Following the requirements specified in Section 4.3.2 and Table 4.3, the **elevator test** runscript for the JSBSim simulation is:

```
<?xml version="1.0" encoding="utf-8"?>
<?xml-stylesheet type="text/xsl" href="http://jsbsim.sf.net/JSBSimScript.xsl"?>
<runscript xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://jsbsim.sf.net/JSBSimScript.xsd"
  name="Initial test">

  <description>
    EPP-FPV Elevator test
  </description>

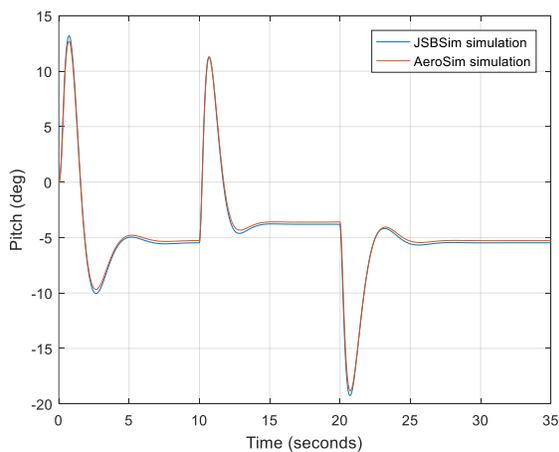
  <use aircraft="EPPFPV" initialize="ini00"/>
  <run start="0" end="35" dt="0.01">

    <event name="Set engine throttle">
    </event>

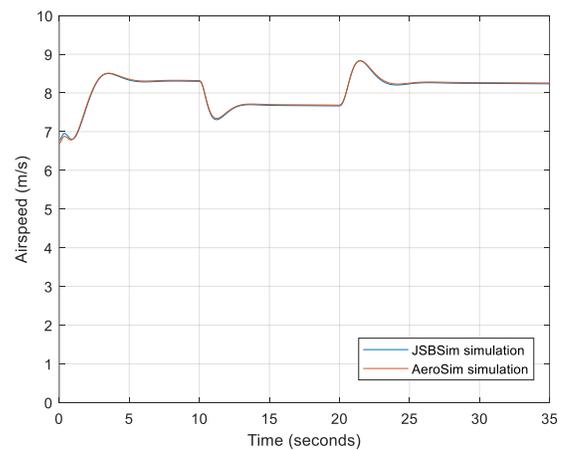
    <event name="Set elevator max">
      <condition>simulation/sim-time-sec ge 10.0</condition>
      <set name="fcs/elevator-cmd-norm" action="FG_STEP" value="- 0.174533
tc="0.1"/>
      <notify/>
    </event>

    <event name="Set elevator back to zero">
      <condition>simulation/sim-time-sec ge 25.0</condition>
      <set name="fcs/elevator-cmd-norm" action="FG_STEP" value="0.0" tc="0.1"/>
      <notify/>
    </event>

  </run>
</runscript>
```



**Figure 4.8.** EPP FPV elevator test. Pitch angle

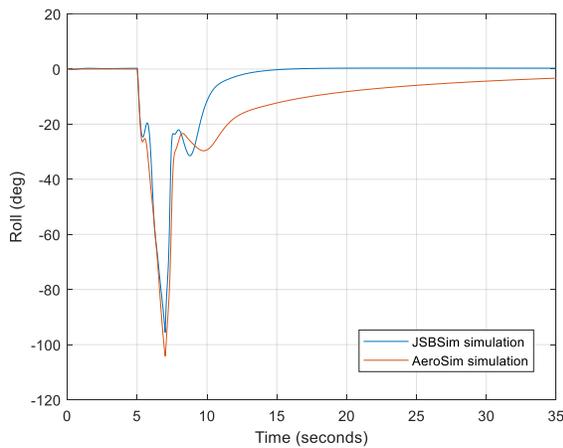


**Figure 4.9.** EPP FPV elevator test. Airspeed

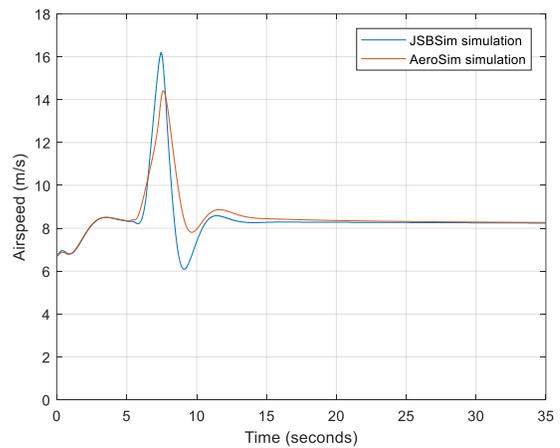
According to the forces and moments produced, when there is a negative elevator deflection, the angle of attack is greater producing an increment in the pitch angle (Figure 4.8). The coherence of the performance is also demonstrated with the airspeed response (Figure 4.9): when the aircraft angle of attack increases due to a downward lift created by a negative elevator deflection, the nose pitches up and the airspeed decreases; the opposite effect occurs when the elevator deflection is positive. This behaviour indicates that both models follow the same sign convention and their performance is similar in terms of airspeed when the elevator changes come from a steady situation. The initial seconds of the simulation show that both FDMs have positive static stability in pitch; the aircraft tends to return to a stable state.

The maximum error produced in pitch represents 1% of the response and the error in the airspeed represents a 2.5% with a maximum of 5%. With a low discrepancy between models, they are considered equivalent.

For the **aileron test**, a maximum deflection (limited by the pilot's recommendation) is sent as the input to the model going back to zero, 2s after:



**Figure 4.10.** EPP FPV aileron test. Roll angle



**Figure 4.11.** EPP FPV aileron test. Airspeed

```

<?xml version="1.0" encoding="utf-8"?>
<?xml-stylesheet type="text/xsl" href="http://jsbsim.sf.net/JSBSimScript.xsl"?>
<runscript xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://jsbsim.sf.net/JSBSimScript.xsd"
  name="Initial test">

  <description>
    EPP-FPV Aileron test
  </description>

  <use aircraft="EPPFPV" initialize="ini00"/>
  <run start="0" end="35" dt="0.01">

    <event name="Set engine throttle">
    </event>

    <event name="Set elevator max">
      <condition>simulation/sim-time-sec ge 5.0</condition>
      <set name="fcs/aileron-cmd-norm" action="FG_STEP" value="- 0.349066"
tc="0.1"/>
      <notify/>
    </event>

    <event name="Set elevator back to zero">
      <condition>simulation/sim-time-sec ge 7.0</condition>
      <set name="fcs/aileron-cmd-norm" action="FG_STEP" value="0.0" tc="0.1"/>
      <notify/>
    </event>

  </run>
</runscript>

```

When banking due to a turning manoeuvre caused by the ailerons, the aircraft pitches down due to a general loss in lift. As a consequence, the RPA airspeed increases because of the glide. This flight performance is reflected in Figure 4.11 supporting the coherence of the system from the physical point of view; the airspeed increases due to a turn and comes back to its initial state when the ailerons return to zero.

Two aspects are observed in both responses (excluding differences from the mathematical perspective):

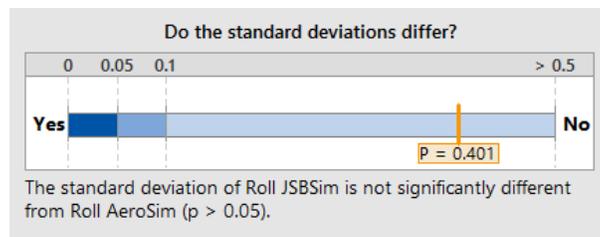
- JSBSim has greater positive dynamic stability in roll (Figure 4.10), meaning that the ability to control the aircraft is also greater.

- The airspeed reached by the JSBSim model is approximately 15% more than the airspeed in the AeroSim simulation. This indicates that the JSBSim model is 15% faster in performance (Figure 4.11).

When the FDM recovers from the turn, as shown in roll Figure 4.10, JSBSim comes back to its initial state in 10s and as a result, the airspeed reached when recovering is 2m/s greater than the AeroSim model as noted in Figure 4.11. Both models are solved by using the first principle of Newton’s second law, and any differences are assumed to be a consequence of the simulation solving method (syntax error).

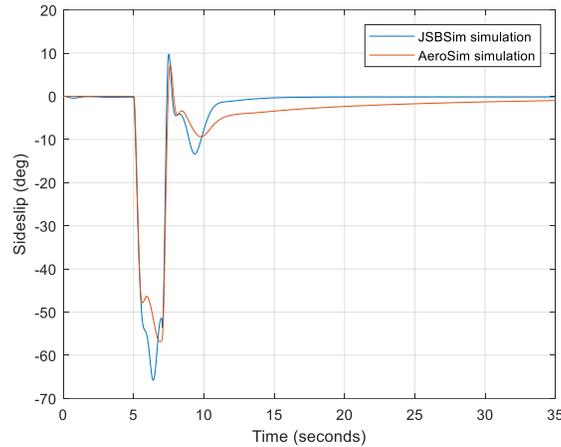
The recovery transition depends on the FDM itself whereas in this test the attitude response is evaluated. Therefore, in order to conclude whether both models are similar under the aileron test, a statistical analysis is done for the relevant time interval between 5 and 7s.

Using Minitab 17 (“Minitab,” 2019) as the statistical software and using the two-sample standard deviation test ( $\sigma/\sigma$ ) to evaluate how the data is spread, it can be concluded that, for Bonett’s method with a 95% confidence level, the JSBSim and AeroSim roll responses are equally spread and not significantly different (Figure 4.12).



**Figure 4.12.** 2-Sample standard deviation test for roll. Aileron test in the interval [5, 7)

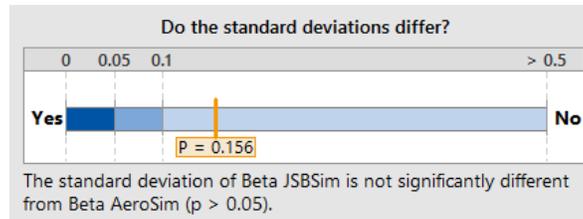
As stated by the pilots in Phase 2, changes in the ailerons will need the support of the rudder in a real flight. Thus, the sideslip angle is also evaluated in order to discuss the effects of the aileron deflection on the lateral stability.



**Figure 4.13.** EPP FPV aileron test. Sideslip

According to the results shown in Figure 4.13, there is a significant effect of roll on the sideslip motion, being more noticeable in the JSBSim case. Most aircraft can perform a smooth turn using ailerons alone but it varies depending on the aircraft. This special fact supports the comments from the pilots on the suggested use of a combination of ailerons and rudder for turning manoeuvres.

From examining the  $\beta$  time response (Figure 4.13) it is difficult to confirm whether both FDM could be considered equivalent. However, a further statistical analysis for the interval [5,8) in Figure 4.14 shows that both responses are similar with 95% of confidence level. This means that the coupling effect that exists between yaw and roll is the same or has the same effect in both FDMs.



**Figure 4.14.** 2-Sample Standard Deviation Test for  $\beta$ . Aileron test in the interval [5, 7)

An unnecessary use of the rudder control might generate a skidding turn resulting in an excessive sideslip. In most fixed-wing RPAs, the turns and changes in heading are usually controlled by the ailerons alone due to the cross-coupling effect between yaw and roll. But despite that, a rudder test cannot be ignored since the pilots commented on the RPA needing a rudder support when banking and this effect has also been shown in the aileron test.

As pointed out in Table 4.3, the **rudder test** in the computer simulation is defined as:

```
<?xml version="1.0" encoding="utf-8"?>
<?xml-stylesheet type="text/xsl" href="http://jsbsim.sf.net/JSBSimScript.xsl"?>
<runscript xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://jsbsim.sf.net/JSBSimScript.xsd"
  name="Initial test">

  <description>
    EPP-FPV Rudder test
  </description>

  <use aircraft="EPPFPV" initialize="ini00"/>
  <run start="0" end="35" dt="0.01">

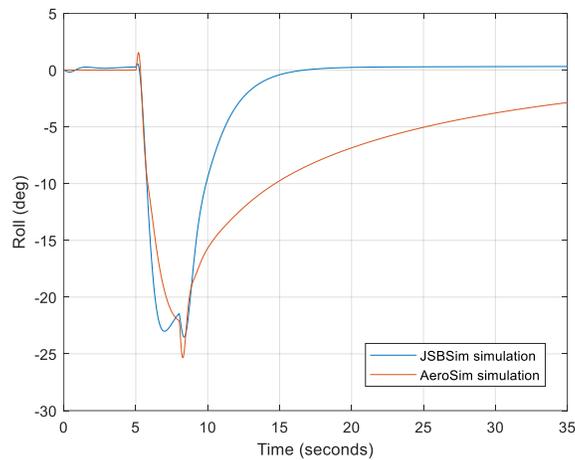
    <event name="Set engine throttle">
      </event>

      <event name="Set rudder">
        <condition>simulation/sim-time-sec ge 5.0</condition>
        <set name="fcs/rudder-cmd-norm" action="FG_STEP" value="0.261799" tc="0.1"/>
        </set>
        <notify/>
      </event>

      <event name="Set rudder back to zero">
        <condition>simulation/sim-time-sec ge 8.0</condition>
        <set name="fcs/rudder-cmd-norm" action="FG_STEP" value="0.0" tc="0.1"/>
        <notify/>
      </event>

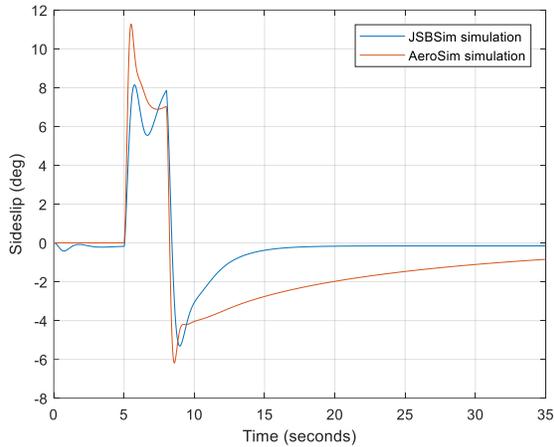
    </run>
  </runscript>
```

When the command is set to its maximum deflection, both roll ( $\phi$ ) and pitch ( $\theta$ ) angles do not add extra relevant information to previous tests. For example, in the roll angle response (Figure 4.15), JSBSim shows a greater stability in roll axis similar to Figure 4.10 in the aileron test. Positive static stability is preferable when controlling the aircraft and no modifications have been done until Phase 3B when flight tests will reject or accept this performance.

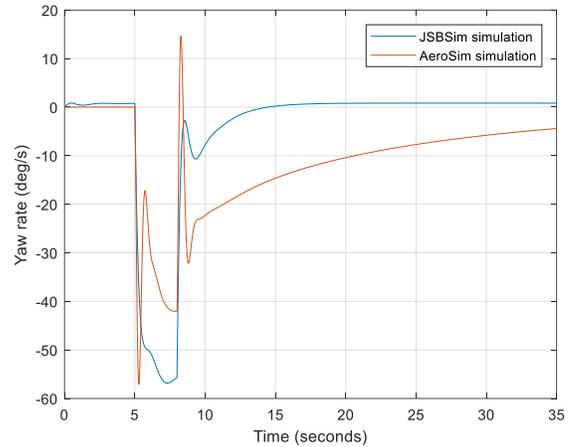


**Figure 4.15.** EPP FPV rudder test. Roll angle

Changes in rudder angle lead to changes in heading and the effects can be examined in the sideslip angle ( $\beta$ ); a constant heading means no sideslip whereas a direction change shows a positive or negative sideslip angle. Following the same concept, the yaw rate ( $r$ ) gives the angular velocity in the horizontal axis, which expresses the rate of change of the heading. Analyzing  $r$  and  $\beta$  provides a more accurate visualization of the effects of the rudder in the horizontal axis and heading.



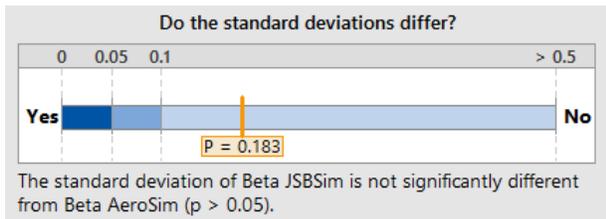
**Figure 4.16.** EPP FPV rudder test. Sideslip



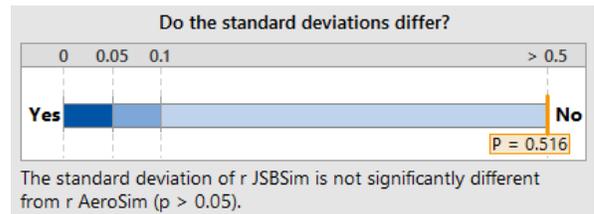
**Figure 4.17.** EPP FPV rudder test. Yaw rate

Figure 4.16 and Figure 4.17 show an appreciable difference between both responses, especially when the rudder is set to a maximum value. However, a difference in the error does not mean that both responses are not similar and a more detailed statistical analysis is needed in order to confirm this fact.

From both properties ( $\beta$ ,  $r$ ) in Figure 4.18 and Figure 4.19, it can be confirmed that, at 95% confidence level, the lateral response of both FDMs is not significantly different in terms of its data distribution and, therefore, JSBSim and AeroSim are considered equivalent under the rudder test.



**Figure 4.18.** 2-Sample standard deviation test for  $\beta$ . Rudder test in the interval [5, 8)



**Figure 4.19.** 2-Sample standard deviation test for yaw rate. Rudder test in the interval [5, 8)

#### 4.3.3.2. Phase 3: Flight Test

For this test, an EPP FPV RPA was flown following the current regulations established by Transport Canada (TC) (Transport Canada, 2018a) in the allowed areas surrounding St. John's, Newfoundland and Labrador (NL), Canada, in Class G airspace. The mission included an R/C controller, a Ground Control Station (GCS), an EPP FPV R/C and an onboard autopilot (ArduPilot, APM 2.6 ("ArduPilot documentation," 2017)) for recording the aircraft performance during flight. Considerations on this test included:

- The tasks of taking off and landing are removed from the analysis since they do not give significant information on a regular flight.
- The turns were only commanded by the ailerons leaving the rudder out from the analysis.
- The throttle control is ~50% during regular flight and assumed to be of that value for the entire Phase 3B of the EPP FPV case study.
- The output generated by the autopilot during the flight test includes the airspeed ( $V_a$ ), the Euler angles roll ( $\phi$ ), pitch ( $\theta$ ) and yaw ( $\psi$ ), the linear ( $\dot{u}, \dot{v}, \dot{w}$ ) and angular ( $p, q, r$ ) rates, and the location in terms of latitude, longitude and altitude.

The full flight mission has been narrowed down to a section where small and medium signals have been recorded as inputs to the computer model (Figure 4.20). The goal of adjusting the model for a particular range of inputs is to have the most accurate possible computer model for a specific task defined in the context. The selected simulation lasts 14s with an initial model transition of 5s expressed in the script file below.

```

<?xml version="1.0" encoding="utf-8"?>
<?xml-stylesheet type="text/xsl" href="http://jsbsim.sf.net/JSBSimScript.xsl"?>
<runscript xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://jsbsim.sf.net/JSBSimScript.xsd"
  name="">

  <description>
    Flight test - small and madium signals
  </description>

  <use aircraft="EPPFPV" initialize="inis2"/>
  <run start="0" end="14" dt="0.02">

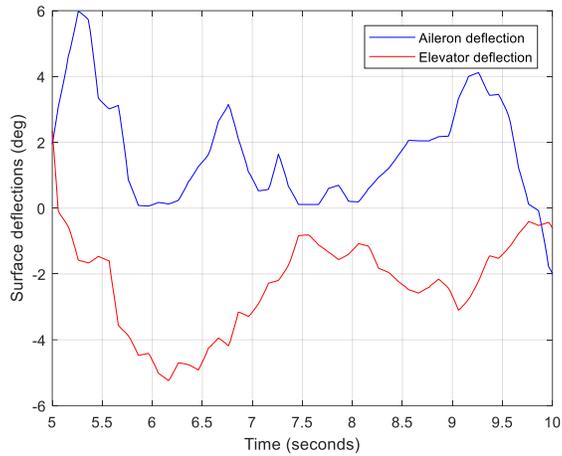
    <event name="Set engine throttle">
      <condition>simulation/sim-time-sec ge 0.5</condition>
      <set name="fcs/throttle-cmd-norm" action="FG_RAMP" value="0.0" tc="5.0"/>
      <set name="fcs/elevator-cmd-norm" action="FG_RAMP" value="0.006318266"
tc="5.0"/>
      <set name="fcs/aileron-cmd-norm" action="FG_RAMP" value="-0.05624498"
tc="5.0"/>
      <notify/>
    </event>

    <event name="Set the inputs from file" continuous="true">
      <condition>simulation/sim-time-sec ge 5.0</condition>
      <set name="fcs/elevator-cmd-norm">
        <function>
          <product>
            <value>0.5</value>
            <table>
              <independentVar lookup="row"> simulation/sim-time-sec
</independentVar>
              <tableData>
                5      0.006318266
                5.02  0.00356658
                ...
                13.98 -0.002128429
                14    -0.002128429
              </tableData>
            </table>
          </product>
        </function>
      </set>

      <set name="fcs/aileron-cmd-norm">
        <function>
          <table>
            <independentVar lookup="row"> simulation/sim-time-sec
</independentVar>
            <tableData>
              5      -0.05624498
              5.02  -0.058899626
              ...
              13.98  0.00619941
              14    0.005185199
            </tableData>
          </table>
        </function>
      </set>
    </event>

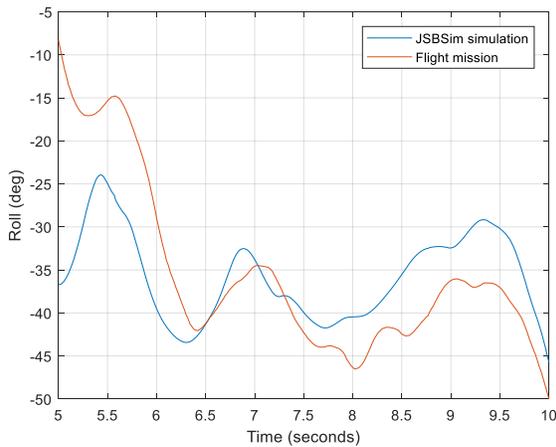
  </run>
</runscript>

```

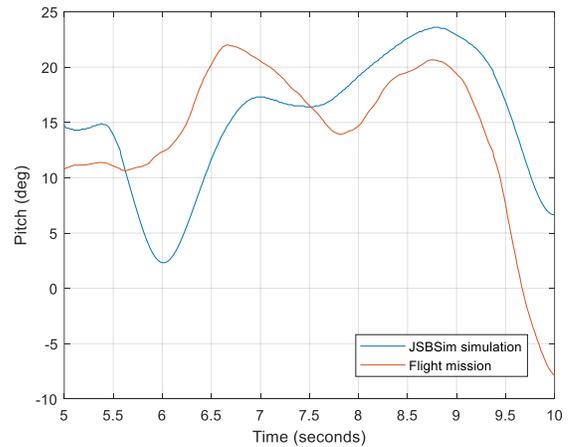


**Figure 4.20.** Aileron and elevator deflection for small and medium signals in Phase 3B

For this particular test, the relevant variables are roll ( $\phi$ ) and pitch ( $\theta$ ) angles. This stage is essential for the correct development of the system since the real flight data is the resource to use as a reference. The first 5 seconds of the simulation correspond to the transient response from the initial to a neutral state and thus, they are omitted from the study.



**Figure 4.21.** Computer model vs. real model. Roll output



**Figure 4.22.** Computer model vs. real model. Pitch output

The performance in Figure 4.21 and Figure 4.22 express the final model performance obtained after iterations and adjustments to the model. To that end, the next steps have been followed:

- The initial offset has been removed by identifying the initial conditions in the simulation; the FDM comes from a neutral state whereas, in the flight mission, the R/C comes from a previous state that will induce changes in the performance observed in the following seconds.
- The gains adjustment is given by scaling the inputs based on the aero surfaces.
- The communication delays between the controller and the actuator/servos have been added with a shift to the FDM. This time discrepancy is foreseeable due to the configuration of the electronic equipment on board. For this particular case, the model showed a delay of 0.25s in roll and 0.5s in pitch.
- The responsiveness of the computer model has been tuned by adapting the aerodynamic coefficients that relate the aileron or elevator deflection to the correspondent roll or pitch performance.

The similarity between both models is quantified by the Pearson correlation coefficient,  $r$ , which measures how well two vectors or sets of data are related. It ranges from +1 to -1, where a value close to 0 indicates that there is no connection between the two variables and values around -1 and +1 indicate a negative and positive connection respectively. The correlation between two vectors  $X$  and  $Y$  of length  $n$  is defined as:

$$r(X, Y) = \frac{\frac{1}{n} \sum_i^n X_i Y_i - \mu_X \mu_Y}{\sigma_X \sigma_Y} \quad (4.1)$$

Where  $\mu_X$  and  $\mu_Y$  are the means of the vectors  $X$  and  $Y$  and  $\sigma_X$  and  $\sigma_Y$  are the standard deviations of the same vectors  $X$  and  $Y$ .

**Table 4.4.** Pearson correlation coefficients on results in Figure 4.21 and Figure 4.22

Angle	Pearson correlation coefficient
Roll	0.61645
Pitch	0.49744

For this particular case, even though the Pearson correlation coefficient shows a fit between the simulation model and the real model (Table 4.4), the match is around 50-60%. Real models and especially, small fixed-wing aircraft are made from materials that might introduce random errors into the system. In the EPP FPV aircraft, the carbon strip that connects the aero surface and the servo (Figure 4.23) is flexible and as a consequence, it affects the way the deflection is transmitted to the actuator.



**Figure 4.23.** Rudder horn and carbon strip in the EPP FPV (HobbyKing.com, n.d.)

The correlation coefficients showed in Table 4.4 are particular of the conducted flight mission and the range of deflections used. This means that the computer model may not be a representation of the real model under other (different) conditions.

Additionally, existing forces to the FDM (wind conditions) have not been modelled in the simulation but are present during the flight mission. This will impact the performance of the EPP FPV in the computer environment and this behaviour is noticed in the responses (Figure 4.21 and Figure 4.22). When addressing this issue, the designer must be aware of the model limitations. For example, the maximum wind speed for flying a light aircraft is 8-12mph whereas the mean (min/max) wind speed in St. John’s, NL is 14mph (“St. John’s Historical Wind Speed,” n.d.).

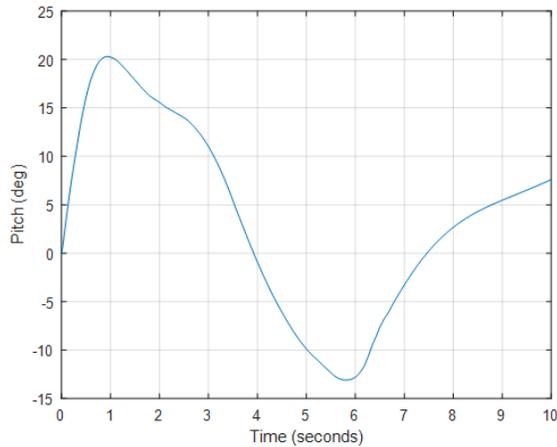
### 4.1.3. Simulation of SAA Manoeuvres

The final stage of this procedure is to show the EPP FPV JSBSim model added in its final application knowing the context of the simulation where the computer model is integrated. The framework is defined by an SAA manoeuvre as a consequence of an encounter between the RPA and a piloted aircraft. The following table offers the minimum stages of a vertical manoeuvre consisting of the initial conditions, the avoidance and the recovery:

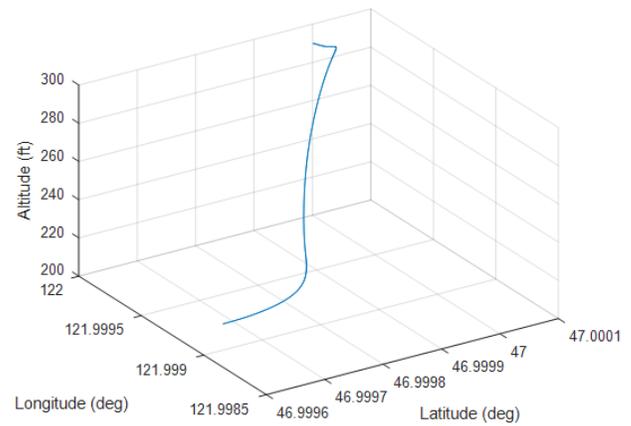
**Table 4.5.** Collision avoidance events for testing the EPP FPV in SAA (3D) manoeuvres

Event/ Task	Throttle (%)	Elevator (rad)	Aileron (rad)
Initial state (0.0s)	70.0	0.0	-0.1
Avoidance manoeuvre (1.5s)	0.0	0.2618	0.015
Recovery (h<200ft in 6s)	30.0	-0.17	-0.1

Assuming that the aircraft comes from the initial condition specified in the table, the manoeuvre is performed by the throttle level and the aileron and elevator aero surfaces at the same time, creating a change in the trajectory as shown in Figure 4.25. The deflections in both aero surfaces were randomly selected but, when the elevator reaches its maximum value, the aircraft flies down to 200ft in less than 5s.



**Figure 4.24.** EPP FPV avoidance manoeuvre. Pitch angle



**Figure 4.25.** EPP FPV avoidance manoeuvre. Trajectory

The relevance of this application is easily noticed from the rate of descent of around 1,200fpm provided by the EPP FPV. Transport Canada defines an abnormal rate of descent as more than 500fpm for an unpressurized flight with passengers on board (Transport Canada, 2010b) (not recommended), meaning that, the EPP FPV RPA has a rate of decent 2.5 times greater and might guarantee the avoidance in case of an NMAC scenario. This claim would need further analysis that falls beyond the scope of this thesis.

### 4.3. Summary

When implementing FDMs in OSS platforms, the model certainty is always questioned. The model must be approved and validated against similar and real models in order to frame its limitations. In this chapter, a set of minimal requirements for developing and validating RPA flight simulators has been presented based on the current advisories and the differences between piloted and unpiloted flight simulators. From a general validation procedure, specific tests have been adapted to match the simulation application. In its final version, the steps taken to develop

and validate the FDM for testing SAA strategies have shown to be a valuable tool for the development of RPA fit-for-purpose FDMs.

The model reliability has been tested in each phase and adjusted for an improved performance with the EPP FPV as the example. The JSBSim 6-DoF FDM has also been shown as a valuable and functional method for simulating RPAs in any context and particularly in SAA where two aircraft share the same environment. For more detailed information about the software and the complementary user guide of this chapter, please refer to Appendix A.

The main limitations of this chapter are related to the RPA model and not the validation procedure in particular. Even though the EPP FPV computer model and the real aircraft showed a correlation coefficient of over 50%, its reliability is still questionable. The EPP FPV R/C aircraft is sensitive to wind changes and its configuration is simple with low-quality materials. Particular structural constructions (even in larger commercial aircraft) often add errors to the aircraft performance. This means that the flight data collected from one mission corresponding to one single real aircraft is not enough to prove the reliability of the computer model. The functionality of the validation procedure has been shown using this particular case but the RPA model remains to be improved. In that case, flight data from several missions should be collected (see recommendations and limitations in Chapter 6).

# Chapter 5

## *Giant Big Stik Computer Model Development with Sense and Avoid Applications*

A second Remotely Piloted Aircraft (RPA) computer model is developed by following the intuitive methodology introduced in Chapter 4. The Giant Big Stik computer model is used as a representative aircraft with the purpose of implementing it in scenarios with piloted aircraft for the study of avoidance manoeuvres (Cereceda et al., 2019). In this particular case, the approach focuses on reflecting the aircraft capabilities in one specific axis where the avoidance performance will take place.

When two aircraft are in conflict and nearly invading their collision volume, the options are minimal (Appendix C) and the aircraft is obliged to perform an extreme manoeuvre. The main issue with an extreme manoeuvre is the ability of the aircraft to safely recover from a critical state. The complex stability of the aircraft requires studies from its physical point of view that can be simplified by defining the simulation context and its limitations.

This chapter, in its first part, includes the Giant Big Stik computer development. The procedure followed to create the model is the methodology presented in Chapter 4. The code has not been included in this case to avoid redundancies but it is found in Appendix D.

The Sense and Avoid (SAA) application is introduced in the second part of this chapter. It consists of the integration of the Giant Big Stik as a representative RPA into a simulation with a Cessna 172 to evaluate their interaction in a close encounter. This second part includes: (1) a brief description of the simulation and context, (2) a methodology used for solving the SAA problem, and (3) a discussion of the implications of close encounters between a Giant Big Stik and a Cessna 172 aircraft.

This study does not discuss the conflict resolution in a collision scenario given the critical circumstance of a collision. Initial discussion and results that originated in the application of this chapter were presented during a Canadian conference (Cereceda, Rolland, & O'Young, 2018) and is currently under a second revision for publication on their special issue. This particular work focuses on encounters between two piloted aircraft and is out of the scope of this thesis.

## **5.1. Giant Big Stik Computer Model Development**

The Giant Big Stik is the largest wooden aircraft belonging to the Stik family developed by Great Planes (Great Planes, 2005a). It is mainly oriented to sports aerobatics with a nearly unlimited flight envelope making it a perfect RPA for the study of extreme avoidance manoeuvres. The thrust is provided by a 16x8in propeller and a 1.55cu-in Zenoah G26 Air Engine (Zenoah, 2007).

According to the methodology diagram expressed in Chapter 4 (Figure 4.3), the validation of the Giant Big Stik can be broken down into the following items: (1) Modelling and development and

(2) Computer model validation. The first part focuses on defining the model requirements and modelling whereas the second part focuses on the model validation. It is important to clearly define the application context (SAA) and its basis are further explained in Appendix C.

### ***5.1.1. Giant Big Stick Simulation Context (Phase 1)***

The model developed in this section serves as a representative RPA for the study of avoidance manoeuvres in close encounter scenarios with piloted aircraft. The simulated critical manoeuvres will focus on a vertical-only diving performance and as a consequence, the computer model development focuses mainly on the pitch axis.

The context of the task described in Phase 1 (as included in the diagram expressed in Figure 4.3) can be broken down into the following items:

- **Simulation objective:** performance in a diving manoeuvre similar to a real flight. Although the computer improvements of the model will focus on large signals in the pitch axis, the computer test will cover all axes since it is important that no instabilities are present and the performance is coherent.
- **Model requirements:** In Phase 3A, large signals in elevator, ailerons, and rudder are simulated in a computer environment. The model performance is evaluated by inspection and statistical analysis with a confidence level of 95% for the Pearson correlation coefficient. In Phase 3B, the model is initially adjusted in roll ensuring that this axis does not create any critical instability that might affect the pitch axis. In the second fine-tuning, the computer model is further adjusted from the pitch axis perspective.

- **Code evaluation, programming, and development of the Flight Dynamics Model (FDM):** similar to the EPP FPV case study, the JSBSim package is used in the model development and the Giant Big Stik aircraft model performance is visualized in FlightGear. The AeroSim toolkit in MATLAB/Simulink is used again as a reference model for this second computer model development example.

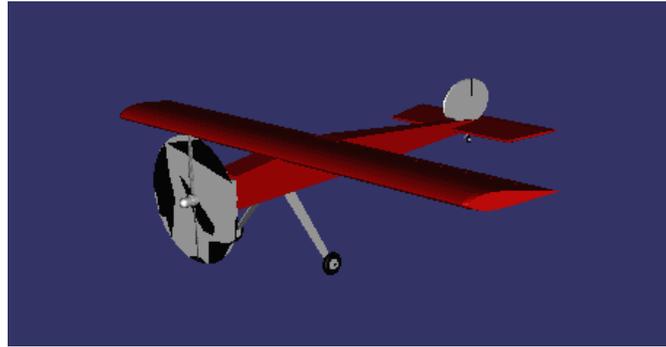
Similar to the EPP FPV case, the wind is present during the flight tests and the real model is expected to show differences with the computer model in Phase 3B.

### ***5.1.2. Giant Big Stik Computer Modelling and Development (Phase 1)***

The modelling in JSBSim starts with the identification of the parameters and aerodynamic coefficients (Table 5.1 and Table 5.2). The parameters and other dimensional information were obtained by manually measuring the aircraft and consulting the manufacturer's information. The aerodynamic parameters were initially calculated by Dr J. Stevenson (J. D. Stevenson, 2015), by following the same approach as in Appendix B for the EPP FPV RPA. Later on, the coefficients were modified according to the model development and validation methodology presented in Chapter 4 (Figure 4.3). The visual model in FlightGear (Figure 5.1) was designed from scratch by using SketchUp ("SketchUp," 2019) and integrated into the simulation.

The aircraft configuration files, as well as the engine and propeller files, are included in Appendix D, Sections D.2.2, D.3.2, and D.4.2, respectively.

The range of control surfaces is provided by the manufacturer and included in Table 5.3. Unlike the EPP FPV case, the pilots in our team are confident about the capabilities of the Giant Big Stik and the range of control deflections are kept as they were given by the manufacturer.



**Figure 5.1.** The Giant Big Stik aircraft in FlightGear

**Table 5.1.** Giant Big Stik Parameters

<metrics>		<mass_balance>	
Wing area (ft2)	10.538	Ixx (Slug*ft2)	0.3046
Wing span (ft)	6.709	Iyy (Slug*ft2)	0.4752
Wing incidence	2.00	Izz (Slug*ft2)	0.7036
Chrod (ft)	1.148	Ixy (Slug*ft2)	0
H tail area (ft2)	1.69	Ixz (Slug*ft2)	0.0951
H tail arm (ft)	2.36	Iyz (Slug*ft2)	0
V tail area (ft2)	1.05	Empty weight (lbs)	13
V tail arm (ft)	2.27	CG (in)	[14.4881, 0, 0]
AERORP (in)	[18.4961, 0, 2.5591]		

**Table 5.2.** Giant Big Stik aerodynamic coefficients

Lift Coefficient		Drag Coefficient		Side Coefficient		Roll Coefficient		Pitch Coefficient		Yaw Coefficient	
		$C_{D0}$	0.1					$C_{m0}$	0.15		
$C_L^\alpha$	5.32	$k$	0.087					$C_m^\alpha$	-1.9		
$C_L^{\dot{\alpha}}$	1.7							$C_m^{\dot{\alpha}}$	-3.5		
				$C_Y^\beta$	-0.83	$C_l^\beta$	-0.034			$C_n^\beta$	0.071
						$C_l^p$	-0.41			$C_n^p$	-0.0575
$C_L^q$	3.9							$C_m^q$	-6.813		
						$C_l^r$	0.107			$C_n^r$	-0.12032
$C_L^\delta$	Elevator	$C_D^\delta$	0.0135	$C_Y^\delta$	0.0	$C_l^\delta$	0.0	$C_m^\delta$	-1.458	$C_n^\delta$	0.0
	Aileron		0.0302		-0.075		-0.2		0.0		0.0108
	Rudder		0.0303		0.1914		-0.107		0.0		-0.062

**Table 5.3.** Control surface deflections range

Control surface	Manufacturer	
	$\delta_{min}$	$\delta_{max}$
Elevator	-26° (-0.4643rad)	26° (0.4643rad)
Aileron	-32.6° (-0.569rad)	32.6° (0.569rad)
Rudder	-31.6° (-0.546rad)	31.6° (0.546rad)

### 5.1.3. *Giant Big Stik Computer Model Validation (Phases 2 and 3)*

In the first stage, the aircraft model configuration files expressed in JSBSim were integrated into FlightGear allowing to test Phase 2 in the validation process. The recommended flying procedures in the Giant Big Stik manual (Great Planes, 2005b) were imprecise but tested, and other online reviews were followed and compared for this phase (RCGroups, 2018).

The pilots overall mentioned that the engine model in JSBSim needed to be tuned; this comment will also be reflected in the following sections and it will be adjusted in the final stage.

#### 5.1.3.1. *Giant Big Stik Phase 3A: Computer Test*

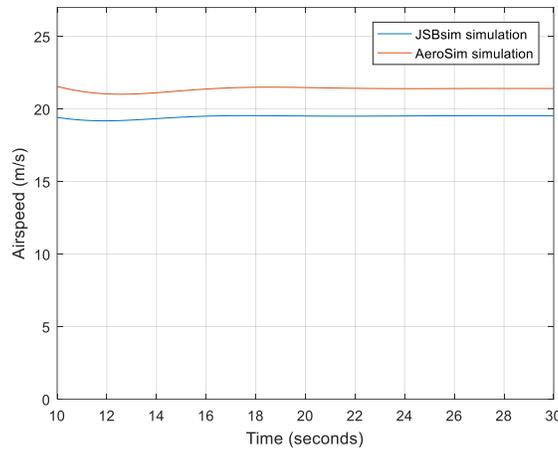
When comparing JSBSim with AeroSim, different scenarios for the commands were independently tested and evaluated in order to reach conclusions and define JSBSim FDM limitations for RPA applications.

The control inputs to the FDM are the primary flight control deflection angles associated with the elevator, ailerons and rudder surfaces. Therefore, the simulations were divided into four categories similar to the EPP FPV case study: static test, elevator test, aileron test and rudder test. There is a special interest in carrying out open-loop tests where the aircraft characteristics and

performance are evaluated for simple primary control inputs, since it gives an excellent opportunity to check the physical dynamic performance and its coherence.

Steady-state conditions take place when there are no variables that make the system change in time; which is an unlikely situation in a real flight but it is a useful way to identify the dynamics of the system. In this first test, the airspeed is compared in a steady-state situation for both models with the aircraft in open-loop without any throttle command.

Given the common initial conditions of 300m and 20m/s, all simulations for the Giant Big Stik case study last 30s.



**Figure 5.2.** Giant Big Stik static test. Steady-state airspeed<sup>3</sup>

From the results in Figure 5.2, and as expected due to the pilot's comments, the JSBSim model does not perform as fast as the AeroSim FDM; there is an offset error of 7% in True Airspeed (TAS). Despite this value, the offset is constant during the simulation, meaning that the dynamics are the same for both models in open-loop and a slight difference will be expected in

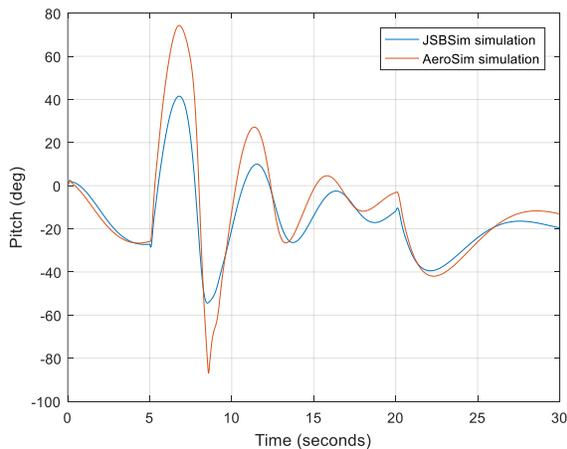
---

<sup>3</sup>The first seconds of the simulation are neglected since it shows the transient response from the initial conditions to the steady state.

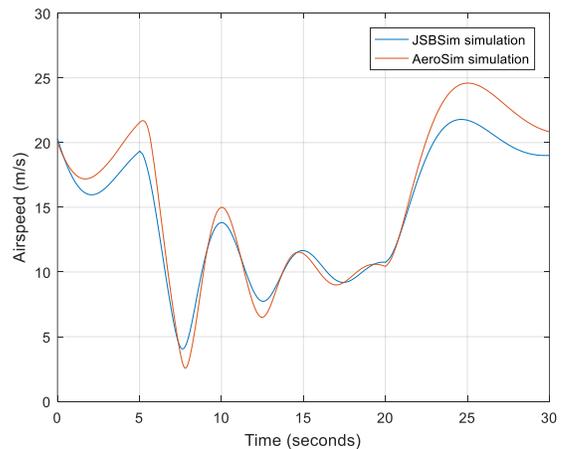
the transient response in the next simulations. This error is later eliminated by adding more power in the engine configuration file.

In the remaining simulations in Phase 3A, the system, represented by the FDM, is assessed for input combinations of the flight control surfaces with no feedback. The isolation of the inputs one at a time assists with the design of a control system when developing an autopilot in future studies.

The elevator is set at the maximum value ( $-0.4643\text{rad}$ ,  $-20^\circ$ ) at 5s and goes back to zero at 20s in the **elevator test**. Following the same flight control principles as the EPP FPV also included in Section 2.3.5, a negative elevator produces an increment in the pitch angle.



**Figure 5.3.** Giant Big Stik elevator test. Pitch angle

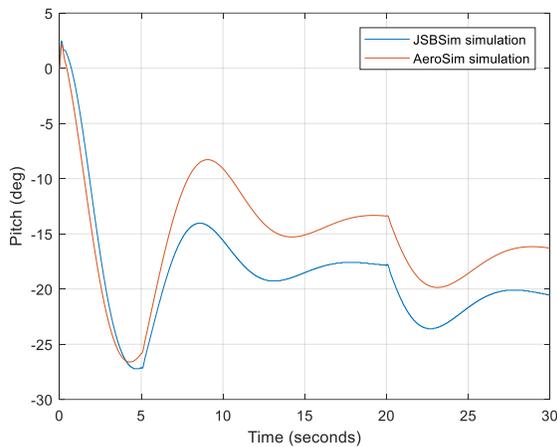


**Figure 5.4.** Giant Big Stik elevator test. Airspeed

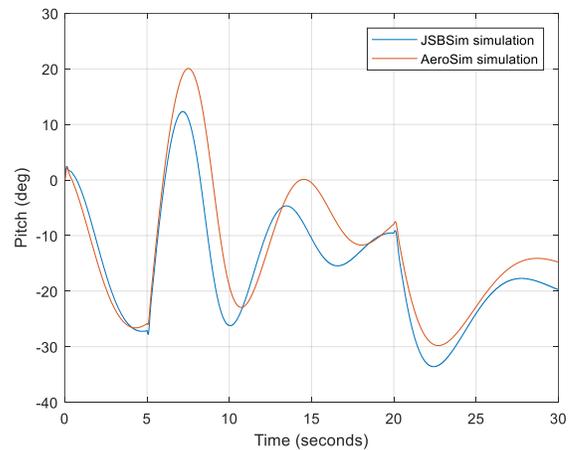
This direct effect in pitch angle due to changes in elevator deflection is noticeable in Figure 5.3; the pitch angle shows the same shape in both cases but when a sudden change in the elevator happens, the AeroSim system is up to 40% more responsive in pitch.

The coherence of the performance is also observed in Figure 5.4: with an increase in the angle of attack, the aircraft slows down.

In a real mission, it is unlikely that the elevator command is set to its maximum value. When an aircraft's physical performance and stability are to be tested in the field, a small signal test is usually done instead –e.g.  $-0.015\text{rad}$ ,  $-1^\circ$ , in elevator done in this study. Considering a regular climb, a second elevator test of  $-0.10\text{rad}$ ,  $-6^\circ$ , is carried out.



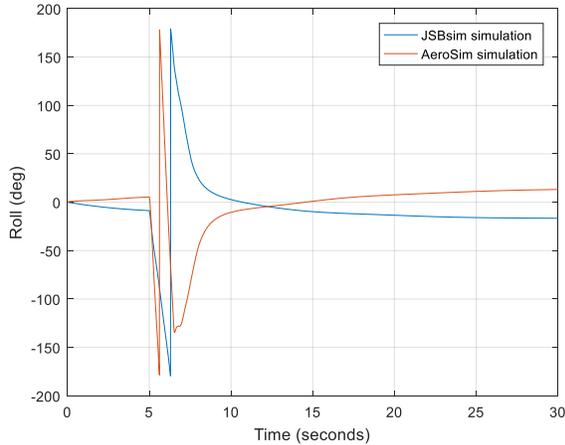
**Figure 5.5.** Giant Big Stik elevator test for small signal ( $-0.015\text{rad}$ ,  $-1^\circ$ ). Pitch angle



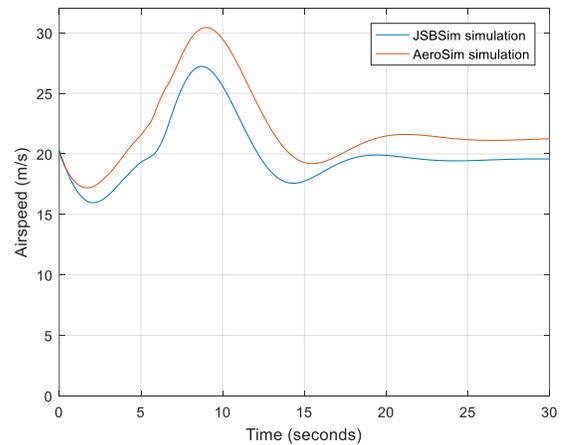
**Figure 5.6.** Giant Big Stik elevator test for a medium signal ( $-0.10\text{rad}$ ,  $-6^\circ$ ). Pitch angle

It is noticeable once again that the AeroSim model is affected by the differences in engine power for both low and medium elevator commands. Additionally, Figure 5.6 shows that for a medium elevator signal, the JSBSim model attempts to come back to the initial state in a shorter time; the difference between the maximum and minimum peaks of the response is slightly smaller. Interestingly, there is an offset (Figure 5.5), created by the differences in engine power, that remains even when the elevator is set back to zero. This means that in order to get the same performance as the AeroSim model –reference FDM in this study-, the JSBSim model should be adjusted either manually with a joystick, or in the control with an added gain.

The ailerons are set at the maximum value ( $-0.569\text{rad}$ ,  $-32.6^\circ$ ), turning left at 5s and going back to the initial situation at 6.5s in the **aileron test**.



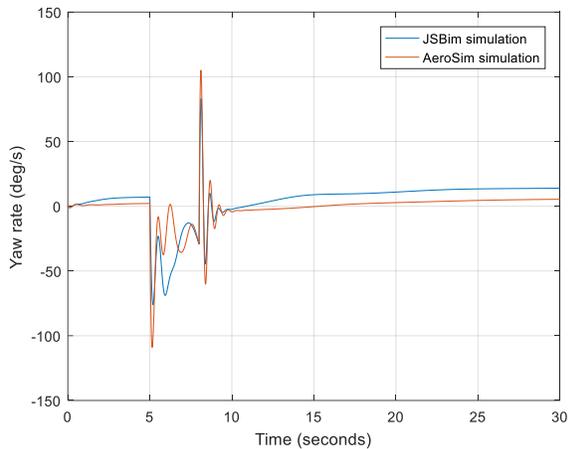
**Figure 5.7.** Giant Big Stik aileron test. Roll angle



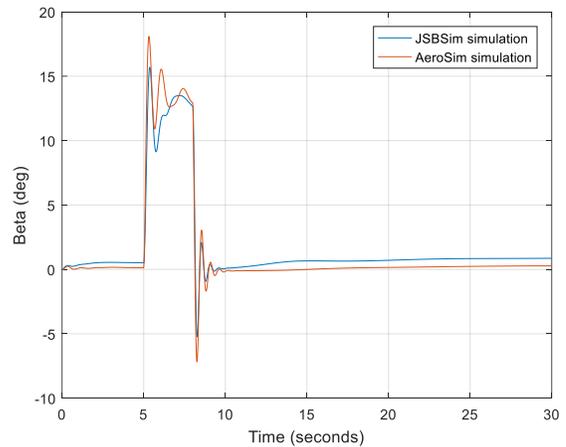
**Figure 5.8.** Giant Big Stik aileron test. Airspeed

As shown in the roll angle (Figure 5.7), when the ailerons are set to the maximum value from a flat situation, AeroSim is again more responsive than JSBSim. In 1s, the AeroSim model has turned  $360^\circ$ , whereas the JSBSim model turned  $180^\circ$ . Despite the difference, the JSBSim context is more likely to take place in a real situation because of the physical configuration of the aircraft. Considering how the airspeed develops during that timeframe (Figure 5.8) and regardless of the turns, the aircraft model in both cases tends to come back to its initial value.

The rudder control is not commonly used in a regular flight due to its responsive nature, which might destabilize the aircraft. Although the turns and changes in heading are usually commanded by the ailerons instead, the validation procedure includes a **rudder test** where the rudder is set at its maximum value ( $0.546\text{rad}$ ,  $31.6^\circ$ ) at 5s and goes back to zero 3s later, making the system turn right. Following the same concept mentioned in the EPP FPV case where roll and pitch angles do not give extra relevant information, in this test the yaw rate ( $r$ ) and the sideslip angle ( $\beta$ ) are analyzed.



**Figure 5.9.** Giant Big Stik rudder test. Yaw rate



**Figure 5.10.** Giant Big Stik rudder test. Sideslip angle ( $\beta$ )

Both models are considerably similar when there is a rudder deflection (Figure 5.9 and Figure 5.10). In contrast to the elevator and ailerons test, when there is a lateral disturbance both models perform similarly.

### 5.1.3.2. *Giant Big Stik Phase 3: Flight Test*

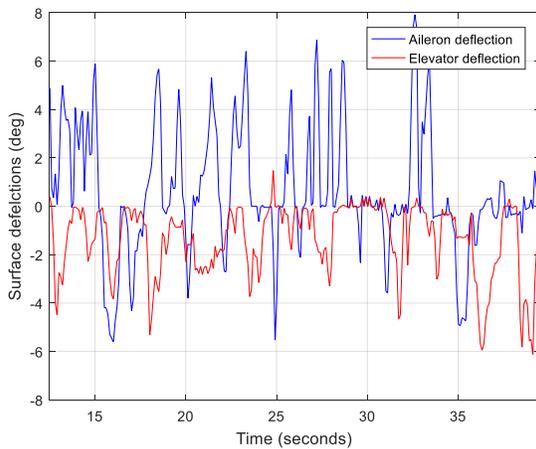
The final stage of the Giant Big Stik computer model development uses test scenarios and empirical validation by comparing model outputs with real measurements from a flight mission.

A Giant Big Stik was flown following the current regulations established by TC (Transport Canada, 2018a) in the allowed areas surrounding St. John’s, NL, Canada, belonging to Class G airspace. The mission included an R/C controller, a Ground Control Station (GCS), a Giant Big Stik vehicle and an autopilot (Piccolo II (“Piccolo II Autopilot,” n.d.)) onboard for recording the control inputs and the performance parameters over time.

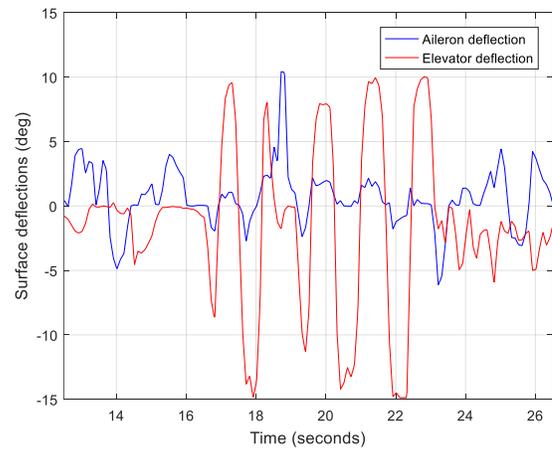
In the analysis, the tasks of taking off and landing were neglected. The rudder control command was also removed from the study since it was not used during flight and there is also a special

interest in the performance of the pitch axis. Likewise, the throttle during the mission was ~50% at all times and therefore, considered constant in the validation process.

The full flight test was divided into sections where the relevant signals were identified as small, medium and large signals. Two main sections were selected where the commands were either large or medium. In the first section (Figure 5.11), the aileron deflection was kept at medium signals while the elevator deflection was left at small signals. The main interest of starting with this section, even though the focus is on the elevator and pitch performance, is because it is important to ensure that no signals in the ailerons could create instabilities in the pitch axis. However, the selected Section 2 (Figure 5.12) focuses on sudden changes in the elevator by approximately  $25^\circ$ , which represent the same deflection that the aircraft might experience during an extreme manoeuvre in a Near Mid-Air Collision (NMAC) scenario.

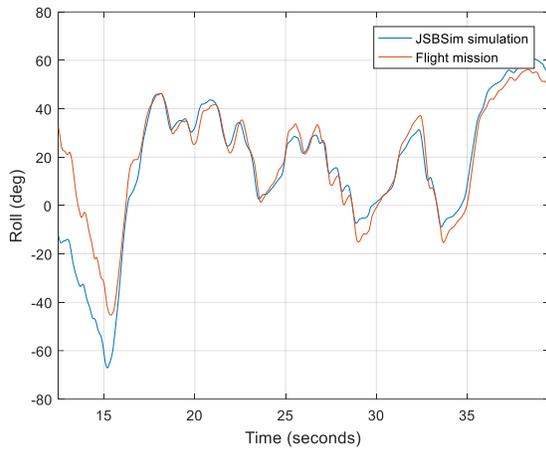


**Figure 5.11.** Aileron and elevator deflections in Section 1

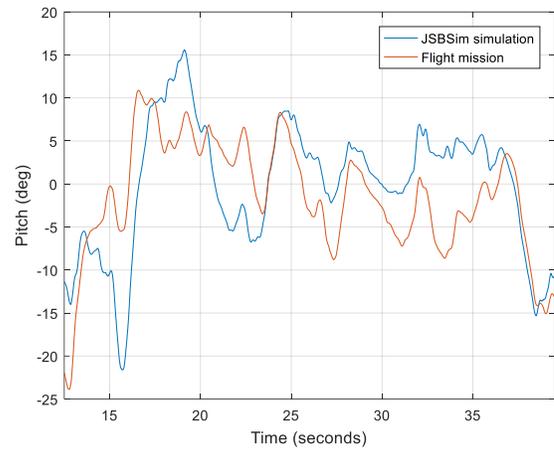


**Figure 5.12.** Aileron and elevator deflections in Section 2

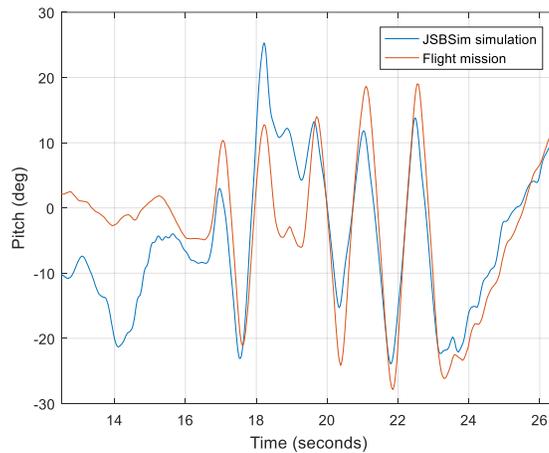
The study in both cases starts at 12.5s, allowing the computer model to stabilize. The first section lasts 39.5s with a time range study of 27s, whereas the second section lasts 26.5s with a total simulation time of 14s. The relevant analyzed variables are the roll and pitch angles, which are directly related to their corresponding axis.



**Figure 5.13.** Section 1. Computer model vs. real model. Roll angle



**Figure 5.14.** Section 1. Computer model vs. real model. Pitch angle



**Figure 5.15.** Section 2. Computer model vs. real model. Pitch angle

The results shown in this section are the final outcome of the computer model after the complete validation and development. Although the offsets, delays, and gains have been adjusted to match the real flight data, differences are allowed in the system due to the uncontrollable external forces (e.g. the wind and the inability of replicating the same real wind during the mission in a computer simulation) and the different initial state where both systems (the real and the computer model) have their starting point.

Although the similarities are observable, the final application of the model is the study of Collision Avoidance (CA) manoeuvres in encounters with piloted aircraft. Therefore, it is important that the fit is realistic and quantifiable. Following equation (4.1) for calculating the Pearson correlation coefficient, and considering the data shown in Figure 5.13, Figure 5.14, and Figure 5.15, the corresponding coefficients are:

**Table 5.4.** Pearson correlation coefficients on results in Figure 5.13, Figure 5.14 and Figure 5.15

Section	Angle	Pearson correlation coefficient
1	Roll	0.9304
1	Pitch	0.60664
2	Pitch	0.74044

The first section in Figure 5.13 indicates a match between both models in roll axis with a Pearson correlation coefficient close to +1 (Table 5.4). When focusing on roll axis in the first section, it is evident that changes in roll will not create instabilities in the pitch axis and the aircraft performance in a simulated environment will be as close as possible to the real flight. The pitch comparison results in Section 1 (Figure 5.14) show that the dynamics of both systems are similar and the differences have been later adjusted in Section 2 (Figure 5.15). The Pearson correlation coefficient in pitch angle has been improved from 0.6 in Section 1 to 0.74 in Section 2 (Table 5.4). In that context, both performances can be considered alike when sudden changes take place in the elevator.

The limitations of this model are related to the range of the commands on the real mission. Even though the fit of the model has been shown during the computer test in Phase 3A, it is recommended that in future studies, further real test include extreme manoeuvres in order to strengthen the computer results.

## **5.2. Giant Big Stik Application: Vertical Collision Avoidance**

A dynamic model of the Cessna 172 is used as the representative intruder aircraft, whereas a dynamic model of the Giant Big Stik RPA performs the avoidance manoeuvre. The Giant Big Stik is selected over the EPP FPV aircraft because of its capabilities. Whereas the EPP FPV (Chapter 4) is a robust aircraft whose performance is compared to a glider, its manoeuvrability is more limited when compared to the Giant Big Stik, and does not serve as a strong representative RPA for this context. The Cessna 172 aircraft represents a traditional general aviation aircraft that is not equipped with a Traffic Collision Avoidance System (TCAS) or any other avoidance system. The SAA aircraft capabilities depend on the pilot, which becomes dangerous when a small RPA flies nearby and the pilot is unable to identify it.

The simulation uses the JSBSim FDM (Berndt & JSBSim Development Team, 2011; Cereceda, Rolland, & O'Young, 2017) along with FlightGear ("FlightGear Flight Simulator," 2019) as the visualization software. The 6-DoF FDM of the Cessna 172 and its visual model have been downloaded from the online JSBSim and FlightGear libraries, respectively ("JSBSim Flight Dynamics Model - Code aircraft/c172x," 2009). The FDM of the Giant Big Stik was validated in the previous section (also published in (Cereceda et al., 2019)), and the visual model was created using SketchUp ("SketchUp," 2019) and later integrated into FlightGear.

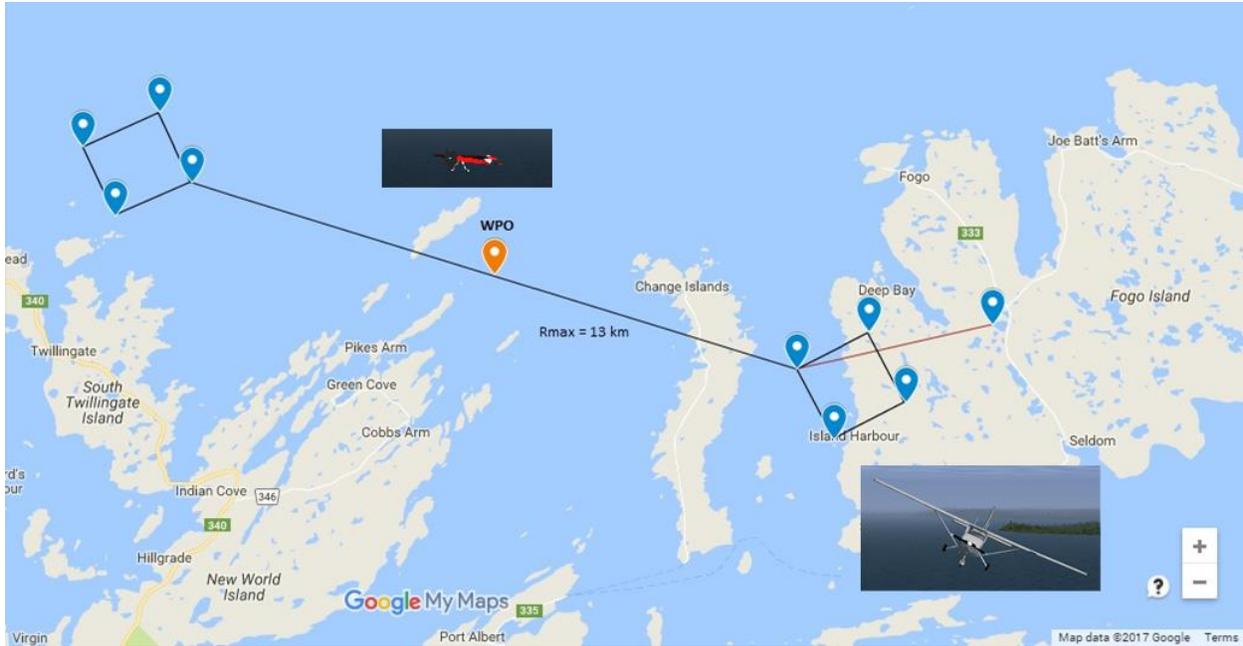
In the initial tests to assess the encounter geometry and identify the conflict points, both aircraft are not carrying any CA system and are assumed to be flying at the same altitude within their

flight envelope. In later studies, the Giant Big Stik carries an SAA system that is able to identify a conflict point and conduct a vertical manoeuvre.

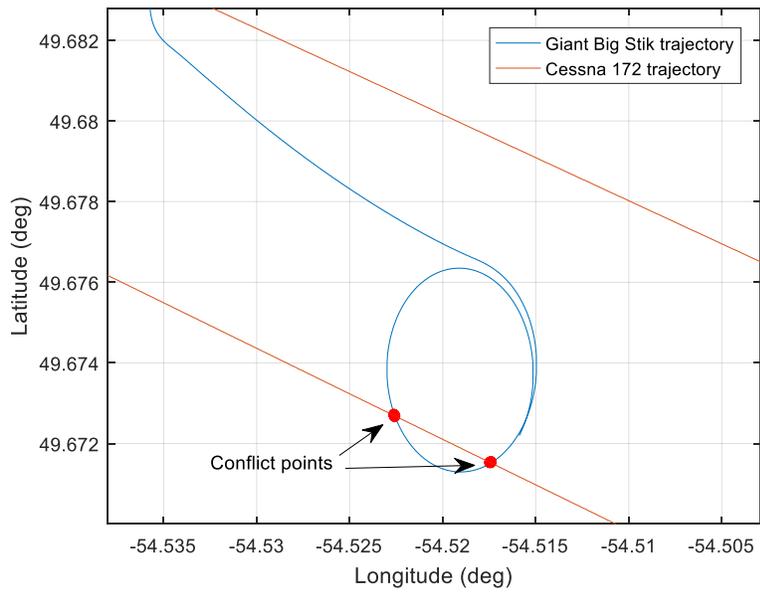
### ***5.2.1. Encounter Geometry and Intruder's Trajectory***

The simulation takes place in the areas surrounding Fogo Island in Newfoundland and Labrador (NL), Canada. The trajectory paths are based on a previous work designed for the development of 4-D encounter geometries (Cereceda & Stevenson, 2014) and described in Appendix D. This geometry (the  $\Phi$  manoeuvre) consists of a dog bone path and an orbit path whose centre is halfway between the returning waypoints of the dog bone's straight segment. The angle of the encounter depends on the radius of the orbit, but for this study the interest is in the frequency of the encounters permitted by the  $\Phi$  manoeuvre instead. Therefore, the orbit path is defined by a single waypoint (the orbital centre) with a turning radius of around 450ft; smaller trajectories would prevent an encounter from happening. The waypoints for both trajectories are expressed in the following map (Figure 5.16) where the Cessna 172 performs a dog bone path and the Giant Big Stik aircraft follows an orbit.

While the Cessna 172 remains in level flight and cruise speed for the entire test, the autopilot in the RPA is set to orbit around the waypoint until the encounter takes place. The trajectories of the Giant Big Stik and the Cessna in Figure 5.17 show two conflict points located around halfway between the waypoints that mark out the straight path.



**Figure 5.16.**  $\Phi$  encounter geometry and its waypoints



**Figure 5.17.**  $\Phi$  manoeuvre: Giant Big Stik and Cessna's trajectories with conflict points

When a conflict point is identified by the Giant Big Stik, the pitch control is disabled to allow the performance of a diving avoidance manoeuvre. However, the roll control remains active for the

entire manoeuvre, since the vertical axis is only affected by the pitch and aircraft controls. The aircraft dives by regulating the primary surface control commands and throttle to avoid the intruder's collision volume. Once the aircraft has completed the avoidance manoeuvre, it resumes its task.

Although the avoidance is vertical, the altitude at which both aircraft are flying is not relevant as long as both converge at the same altitude.

### ***5.2.2. Collision Avoidance Conditions and Methodology for Solving the SAA problem for RPAs***

The minimum requirement for the detection is that there is enough time for the aircraft to perform a manoeuvre and remain safe. The functional boundaries and thresholds defined in Appendix C establish the risk of an airborne collision.

The purpose of the avoidance manoeuvre is to keep the intruder aircraft out of the collision volume (identified as a conflict point in Figure 5.17). The collision volume is determined by a cylinder of 200ft height and 500ft radius (Appendix C) whose centre is the CM of the ownship aircraft. These values indicate that diving at least 100ft after an NMAC situation is identified will prevent a collision. 100ft is quite close-fitting and is considered a theoretical reference for the avoidance. In a real flight, it is recommended that the threshold be extended to around 200ft for safety reasons.

In the recommendations introduced for defining the best practices for Beyond Visual Line-of-Sight (BVLOS) operations (Appendix C), the manoeuvre time ( $\tau$ ) is the time to complete the

avoidance task, which must be minimized in a case of an NMAC. This means that  $\tau$  represents the minimal time to dive, which can be calculated from the maximum diving rate of the aircraft.

A vertical avoidance emulates the TCAS procedure, which is a common practice across commercial aircraft (Appendix C – Figure C.1 and Figure C2). In later studies, the diving manoeuvre could evolve into a more complex manoeuvre, including a roll with the aileron and rudder deflections (Chapter 6). However, the first step is to calculate the maximum diving rate of the representative RPA.

Focusing on the vertical axis, the proposed SAA solving method examines the aircraft's performance in terms of the altitude response. Due to the complexity of the aircraft model, it is important to define the context in order to minimize the error and limit the scope.

Considering the framework provided by the simulation context described at the beginning of Section 5.2, which is based on the CA concepts introduced in Appendix C, when a threat is detected, it is expected that the pilot in command will conduct an avoidance manoeuvre (15s)<sup>4</sup>. However, in case the pilot fails to perform a manoeuvre because either is not aware of the presence of an intruder, or there is no human-in-the-loop, an avoidance procedure must take control of the aircraft in order to avoid a collision. The full process is summarized in the flowchart in Figure 5.18.

---

<sup>4</sup> Recommended best practices in Canada (see Appendix C, Section C.1.1 and Figure C.5)

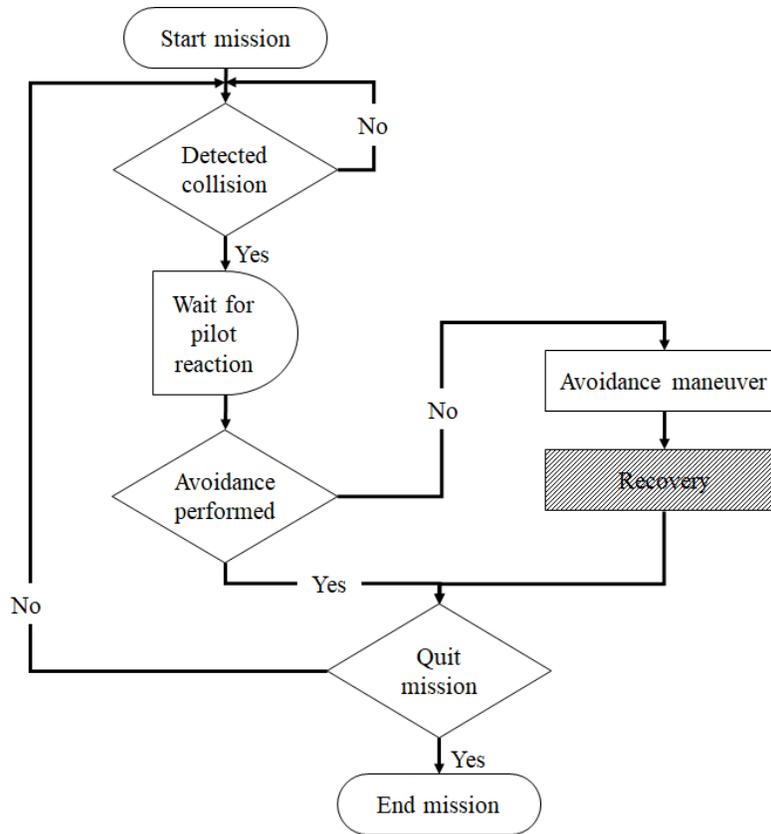


Figure 5.18. CA flowchart procedure<sup>5</sup>

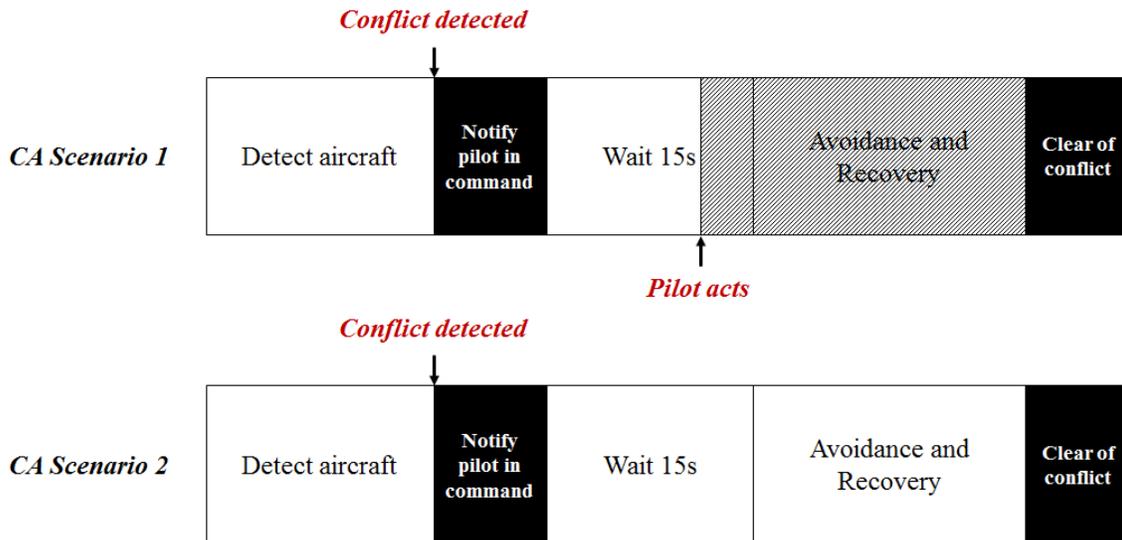


Figure 5.19. CA Scenarios and solving procedure based on Figure 5.18

<sup>5</sup> This structure only contemplates one detection at a time. In case of multiple detections, priority is given to the detection that creates the most critical danger.

The participation of the human pilot in command during the flight mission determines the sequence of actions during the CA procedure. In the first scenario (Figure 5.19 – top diagram), the pilot reacts with an avoidance manoeuvre to the detected conflict within the 15s allowed by the CA system. With the pilot taking control of the aircraft, the avoidance procedure is not activated. In the second scenario (Figure 5.19 – bottom diagram), the pilot does not act on the notification of a possible conflict because either there is no human-in-the-loop or the pilot is not aware of the notification (communication failure or other). Under this circumstance, the CA system executes an avoidance manoeuvre on time to avoid a collision without the authorization from the pilot. When the avoidance has finished, the system issues a “clear of conflict” notification.

The CA controlled procedure can be divided into two different stages: (1) the avoidance manoeuvre and (2) the recovery performance. Whereas the avoidance manoeuvre is crucial for the calculation of  $\tau$  and will be discussed in further detail over the following subsections, the recovery performance remains as future work. At this initial stage, the interest is in assessing whether the RPA has the capability of avoiding the piloted aircraft. In later studies, the performance can be improved for a smoother dive.

Assuming that the aircraft altitude response is to be controlled, the CA problem can be observed from a control’s perspective. The flight-path is initially operated by the autopilot until the intruder flies toward the collision volume. During the avoidance, the aircraft disconnects the autopilot for pitch control and dives out of the NMAC by regulating the primary surface control commands and throttle. Once the aircraft has dived 200ft and is out of a possible NMAC, the task prior to the collision detection is resumed. The simulation and the study of avoidance manoeuvres can be synthesized as follows:

- a) Conditions prior to the encounter: the control of the aircraft is carried out by the autopilot ( $\Phi$  manoeuvre).
- b) Avoidance manoeuvre: the vertical performance is executed by the avoidance system controlling the elevator deflection and the throttle level.
- c) Recovery: the aircraft is stabilized by operating the control commands: elevator and throttle<sup>6</sup>.
- d) End of the avoidance: the autopilot takes control of the aircraft.

Considering that the conditions prior to the encounter have been defined in the introductory paragraph of Section 5.2, the aircraft's capability of performing a diving manoeuvre is governed by  $\tau$  and will determine whether the RPA could effectively avoid a piloted aircraft.

### **5.2.3. *Implications***

Through this study, the value of  $\tau$  is calculated from the combination of two variables that are known to affect the vertical performance of an aircraft: the elevator deflection and the throttle level. The dominating pitch-control surface is the elevator. With the elevator deflection downwards, the tail is pulled up due to an increase of lift force in the tail. This creates a nose-down pitching moment on the aircraft, decreasing the overall lift and the angle of attack (Section 2.3.5 and (Barnard & Philpott, 2010)). Initial tests conducted on general aircraft (Twin Otter) showed that the throttle level does not greatly affect the performance compared to the elevator deflection (Cereceda et al., 2018). This means that for some larger aircraft, the throttle level can be removed as a factor from the estimation of  $\tau$ . Additionally, from a practical standpoint, in a

---

<sup>6</sup>Out of the scope of this work

critical situation where the manoeuvre has to be completed in a few seconds, the throttle level is not usually changed by the pilot when the elevator deflection is controlling the performance.

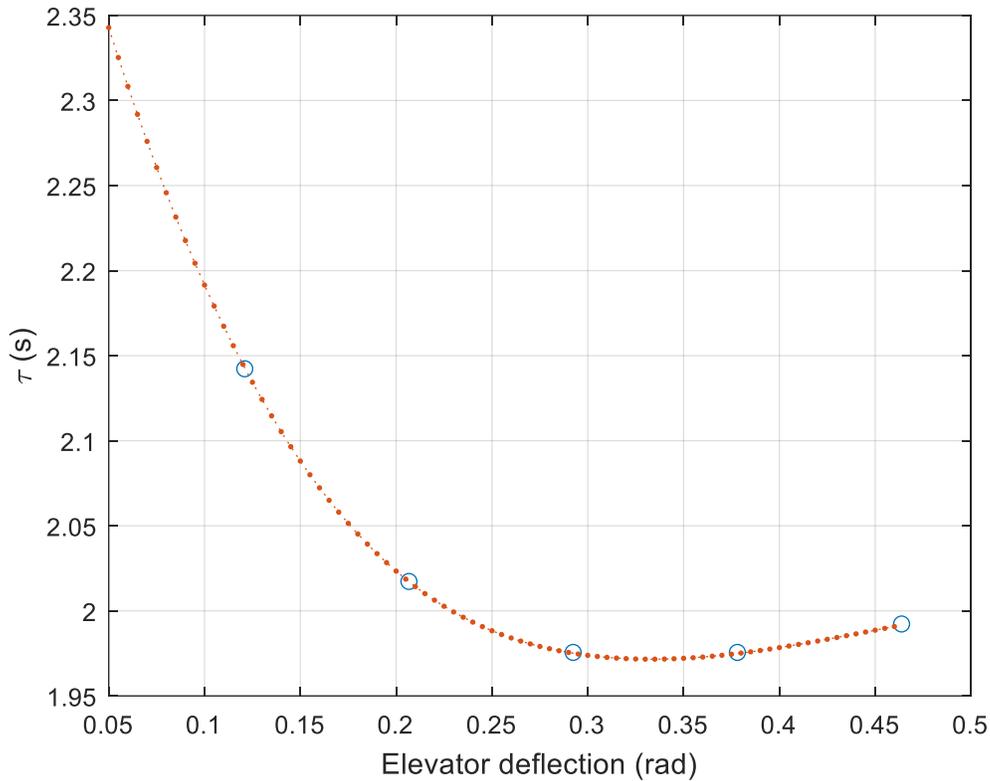
The Giant Big Stik is an overpowered RPA and the throttle level will have a significant impact on the diving rate. For this particular aircraft, the larger the throttle level, the faster the aircraft dives. This is caused by the power of its engine compared to the size, weight, and capabilities of the Giant Big Stik.

Assuming that the throttle is at the maximum level (100%), a series of simulations have been conducted (Table 5.5) to estimate the value of  $\tau$  for a range on the elevator deflection between  $4.5^\circ$  (0.0787rad) and  $26.6^\circ$  (0.4643rad):

**Table 5.5.** Simulation runs for estimating  $\tau$

Elevator deflection $\delta_e$ (rad)	Throttle level $\delta_T$ (%)	Tau $\tau$ (seconds)
0.1215	100	2.1417
0.2072		2.0167
0.2929		1.975
0.3786		1.975
0.4643		1.9917

The following figure shows the correlation between the elevator deflection ( $\delta_e$ ) and  $\tau$  with the throttle level ( $\delta_T$ ) set at 100%. A spline interpolation has been used to calculate  $\tau$  between the test points.



**Figure 5.20.** Elevator deflection vs.  $\tau$

The minimum time to dive 100ft ( $\tau$ ) is estimated to be around 2s; there is a difference of less than 0.5s between the maximum and minimum values, being trivial (Figure 5.20). While the Giant Big Stik diving rate is around 3,000fpm, the recommended descent rate of an unpressurized cabin like the Cessna 172 is no more than 500fpm (Transport Canada, 2010b). The Cessna 172 is not TCAS equipped but, compared to a larger aircraft equipped with TCAS, the Giant Big Stik doubles the diving rate of the issued Resolution Advisories (RAs) (see Appendix C – Figure C.1 and Figure C.2) (Federal Aviation Administration. U.S. Department of Transportation, 2011).

In a scenario with an encounter between a Cessna 172 and a Giant Big Stik where both are flying at the same altitude and based on the estimated diving rates, the latter would successfully avoid a

collision (assuming the context in this section). However, in the case that the intruder’s pilot identifies the RPA and also performs a diving manoeuvre, a further study is recommended.

Based on the 4 stages of the CA solving method described in Section 5.2.2, the avoidance manoeuvre analysis only focuses on the second stage. The control structure and the conditions for switching to the corresponding stage are the following (Table 5.6):

**Table 5.6.** CA stages for the avoidance study

CA stage	Control		Elevator deflection ( $\delta_e$ )	Throttle level ( $\delta_T$ )	Switch condition
	Roll axis	Pitch axis			
Initial state	Closed-loop	Closed-loop	-	-	-
Avoidance	Closed-loop	Open-loop	From 4.5° (0.0787rad) to 26.6° (0.4643rad)	100	Intruder detected + 15s
Recovery <sup>7</sup>	Closed-loop	TBD	Full range	TBD	h < 200ft
End	Closed-loop	Closed-loop	-	-	-

- a) Initial state: before any threat is detected, the aircraft flies under regular control following an orbital path as defined by the  $\Phi$  manoeuvre.
- b) Avoidance: if an intruder is detected and the pilot in command (in case there is a human-in-the-loop supervising the mission) has not taken any measures to avoid the collision after 15s, the CA system initiates a diving manoeuvre by setting the elevator deflection to a range between 4.5° and 26.6° in open-loop, while the roll remains under the control of the autopilot to permit a vertical manoeuvre. According to the results in Figure 5.20, in the case of an encounter between a Cessna and a Giant Big Stik, the latter successfully avoids the piloted aircraft due to a larger diving rate.

<sup>7</sup>An improved performance with a recovery study remains as future work

- c) Recovery7: represents the aircraft dynamic performance seconds after the aircraft has been considered to have dived down 200ft from its initial location. This is a complex study since the purpose of the full avoidance task is to not only take the aircraft out of risk but to also do it safely. It is currently out of the scope of this work.
- d) End: the task prior to the identification of the conflict point is resumed.

### **5.3. Summary**

The Giant Big Stik computer model has been adjusted for more specific vertical avoidance strategies and it is introduced as a representative RPA model with SAA applications. The avoidance task is based on current regulations and recommendations (Appendix C) established by international administrations and Transport Canada for avoiding NMAC. The main goal is to develop a fit-for-purpose FDM for estimating achievable climb/descend rates for CA manoeuvres.

This chapter has also introduced a CA methodology for solving NMAC scenarios between piloted and RPA. It included a series of 4 steps including an avoidance manoeuvre and a recovery performance. When conducting a complete CA design, it is important to first that the RPA is capable of operating an avoidance manoeuvre.

Based on the advisories given by the TCAS in commercial aircraft, a vertical avoidance manoeuvre has been suggested as a direct solution over complex trajectories. At the same time, recommendations suggest a  $2\tau+15$  directive to perform an avoidance, with  $\tau$  being the time to avoid a collision. In an NMAC, the objective is the intruder to remain out of the collision volume, which is defined by a cylindrical volume of 500ft of radius and 200ft of height. This means that  $\tau$

is defined by the time to dive 100ft in case the two aircraft involved are flying at the same altitude. However, diving 100ft is a tight solution and in a real mission, a dive of 200ft is encouraged.

Additionally, calculating  $\tau$  as the time to dive 100ft allowed for the estimation of the aircraft's diving rate, which permitted the completion of a sanity check prior to conducting further CA studies. This avoidance study has been tested with a Cessna 172 as the representative piloted aircraft and the Giant Big Stik as the RPA. For this specific case, the difference in the diving rate between the Cessna and the Giant Big Stik is significant; with 500fpm and 3,000fpm respectively. This suggests that the RPA is capable of avoiding a Cessna under this particular example.

# Chapter 6

## *Discussion, Recommendations, and Conclusion*

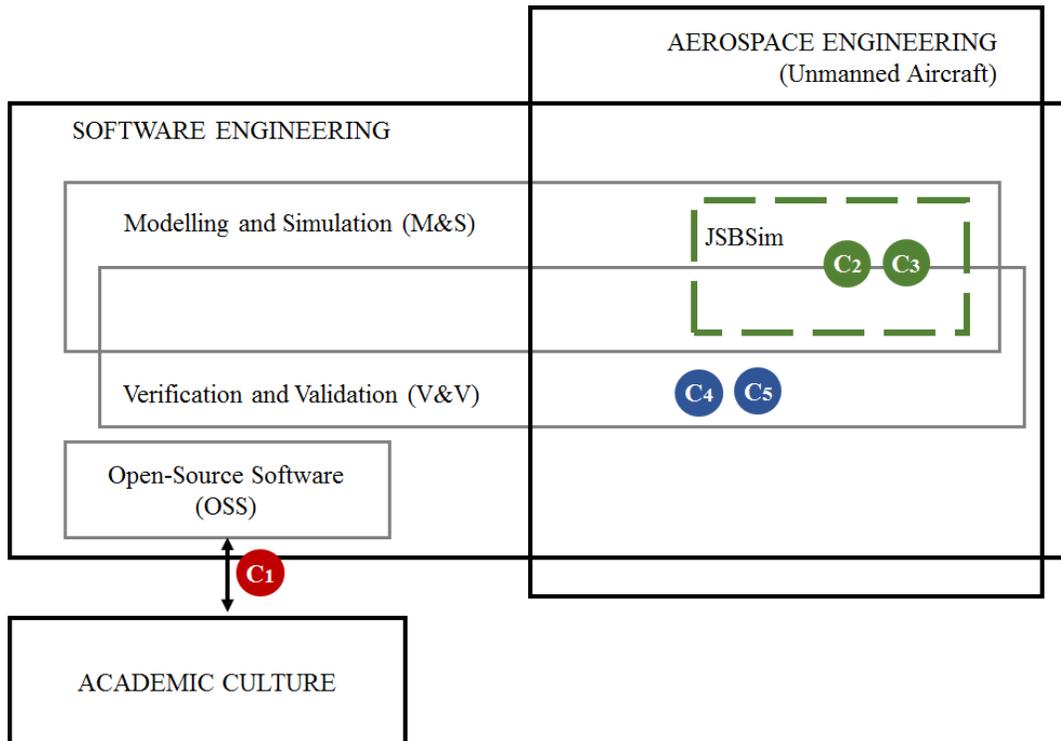
Starting with a discussion on Open-Source Software (OSS) from a graduate student perspective, followed by a methodology for validating Remotely Piloted Aircraft (RPA) computer models, and finishing with the design and validation of a representative RPA computer model in JSBSim for testing Sense and Avoid (SAA) approaches, this research has covered several fields of engineering with aerospace applications. The diverse contributions of this research are further discussed in this chapter. Additionally, a list of possibilities for future work with recommended improvements based on the findings of this research is presented, followed by the conclusions.

### **6.1. Impact and Contributions**

The ultimate goal of this research observed from a wide perspective is to assess the integration of Remotely Piloted Aircraft Systems (RPAS) into the airspace from the flight dynamics modelling point of view. The integration is analyzed by evaluating the effect of encounters between piloted and RPA where the RPA Flight Dynamics Model (FDM) is implemented.

For the representative RPA (Giant Big Stik), a computer FDM has been designed to fit the simulated environment. The computer model was previously validated by a methodology that was based on existing Verification and Validation (V&V) methods and adapted to RPAs. Due to the proprietary nature of most aerospace simulators, the open-source platform JSBSim was used. These collaborative frameworks have particular technical challenges, which have also been discussed in detail.

The fields addressed throughout this document include: (1) aerospace engineering as the application (CA with RPAS in particular), (2) software engineering, including Modelling and Simulation (M&S), V&V, and OSS, and (3) academic culture. In the following Figure 6.1, each contribution is expressed in its corresponding group and field.



**Figure 6.1.** Thesis contributions (updated from Figure 1.1)

### ***6.1.1. Open-Source Software Contribution***

The model and simulation developed in this thesis are based on OSS to avoid proprietary software. However, the project faced many difficulties when using the selected OSS, JSBSim. As an outcome, a discussion on the challenges of OSS in academia emerged, since most of the barriers were not only specific to JSBSim.

Graduate students play a significant role in the OSS loop; they evolve from regular users to contributors throughout their studies, which enriches the OSS community. They also offer an important resource for its sustainability. However, graduate students software contributions are often undervalued by the scientific community (e.g. code contributions are not taken into consideration for promotion in academia). Technical, cultural and practical challenges sometimes discourage graduate students from the continued use of OSS and contributing to their community. This thesis has provided an overview of this problem and how graduate students fit in the OSS environment. By following a series of good practices, the students can benefit from interacting and networking with other students and researchers, while the OSS community can benefit from their contribution to the software. The novelty of this discussion is that it focuses on the graduate student experience. However, this is an institutional issue and everyone in academia (graduate students, supervisors, faculty, administrators, and OSS community members) has been given a series of practices to address this issue and create a better experience for graduate students. The proposed practices would establish guidelines that will then have long-term benefits, if implemented as regular practices.

Overall, the impact of this thesis can be summarized as the following contribution:



*The powerful role of graduate students in the OSS loop: challenges and best practices for a positive experience.*

This discussion has been presented as a workshop at the ACM Canadian Celebration of Women in Computing in November 2018, and during the Teaching and Learning Conference in May 2019 at Memorial University. A formalized and extended version of the work included in this thesis, which analyzes the barriers faced by graduate students and provides a series of recommendations for different representative groups in academia, is being prepared for publication in the open-access Facets journal (Canadian Science Publishing, 2019). Additionally, a blog post on this topic has been commissioned by Nature Careers (“Nature > careers,” n.d.) and is currently under development. This research has been conducted in collaboration with Danielle Quinn, from the Biology department in the Faculty of Science at Memorial University.

### **6.1.2. JSBSim Contribution**

JSBSim is a widely used aerospace tool in academia, research, and industry. Since it is open-source, it has also become popular among enthusiasts who run the software as a flight simulator. Along with the code, there is a large online community that supports and keeps the software active. In order to use JSBSim to its full potential, it is recommended that users review the manual. However, it is expected that users will face difficulties understanding this manual, as it is currently incomplete and several sections can be improved. On top of these issues, there is no specific section for RPAs in the latest version.

With this work, two RPA computer models have been designed to meet the requirements of the research. Based on the experience and the information in the general manual, a simplified document of the JSBSim manual, focusing on the development of RPAs, was created:

“Appendix A: A simplified manual of the JSBSim open-source FDM for fixed-wing RPA applications”. This appendix is under revision to be included as a supplementary manual along with the main package (Cereceda, 2019). An extract of this thesis was also presented at the IEEE Newfoundland Electrical and Computer Engineering Conference (NECEC2017) in November 2017 (Cereceda et al., 2017).

The existing package does not include either RPA models or a manual that opens the software to their use. The impact of this work, included in Chapter 4 and Appendix A, is observed in the large community of online users who will benefit from: (1) a simplified document for the design of RPAs and (2) the two new and validated RPA computer models. JSBSim users will be able to download the manual and integrate the RPA models into their simulations.

Contributions to the JSBSim community:

**C2** *A simplified version of the current JSBSim manual for RPAs including the minimum requirements for the design of an RPA in JSBSim. The package and its application for RPAs are introduced with a case study that aims to help and guide any modellers on the RPA computer design task (Cereceda, 2019).*

**C3** *The EPP FPV and the Giant Big Stik computer models are accessible in the JSBSim online library.*

The code for both computer models is included in “Appendix D: Code and configuration files”.

### **6.1.3. Development and Validation of RPA Computer Models**

The use of the JSBSim computer model raised concerns around its reliability. Based on existing V&V methods, general aircraft validation categories, and regulatory advisories, a validation

method in tutorial format for the development of RPA computer models was developed. The initial purpose of this validation method was to present JSBSim as a trusted FDM for the development of any fixed-wing computer models. The method was later generalized, and is now used as a validation procedure for any RPA computer model.

Official existing advisories do not contemplate RPA validation methods for flight simulators. This research started with an initial analysis of the differences between piloted and remotely piloted flight simulators and listed a series of considerations for RPAS flight simulators. Based on these conclusions and existing practices for general aircraft, a validation method was developed. This method aims to initialize a framework for validating RPA computer models and, eventually, become a standardized practice. The relevance of this thesis as included in Chapter 4 with its application in Chapter 5, is that the V&V is flexible, serves in any simulation context regardless of the requirements, and it is ready for any developer to use.

Overall, the impact of this research in M&S and V&V for RPAS applications can be summarized in the following contributions:

-  *List of minimal requirements for RPAS flight simulators.*
-  *The presented validation methodology is flexible and addresses the lack of validation standards for RPAs by using straightforward methods (e.g. inspection and correlation) based on existing advisories for piloted aviation that could be improved to set up a formal standard.*

The initial approach to this issue, including the Giant Big Stik computer development in absence of flight tests, is described in “Validation discussion of an Unmanned Aerial Vehicle (UAV) using JSBSim Flight Dynamics Model compared to MATLAB/Simulink AeroSim Blockset”,

which was presented during the IEEE Systems, Man and Cybernetics Conference 2016 (SMC2016) (Cereceda et al., 2016). The complete modelling procedure, including flight tests, is published in the open-access MDPI Drones journal (Cereceda et al., 2019).

#### **6.1.4. *Collision Avoidance Application***

The application of this thesis starts with the study of extreme manoeuvres in the circumstance of an encounter between a piloted and an RPA. The encounter geometry simulated was a  $\Phi$  manoeuvre: an improved geometry that was presented at the Newfoundland Electrical and Computer Engineering Conference 2014 (NECEC2014) in collaboration with Dr. Jonathan Stevenson (Cereceda & Stevenson, 2014). The piloted aircraft is represented by a Cessna 172 while the Giant Big Stik is the representative RPA.

The Cessna 172 belongs to the category of general aviation aircraft and does not require to carry a TCAS on board that would give it the ability to identify hazards in the surrounding areas over 1,000ft. This means that the avoidance capability of the Cessna 172 (and other general aviation aircraft) relies on the pilot being able to visually detect a remotely piloted system. This problem is one consequence of the integration of the RPAs into the airspace and is one of interest since both the piloted and the remotely piloted system usually fly at low altitudes. Prior studies have approached this problem with complex methods and calculations that add delays in real-time performance.

Based on the TCAS principles of a vertical avoidance, a study evaluating the Giant Big Stik diving performance has shown that, under the circumstance of a Near Mid-Air Collision (NMAC), the RPA is around 2.5 times faster than the piloted aircraft. This means that the Giant Big Stik can effectively avoid a collision with a Cessna 172 (or any aircraft limited to a diving

rate of 500fpm). However, this statement is limited to the Giant Big Stik aircraft since the diving rate of any RPA depends on the corresponding aircraft capabilities. Therefore, no contribution is claimed in the field of SAA and collision avoidance.

## 6.2. Future Research

With this research, advancement has been made with the initiation of a series of V&V phases for the development of RPA computer models. Associated to that, the JSBSim open-source package has shown to be a successful tool for the simulation of the interaction between piloted and RPA and the design of new RPA FDM. Limited improvements have been made in the SAA application and further studies are needed in order to settle formal claims.

The following recommendations provide a series of suggestions for future research and development activities. The main research lines related to this work that are worth continuing are the improvements in the validation procedure and the avoidance approach for RPAs.

### 6.2.1. *Proposed Improvements to Open-Source Software in Academia*

Although technical difficulties while working with OSS are expected in academia, these difficulties should not add extra challenges for graduate students to conduct their research work. Chapter 3 provided an overview of the most common barriers and the means to overcome them. It is important to bring the recommended solutions in Section 3.2.5 into practice to instigate a positive graduate student experience, and the following (improvements) are suggested:

- 1- Assess the needs of graduate students in-depth:** The discussion on the challenges that the graduate students experience with OSS during their programs was based on known culture. This discussion could become a more formal study by assessing all graduate

students in an academic institution to see what disciplines are the most affected and what barriers have the worst impact. This would help narrow down the problem in order to look for more specific solutions.

- 2- **Initiate a set of guidelines on how to register and cite code with the collaboration of librarians:** Many pieces of code are lost as students graduate since research repositories mostly consist of documented content. It would be of interest to start conversations with librarians to frame a system that could contain research code (recommendations for administrators in Section 3.2.5).
- 3- **Share good coding practices among coding instructors:** Good practices in scientific computing are assumed but rarely implemented. It is important that instructors are more aware of their positive effect and share them with other peers and learners. A series of workshops for instructors would help remind the importance of these practices.

### ***6.2.2. Proposed RPAS Flight Simulator Requirements Improvements***

This thesis has provided an initial analysis of the differences between piloted and RPA flight simulators in order to list the minimal requirements for RPAS flight simulators. However, the role of the pilot needs further discussion since it is a wide topic. In future projects, **the study on the human in RPAS flight simulators could be expanded** to further evaluate and improve flight simulators requirements.

### ***6.2.3. Proposed Computer Model Validation Improvements***

Chapter 4 presented a classic tool that it was later implemented for the development and validation of RPA computer models in that same chapter and Chapter 5 with SAA applications. Currently, the literature is not clear about what the standards are for the correct design of these

particular models. This thesis aimed to initiate a discussion on this problem and to provide a framework to begin its definition. However, more effort should be made to expand this topic further.

The following improvements are proposed to promote the RPA computer model validation:

- 1- Improve V&V phases:** The methodology presented in Chapter 4 is based on classic validation techniques and basic in its definition. It lacks the specifics and the existing phases need a quantitative definition. Due to the lack of references for the development of RPA models, the aerospace community is in need of guidelines that could serve as a standard. This method was introduced here in its early development stages and it needs the specifics for its formalization from an M&S and systems point of view. Although any V&V methodology directly depends on the final application of the project and the model conditions, an expected level of similarity in the observation/inspection should be defined.
- 2- Pilot's Operating Handbook (POH) for RPAs:** When trying to define the specifics of phase 2, it was found that R/C aircraft manufacturers do not provide a similar document to a POH for piloted aviation. The specifications and expected aircraft performance were obtained from speaking with R/C pilots and from online forums. This information is based on the experience of the human pilot and lacks the details; more specific reports should be provided by the manufacturers.
- 3- Design other RPA models:** The V&V methodology has been used to model two representative aircraft that are similar in size but not in performance. This procedure could be tested for smaller and bigger aircraft in order to validate its robustness.
- 4- One factor at a time limitations:** In phase 3 of the computer model validation procedure, each command input was isolated to compare the performance of the RPA models since

the influence of the aerosurface deflections to their correspondent axis is known from the aircraft dynamics. This technique does not allow for evaluating the effect of other deflections on a particular axis. For a finer tune of the model, a different observation method should be carried out.

- 5- V&V for broader applications:** The study of SAA manoeuvres and, CA in particular, was the application of this research and the rationale behind the definition of the V&V methodology. However, it could be expanded to other methods and applications in order to test its viability and flexibility.

The technical difficulties associated with the RPA flight tests have limited the validation to a particular control range, meaning that RPA computer models can only be fully certified after extensive testing. Related to the computer model development but not with the introduced V&V methodology in Chapter 4, the following recommendations are proposed to improve the RPA computer models:

- 1- Improve the EPP FPV FDM reliability:** The EPP FPV structure and fuselage are made from flexible materials that add a random error in the transmission of the control deflection to the aero surfaces. For this particular case, the validation procedure requires more real data from the real system. A larger dataset from several missions would help to identify and correct the additional error and improve the aircraft performance in the simulation.
- 2- Improve the Giant Big Stik computer model performance in pitch:** The flight data used for validating the Giant Big Stik included changes in elevator deflection of around 25°. The flight mission was limited because of the risk associated with extreme manoeuvres. However, an experienced pilot who is familiar with the aerobatic

performance of the Giant Big Stik could conduct a flight that included changes in the elevator deflection from a steady flight. This dataset would help to fine-tune the Giant Big Stik FDM and increase its correlation coefficient.

#### **6.2.4. *Proposed Improvements to CA in RPAs***

With the same scenario and simulation environment, the RPA case in Chapter 5 showed that the Giant Big Stik provides an effective avoidance manoeuvre in an encounter with a Cessna 172 (assuming the simulation context described here only). However, additional research remains to be completed to support this finding and the next recommendations list the improvements on the issue of CA in RPAs:

- 1- Recovery analysis study:** The recovery has not been studied for the RPA case in Chapter 5 because there was greater interest in the implications of  $\tau$  and the diving time. Therefore, a further recovery study should be carried for a full study of the procedure.
- 2- Introduce the scenario where the pilot in the piloted aircraft also performs a diving manoeuvre:** The Giant Big Stik diving rate showed an excellent value compared to the one given by the Cessna 172. It is difficult to indicate that the Giant Big Stik provides a better avoidance manoeuvre since there are more encounter scenarios that have not been examined. The current study could be improved by considering the event of the piloted aircraft diving at the same time the threat is detected.
- 3- Evaluate the implications of the CA methodology with other representative RPAs:** In this document, the Giant Big Stik was selected as a reference RPA to represent its performance compared to the Cessna 172. The results showed that the RPA provided a faster manoeuvre for that particular case. However, it is worth assessing if this statement

could be extended to other RPA and piloted aircraft of similar characteristics to the Giant Big Stik and the Cessna 172 respectively.

### **6.3. Conclusion**

This research has presented a wide analysis of open-source simulations of RPAs with SAA applications. Open-source flight simulators, such as JSBSim and FlightGear, have provided the flexibility demanded by the analysis conducted. The downside of designing models on an existing open-source platform is the learning curve that comes along with it and the difficulties created on the contributor. These barriers have been analyzed from a graduate student perspective and a series of practices to improve the graduate student experience have been presented. The main objective of these recommendations is to provide a positive environment for the student to develop their research work while contributing to OSS.

Particular challenges were faced with the JSBSim open-source FDM and a simplified manual for the development of RPAs was created to answer the lack of documented guidelines. When creating the aircraft models in JSBSim, it was found that there was a lack of standards and official recommendations for the development of RPA computer models and their validation. In a way to answer this added difficulty, an introductory V&V method was proposed and successfully used for the development of two RPA models: the EPP FPV and the Giant Big Stik. This simplified method presents an easy-to-follow procedure for any designer and open to any modifications or adjustments depending on the requirements of the application. It aims to initiate a discussion about the lack of standards on this topic and possibly become a guideline in the

future. However, the technical restrictions of testing extreme manoeuvres with RPAs on the field have limited the applicability of this procedure to only the allowed test ranges.

For the study of avoidance manoeuvres as the application, this document presented a solution based on the classic approach of the TCAS in commercial aviation: in case of a threat detection, the aircraft is recommended to avoid the hazard vertically. By defining the minimum requirements for a diving avoidance manoeuvre, the effect of the factors in  $\tau$  is calculated assuming that  $\tau$  is only dependent on the aircraft's own dynamics. When the  $\tau$  evaluation was conducted on the Giant Big Stik case (RPA representative), a significant difference was observed when comparing it to the Cessna 172 (piloted aircraft representative) vertical performance: the diving time ( $\tau$ ) in the Giant Big Stik RPA largely exceeds the piloted aircraft, which denotes that the RPA provides a faster avoidance. However, this does not mean that the Giant Big Stik will successfully avoid a possible encounter with a general aircraft since there are other possible scenarios that depend on the reaction of the human pilot on the piloted aircraft.

The application has served as a “sanity check” for the study of the safe integration of RPAs into the airspace and work remains to be completed to assess whether RPAs provide a safer avoidance than piloted aircraft. Even so, the computer model developed to test SAA manoeuvres using the proposed V&V is a fitting representation of the real model (for the design signal range).

## **6.4. List of publications**

The following include a list of papers, peer-reviewed manuscripts, and reports related to this thesis:

### ***Journal publications***

1. Oihane Cereceda and Danielle Quinn, *A Graduate Student Perspective on Overcoming Barriers to Interacting with Open-Source Software*, accepted for publication on the Facets open-access journal in November 2019.
2. Oihane Cereceda, Luc Rolland, and Siu O'Young, *Giant Big Stik R/C UAV computer model development in JSBSim for sense and avoid applications*, published on the MDPI Drones open-access journal in June 2019.

### ***Conference publications***

3. Oihane Cereceda, *Collision Avoidance Methodology for Unmanned Aerial Vehicles and the Giant Big Stik as a Case Study*, presented at the 28<sup>th</sup> Newfoundland Electrical and Computer Engineering Conference in November 2019 (St. John's, Canada).
4. Oihane Cereceda, *A Survey of Collision Avoidance Methods for Unmanned Aircraft Systems*, presented at the 27<sup>th</sup> Newfoundland Electrical and Computer Engineering Conference in November 2018 (St. John's, Canada).
5. Oihane Cereceda, Luc Rolland, and Siu O'Young, *Vertical avoidance and recovery analysis of a general aircraft in near mid-air collision scenarios using design and analysis of computer experiments*, presented at the 31<sup>st</sup> Canadian Conference on Electrical and Computer Engineering in May 2018 (Québec City, Canada).
6. Oihane Cereceda, Luc Rolland, and Siu O'Young, *JSBSim open source Flight Dynamics Model for fixed-wing Unmanned Aerial Vehicle applications*, presented at the 26<sup>th</sup> Newfoundland Electrical and Computer Engineering Conference in November 2017 (St. John's, Canada).

7. Oihane Cereceda, Luc Rolland, and Siu O'Young, *Validation discussion of an Unmanned Aerial Vehicle (UAV) using JSBSim Flight Dynamics Model compared to MATLAB/Simulink AeroSim Blockset*, presented at the IEEE Systems, Man, and Cybernetics Conference in October 2016 (Budapest, Hungary).
8. Jonathan D. Stevenson and Oihane Cereceda, *A Simulated Environment for Testing 4D Detect See and Avoid Scenarios for UAVs*, presented at the 23<sup>rd</sup> Newfoundland Electrical and Computer Engineering Conference in November 2014 (St. John's, Canada).

***Technical reports***

9. Oihane Cereceda, *Coefficients Calculation for the EPP FPV R/C Aircraft*, technical report, Memorial University of Newfoundland, May 2019.
10. Oihane Cereceda, *A Simplified Manual of the JSBSim Open-Source Software FDM for Fixed-Wing UAV Applications*, technical report, Memorial University of Newfoundland, May 2019.

# References

- Adaska, J. W., Obermeyer, K., & Schmidt, E. (2014). Robust probabilistic conflict prediction for sense and avoid. *2014 American Control Conference*, 1198–1203.  
<https://doi.org/10.1109/ACC.2014.6859435>
- Allerton, D. (2009). *Principles of Flight Simulation*.  
<https://doi.org/10.1017/CBO9781107415324.004>
- Alligier, R., Allignol, C., Barnier, N., Durand, N., & Wang, R. (2018). Detect and Avoid Algorithm for UAS with 3D-Maneuvers. *International Conference on Research in Air Transportation 2018*. Retrieved from [www.icrat.org](http://www.icrat.org)
- ArduPilot documentation. (2017). Retrieved April 21, 2017, from <http://ardupilot.org/ardupilot/index.html>
- Artacho, B. (2018). *Unmanned Aircraft System (UAS) Integration to Airspace and Collision Risk Assessment* (Memorial University of Newfoundland). Retrieved from <https://research.library.mun.ca/13082/1/thesis.pdf>
- Bacharakis, C. (2018). Cracking the Code — how Mozilla is helping university students contribute to Open Source. Retrieved December 5, 2019, from <https://medium.com/mozilla-open-innovation/cracking-the-code-how-mozilla-is-helping-university-students-contribute-to-open-source-25fa630d8c5c>
- Baker, C. A. B., Ramchurn, S., Teacy, W. T. L., & Jennings, N. R. (2016). Planning Search and Rescue Missions for UAV Teams. *Conference on Prestigious Applications of Intelligent Systems at ECAI 2016, The Hague, NL, 31 Aug - 02 Sep 2016*. IOS Press, 1–6.
- Balachandran, S., & Atkins, E. (2017). Markov Decision Process Framework for Flight Safety

- Assessment and Management. *Journal of Guidance, Control, and Dynamics*, 40(4), 817–830. <https://doi.org/10.2514/1.G001743>
- Barnard, R. H., & Philpott, D. R. (2010). *Aircraft flight. A description of the physical principles of aircraft flight* (4th ed.). Pearson Education.
- Barnes, N. (2010, October 14). Publish your computer code: It is good enough. *Nature*, Vol. 467, p. 753. <https://doi.org/10.1038/467753a>
- Berndt, J. S., & JSBSim Development Team. (2011). *JSBSim, An open source, platform-independent, flight dynamics model in C++*.
- Bharati, S. P., Wu, Y., Sui, Y., Padgett, C., & Wang, G. (2018). Real-Time Obstacle Detection and Tracking for Sense-and-Avoid Mechanism in UAVs. *IEEE Transactions on Intelligent Vehicles*, 3(2), 185–197. <https://doi.org/10.1109/tiv.2018.2804166>
- Boivin, C. (2017). A first in Canada: Drone collides with passenger plane above Quebec City airport. Retrieved August 16, 2018, from CBC News website:  
<http://www.cbc.ca/news/canada/montreal/garneau-airport-drone-quebec-1.4355792>
- Bowlick, F. J., Goldberg, D. W., & Bednarz, S. W. (2017). Computer Science and Programming Courses in Geography Departments in the United States. *Professional Geographer*, 69(1), 138–150. <https://doi.org/10.1080/00330124.2016.1184984>
- Brooker, P., & Wo, Y. (2017). Introducing Unmanned Aircraft Systems into a High Reliability ATC System. *THE JOURNAL OF NAVIGATION*, 66, 719–735.  
<https://doi.org/10.1017/S0373463313000337>
- Buchholz, J. J., Bauschat, J.-M., Hahn, K. U., & Pausder, H. J. (1996). ATTAS & ATTheS In-Flight Simulators. Recent Application Experiences and Future Programs. In *Proceedings of the NATO AGARD Symposium on flight simulation – Where are the challenges?* (AGARD-

- CP-577). Retrieved from <http://www.dtic.mil/cgi-bin/GetTRDoc?AD=ADA310479#page=36>
- Burning Glass Technologies. (2016). *Beyond Point and Click the Expanding Demand for Coding Skills*. Retrieved from [www.burning-glass.com](http://www.burning-glass.com)
- Canadian Aviation Regulations. (2018). Retrieved August 15, 2018, from <http://laws-lois.justice.gc.ca/eng/regulations/SOR-96-433/page-1.html#h-3>
- Canadian Science Publishing. (2019). *FACETS*.
- Carey, M. A., & Papin, J. A. (2018). Ten simple rules for biologists learning to program. *PLOS Computational Biology*, *14*(1), e1005871. <https://doi.org/10.1371/journal.pcbi.1005871>
- Caris, M., Stanko, S., Palm, S., Sommer, R., & Pohl, N. (2015). Synthetic aperture radar at millimeter wavelength for UAV surveillance applications. *2015 IEEE 1st International Forum on Research and Technologies for Society and Industry Leveraging a Better Tomorrow (RTSI)*, 349–352. <https://doi.org/10.1109/RTSI.2015.7325145>
- Cereceda, O. (2019). *A Simplified Manual of the JSBSim Open-Source Software FDM for Fixed-Wing UAV Applications*. Retrieved from Faculty of Engineering and Applied Science, Memorial University of Newfoundland website: <https://research.library.mun.ca/13798/>
- Cereceda, O., Rolland, L., & O'Young, S. (2016). Validation discussion of an Unmanned Aerial Vehicle (UAV) using JSBSim Flight Dynamics Model compared to MATLAB/Simulink AeroSim Blockset. *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 3989–3994. <https://doi.org/10.1109/SMC.2016.7844857>
- Cereceda, O., Rolland, L., & O'Young, S. (2017, November 15). *JSBSim Open-Source Flight Dynamics Model for Fixed-Wing Unmanned Aerial Vehicle Applications*. Retrieved from <https://research.library.mun.ca/13801/>

- Cereceda, O., Rolland, L., & O'Young, S. (2018). Vertical avoidance and recovery analysis of a general aircraft in near mid-air collision scenarios using design and analysis of computer experiments. *IEEE Canadian Conference on Electrical and Computer Engineering (CCECE2018)*, 2–5.
- Cereceda, O., Rolland, L., & O'Young, S. (2019). Giant Big Stik R/C UAV Computer Model Development in JSBSim for Sense and Avoid Applications. *Drones*, 3(2), 48.  
<https://doi.org/10.3390/drones3020048>
- Cereceda, O., & Stevenson, J. D. (2014, November 3). *A Simulated Environment for Testing 4D Detect See and Avoid Scenarios for UAVs*. Retrieved from <https://research.library.mun.ca/13799/>
- Chanel, C. P. C., Teichteil-Königsbuch, F., & Lesire, C. (2012). POMDP-based online target detection and recognition for autonomous UAVs. *Frontiers in Artificial Intelligence and Applications*, 242, 955–960. <https://doi.org/10.3233/978-1-61499-098-7-955>
- Chen, R. T. N. (1989). *A Survey of Nonuniform Inflow Models for Rotorcraft Flight Dynamics and Control Applications*. Retrieved from <https://ntrs.nasa.gov/search.jsp?R=19900006622>
- Cook, S. P., Brooks, D., Cole, R., Hackenberg, D., & Raska, V. (2015). Defining Well Clear for Unmanned Aircraft Systems. *AIAA Infotech @ Aerospace*. <https://doi.org/10.2514/6.2015-0481>
- Cooper, G. E., & Harper, R. P. (1969). *The use of pilot training in the evaluation of aircraft handling qualities*. Retrieved from <https://ntrs.nasa.gov/search.jsp?R=19690013177>
- Crichton, D. (2018). Open source sustainability. Retrieved September 12, 2018, from TechCrunch website: <https://techcrunch.com/2018/06/23/open-source-sustainability/>
- Desaraju, V., & Michael, N. (2014). Vision-based Landing Site Evaluation and Trajectory

Generation Toward Rooftop Landing. *Rss*. <https://doi.org/10.15607/rss.2014.x.044>

Diston, D. J. (2010). *Computational modelling and simulation of aircraft and the environment*.

Retrieved from

[https://books.google.ca/books/about/Computational\\_Modelling\\_and\\_Simulation\\_o.html?id=0v2hQwAACAAJ&redir\\_esc=y&hl=en](https://books.google.ca/books/about/Computational_Modelling_and_Simulation_o.html?id=0v2hQwAACAAJ&redir_esc=y&hl=en)

Dunn, C. (2018). Unauthorized drone forces wildfire-fighting aircraft away from B.C. blaze.

Retrieved August 16, 2018, from CBC News website:

<https://www.cbc.ca/news/canada/british-columbia/unauthorized-drone-forces-wildfire-fighting-aircraft-away-from-b-c-blaze-1.4759529>

Durham, W. (2013). *Aircraft Flight Dynamics and Control*. Wiley.

Fang, S. X. (2014). *UAV 4D Synchronization*.

Fang, S. X. (2018). *Risk-based Supervisory Guidance for Detect and Avoid involving Small*

*Unmanned Aircraft Systems* (Memorial University of Newfoundland). Retrieved from

<https://research.library.mun.ca/13204/1/thesis.pdf>

Fasano, G., Accado, D., Moccia, A., & Moroney, D. (2016). Sense and Avoid for Unmanned Aircraft Systems. *IEEE Aerospace and Electronic Systems Magazine*, (10).

<https://doi.org/10.1117/12.720867>

Federal Aviation Administration. U.S. Department of Transportation. (2011). *Introduction to TCAS II*.

Federal Aviation Administration. (2013a). *Integration of Civil Unmanned Aircraft Systems (UAS) in the National Airspace System (NAS) Roadmap*. 74.

Federal Aviation Administration. (2013b). *Sense and Avoid (SAA) for Unmanned Aircraft Systems (UAS). Second Caucus Workshop Report*.

- Federal Aviation Administration. (2019). Flight Simulation Training Device Qualification Guidance – Advisory Circulars (AC). Retrieved August 28, 2018, from <https://www.faa.gov/about/initiatives/nsp/ac/>
- Finn, A., & Franklin, S. (2011). Acoustic sense & avoid for UAV's. *2011 Seventh International Conference on Intelligent Sensors, Sensor Networks and Information Processing*, 586–589. <https://doi.org/10.1109/ISSNIP.2011.6146555>
- Flight Simulator : Plane Pilot - Microsoft. (2018). Retrieved December 9, 2018, from <https://www.microsoft.com/en-us/p/flight-simulator-plane-pilot/9wzdnrcd8s11?activetab=pivot:overviewtab>
- FlightGear Flight Simulator. (2019). Retrieved October 9, 2017, from <http://www.flightgear.org/>
- Forsberg, K., & Mooz, H. (1991). The Relationship of System Engineering to the Project Cycle. *INCOSE International Symposium*, 1(1), 57–65. <https://doi.org/10.1002/j.2334-5837.1991.tb01484.x>
- Galbraith, B. (2010). *JSBSim Script Tutorial 1*. Retrieved from [http://www.holycows.net/JSBSim\\_Script\\_Tutorial.pdf](http://www.holycows.net/JSBSim_Script_Tutorial.pdf)
- Gardner, R. W., Genin, D., McDowell, R., Rouff, C., Saksena, A., & Schmidt, A. (2016). Probabilistic model checking of the next-generation airborne collision avoidance system. *2016 IEEE/AIAA 35th Digital Avionics Systems Conference (DASC)*, 1–10. <https://doi.org/10.1109/DASC.2016.7777963>
- Gazebo. (2014). Retrieved April 28, 2017, from <http://gazebo.org/>
- Geiger, R. S., Varoquaux, N., Mazel-Cabasse, C., & Holdgraf, C. (2018). The Types, Roles, and Practices of Documentation in Data Analytics Open Source Software Libraries: A Collaborative Ethnography of Documentation Work. *Computer Supported Cooperative*

Work: *CSCW: An International Journal*, 27(3–6), 767–802. <https://doi.org/10.1007/s10606-018-9333-1>

Gnuplot. (n.d.). Retrieved September 20, 2018, from <http://www.gnuplot.info/>

Goldberg, B. E., Everhart, K., Stevens, R., Babbitt III, N., Clemens, P., & Stout, L. (1994). System Engineering “ Toolbox ” for Design-Oriented Engineers. *NASA Reference Publication 1358*, (December).

Gong, M., De Marco, A., & Berndt, J. S. (n.d.). JSBSim Commander. Retrieved December 6, 2017, from <http://jsbsimcommander.sourceforge.net/>

Government of Canada. (2019). Canadian Aviation Regulations SOR/96-433. Retrieved from <https://laws-lois.justice.gc.ca/eng/regulations/SOR-96-433/FullText.html#s-900.01>

Great Planes. (2005a). *Giant Big Stik ARF*. Retrieved from <https://www.greatplanes.com/airplanes/gpma1224.php>

Great Planes. (2005b). *Giant Big Stik ARF Instruction Manual*. Retrieved from <http://manuals.hobbico.com/gpm/gpma1224-manual.pdf>

Great Planes. (2005c). *Giant Big Stik ARF Instruction Manual*. (217), 13–15.

Griffith, J., Kochenderfer, M., & Kuchar, J. (2008). Electro-Optical System Analysis for Sense and Avoid. *AIAA Guidance, Navigation and Control Conference and Exhibit*.

<https://doi.org/10.2514/6.2008-7253>

Gupta, L., Jain, R., & Vaszkun, G. (2016). Survey of Important Issues in UAV Communication Networks. *IEEE Communications Surveys & Tutorials*, 18(2), 1123–1152.

<https://doi.org/10.1109/COMST.2015.2495297>

Heppe, S. B. (2015). Problem of UAV Communications. In *Handbook of Unmanned Aerial Vehicles* (pp. 715–748). [https://doi.org/10.1007/978-90-481-9707-1\\_30](https://doi.org/10.1007/978-90-481-9707-1_30)

HobbyKing.com. (n.d.). EPP-FPV Instructions. Retrieved April 21, 2017, from <https://hobbyking.com/media/file/163971972X102667X57.pdf>

Institution of Electrical Engineers., R., Drolet, G., & Bray, J. R. (2017). IEE proceedings. Radar, sonar, and navigation. In *IET Radar, Sonar & Navigation* (Vol. 11). Retrieved from <http://digital-library.theiet.org/content/journals/10.1049/iet-rsn.2016.0520>

Jackson, E. B. (1995). Manual for a Workstation-based Generic Flight Simulation Program ( LaRCsim ) Version 1 . 4. *Nasa Technical Memorandum 110164*.

Journal of Open Research Software. (2018). <https://doi.org/10.5334/jors.223>

JSBSim - FlightGear wiki. (2018). Retrieved April 21, 2017, from <http://wiki.flightgear.org/JSBSim>

JSBSim Development Team. (2001). JSBSim Flight Dynamics Model download. Retrieved September 20, 2018, from <https://sourceforge.net/projects/jsbsim/>

JSBSim Development Team. (2005a). Aeromatic for the JSBSim Open Source Flight Dynamics Model. Retrieved April 21, 2017, from <http://jsbsim.sourceforge.net/aeromatic2.html>

JSBSim Development Team. (2005b). JSBSim Open Source Flight Dynamics Model. Retrieved October 9, 2017, from <http://jsbsim.sourceforge.net/>

JSBSim Development Team. (2017). JSBSim Flight Dynamics Model: JSBSim. Retrieved October 9, 2017, from <http://jsbsim.sourceforge.net/JSBSim/>

JSBSim Development Team. (2018). JSBSim Flight Dynamics Model - Browse Files at SourceForge.net. Retrieved June 10, 2019, from <https://github.com/JSBSim-Team/jsbsim>

JSBSim Flight Dynamics Model - Code aircraft/c172x. (2009). Retrieved September 18, 2018, from <https://sourceforge.net/p/jsbsim/code/ci/master/tree/aircraft/c172x/>

JSBSim Flight Dynamics Model - Code aircraft/DHC6. (n.d.). Retrieved September 18, 2018,

- from <https://sourceforge.net/p/jsbsim/code/ci/master/tree/aircraft/DHC6/>
- JSBSim Flight Dynamics Model - Code aircraft/minisgs. (2016). Retrieved December 17, 2018, from <https://sourceforge.net/p/jsbsim/code/ci/master/tree/aircraft/minisgs/>
- Ke, Y., Wang, K., & Chen, B. M. (2018). Design and Implementation of a Hybrid UAV with Model-Based Flight Capabilities. *IEEE/ASME Transactions on Mechatronics*, 23(3), 1114–1125. <https://doi.org/10.1109/TMECH.2018.2820222>
- Kesteloo, H. (2018). Suspected drone collision with Cessna 172 causes \$4,000 in damage. Retrieved August 16, 2018, from DroneDJ website: <https://dronedj.com/2018/02/23/drone-collision-with-cessna-172/>
- Kochenderfer, M. J., & Chryssanthacopoulos, J. P. (2013). Collision Avoidance Using Partially Controlled Markov Decision Processes. *Agents and Artificial Intelligence. ICAART 2011. Communications in Computer and Information Science*. Retrieved from [https://link.springer.com/content/pdf/10.1007%2F978-3-642-29966-7\\_6.pdf](https://link.springer.com/content/pdf/10.1007%2F978-3-642-29966-7_6.pdf)
- Kochenderfer, M. J., Holland, J. E., & Chryssanthacopoulos, J. P. (2012). *Next-Generation Airborne Collision Avoidance System*. Retrieved from <http://www.dtic.mil/docs/citations/AD1014875>
- Kozuba, J. (2011). Impact of human factor on likelihood of aircraft accident. *Archives of Transport System Telematics*, Vol. 4, iss. 2, 29–36. Retrieved from <http://yadda.icm.edu.pl/baztech/element/bwmeta1.element.baztech-article-BSL7-0054-0014>
- Lakhan, S. E., & Jhunjhunwala, K. (2008). Open Source Software in Education. *EDUCAUSE Quarterly Magazine*, 31(2), 32–40. Retrieved from <http://www.educause.edu/EDUCAUSE+Quarterly/EDUCAUSEQuarterlyMagazineVolum/OpenSourceSoftwareinEducation/162873>

- Law, A. M. (2014). *Simulation modeling and analysis*. Retrieved from [https://books.google.ca/books?id=5clDnAEACAAJ&source=gbs\\_book\\_other\\_versions](https://books.google.ca/books?id=5clDnAEACAAJ&source=gbs_book_other_versions)
- List, M., Ebert, P., & Albrecht, F. (2017). Ten Simple Rules for Developing Usable Software in Computational Biology. *PLOS Computational Biology*, *13*(1), e1005265. <https://doi.org/10.1371/journal.pcbi.1005265>
- Liu, M., Egan, G. K., & Santoso, F. (2015). Modeling, Autopilot Design, and Field Tuning of a UAV with Minimum Control Surfaces. *IEEE Transactions on Control Systems Technology*, *23*(6), 2353–2360. <https://doi.org/10.1109/TCST.2015.2398316>
- Londner, E. H., & Moss, R. J. (2017). A Bayesian Network Model of Pilot Response to TCAS Resolution Advisories. In *Europe Air Traffic Management Research and Development Seminar*. Retrieved from [http://www.atmseminarus.org/seminarContent/seminar12/papers/12th\\_ATM\\_RD\\_Seminar\\_paper\\_46.pdf](http://www.atmseminarus.org/seminarContent/seminar12/papers/12th_ATM_RD_Seminar_paper_46.pdf)
- M.V. Cook. (2007). *Flight Dynamics Principle* (2nd ed.).
- Marston, M., Operations, N. A., & Baca, G. (2015). *ACAS-Xu / Initial Self-Separation Flight Tests Flight Test Report*. Retrieved from <https://ntrs.nasa.gov/search.jsp?R=20150008347>
- Mccallie, D., Butts, J., & Mills, R. (2011). Security analysis of the ADS-B implementation in the next generation air transportation system. *International Journal of Critical Infrastructure Protection*, *4*, 78–87. <https://doi.org/10.1016/j.ijcip.2011.06.001>
- Mcfadyen, A., & Mejias, L. (2016, January 1). A survey of autonomous vision-based See and Avoid for Unmanned Aircraft Systems. *Progress in Aerospace Sciences*, Vol. 80, pp. 1–17. <https://doi.org/10.1016/j.paerosci.2015.10.002>
- McGovern, S. M. (2007). Categories for classification of aircraft flight model validation.

*AIAA/IEEE Digital Avionics Systems Conference.*

<https://doi.org/10.1109/DASC.2007.4391961>

Melnyk, R., Schrage, D., Volovoi, V., & Jimenez, H. (2014). Sense and avoid requirements for unmanned aircraft systems using a target level of safety approach. *Risk Analysis*, 34(10), 1894–1906. <https://doi.org/10.1111/risa.12200>

Minitab. (2019). Retrieved April 21, 2017, from <http://www.minitab.com/en-us/>

Mueller, E. R., Isaacson, D. R., & Stevens, D. (2016). *Air Traffic Controller Acceptability of Unmanned Aircraft System Detect-and-Avoid Thresholds*. Retrieved from <http://www.sti.nasa.gov>

Mueller, M., Smith, N., & Ghanem, B. (2016). A benchmark and simulator for UAV tracking. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 9905 LNCS, 445–461. [https://doi.org/10.1007/978-3-319-46448-0\\_27](https://doi.org/10.1007/978-3-319-46448-0_27)

Murray-Smith, D. J. (2012). *Modelling and simulation of integrated systems in engineering : issues of methodology, quality, testing and application*. Retrieved from [https://books.google.ca/books?id=OX9wAgAAQBAJ&lpg=PP1&pg=PP1&redir\\_esc=y#v=onepage&q&f=false](https://books.google.ca/books?id=OX9wAgAAQBAJ&lpg=PP1&pg=PP1&redir_esc=y#v=onepage&q&f=false)

Nangia, U., & Katz, D. S. (2017). Track 1 Paper: Surveying the US National Postdoctoral Association Regarding Software Use and Training in Research. *Copyright Nangia and Katz*. <https://doi.org/10.6084/m9.figshare.5328442>

Narkawicz, A., Muñoz, C. A., & Dutle, A. (2016). Coordination Logic for Repulsive Resolution Maneuvers. *16th AIAA Aviation Technology, Integration, and Operations Conference*. <https://doi.org/10.2514/6.2016-3156>

- National Museum of the US Air Force™. (2015). Kettering Aerial Torpedo “Bug.” Retrieved August 12, 2019, from <https://www.nationalmuseum.af.mil/Visit/Museum-Exhibits/Fact-Sheets/Display/Article/198095/kettering-aerial-torpedo-bug/>
- Nature > careers. (n.d.). Retrieved March 12, 2019, from Nature website: <https://www.nature.com/careers>
- Newcome, L. R. (2005). *Unmanned aviation : a brief history of unmanned aerial vehicles*. Retrieved from [https://books.google.ca/books/about/Unmanned\\_Aviation.html?id=EnqjAwAAQBAJ&redir\\_esc=y](https://books.google.ca/books/about/Unmanned_Aviation.html?id=EnqjAwAAQBAJ&redir_esc=y)
- Nikolai, B. (1999). Open Source Software Development as a Special Type of Academic Research (Critique of Vulgar Raymondism). *First Monday*, 4(10).
- OpenEagles. (n.d.). Retrieved October 19, 2017, from <http://www.openeagles.org/wiki/doku.php?id=start>
- Ott, J. T. (2015). Well Clear: General Aviation and Commercial Pilot’s Perception of Unmanned Aerial Vehicles in the National Airspace System. *Human Factors and Ergonomics Society 59th Annual Meeting*. <https://doi.org/10.1177/1541931215591013>
- Pajares, G. (2015). Overview and Current Status of Remote Sensing Applications Based on Unmanned Aerial Vehicles (UAVs). *Photogrammetric Engineering & Remote Sensing*, 81(4), 281–330. <https://doi.org/10.14358/PERS.81.4.281>
- Pandey, R. K., & Tiwari, V. (2011). *Reliability Issues in Open Source Software*. 34(1), 34–38. <https://doi.org/10.5120/4065-5849>
- Perkel, J. (2016). Democratic databases: Science on GitHub. *Nature*, 538(7623), 127–128. <https://doi.org/10.1038/538127a>

- Piccolo II Autopilot. (n.d.). Retrieved September 13, 2018, from Cloud Cap Technology website:  
<http://www.cloudcaptech.com/products/detail/piccolo-ii>
- Prabhu, P., Zhang, Y., Ghosh, S., August, D. I., Huang, J., Beard, S., ... Walker, D. (2011). A survey of the practice of computational science. *State of the Practice Reports on - SC '11*, 1. <https://doi.org/10.1145/2063348.2063374>
- Ramasamy, S., & Sabatini, R. (2015). A unified approach to cooperative and non-cooperative Sense-and-Avoid. *2015 International Conference on Unmanned Aircraft Systems, ICUAS 2015*. <https://doi.org/10.1109/ICUAS.2015.7152360>
- Ramasamy, S., Sabatini, R., Gardi, A., & Liu, J. (2016). LIDAR obstacle warning and avoidance system for unmanned aerial vehicle sense-and-avoid. *Aerospace Science and Technology*, 55, 344–358. <https://doi.org/10.1016/J.AST.2016.05.020>
- Ray Young, & Brenton, S. (2016). Establishing baseline requirements for a UAS ground-based sense and avoid system. *2016 Integrated Communications Navigation and Surveillance (ICNS)*, 8D4-1-8D4-10. <https://doi.org/10.1109/ICNSURV.2016.7486385>
- Raymond, E. S. (2001). *The cathedral and the bazaar : musings on Linux and open source by an accidental revolutionary*. Retrieved from [https://books.google.ca/books/about/The\\_Cathedral\\_the\\_Bazaar.html?id=F6qgFtLwpJgC&redir\\_esc=y](https://books.google.ca/books/about/The_Cathedral_the_Bazaar.html?id=F6qgFtLwpJgC&redir_esc=y)
- RCGroups. (2017). Remote Control, Radio Control Planes, Drones, Cars and Boats. Retrieved April 23, 2017, from <https://www.rcgroups.com/forums/index.php>
- RCGroups. (2018). Great Planes Giant Big Stik XL 55-61cc Gas/EP ARF - RCGroups Review - RC Groups. Retrieved April 19, 2018, from <https://www.rcgroups.com/forums/showthread.php?3000589-Great-Planes-Giant-Big-Stik->

XL-55-61cc-Gas-EP-ARF-RCGroups-Review

- Rhodes, D. (2017). Ground based sense and avoid key piece of the BLOS puzzle. *2017 Integrated Communications, Navigation and Surveillance Conference (ICNS)*, 1–15.  
<https://doi.org/10.1109/ICNSURV.2017.8012050>
- Roads and Bridges: History and Background of Digital Infrastructure*. (n.d.). Retrieved from  
<https://www.fordfoundation.org/media/2976/roads-and-bridges-the-unseen-labor-behind-our-digital-infrastructure.pdf>
- Rodriguez-Fernandez, V., Menendez, H. D., & Camacho, D. (2015). Design and development of a lightweight multi-UAV simulator. *Proceedings - 2015 IEEE 2nd International Conference on Cybernetics, CYBCONF 2015*, 255–260.  
<https://doi.org/10.1109/CYBConf.2015.7175942>
- ROS. (2008). Retrieved April 28, 2017, from <http://www.ros.org/>
- Sahawneh, L. R., Mackie, J., Spencer, J., Beard, R. W., & Warnick, K. F. (2015). Airborne Radar-Based Collision Detection and Risk Estimation for Small Unmanned Aircraft Systems. *Journal of Aerospace Information Systems*, *12*(12), 756–766.  
<https://doi.org/10.2514/1.I010284>
- Salas, E., Maurino, D., & Curtis, M. (2010). Human Factors in Aviation. In *Human Factors in Aviation* (Second edi). <https://doi.org/10.1016/B978-0-12-374518-7.00001-8>
- Selig, M. S., Deters, R., & Dimock, G. (2002). Aircraft Dynamic Models for Use with FlightGear. Retrieved April 28, 2017, from <http://m-selig.ae.illinois.edu/apasim/Aircraft-uiuc.html>
- Shaheen E. Lakhan, K. J. (2008). *Open-Source Software in Education*. Retrieved from <https://er.educause.edu/articles/2008/5/open-source-software-in-education>

- Shevell, R. S. (1989). *Fundamentals of flight*. Retrieved from [https://books.google.ca/books?id=9MOQQgAACAAJ&dq=Fundamentals+of+flight&hl=en&sa=X&ved=0ahUKEwjH3sDJtrbTAhVE\\_IMKHZ2FAdEQ6AEIKTAB](https://books.google.ca/books?id=9MOQQgAACAAJ&dq=Fundamentals+of+flight&hl=en&sa=X&ved=0ahUKEwjH3sDJtrbTAhVE_IMKHZ2FAdEQ6AEIKTAB)
- SimplePlanes. (2019). SGS-126 N5702S Mini Glider. Retrieved December 17, 2018, from <https://www.simpleplanes.com/a/kpSAvF/SGS-126-N5702S-Mini-Glider>
- SketchUp. (2019). Retrieved April 19, 2018, from <https://www.sketchup.com/>
- Smith, A. M., Katz, D. S., & Niemeyer, K. E. (2016). Software citation principles. *PeerJ Computer Science*, 2, e86. <https://doi.org/10.7717/peerj-cs.86>
- Smith, A. M., Niemeyer, K. E., Katz, D. S., Barba, L. A., Githinji, G., Gymrek, M., ... Vanderplas, J. T. (2018). Journal of Open Source Software (JOSS): design and first-year review. *PeerJ Computer Science*. <https://doi.org/10.7717/peerj-cs.147>
- Smith, R. (2001). Open Dynamics Engine. Retrieved April 28, 2017, from <http://www.ode.org/>
- SoftwareX. (2019). Retrieved September 12, 2018, from SoftwareX website: <https://www.journals.elsevier.com/softwarex>
- St. John's Historical Wind Speed. (n.d.). Retrieved December 17, 2018, from [https://stjohns.weatherstats.ca/metrics/wind\\_speed.html](https://stjohns.weatherstats.ca/metrics/wind_speed.html)
- Starmetro Staff. (2018). WestJet flight has close call with drone while en route to Edmonton airport Tuesday. Retrieved August 16, 2018, from The Star website: <https://www.thestar.com/edmonton/2018/08/01/westjet-flight-has-close-call-with-drone-while-en-route-to-edmonton-airport-tuesday.html>
- Stegagno, P., Basile, M., Bulthoff, H. H., & Franchi, A. (2014). A semi-autonomous UAV platform for indoor remote operation with visual and haptic feedback. *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 3862–3869.

<https://doi.org/10.1109/ICRA.2014.6907419>

Stevens, B. L., Lewis, F. L., & Johnson, E. N. (1992). *Aircraft control and simulation*. Retrieved from

[https://books.google.ca/books?id=XIabCgAAQBAJ&printsec=frontcover&dq=Aircraft+control+and+simulation&hl=en&sa=X&ved=0ahUKEwjg-\\_mntrbTAhVC2oMKHY6bCBoQ6AEIJDA#v=onepage&q=Aircraft control and simulation&f=false](https://books.google.ca/books?id=XIabCgAAQBAJ&printsec=frontcover&dq=Aircraft+control+and+simulation&hl=en&sa=X&ved=0ahUKEwjg-_mntrbTAhVC2oMKHY6bCBoQ6AEIJDA#v=onepage&q=Aircraft%20control%20and%20simulation&f=false)

Stevenson, J. D. (2013). *Small UAV 4D Simulation in MATLAB/Simulink and FlightGear. User Description Document*.

Stevenson, J. D. (2015). *Assessment of the equivalent level of safety requirements for small unmanned aerial vehicles*. Memorial University of Newfoundland.

Stevenson, Jonathan D. (2015). *Assessment of the Equivalent Level of Safety Requirements for Small Unmanned Aerial Vehicles* (Memorial University of Newfoundland). Retrieved from <https://research.library.mun.ca/8517/1/thesis.pdf>

Strohmeier, M., Schafer, M., Lenders, V., & Martinovic, I. (2014). Realities and challenges of nextgen air traffic management: the case of ADS-B. *IEEE Communications Magazine*, 52(5), 111–118. <https://doi.org/10.1109/MCOM.2014.6815901>

Taking flight. (2017). Retrieved December 9, 2018, from The Economist website: <https://www.economist.com/technology-quarterly/2017-06-08/civilian-drones>

Temizer, S., Kochenderfer, M. J., Kaelbling, L. P., Lozano-Pérez, T., & Kuchar, J. K. (2010). *Collision Avoidance for Unmanned Aircraft using Markov Decision Processes* \*. Retrieved from <https://people.csail.mit.edu/lpk/papers/aiaa10.pdf>

Textron Aviation. (n.d.). Cessna Skyhawk.

The Journal of Open Source Software. (2018). Retrieved September 12, 2018, from Journal of Open Source Software website: <http://joss.theoj.org>

The Open Source Definition. (2007). Retrieved September 12, 2018, from Open Source Initiative website: <https://opensource.org/osd>

Transport Canada. (1998). *Aeroplane and Rotorcraft Simulator Manual*. Retrieved from <https://www.tc.gc.ca/Publications/en/tp9685/pdf/hr/tp9685e.pdf>

Transport Canada. (2010a). National Simulator Evaluation Program. Retrieved August 28, 2018, from <https://www.tc.gc.ca/eng/civilaviation/opssvs/nationalops-airline-simulator-menu-850.htm>

Transport Canada. (2010b). Single Crew Aeroplane Standard Operating Procedures - Definitions. Retrieved from <https://www.tc.gc.ca/eng/civilaviation/standards/commerce-manuals-singlecrewsop-menu-1321.htm>

Transport Canada. (2018a). Drone Safety. Retrieved April 21, 2018, from <https://www.tc.gc.ca/eng/civilaviation/drone-safety.html>

Transport Canada. (2018b). *Flying for fun? Rules for recreational drone users*. Retrieved from <https://www.tc.gc.ca/en/services/aviation/documents/rules-recreational-drones.pdf>

Transport Canada. (2019a). Civil Aviation Daily Occurrence Report System. Retrieved August 15, 2018, from <http://wwwapps.tc.gc.ca/Saf-Sec-Sur/2/CADORS-SCREAQ/m.aspx?lang=eng>

Transport Canada. (2019b). Drone safety. Retrieved July 15, 2019, from <https://www.tc.gc.ca/en/services/aviation/drone-safety/proposed-rules-drones-canada.html>

Transport Canada. (2019c). Find your category of drone operation. Retrieved July 30, 2019, from <https://www.tc.gc.ca/en/services/aviation/drone-safety/find-category-drone-operation.html>

- Transportation Safety Board of Canada. (2017). Statistical Summary: Aviation Occurrences 2017. Retrieved September 4, 2018, from <http://www.bst-tsb.gc.ca/eng/stats/aviation/2017/ssea-ssao-2017.asp>
- Universita degli Studi di Napoli Federico II. (n.d.). ADAG | Aircraft Design & AeroFlightDynamics Group. Retrieved October 10, 2017, from <http://www.adag.unina.it/english/index.html>
- Unmanned Dynamics. (2006). *AeroSim Aeronautical simulation blockset version 1.2*. Retrieved from [http://www.u-dynamics.com/aerosim/aerosim\\_ug.pdf](http://www.u-dynamics.com/aerosim/aerosim_ug.pdf)
- Unmanned Systems Canada. (2017). *Small Remotely Piloted Aircraft System ( RPAS ) Best Practices for BVLOS Operations*. 1–74.
- van de Schoot, R., Yerkes, M. A., Mouw, J. M., & Sonneveld, H. (2013). What Took Them So Long? Explaining PhD Delays among Doctoral Candidates. *PLoS ONE*, 8(7), e68839. <https://doi.org/10.1371/journal.pone.0068839>
- Vanhanen, J., Lehtinen, T. O. A., & Lassenius, C. (2012). Teaching real-world software engineering through a capstone project course with industrial customers. *2012 First International Workshop on Software Engineering Education Based on Real-World Experiences (EduRex)*, 29–32. <https://doi.org/10.1109/EduRex.2012.6225702>
- Villa, T., Salimi, F., Morton, K., Morawska, L., & Gonzalez, F. (2016). Development and Validation of a UAV Based System for Air Pollution Measurements. *Sensors*, 16(12), 2202. <https://doi.org/10.3390/s16122202>
- Vogeltanz, T. (2015). A Survey of Free Software for the Design, Analysis, Modelling, and Simulation of an Unmanned Aerial Vehicle. *Archives of Computational Methods in Engineering*. <https://doi.org/10.1007/s11831-015-9147-y>

- Vogeltanz, T., & Jašek, R. (2015). JSBSim library for flight dynamics modelling of a mini-UAV. *International Conference on Numerical Analysis and Applied Mathematics 2014 (ICNAAM-2014)*. <https://doi.org/10.1063/1.4912770>
- Wang, Z., Lin, X., Xiang, X., Blasch, E., Pham, K., Chen, G., ... Wang, G. (2016, May 17). *An airborne low SWaP-C UAS sense and avoid system* (K. D. Pham & G. Chen, Eds.). <https://doi.org/10.1117/12.2227221>
- Wheeler, D. A. (2015, January). *Why Open Source Software / Free Software (OSS/FS, FLOSS, or FOSS)? Look at the Numbers!* (Vol. 12, pp. 61–78). Vol. 12, pp. 61–78. <https://doi.org/10.1046/j.1365-2575.2002.00118.x>
- Wilson, G., Bryan, J., Cranston, K., Kitzes, J., Nederbragt, L., & Teal, T. K. (2017). Good Enough Practices in Scientific Computing. *PLoS Computational Biology*, *13*, 1–20. <https://doi.org/10.1371/journal.pcbi.1005510>
- Wong, D. R., Ou, Q., Sinclair, M., Li, Y. J., Chen, X. Q., & Marburg, A. (2008). Unmanned Aerial Vehicle flight model validation using on-board sensing and instrumentation. *15th International Conference on Mechatronics and Machine Vision in Practice, M2VIP'08*. <https://doi.org/10.1109/MMVIP.2008.4749516>
- X-Plane 11 Flight Simulator. (2018). Retrieved December 9, 2018, from <https://www.x-plane.com/>
- YASim - FlightGear. (2018). Retrieved April 21, 2017, from <http://wiki.flightgear.org/YASim>
- Young, R. (2018). UAS ground-based detect and avoid capability. *2018 Integrated Communications, Navigation, Surveillance Conference (ICNS)*, 2B2-1-2B2-14. <https://doi.org/10.1109/ICNSURV.2018.8384837>
- Yu, X., & Zhang, Y. (2015). Sense and avoid technologies with applications to unmanned

aircraft systems: Review and prospects. *Progress in Aerospace Sciences*, 74, 152–166.

<https://doi.org/10.1016/j.paerosci.2015.01.001>

Yun, C., Li, X., & Zheng, Z. (2013). Design of UAV Simulator Based on Man-in-Loop Simulation Platform. *International Journal of Science, Environment and Technology*, 2(3).

Zekry, A., Nabil Mobarez, E., Ouda, A. N., & Zekry, A. A. (2016). Mathematical Representation, Modeling and Linearization for Fixed Wing UAV. *Article in International Journal of Computer Applications*, 147(2), 975–8887. <https://doi.org/10.5120/ijca2016910999>

Zeng, Y., Zhang, R., & Lim, T. J. (2016). Wireless communications with unmanned aerial vehicles: opportunities and challenges. *IEEE Communications Magazine*, 54(5), 36–42. <https://doi.org/10.1109/MCOM.2016.7470933>

Zenoah. (2007). *Engine manual G26 Air G26/G231 Marine G26/G231 Heli G38 G62 GT80 Twin G45*. Retrieved from [https://www.horizonhobby.com/pdf/ZEN\\_Manual\\_New-03-02-2007.pdf](https://www.horizonhobby.com/pdf/ZEN_Manual_New-03-02-2007.pdf)

Zhang, Y., & MCGovern, S. (2009). Mathematical Models for Human Pilot Maneuvers in Aircraft Flight Simulation. *ASME 2009 International Mechanical Engineering Congress and Exposition*. Retrieved from [https://ntl.bts.gov/lib/33000/33700/33763/Mathematical\\_Models\\_for\\_Human\\_Pilot\\_Maneuvers.pdf](https://ntl.bts.gov/lib/33000/33700/33763/Mathematical_Models_for_Human_Pilot_Maneuvers.pdf)

# Appendix A

## *A Simplified Manual of the JSBSim Open-source FDM for Fixed-wing RPA Applications*

Simulation packages provide a valuable framework or environment to study the interaction between aircraft, including Remotely Piloted Aircraft (RPAs), and the existing air traffic in Near Mid-Air Collision (NMAC) scenarios. The described simulation package is based on the open-source JSBSim Flight Dynamics Model (FDM), which has been validated and tested in RPA computer models for 4D encounters and avoidance manoeuvres. The objective of this appendix is to provide a simplified version of the current package, including the minimum requirements for the design of an RPA in JSBSim, and to guide any modellers on the RPA computer design task. Introductory concepts and the dynamics behind this package were introduced in Chapter 2 and will not be restated here.

This appendix is an adapted version of a previous work presented at the IEEE Newfoundland Electrical and Computer Engineering Conference (NECEC2017) in November 2017 under the title “JSBSim open-source Flight Dynamics Model for fixed-wing Unmanned Aerial Applications” (Cereceda et al., 2017). It is also a variant of a document that is being prepared for submission to the JSBSim online community as part of the contributions C4 and C5 listed in Chapter 6. This appendix begins with a brief introduction of JSBSim structure and simulation modes. The source code classes are introduced in Section A.2 followed by the set instructions for

the additional feature of the multiplayer mode used in 4D encounters. The appendix concludes with an RPA example case study.

## **A.1. High-Level Simulation Structure**

JSBSim (Berndt & JSBSim Development Team, 2011) is an FDM package that consists of a series of classes integrated together to simulate an aircraft and its environment. The package is stable and ready to use from the command/shell window. The basic version also includes a large aircraft library and simple demos. The most remarkable work done on JSBSim are the motion base simulator at the University of Naples, Italy (Universita degli Studi di Napoli Federico II, n.d.) and the human pilot math model with JSBSim as the 6-DoF simulation core, developed by the U.S. Department of Transportation (Zhang & Mcgovern, 2009).

However, JSBSim does not contain any visual environments or models associated with it and additional software –FlightGear (“FlightGear Flight Simulator,” 2019)– is required if the performance needs to be observed.

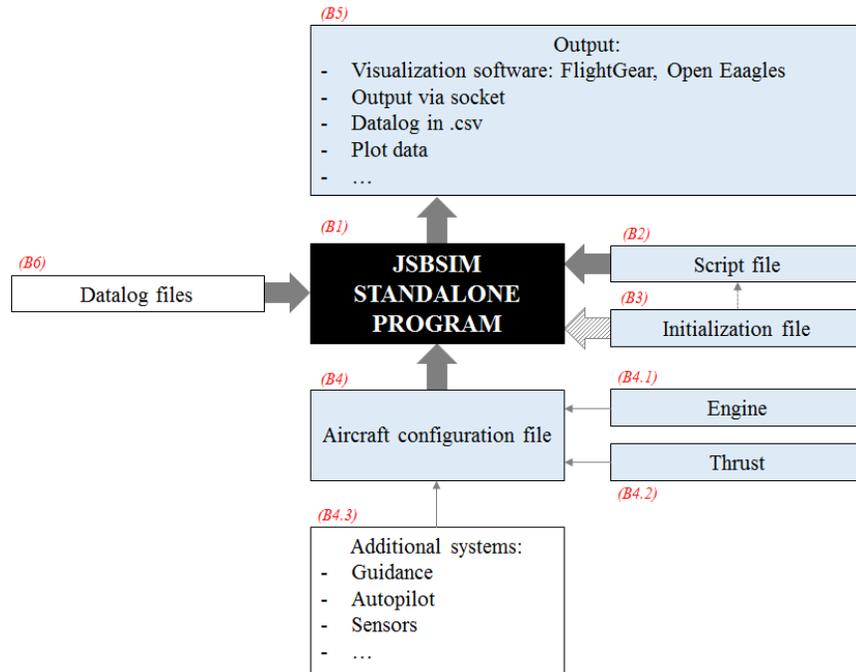
JSBSim can be downloaded online (JSBSim Development Team, 2001) and more information can be found on its website (JSBSim Development Team, 2005b). Although the developers claim that it can be integrated into MATLAB/Simulink, the system is still in development and needs significant improvement.

### ***A.1.1. Standalone Mode: Scripting***

The standalone simulation mode of JSBSim only requires the source code, the set of engines, and aircraft. The package is in constant development with periodic releases; the source code is considered stable whereas new aircraft and other systems are uploaded to the repository after

they are verified. If the user needs to compile and build the program, they should follow the instructions in the manual (Berndt & JSBSim Development Team, 2011).

A JSBSim standalone simulation structure can be summarized by the following blocks (Figure A.1):



**Figure A.1.** JSBSim standalone mode structure

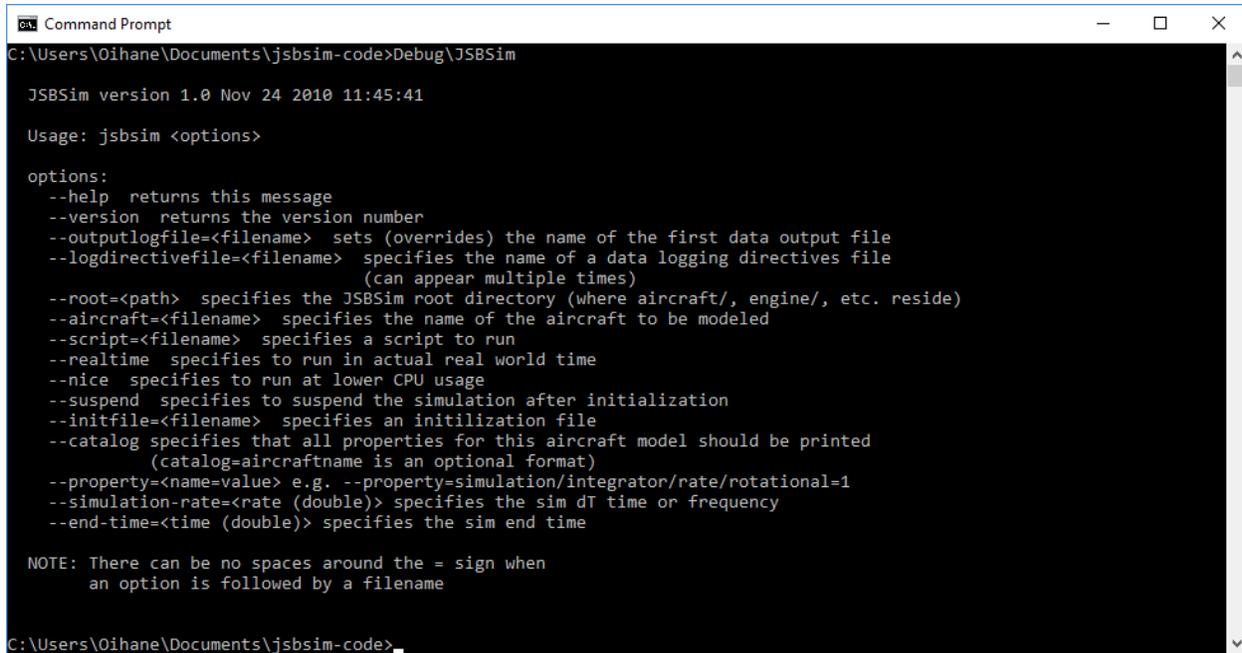
- The **JSBSim source code (B1)** is formed by the full 6-DoF FDM, including the dynamics of the system and all the models that exchange properties with the aircraft or computer models, such as wind and atmosphere.
- The **script file (B2)** is expressed in .xml format and describes the tasks to be performed by the computer model.
- The **initialization file (B3)** includes the information related to the initial state of the aircraft. It can be called either from **B1** or **B2**.

- The **aircraft configuration file (B4)** contains all the parameters for a specific aircraft. The main file includes the metrics, mass and aerodynamics, among others. The propulsion system formed by the **engine (B4.1)** and **thrust (B4.2)** files are called from **B4. Extra files (B4.3)** can also be added depending on the final purpose of the simulation. Examples include an autopilot, guidance system, or more specific information about elements such as sensors or a control system.
- The type of **output (B5)** is defined at the end of the aircraft configuration file (**B4**) and generated by the source code. The output can be generated through a series of datalogs or, become a visual performance using complementary software such as Flight Gear or OpenEagles (“OpenEagles,” n.d.).
- An **additional datalog (B6)** with specific parameters or a set of parameters to a particular package can be added as well. This feature has not yet been tested.

Therefore, the minimum and most basic configuration consists of the source code (**B1**) with the aircraft configuration file, including the propulsion system (**B4, B4.1, B4.2**), the script files defining the task (**B2**), the initialization file (**B3**), and the output generation files or interface (**B5**). The corresponding blocks are highlighted in light blue in Figure A.1. The remaining blocks are complementary depending on the task to be completed. A straightforward simulation can be run with the Debug command shown in Figure A.2, indicating the minimum files and configuration/execution simulation parameters.

If run in batch mode, the complete simulation is executed in a line code (Figure A.3). That batch file should include the basic and minimum command lines for the simulation as described below in the code box. This example eliminates files generated from previous simulations, runs the

JSBSim package by indicating the correspondent script, moves the output file and plots the results in Gnuplot (“Gnuplot,” n.d.).



```
Command Prompt
C:\Users\Oihane\Documents\jsbsim-code>Debug\JSBSim

JSBSim version 1.0 Nov 24 2010 11:45:41

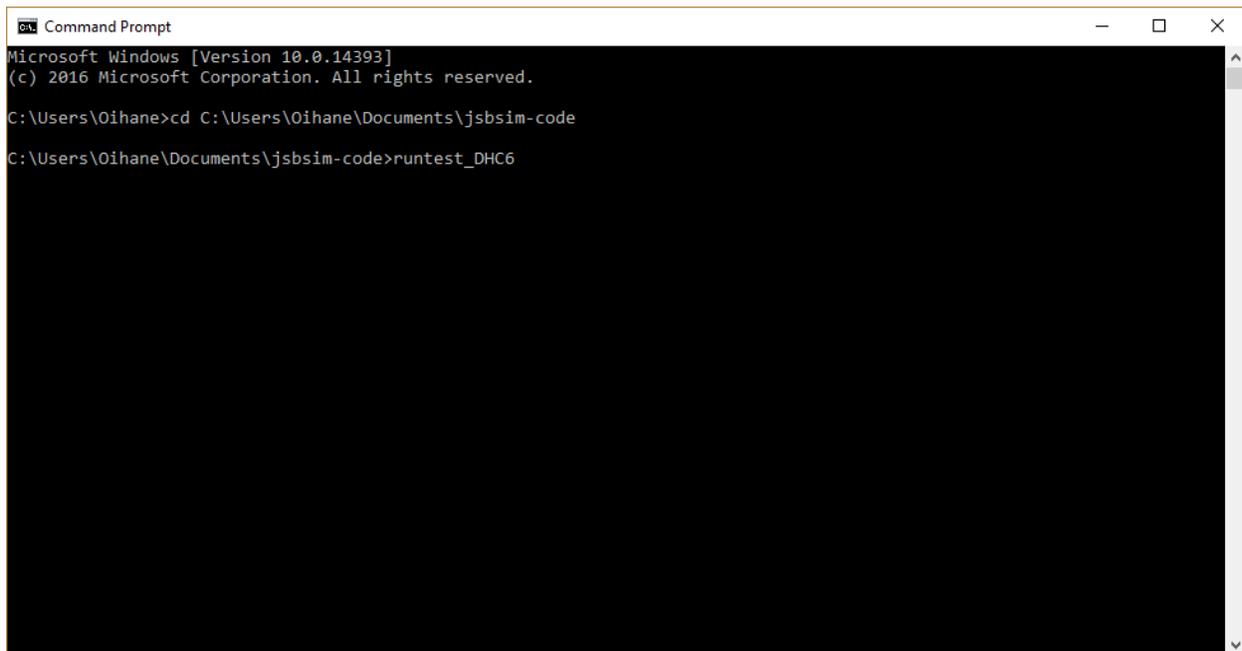
Usage: jsbsim <options>

options:
--help returns this message
--version returns the version number
--outputlogfile=<filename> sets (overrides) the name of the first data output file
--logdirectivefile=<filename> specifies the name of a data logging directives file
    (can appear multiple times)
--root=<path> specifies the JSBSim root directory (where aircraft/, engine/, etc. reside)
--aircraft=<filename> specifies the name of the aircraft to be modeled
--script=<filename> specifies a script to run
--realtime specifies to run in actual real world time
--nice specifies to run at lower CPU usage
--suspend specifies to suspend the simulation after initialization
--initfile=<filename> specifies an initialization file
--catalog specifies that all properties for this aircraft model should be printed
    (catalog=aircraftname is an optional format)
--property=<name=value> e.g. --property=simulation/integrator/rate/rotational=1
--simulation-rate=<rate (double)> specifies the sim dt time or frequency
--end-time=<time (double)> specifies the sim end time

NOTE: There can be no spaces around the = sign when
      an option is followed by a filename

C:\Users\Oihane\Documents\jsbsim-code>
```

Figure A.2. JSBSim program command line and options



```
Command Prompt
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\Users\Oihane>cd C:\Users\Oihane\Documents\jsbsim-code
C:\Users\Oihane\Documents\jsbsim-code>runtest_DHC6
```

Figure A.3. JSBSim program command line in batch mode

```
rem Remove the old results file
del /Q aircraft\DHC6\test\TestDHC6_Out.csv
del /Q aircraft\DHC6\test\JSBoutDHC6.csv

rem Run the test
Debug\JSBSim --script=aircraft\DHC6\test\DHC6-test.xml

rem Copy the csv file to another location and change its name
copy JSBoutDHC6.csv aircraft\DHC6\test\
ren aircraft\DHC6\test\JSBoutDHC6.csv TestDHC6_Out.csv

rem Generate gnuplot to the screen
gnuplot aircraft\DHC6\test\PlotOrbitDHC6.p
```

### *A.1.1.1. Script Definition*

The simulation task is defined in the script alongside the aircraft and its initial state. Events activate when a condition (declared within JSBSim) is met and a series of actions are activated. Each event is triggered once unless the condition associated with it is declared "continuous" or "persistent", making it constantly evaluated during the simulation.

In the example framed below extracted from a script demo, the Cessna 172 aircraft is tested for the autopilot hold and cruise performance. The aircraft and its initial state is declared with `<use aircraft=" " initialize=" "/>` and the simulation conditions are defined with `<run start=" " end=" " dt=" ">`. The event extracted from the script shows that a series of autopilot properties get updated 5 seconds after the simulation starts. With the `</notify>` function, the Euler angles and the altitude are displayed on the command window when the event is triggered.

```

<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl"
href="http://jsbsim.sourceforge.net/JSBSimScript.xsl"?>
<runscript xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://jsbsim.sf.net/JSBSimScript.xsd"
  name="C172 cruise at 4K, 100% power">
  <description>This run is for testing the C172 altitude hold autopilot and cruise
performance</description>
  <use aircraft="c172x" initialize="reset01"/>

  <run start="0.0" end="50" dt="0.0083333">

    <property value="0"> simulation/run_id </property>

    <event name="Hold heading and altitude">
      <condition>simulation/sim-time-sec ge 5</condition>
      <set name="ap/heading_setpoint" value="200"/>
      <set name="ap/heading_hold" value="1"/>
      <set name="ap/altitude_setpoint" value="4000.0"/>
      <set name="ap/altitude_hold" value="1"/>
      <notify>
        <property> attitude/psi-rad </property>
        <property> attitude/theta-rad </property>
        <property> attitude/phi-rad </property>
        <property> position/h-agl-ft </property>
      </notify>
    </event>

  </run>
</runscript>

```

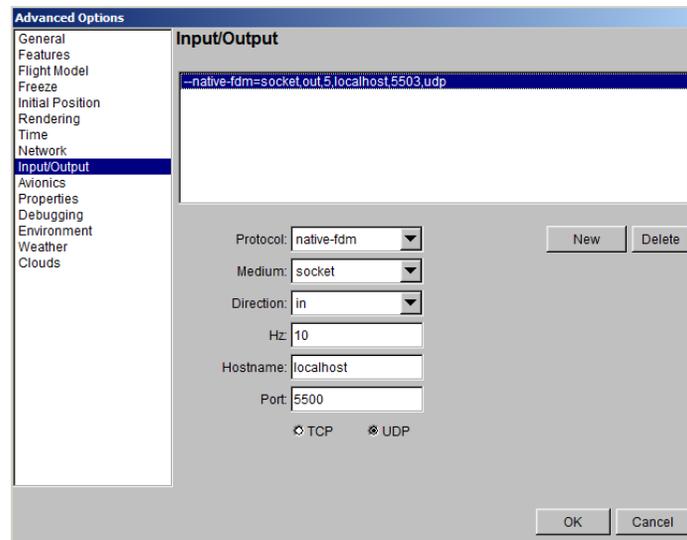
### A.1.1.2. Visualization in FlightGear

By default, the standalone mode does not allow for visualization. With an extra line of code in the aircraft configuration file, the model can share its output with FlightGear, enabling the visual performance. For a successful exchange of properties, the same input/output information must be configured in FlightGear, since JSBSim runs as an external model.

Assume that following line of code is declared in the JSBSim in the aircraft configuration file:

```
<output name ="localhost" type="FLIGHTGEAR" port="5500" protocol="UDP" rate="10"> </output>
```

Then, the FDM in FlightGear must be selected as external and the input property in FlightGear must be:



**Figure A.4.** FlightGear interface. Advanced options. Input/output properties

The visualization of the performance will start as soon as the simulation launches. It should be noted that the visualization will depend on the JSBSim delta-time unless otherwise indicated.

### ***A.1.2. Integration in FlightGear***

The second way to run an aircraft designed in JSBSim is by integrating it directly into FlightGear (“JSBSim - FlightGear wiki,” 2018). The main difference between this mode and the mode explained in Section A.1.1.2. is that in this case, only the aircraft configuration and its related files are needed since FlightGear holds the simulation. This is extremely useful when the aircraft is flown manually with a controller, whereas it becomes impractical when choosing a pre-defined task.

For a better understanding, the Figure A.5 shows a simplified structure of FlightGear formed by aircraft (expressed in JSBSim FDM), Air Traffic Control (ATC), airport, scenery and other models:

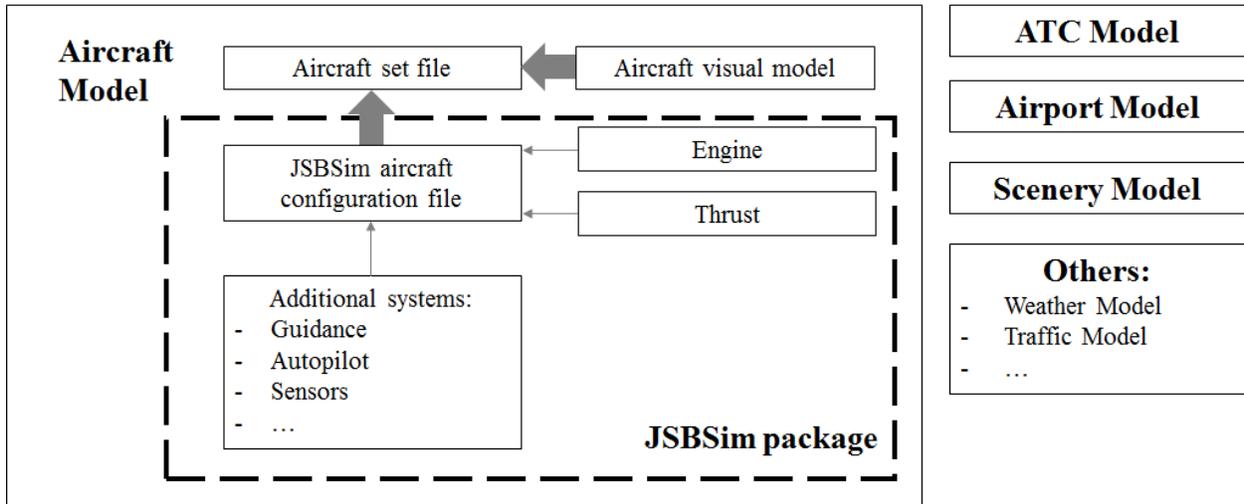


Figure A.5. FlightGear block structure

Copy the required files in the source folder, including the aircraft information, into FlightGear to run the simulation. If everything has been done correctly, the selected aircraft will be listed as available. Before launching the simulation, ensure that the flight model option “jbs” is selected in Advanced Options (Figure A.6). The model will start on the ground or in the air and can be initiated and controlled by a flight controller, a joystick, or with the computer keyboard.

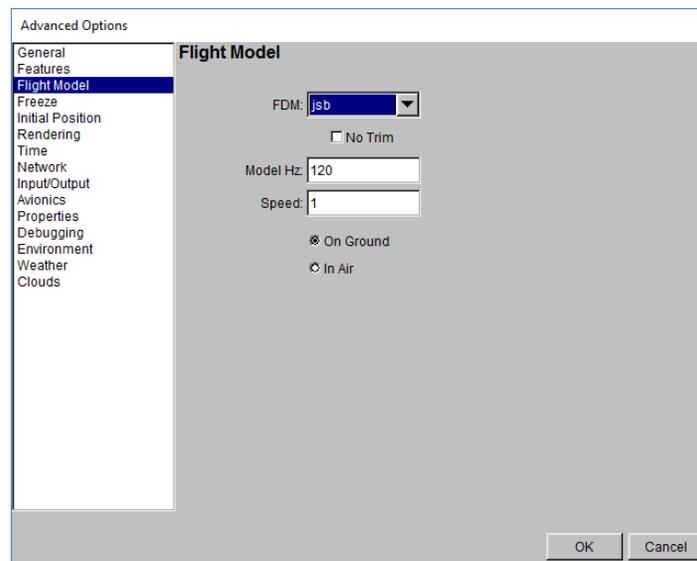


Figure A.6. FlightGear interface. Advanced options. Flight Model

Note that in this mode there is no need to add any shared properties in the aircraft configuration file, since FlightGear will be generating the output straight from the simulation.

## A.2. Classes, Class Hierarchy and Model Class

The JSBSim complete package includes all the source codes for a basic simulation; however, the user can create new systems depending on the required task. These systems might include autopilot, Guidance, Navigation and Control (GNC) and specific onboard instruments. JSBSim is designed to be modified depending on the vehicle and the user's needs. C++ represents an excellent programming language to cover all the requirements in JSBSim; it provides the required management tool for extracting, calculating and propagating data between classes as part of its object-oriented programming.

The JSBSim source code structure works like any other computer model in C++. The location and dynamics of the aircraft are given by the mathematical expressions in Chapter 2 (Section 2.3). The package models the metrics, the computation of the forces and moments, and the propagation/output of the dynamics, among others. Likewise, JSBSim also needs basic mathematical elements such as functions, tables, and quaternions to handle mathematical transformations.

The code is distributed in classes depending on their function. The collection of described high-level classes could be classified in:

- The **executive class** *FGFDMExec* initializes and runs the package from the application calling JSBSim. The simulation starts and finishes according to the script that includes

the simulation details. When *FGFDMExec* class is initialized, it creates model objects that define the aircraft according to the aircraft configuration file defined by the user.

- The **model class** *FGModel* represents the real physical classes and elements in the aircraft. The model classes are executed in order including *FGAerodynamics*, *FGAircraft*, *FGAtmosphere*, etc.
- The **math classes** (*FGColumnVector3*, *FGQuaternion*, *FGTable*, etc.) contain the mathematical operations needed to solve equations, transformations, and other relations in JSBSim.
- **I/O and initialization classes** (*FGInputSocket*, *FGOutputTextFile*, etc.) handle inputs and outputs to the system.
- The **basic classes** (*FGJSBBase* and *FGState*) provide common capabilities to all the classes such as message handling.

Focusing on the RPA computer model design, only the Model classes differ from the general aviation aircraft case; the remaining classes are necessary, and include the operators and simulation characteristics required for the correct implementation of JSBSim. The model classes inherited from *FGModel* are listed below, including a brief description and special conditions (if applicable) for the RPA case:

- The input class (*FGInput*)<sup>8</sup> manages the inputs to the model with `<input>` elements in the aircraft configuration file. When the software reads `<input>`, a communication between classes is open and an appropriate action taken.
- The atmosphere class (*FGAtmosphere*)<sup>8</sup> models the 1976 standard atmosphere including winds, turbulence and giving the values of pressure, density and temperature, depending

---

<sup>8</sup>This class is required in all types of aircraft and fixed-wing RPAs.

on the location of the aircraft. Beware that the RPA case will only consider low altitude (under 1000ft)<sup>9</sup> and therefore, the model may be simplified to consider only the troposphere conditions. However, the author recommends keeping this model as it is for possible future uses.

- The FCS class (*FGFCS*)<sup>8</sup> manages a collection of flight control classes defined by the aircraft components (surface control elements, throttle, autopilot). In a simple RPA simulation, the primary controls for the model are the surface command elements of the aircraft which include the ailerons, elevator, and rudder.
- The propulsion class (*FGPropulsion*)<sup>8</sup> manages from 0 to n number of engines (*FGEngine*). JSBSim includes different kinds of propulsion systems depending on the engines used to generate thrust. They include a piston engine model (*FGPiston* → *FGEngine*), a jet turbine engine model (*FGTurbine* → *FGEngine*), a turboprop engine model (*FGTurboProp* → *FGEngine*), a rocket engine model (*FGRocket* → *FGEngine*) and an electric engine model (*FGElectric* → *FGEngine*). However, only the piston and electric models are considered in the case of RPAs. The thrust generation (*FGThruster* → *FGForce*) presents the same scenario where among the options found in JSBSim –direct, nozzle (*FGNozzle* → *FGThruster*), propeller (*FGPropeller* → *FGThruster*) and rotor (*FGRotor* → *FGThruster*)– only the propeller is used in the fixed-wing RPA case. The propulsion system, including the engine and the origin of the thrust generation, are called from the aircraft configuration file.
- The mass balance class (*FGMassBalance*)<sup>8</sup> calculates the moments of inertia, Center of Gravity (CG), and mass over time. At initialization, the `<mass_balance>` section in the

---

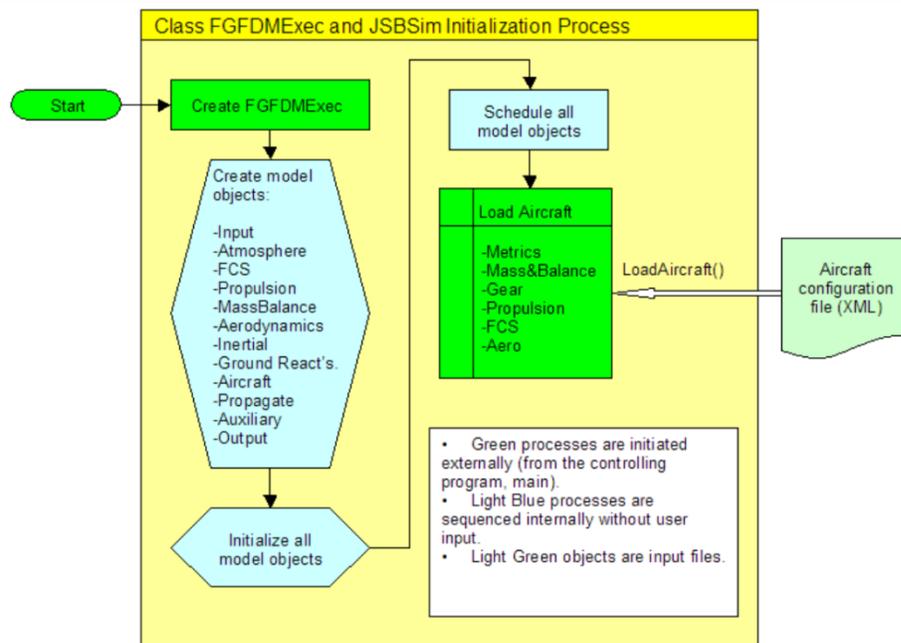
<sup>9</sup>This value might differ with the upcoming Transport Canada (TC) regulations to be implemented in 2019.

aircraft configuration file is read and for each sample time, the CG and mass are updated. This class is relevant in the RPA case when the aircraft is carrying a piston engine since the fuel consumption will highly update those values.

- The aerodynamics class (*FGAerodynamics*)<sup>8</sup> is a collection of manager classes with individual force and moment definitions. When `<aerodynamics>` is called in the aircraft configuration file, this class handles the corresponding aerodynamic calculations obtaining the forces and moments calculated for each of the axes.
- The inertial class (*FGInertial*)<sup>8</sup> initializes the radius and reference acceleration values.
- The ground reactions class (*FGGroundReactions*) models the ground reactions defined in the aircraft configuration file as `<ground_reactions>`. The two types of contacts are BOGEY, which is directly related to the landing gear and its contacts, and STRUCTURE, which is used to locate any aircraft contact type that is not part of the landing gear (wingtips, nose and tail). JSBSim models the landing gear set as a spring/damper model with `<spring_coeff>` and `<damping_coeff>`. It also models retractable landing gear for contacts but it is not applicable to most RPAs as their landing gear is non-retractable.
- The aircraft class (*FGAircraft*)<sup>8</sup> gathers all systems together; it initializes the aircraft model loading its properties with `<metrics>`, and obtains the contribution of each of the systems in the generation of forces and moments.
- The propagate class (*FGPropagate*)<sup>8</sup> models the Equations of Motion (EOM), giving the state of the vehicle from the forces and moments generated during flight.
- The auxiliary class (*FGAuxiliary*) models pilot sensed accelerations and other auxiliary parameters used for acceleration calculations in inertial space. This class is only required in case the RPA carries motion-based sensors onboard.

- The output class (*FGOutput*)<sup>8</sup> handles the simulation output. The desired output is defined in the aircraft configuration file. The classes generated include: CSV (datalog in csv), SOCKET (data sent to a socket output defined by an IP address), FLIGHTGEAR (socket to FlightGear) and TABULAR (columnar data).

Figure A.7 expresses the JSBSim FGModel in operation; when JSBSim is initialized, FGFDMExec is executed generating all the models' objects that will be uploaded with the information contained in the aircraft configuration file. This allows the assembly of the 6-DoF aircraft computer model.



**Figure A.7.** FGFDMExec and JSBSim Initialization process (Berndt & JSBSim Development Team, 2011)

For a full collection of JSBSim classes, see reference (JSBSim Development Team, 2017).

### **A.3. Additional Features: Multiplayer Mode**

An additional feature in FlightGear allows for the visualization of several independent models in one simulation by communicating with each other. The setup is similar to the FlightGear visualization in the JSBSim standalone mode (Section A.1.1.2) except that the properties shared between one and another must be opposite.

For the multiplayer mode setup instructions (extracted from (Stevenson, 2013)), assume that one model is called “aircraft 1” (A1) and another “aircraft 2” (A2). For this mode, the IP of both computers is required to enable the communication; use the ipconfig command to retrieve that information.

In A1 follow these steps:

- 1- Launch FlightGear selecting the aircraft and the airport.
- 2- Select AI Models and Random Objects.
- 3- Select Multiplayer mode:
  - Callsign: GS\_TEST1
  - Hostname: IP of A1
  - In: 5510
  - Out: 5520
- 4- Select Advanced options.
- 5- Flight model. Select the most appropriate model depending on the run mode (jsb, external, etc.).
- 6- Input/Output: Include the following variables:

--native-fdm=socket, in, 10, localhost, 5500, udp (external FDM – JSBSim output)

--native-ctrls=socket, out, 5, localhost, 5501, udp

--native-fdm=socket, out, 5, IP of C2, 5504, udp (communication to A2)

7- Return and Run.

In A2, do the following:

1- Launch FlightGear, selecting the appropriate aircraft and the same airport as in A1.

2- Select AI Models and Random Objects.

3- In Multiplayer mode:

- Callsign: GS\_TEST2

- Hostname: IP of A1

- In: 5520

- Out: 5510

4- Select Advanced Options.

5- Flight model: jsb, external, etc.

6- Input/Output: Add the following:

--native-fdm=socket, in, 10, localhost, 5502, udp (external FDM – JSBSim output)

--native-ctrls=socket, out, 5, localhost, 5503, udp

7- Return and Run.

Both aircraft should be displayed on the screen of A1 from the perspective of A1 (Figure A.8).

This particular Multiplayer mode allows one computer (A1) to be the host of the online simulation, while A2 only simulates the performance of the second aircraft.



**Figure A.8.** Multiplayer mode in FlightGear with a Cessna 172 as seen from the cockpit of another aircraft

## A.4. Case Study: Giant Big Stik Fixed-wing RPA

The Giant Big Stik R/C RPA (Figure A.9) (Great Planes, 2005c) is an aerobatic sport-scale aircraft belonging to the “Stik” series, manufactured by GreatPlanes (Great Planes, 2005a). It is powered by a fuel engine (Zenoah 26A) and flies like a full-size aeroplane. This model aeroplane has been widely used in the Remote Aerial Vehicles for Environment-monitoring (RAVEN) group at Memorial University of Newfoundland for many years. Its aerobatic characteristics are the most significant highlight of this model.



Figure A.9. Giant Big Stik on the field before tests

#### A.4.1. Giant Big Stik Aircraft Modelling

Based on the package structure described in Figure A.1, the minimum set of blocks are created as part of the aircraft modelling:

- B3-initialization file: the simulation is held near Clarenville, Newfoundland, Canada with a certain initial RPA airspeed and no wind.

```

<initialize name="reset03">
  <vc unit="KTS"> 38.87689112646 </vc>
  <longitude unit="DEG"> -53.91752033 </longitude>
  <latitude unit="DEG"> 48.2778129 </latitude>
  <phi unit="DEG"> 0.0 </phi> <!-- Roll -->
  <theta unit="DEG"> 0.0 </theta> <!-- Pitch -->
  <psi unit="DEG"> 0.0 </psi> <!-- Yaw -->
  <altitude unit="FT"> 984.252 </altitude>
  <hwind> 0.0 </hwind>
</initialize>

```

- B4-aircraft configuration file. It includes all the coefficients and parameters specific to the Giant Big Stik RPA, which are introduced in Tables 5.1 and 5.2 in Section 5.1.2 – Chapter 5. For this example, two cases are considered: an output file and a visualization while the simulation is running.

```

<fdm_config name="GBS_3" version="2.0" release="ALPHA"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://jsbsim.sourceforge.net/JSBSim.xsd">

  <fileheader>
    <author> Oihane Cereceda </author>
    <filecreationdate> 2015-11-11 </filecreationdate>
    <version> v4 </version>
    <description> Giant Big Stik JSBSim model
    v1: Files and values created from Aeromatic v0.82
    v2: Files modified according to more specific data
    v3: Validate model using scripts and plotting data </description>
    v4: Output to FlightGear to visualize the performance
  </fileheader>

  <metrics>

  <mass_balance>

  <ground_reactions>

  <propulsion>

  <flight_control name="FCS: unnamed">

  <aerodynamics>

  <output name="localhost" type="FLIGHTGEAR" port="5500" protocol="UDP" rate="10">
</output>

  <output name="GBS_3Out.csv" type="CSV" rate="100">

</fdm_config>

```

#### ***A.4.2. Giant Big Stik in Standalone Mode***

As an example of the use of JSBSim with RPA applications, a simple task is defined where the model turns left using the ailerons at maximum deflection in open-loop. In this example, which has been used in previous work (Cereceda et al., 2016) for validation purposes, the aileron is set to its maximum value in 5 seconds and returns back to 0.0 in 6.5 seconds.

```

<use aircraft="GBS" initialize="reset03"/>
<run start="0" end="35" dt="0.01">

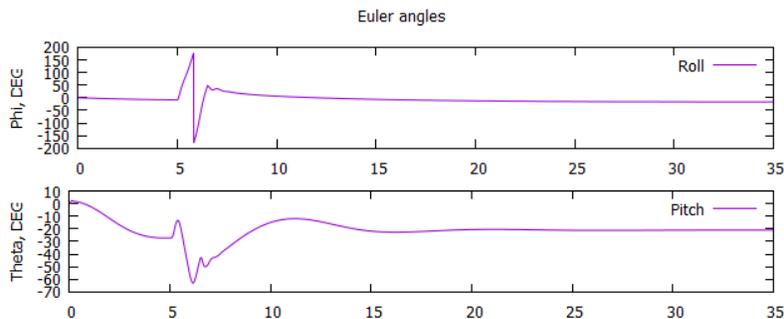
  <event name="Set engine throttle">

<event name="Set aileron max. Turn left">
  <condition> simulation/sim-time-sec ge 5.0 </condition>
  <set name="fcs/aileron-cmd-norm" action="FG_STEP" value="-0.569" tc="1"/>
  <notify/>
</event>

<event name="Set aileron to zero">
  <condition> simulation/sim-time-sec ge 6.5 </condition>
  <set name="fcs/aileron-cmd-norm" action="FG_STEP" value="0.0" tc="1"/>
  <notify/>
</event>

</run>

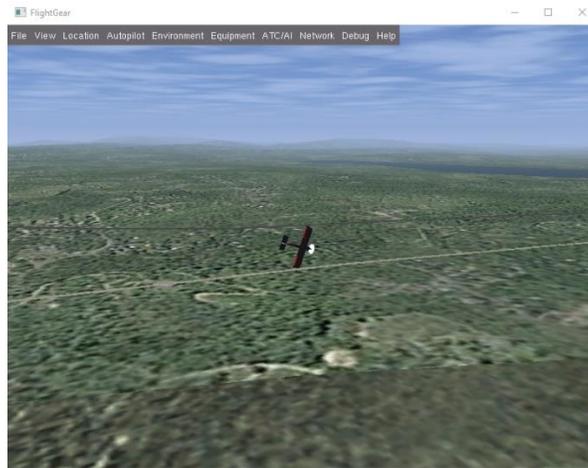
```



**Figure A.10.** RPA roll and pitch angles

The simulation starts with the RPA in the air, meaning that it needs a few seconds to reach a stable situation before any command is sent to the model. This is also reflected when, in the case of real flight, the R/C model switches from manual to remotely piloted mode. The simulation test shown here expresses a special case where the ailerons are shifted to their maximum value in order to evaluate the dynamics of the system when performing an extreme manoeuvre. Figure A.10 (plots generated using Gnuplot from JSBSim datalog output) shows how the roll angle is affected by changes in the aileron deflection; the aircraft quickly spins around and stabilizes as soon as the ailerons are back to 0.0.

The coherence of the system is noticeable. When the aircraft rolls, the lift vertical component is no longer balanced and the weight creates a loss in altitude and pitch angle. The response is presented in Figure A.11 where FlightGear has been used to visualize the performance of the aircraft during the simulation.



**Figure A.11.** Giant Big Stik in FlightGear v2.0.0 performing an aerobatic manoeuvre in JSBSim standalone mode

#### ***A.4.3. Giant Big Stik Integrated into FlightGear***

This JSBSim feature allows the user to manually fly the model using the FlightGear interface. Following the setup described in Section A.1.2, and using the Futaba Interlink Elite Controller to fly the computer model, the simulation views are the following:



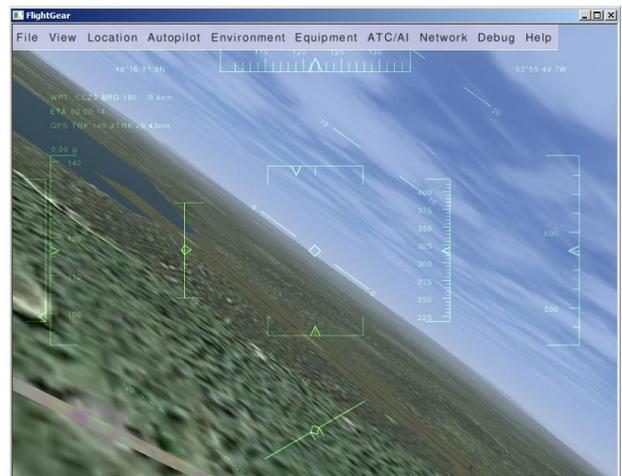
(a) Chase view of a turning manoeuvre



(b) Straight flight over the runway, chase view



(c) Fly-by view simulating what an R/C pilot would see from the ground



(d) Cockpit view during flight

**Figure A.12.** Giant Big Stik manual flight in FlightGear v2.0.0

# Appendix B

## *Coefficients calculation for the EPP FPV R/C*

Prior to the creation of the aircraft configuration files of the EPP FPV in JSBSim, the aircraft's metrics and certain parameters have to be calculated. These define the response and performance of the computer model that must be representative of the real model.

This appendix includes the calculations carried out to obtain the mass moment of inertia matrix and the initial aerodynamic coefficients that will serve as a base for the validation procedure in Chapter 4.

### **B.1 Inertia Coefficients**

The moment of inertia is a rotational inertial measurement or the tendency of a body to resist the changes on its rotational movement. Although the moment of inertia must be constant for a rigid body, it is always around a particular axis and can hold different values for other axes. The moment of inertia also depends on the mass distribution with respect to the rotational axis. The moment of inertia of a continuously distributed mass is:

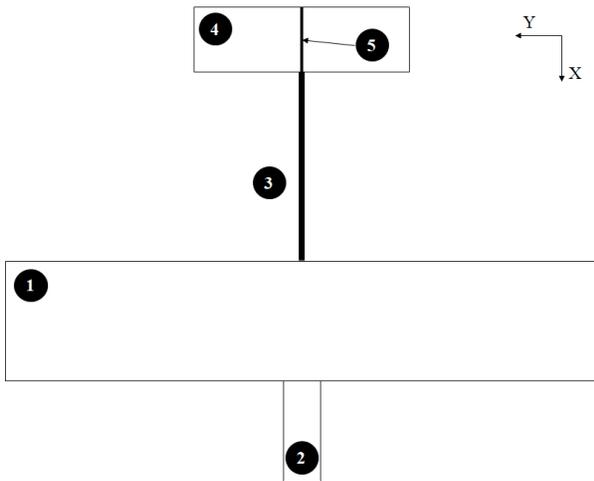
$$I = \int r^2 dm \quad (\text{B.1})$$

Where  $dm$  is the differential mass of the body and  $r^2$  is the minimum distance to the rotational axis. Assuming that the even distribution of the mass is unknown, the mass can be substituted by the density times the volume. Then, the inertia based on the volume is calculated by:

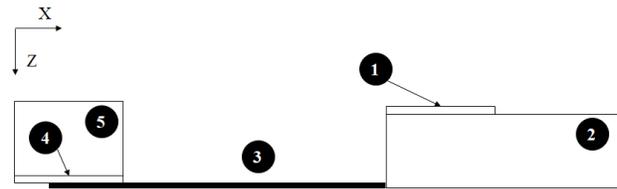
$$I = \int r^2 \rho dV \quad (\text{B.2})$$

Where  $\rho$  is the volumetric density expressed by mass/volume and  $dV$  is the differential volume.

If the body is composed of a series of geometric elements, the moment of inertia is calculated by (1) splitting each of those elements, (2) calculating their moment of inertia, and (3) adjusting the moment of inertia by knowing the distance between the element's Centre of Mass (CM) and the system's CM. The EPP FPV aircraft is formed approximately by the following series of geometries in the corresponding projection plane:



**Figure B.1.** Top view of the RPA



**Figure B.2.** Side view of the RPA

**Table B.1.** List of simple geometries in the EPP FPV

Element #	Fuselage/aerosurface name
1	Wings rectangle
2	Fuselage rectangle
3	Tube of the fuselage rectangle
4	Elevator stabilizer rectangle
5	Vertical fin rectangle

Additionally, the Steiner's Theorem (commonly known as the Parallel Axis Theorem) establishes that the moment of inertia, with respect to any axis parallel to an axis that crosses the CM, is equal to the moment of inertia of the axis that crosses the CM plus the product of the mass by the square of the distance between both axes:

$$I_{axis} = I_{axis}^{(CM)} + Mh^2 \quad (B.3)$$

Where  $I_{axis}$  is the moment of inertia of the parallel axis,  $I_{axis}^{(CM)}$  is the moment of inertia of the axis that crosses the CM and is parallel to the first one,  $M$  is the mass, and  $h$  is the distance between both axes.

### ***B.1.1. Moment of Inertia around X Axis ( $I_X$ )***

Due to the aircraft symmetry along the X axis, each CM of the geometric figures belongs to that X axis (Figure B.1). This means that the moments of inertia of each of the geometries around their CM are the same moments on the X axis. Therefore, the moment of inertia around the X axis is the sum of the moments of inertia generated by all the elements in the aircraft:

**Table B.2.** Moment of inertia around X axis

Element #	Fuselage/aerosurface name	$I_X^{(CM)}$ (kg m <sup>2</sup> )
1	Wings rectangle	0.048811653
2	Fuselage rectangle	0.000667668
3	Tube of the fuselage rectangle	7.16102E-06
4	Elevator stabilizer rectangle	0.000854486
5	Vertical fin rectangle	1.38669E-07
		$I_{XX} = \sum I_X^{(CM)} = 0.050341107$

### ***B.1.2. Moment of Inertia around Y Axis ( $I_Y$ )***

The RPA is not symmetric around the Y axis and, therefore, the moment of inertia is calculated by following the parallel axis theorem in two steps: (1) calculating the moment of inertia around

the Y axis of the corresponding element and (2) applying the theorem of the parallel axes (equation B.3).

Based on the geometry of the aircraft in Figure B.2, the moments of inertia around their own CM and the adjusted moment of inertia around the aircraft CM are:

**Table B.3.** Moment of inertia around Y axis

Element #	Fuselage/aerosurface name	$I_Y^{(CM)}$ (kg m <sup>2</sup> )	$I_Y$ (kg m <sup>2</sup> )
1	Wings rectangle	0.0007292	0.000883194
2	Fuselage rectangle	0.0080239	0.009035895
3	Tube of the fuselage rectangle	0.0059711	0.00700636
4	Elevator stabilizer rectangle	5.148E-05	7.02006E-05
5	Vertical fin rectangle	1.123E-05	1.71092E-05
			$I_{YY} = \sum I_Y = 0.017012758$

### ***B.1.3. Moment of Inertia around Z Axis ( $I_Z$ )***

The aircraft view for the calculation of the moment of inertia around the Z axis is the projection on the YZ plane. The measurements needed for the calculation of  $I_Z$  can be extracted from the projections in Figure B.1 and Figure B.2. Since the aircraft is not symmetric in the Z axis, the Steiner's Theorem is required (equation B.3). Similarly, as calculated for  $I_{YY}$ ,  $I_{ZZ}$  is obtained in two steps: (1) calculate  $I_Z^{(CM)}$  for each of the elements in the aircraft and (2) adjust  $I_Z$  for each element by using the parallel axis theorem:

**Table B.4.** Moment of inertia around Z axis

Element #	Fuselage/aerosurface name	$I_Z^{(CM)}$ (kg m <sup>2</sup> )	$I_Z$ (kg m <sup>2</sup> )
1	Wings rectangle	0.049540815	0.050101808
2	Fuselage rectangle	0.008691551	0.023759408
3	Tube of the fuselage rectangle	0.005971099	0.029063534
4	Elevator stabilizer rectangle	0.000905967	0.021889738
5	Vertical fin rectangle	1.13709E-05	0.004589648
			$I_{ZZ} = \sum I_Z = 0.129404136$

#### ***B.1.4. Product Moment of Inertia***

By definition, the product-moment of inertia of a system, with respect to two planes, is the sum of the products of all the masses of the system times the distance to each of the planes. Assuming that the product moment of inertia is between the axis X and Y, then  $I_{XY}$  (or  $I_{YX}$ ) is:

$$I_{XY} = I_{YX} = \sum_{i=1}^n m_i x_i y_i \quad (\text{B.4})$$

Where  $m_i$  is the mass of each element in the system and  $x_i$  and  $y_i$  are the distance to the planes X and Y respectively.

Similarly,  $I_{XZ} = I_{ZX}$  and  $I_{YZ} = I_{ZY}$  are:

$$I_{XZ} = I_{ZX} = \sum_{i=1}^n m_i x_i z_i \quad (\text{B.5})$$

$$I_{YZ} = I_{ZY} = \sum_{i=1}^n m_i y_i z_i \quad (\text{B.6})$$

Based on equation B.4,  $I_{XY}$  is 0 since the aircraft is symmetric in that plane and the aircraft has its opposite on the other side of the axis, making the total sum equal to 0. The aircraft is also symmetric in the YZ plane and, therefore,  $I_{YZ} = I_{ZY} = 0$ .

However, there is no symmetry on the XZ plane as it is observed from Figure B.2. The product of inertia for this case is calculated by using equation B.5, and knowing the distance between each of the CMs and the corresponding X and Z axes:

**Table B.5.** Product moment of inertia in XZ

Element #	Fuselage/aerosurface name	$I_{xz}$ (kg m <sup>2</sup> )
1	Wings rectangle	-0.000321188
2	Fuselage rectangle	0.003813103
3	Tube of the fuselage rectangle	0.004833909
4	Elevator stabilizer rectangle	0.000622633
5	Vertical fin rectangle	0.000162956
		$I_{xz} = \sum I_{xz} = 0.009111413$

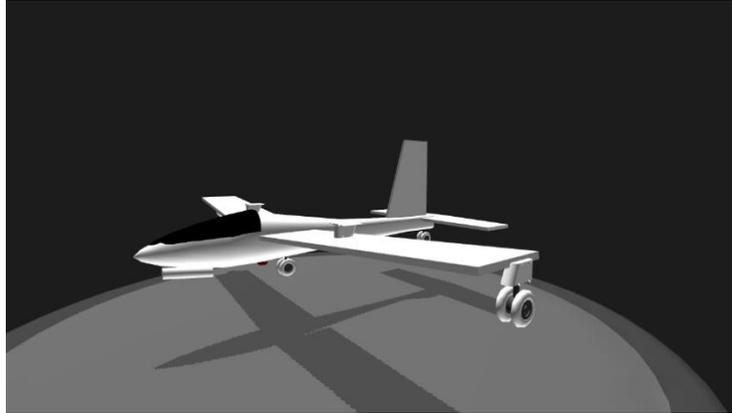
Finally, the matrix moment of inertia is built with all the moments of inertia around their own axis in the diagonal and the products of inertia in the off-diagonal elements:

$$\begin{bmatrix} I_{XX} & I_{XY} & I_{XZ} \\ I_{YX} & I_{YY} & I_{YZ} \\ I_{ZX} & I_{ZY} & I_{ZZ} \end{bmatrix} = \begin{bmatrix} 0.0503411 & 0 & 0.00911141 \\ 0 & 0.017012758 & 0 \\ 0.00911141 & 0 & 0.12940414 \end{bmatrix} \quad (\text{B.7})$$

## B.2 Aerodynamic coefficients

Contrary to the moments of inertia, which were calculated from physical elements on the aircraft, the aerodynamic coefficients require wind tunnel experiments or similar. A common practice is to use the aerodynamic coefficients from known aircraft that have similar structure and response.

The mini SGS-126 (Figure B.3) is a glider similar in shape to the EPP FPV. The aerodynamic responses of both are similar and, therefore, the aerodynamic coefficients of the mini SGS-126 were used as a basis for the calculation of the EPP FPV parameters.



**Figure B.3.** Mini SGS-126 Glider (SimplePlanes, 2019)

The original aerodynamic coefficients from the mini SGS-126 are (“JSBSim Flight Dynamics Model - Code aircraft/minisgs,” 2016):

**Table B.6.** Mini SGS-126 drag force aerodynamic coefficients

		DRAG					
$C_{D0}$	Drag at zero lift			0.007			
$C_D^\alpha$	Drag due to $\alpha$	$\alpha$	-0.0175	0.0	...	1.3963	1.5708
		$C_D^\alpha$	0.01	0.015	...	1.5	1.46
$C_D^{\delta^e}$	Drag due to elevator deflection	Elevator	-1.0		0.0		1.0
		$C_D^{\delta^e}$	0.114		0.0		0.114

**Table B.7.** Mini SGS-126 side force aerodynamic coefficients

		SIDE	
$C_Y^\beta$	Side force due to $\beta$		-0.2850
$C_Y^{\delta^a}$	Side force due to aileron deflection		-0.0456
$C_Y^{\delta^r}$	Side force due to rudder deflection		0.1880

**Table B.8.** Mini SGS-126 lift force aerodynamic coefficients

		LIFT					
$C_L^\alpha$	Drag due to $\alpha$	$\alpha$	-0.1571	-0.1369	...	1.369	1.5708
		$C_L^\alpha$	0.0	0.06	...	0.26	0.03
$C_L^{\delta^e}$	Drag due to elevator deflection			-0.3420			

**Table B.9.** Mini SGS-126 roll moment aerodynamic coefficients

ROLL		
$C_l^\beta$	Roll moment due to $\beta$	-0.0513
$C_l^p$	Roll moment due to roll rate	-0.4700
$C_l^r$	Roll moment due to yaw rate	0.1500
$C_l^{\delta^a}$	Roll moment due to aileron deflection	0.2500
$C_l^{\delta^r}$	Roll moment due to rudder deflection	0.0046

**Table B.10.** Mini SGS-126 pitch moment aerodynamic coefficients

PITCH		
$C_{m0}$	Pitch moment at zero lift	0.0
$C_m^\alpha$	Pitch moment due to $\alpha$	-0.5730
$C_m^q$	Pitch moment due to pitch rate	-9.0000
$C_m^{\dot{\alpha}}$	Pitch moment due to $\alpha$ rate	-5.2000
$C_m^{\delta^e}$	Pitch moment due to elevator deflection	-1.2610

**Table B.11.** Mini SGS-126 yaw moment aerodynamic coefficients

YAW		
$C_n^\beta$	Yaw moment due to $\beta$	0.0170
$C_n^p$	Yaw moment due to roll rate	-0.1800
$C_n^r$	Yaw moment due to yaw rate	-0.0250
$C_n^{\delta^a}$	Yaw moment due to aileron deflection	0.0115
$C_n^{\delta^r}$	Yaw moment due to rudder deflection	-0.0370

Based on the validation procedure presented in Chapter 3 and the adjustments carried out in Chapter 4, the following coefficients have been modified in order to match the real aircraft performance:

Coefficient	Mini SGS-126	EPP FPV
$C_Y^\beta$	-0.2850	-0.83
$C_l^\beta$	-0.0513	-0.0313
$C_l^{\delta^r}$	0.0046	-0.0046
$C_{m0}$	0.0	0.102
$C_m^\alpha$	-0.5730	-1.573

# Appendix C

## *Extended Sense and Avoid Basis and Existing Work*

Threat detection and avoidance manoeuvres are complex fields of study. With the recent integration of RPAS into the airspace (Federal Aviation Administration, 2013a), collision avoidance methods have become a growing topic in engineering. Commercial aeroplanes and big aircraft carry instrumentation on board, such as the Traffic Collision Avoidance System (TCAS) (Federal Aviation Administration. U.S. Department of Transportation, 2011) (

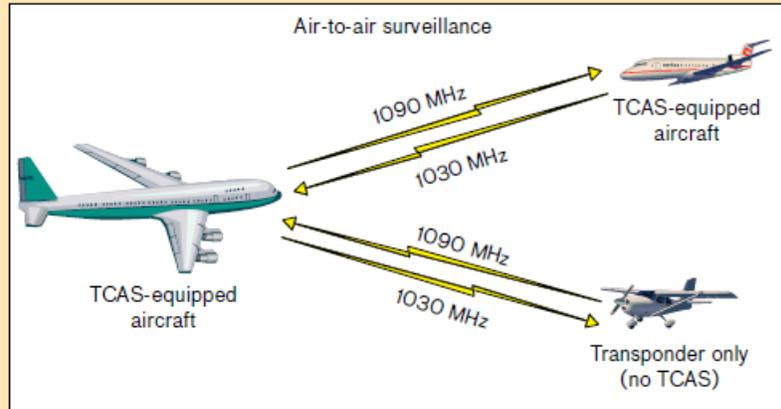
Figure C.1 and Figure C.2), which is able to monitor in real-time the existence of threats and provide the most appropriate avoidance for a particular case. However, this device is inhibited at any altitude below 1,000ft to prevent dangerous advisories at low altitude. According to (Federal Aviation Administration. U.S. Department of Transportation, 2011), General Aviation (GA) aircraft are not required to carry TCAS on board. In the absence of TCAS and transponders that provide a pilot-initiated collision avoidance system, other methods are required.

This appendix wants to give a deeper overview of the general Sense and Avoid (SAA) concept introduced and summarized in Chapter 2.

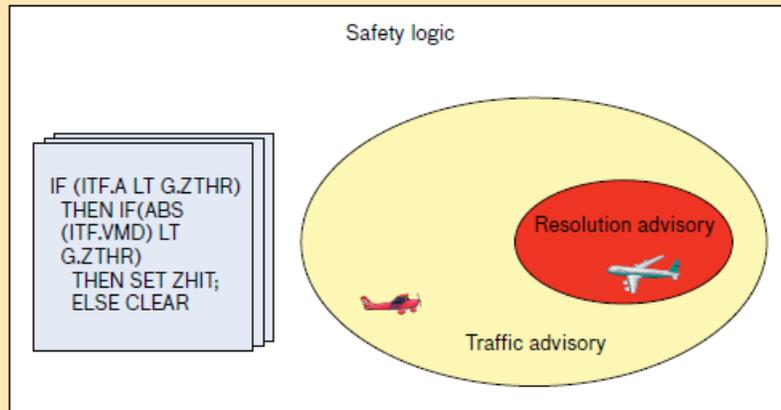
# How TCAS Works

There are four main components of TCAS: airborne surveillance, safety logic, vertical advisories, and a pilot interface. If another airborne aircraft is a potential threat, TCAS issues a traffic advisory (TA), which gives the pilots an audio announcement "Traffic, Traffic" and highlights the intruder on a traffic display. The TA is intended to help pilots achieve visual acquisition of other aircraft and prepare the pilots for a potential avoidance maneuver. If a maneuver becomes necessary, the system will issue a resolution advisory (RA), instructing the pilots to climb or descend to maintain a safe distance. There is an audio announcement of the required vertical maneuver, and the range of acceptable vertical rates is shown on the vertical speed indicator. On some aircraft, additional pitch guidance is provided to pilots.

TCAS may issue a variety of different advisories, including do not climb or descend, limit climb or descend to 500, 1,000, or 2,000 ft/min, level-off, climb or descend at 1,500 ft/min, increase climb or descend to 2,500 ft/min, or maintain current vertical rate. Depending on how the encounter evolves, TCAS may strengthen, weaken, or reverse the direction of the advisory. Note that an RA provides vertical guidance only; TCAS does not issue

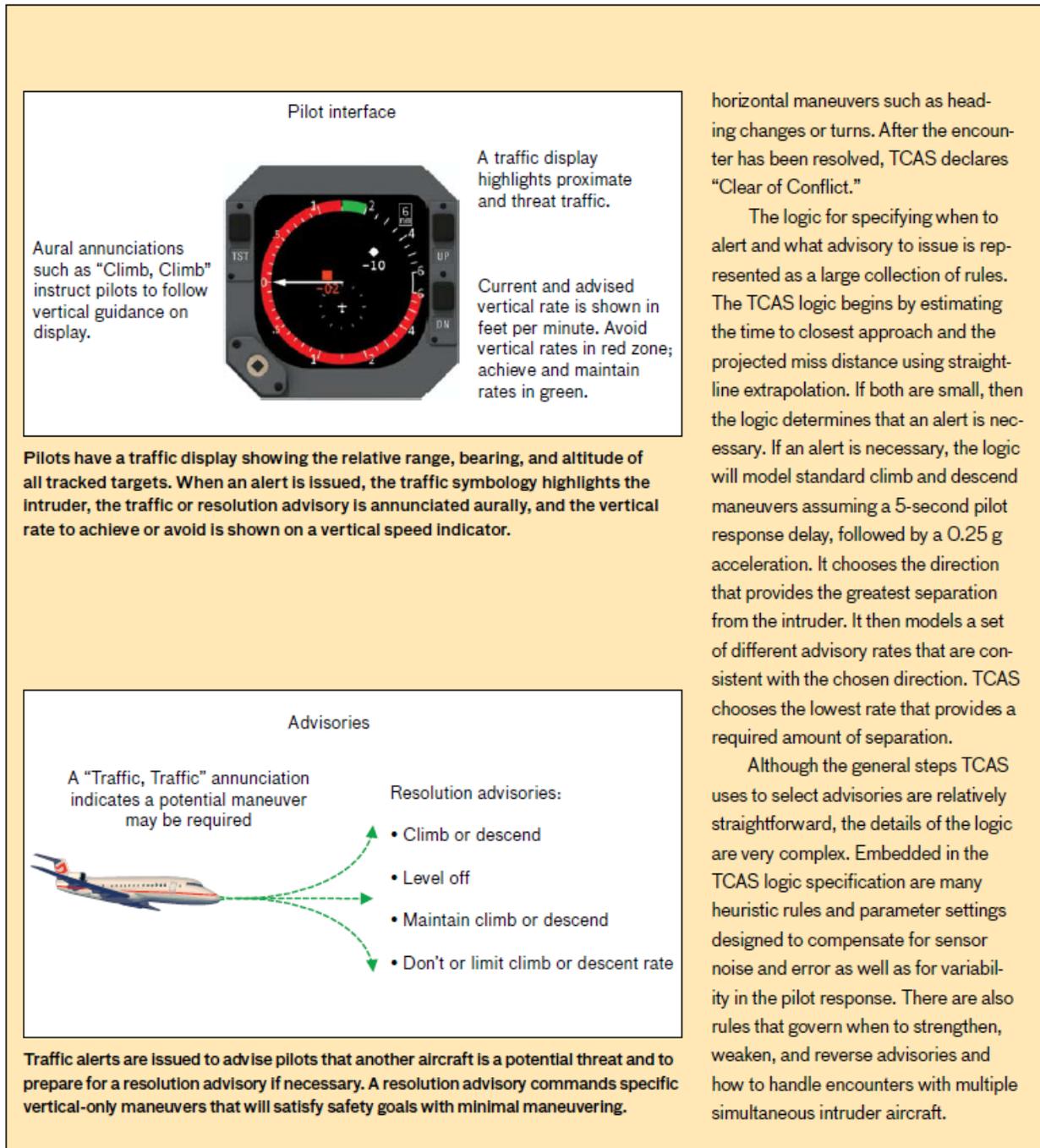


The TCAS surveillance unit interrogates nearby transponder-equipped aircraft. Traffic range, bearing, and altitude estimates are calculated based on the received time, location, and content of the reply. If the tracked aircraft is declared a threat and is also TCAS-equipped, the two TCAS units coordinate complementary advisories through discrete messages.



Advisory logic uses deterministic and heuristic rules to issue alerts against a potential threat on the basis of time of closest approach and projected miss distance.

**Figure C.1.** How TCAS Works (1/2) (Kochenderfer, Holland, & Chryssanthopoulos, 2012)



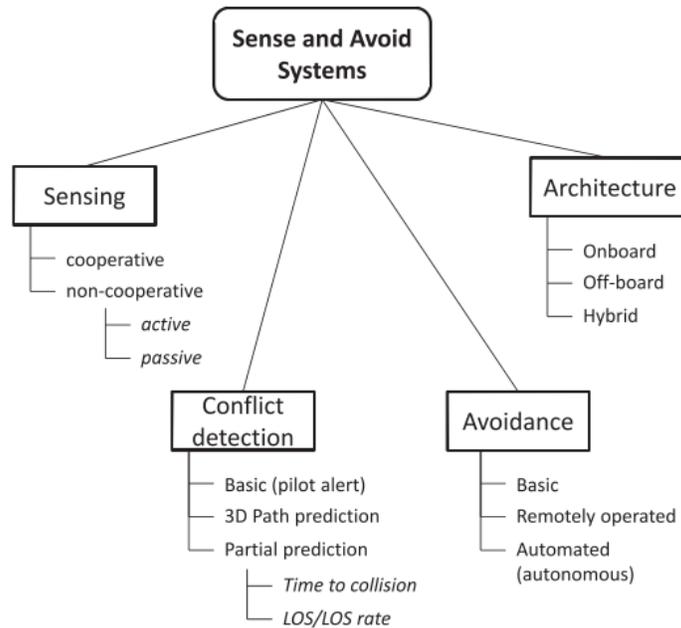
**Figure C.2.** How TCAS Works (2/2) (Kochenderfer et al., 2012)

## **C.1. SAA: Definitions and General Concepts**

The concept of sense and avoid is described as the capability of the aircraft to detect a hazard, track the intruder, estimate a possible collision, and avoid.

The fundamental objective of SAA methods is to provide the RPA pilot with equivalent “first-person view” capabilities to a piloted aircraft. In a wider perspective, the safe integration of RPAs into the airspace is not the only current application of SAA methods. Some examples can be found in the literature where SAA concepts are applied to landing approaches (Desaraju & Michael, 2014), target detection and recognition (Chanel, Teichteil-Königsbuch, & Lesire, 2012), and search and rescue with multiple RPAs (Baker, Ramchurn, Teacy, & Jennings, 2016).

A more detailed SAA structure (Figure C.3) can be found in the literature (Fasano et al., 2016) where the authors make a distinction between the conflict detection to identify the nature of an intrusion and the avoidance manoeuvre. As a general concept, SAA covers all the systems and sources of information involved to mitigate the lack of the capability for the first-person view of the RPAS.



**Figure C.3.** Taxonomy of SAA systems (Fasano et al., 2016)

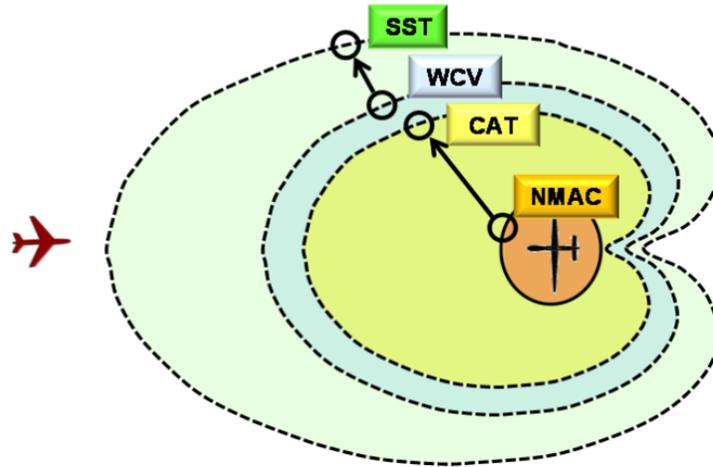
The focus of this work is on the action of avoidance. Therefore, a brief description of the other components is included in Sections C.1.2. in order to provide an overall context of the full SAA task.

### ***C.1.1. SAA Requirements***

In pursuance of simulating and testing avoidance manoeuvres, a near-collision situation needs to be replicated and subsequently, different scenarios must be defined depending on the risk level.

The functional boundaries expressed in Figure C.4 indicate the risk to a collision. The two main components are Self-Separation (SS) and Collision Avoidance (CA). The SS function aims to reduce the probability of a collision by ensuring that the aircraft remains well-clear. Therefore, the initial goal of the avoidance system is to start a procedure that ends before the Collision Avoidance Threshold (CAT). When the SS is lost by trespassing the Well-Clear Violation (WCV)

boundary and no action has been taken, the CA component engages immediate manoeuvres in a short period of time before an NMAC situation.



**Figure C.4.** Thresholds (Federal Aviation Administration, 2013b). SST: Self-separation threshold. WCV: Well-clear violation. CAT: Collision avoidance threshold. NMAC: Near mid-air collision

According to the current recommendations in Canada (Unmanned Systems Canada, 2017), the collision volume is defined by a cylindrical volume with a horizontal radius of 500ft and height of 200ft. The manoeuvre time ( $\tau$ ) is the time required by the aircraft to complete the task of avoiding the collision volume and the conflict point is the time to a predicted collision. Considering the human factor involved in the procedure that includes a 15s delay (Figure C.5), the minimum warning time for the pilot is then,  $2\tau+15^{10}$ ;  $\tau$  is doubled in order to increase the safety margin. The avoidance system should execute an avoidance manoeuvre  $2\tau$  seconds before the conflict point. Depending on the manoeuvre time, the aircraft capabilities, and the environment, the manoeuvre task varies.

<sup>10</sup> Recommended best practices in Canada (Unmanned Systems Canada, 2017).

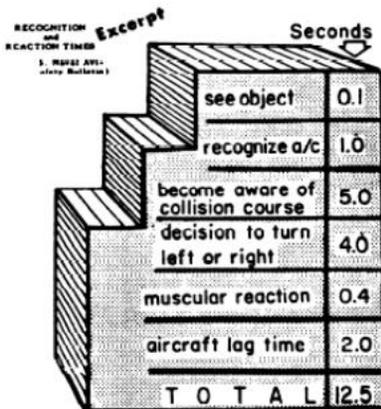


Figure C.5. Aircraft recognition and reaction time (Unmanned Systems Canada, 2017)

### C.1.2. Sensing and Conflict Detection

In the first stage of the SAA task, the aircraft identifies conflicting traffic. Extensive work has been done on sensors for environment surveillance and threat detection over the years (Pajares, 2015; Yu & Zhang, 2015). This thesis focuses on CA and for that reason, there is no interest in broadly studying these sensors. However, in order to provide some context to the CA problem, the most significant devices are summarized in Table C.1.

Overall, sensing can be classified as cooperative (e.g. transponders) or non-cooperative depending on if the system is able to interrogate and share information with other aircraft in the airspace. Cooperative sensors aim to emulate the pilot capabilities of detection and identification but to date, no system has been an accurate and real representation of a pilot.

The non-cooperative sensors include active (e.g. RADAR) or passive sensors (e.g. cameras). Whereas active sensors are attractive but heavy and expensive, passive sensors are lighter and more effective for object detection.

**Table C.1.** Most significant sensing technology for SAA

Sensor	Highlights	Limitations
ADS-B (Strohmeier, Schafer, Lenders, & Martinovic, 2014)	<ul style="list-style-type: none"> <li>• Broadcast the aircraft location based on GPS information</li> <li>• Cooperative sensing solution</li> </ul>	<ul style="list-style-type: none"> <li>• Only useful if other aircraft are using the same system</li> </ul>
RADAR (Caris, Stanko, Palm, Sommer, & Pohl, 2015; Institution of Electrical Engineers., Drolet, & Bray, 2017)	<ul style="list-style-type: none"> <li>• Best sensing solution for airborne surveillance</li> <li>• All-weather and all-time</li> </ul>	<ul style="list-style-type: none"> <li>• Large size</li> <li>• False objects might be detected if the pulse frequency is not correctly calculated</li> </ul>
LIDAR (Ramasamy, Sabatini, Gardi, & Liu, 2016)	<ul style="list-style-type: none"> <li>• Similar concept as RADARs but smaller size</li> <li>• Effective when used with other sensors as a secondary device</li> </ul>	<ul style="list-style-type: none"> <li>• Slow scanning process</li> <li>• Only feasible for short-range if used as a primary sensor</li> </ul>
Acoustic sensors (Finn & Franklin, 2011)	<ul style="list-style-type: none"> <li>• Low cost</li> <li>• Auxiliary sensor</li> </ul>	<ul style="list-style-type: none"> <li>• Rough obstacle position detection</li> <li>• Large disturbances</li> </ul>
EO/IR cameras (Griffith, Kochenderfer, & Kuchar, 2008)	<ul style="list-style-type: none"> <li>• Low cost</li> <li>• Accurate detection</li> <li>• Used as a primary sensor</li> </ul>	<ul style="list-style-type: none"> <li>• Low-cost solutions are limited to day-time detection</li> <li>• Blur effect due to aircraft motion</li> </ul>
Ground-based sensors (Young, 2018)	<ul style="list-style-type: none"> <li>• Accurate range and bearing information</li> <li>• Absolute location and velocity of the intruding aircraft</li> <li>• Not RF link dependent</li> <li>• No additional SWaP onboard</li> </ul>	<ul style="list-style-type: none"> <li>• Limited to a fixed area</li> </ul>

## **C.2. CA Methods**

Autonomous avoidance techniques remain a great research topic. Depending on factors such as sensor detection range and aircraft capabilities, the avoidance techniques ranges; distinct methods should be implemented in order to keep the system safe at all times.

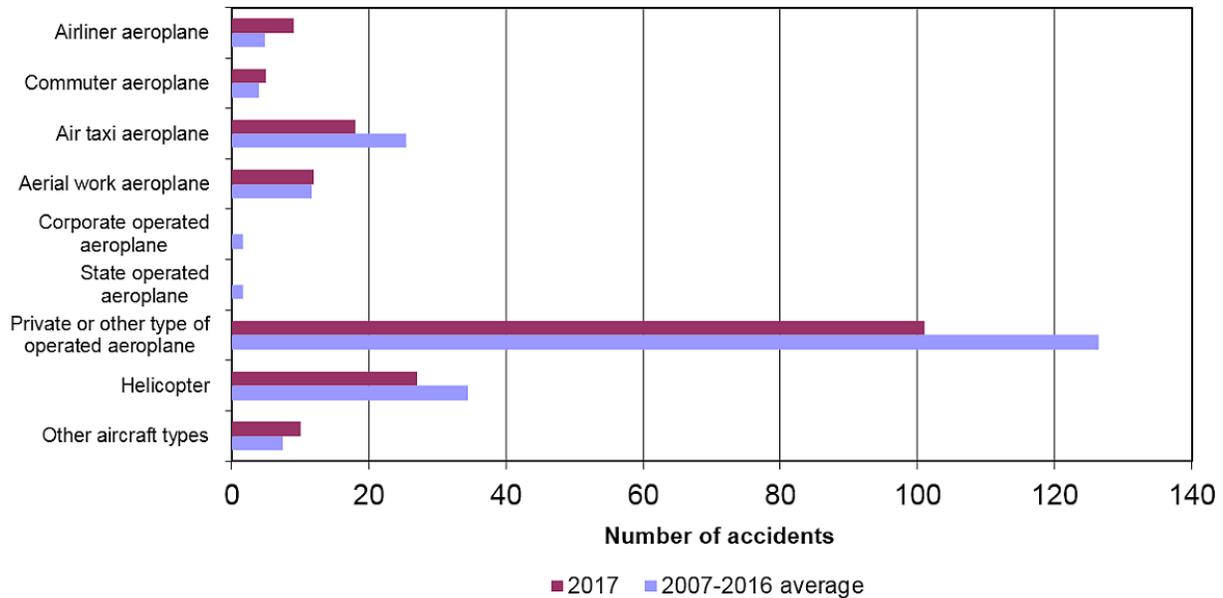
With advances in technology, the current air traffic system is undergoing a revision and modernization. The current airspace is overloaded and the Air Traffic Management (ATM) system that was defined in the 70s does not support it. The NextGen project aims to provide a new ATM system able to accommodate all new smaller classes of vehicles, such as RPAs in the airspace and provide a collision avoidance system for all aircraft (Kochenderfer et al., 2012; McCallie, Butts, & Mills, 2011). The work included in this document only considers the current airspace structure but the author is aware that a new system may replace the topics and methods discussed here.

### ***C.2.1. CA for GA***

TCAS is the standard collision avoidance system in commercial aircraft (

Figure C.1 and Figure C.2). However, GA is usually not equipped with this system since TCAS II is not mandatory for aircraft with less than 30 seats and below 33,000 lbs (Federal Aviation Administration. U.S. Department of Transportation, 2011). As an example, the Twin Otter and the Cessna 172 aircraft, which are the representative GA aircraft included in this thesis have 21 seats-5,900lbs and 4 seats-1,700lbs respectively. These aircraft usually fly under Visual Flight Rules (VFR) as opposed to Instrument Flight Rules (IFR) where the aircraft operation is mainly carried out by instruments reference.

The issues around GA not carrying any avoidance systems onboard are reflected in the number of air accidents over the last year (Figure C.6). Although these occurrences were lower than the average for the previous 10 years, there is still a significant difference between commercial aircraft and GA.



**Figure C.6.** Air occurrences in 2017. Accidents involving Canadian-registered aircraft, by operation type, 2017  
(Transportation Safety Board of Canada, 2017)

A simpler version of the current TCAS, TCAS I, was intended to be used in GA to issue Traffic Advisories (TAs) to pilots. Although some aircraft and helicopters carry a TCAS system onboard, most GA do not because it is not mandatory or standardized.

Based on the existing TCAS II concept, an alternative is to implement the system in GA. However, the Resolution Advisories (RAs) suggested actions were outside the aircraft limits, since GA flights fall into a different category. Therefore, the full TCAS SAA system would need

to be modified and replaced according to the aircraft requirements, which is not an effective procedure.

As a second approach, a modified system was tested with descend/climb coordination with successful results (Narkawicz, Muñoz, & Dutle, 2016). This system was based on the Vertical Resolution Advisory Complement (VRAC) TCAS II responsive coordination. However, the system always issued a descend/climb manoeuvre when an encounter happened, regardless of the nature of the threat. The coordination of the advisories between different systems introduces another challenge. This must be adjusted by designing compatible devices since different surveillance systems might lower the reliability of the entire avoidance system. This problem can be mitigated by using an algorithm that evaluates the error with an added cost to the system.

More refined approaches have tried to estimate the future location of the other aircraft by optimization. Those are based on probabilistic models from a Markov Decision Process (MDP) (Balachandran & Atkins, 2017; Gardner et al., 2016; Kochenderfer & Chryssanthacopoulos, 2013) whose intent is to model the encounter to increase the reliability of the CA algorithm.

The pilot response is another issue that CA in GA must account for, since the pilot is currently the system component that makes the final decision to perform a particular manoeuvre. Implementing a new avoidance system means a process of familiarization that might lead to more risk situations and incompatibilities during the early stages of its integration. The human factor issue is a large research topic that is outside the scope of this thesis. Those interested in the subject can consult the literature in references (Kozuba, 2011; Salas, Maurino, & Curtis, 2010).

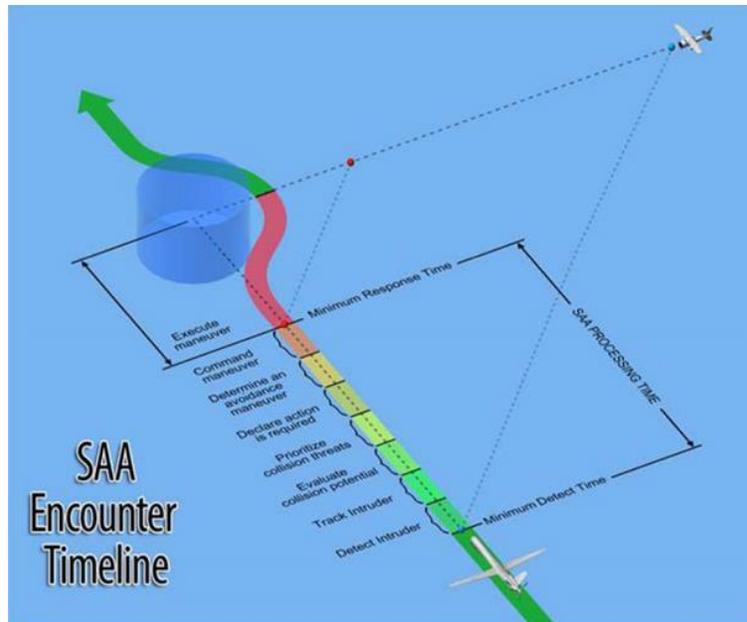
Primarily, the CA topic in GA remains a strong ongoing research topic that investigates the problems of human factors, coordination with the current TCAS II and CA methods.

### ***C.2.2. CA for RPAs***

With the recent integration of RPAS into the airspace, the SAA in RPAS has become a topic of interest. RPAS technology is rapidly developing, but the lack of a human pilot onboard introduces challenges when it comes to providing the RPAS with the same safety level as a piloted aircraft. The first intuitive approach, similar to the GA case, is to examine TCAS alternatives for RPAS (Brooker & Wo, 2017). The current ATM structure does not manage small aircraft and the TCAS avoidance system would require severe improvements (e.g. ACAS-Xu project (Marston, Operations, & Baca, 2015)); a process that would be costly and lengthy.

Taking into account the variety of sensors and RPAS classes, the detection time is generally limited by the sensor efficiency. This means that designing a specific avoidance procedure for each of the different sensors or aircraft classes is unreasonable; standards and general methods must be designed and discussed. For example, risk assessment studies for CA have shown strong results by focusing on RADAR data (Artacho, 2018; Fang, 2018).

In order to eliminate the sensor issue, CA methods can assume that an object has already been detected. Therefore, the remaining tasks are: first, the decision that selects the best manoeuvre to perform and second, the control actions in response to that situation (Figure C.7).



**Figure C.7.** SAA encounter timeline (Federal Aviation Administration, 2013b)

Since RPAS share the airspace with piloted aircraft, certain CA methods try to model the pilot's behaviour as a way to estimate the piloted aircraft's performance and select the best manoeuvre (Londner & Moss, 2017; Zhang & Mcgovern, 2009). The MDP and Monte-Carlo methods are the most common approaches for modelling the pilot's performance (Adaska, Obermeyer, & Schmidt, 2014; Sahawneh, Mackie, Spencer, Beard, & Warnick, 2015; Temizer, Kochenderfer, Kaelbling, Lozano-Pérez, & Kuchar, 2010). Although it is important that the RPAS understands human behaviour in order to choose the most appropriate solution, the reliability of these methodologies is questioned since it is nearly impossible to reproduce and predict a human's behaviour using statistical models.

In order to eliminate this issue, some approaches follow the TCAS convention of permitting a vertical-only manoeuvre (Marston et al., 2015). However, vertical avoidance is not always the fastest and most effective measure to lead the aircraft out of a collision. In response, some

approaches have designed 3D manoeuvres that require complex calculations (Alligier, Allignol, Barnier, Durand, & Wang, 2018).

There are two main ways to give the aircraft the capability of avoidance: (1) pilot-in-the-loop control and (2) automatic control. Remote avoidance systems are limited to VLOS missions, where the pilot has full control of the aircraft performance (Stegagno, Basile, Bulthoff, & Franchi, 2014). BVLOS operations have more autonomy to follow a manoeuvring procedure with the human supervising; though the mission scope widens, the system has to rely on the sensors.

Aircraft automated capabilities are given either by Ground-Based Sense and Avoid (GBSAA) or by Airborne-Based Sense and Avoid (ABSAA). GBSAA systems are exposed to communication delays or misses between the GS and the vehicle. This limitation makes GBSAA procedures unreliable for CA, since the time to a collision is reduced to a few seconds. Per contra, ABSAA systems permit the integration of a wider range of sensors onboard, eliminating the communication problem and allowing BVLOS avoidance. This structure gives the RPAS the required autonomy to identify the hazard, make a decision on the avoidance and perform a manoeuvre. This means that the entire task relies on sensors that could be noisy and might not reflect the changes in aircraft dynamics correctly. Whereas most of the research around this topic has focused on GBSAA methods because it eliminates the Size, Weight and Power (SWaP) problem in small fixed-wing aircraft (Ray Young & Brenton, 2016; Rhodes, 2017), fully ABSAA approaches remain an active research topic, since the approach depends on the sensor that determines the detection range and, therefore, the avoidance operation (Wang et al., 2016).

Another issue associated with SAA in RPAS, but not related to the aircraft performance, is the minimum level of safety that the SAA task must provide (Stevenson, 2015). Other research has tried to answer this challenge by defining a general framework for all RPAS classes in the airspace (Melnyk, Schrage, Volovoi, & Jimenez, 2014).

Whereas CA only permits a few seconds to execute a fast manoeuvre, most of the work found in the literature focuses on the SS task, since it allows more time to perform an avoidance (Cook, Brooks, Cole, Hackenberg, & Raska, 2015; E. R. Mueller, Isaacson, & Stevens, 2016; Ott, 2015). The CA approach in this document examines the aircraft capabilities to dive at its maximum rate in order to avoid a collision without the limitations of the sensing systems, communication delays with the GCS, and complex manoeuvres.

Overall, the minimum requirements for a CA manoeuvre are:

- 1- Time to a collision, which is determined by the boundaries and thresholds expressed in Figure C.4; the closer to an NMAC the faster the manoeuvre must be.
- 2- Vehicle capabilities, which vary from gliders to GA and RPAs.
- 3- Environmental conditions.

# Appendix D

## *Encounter Geometries*

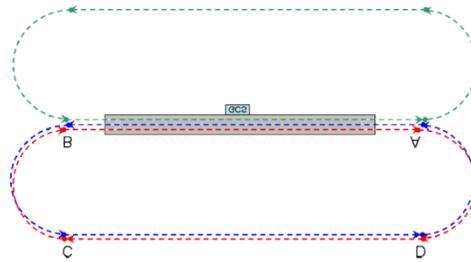
It is intuitive to think that the geometry of the encounter will delimit the time to a collision. In fact, the encounter geometry directly affects the detection range of the sensing system. Since the sensing part of Sense and Avoid (SAA) is outside of the scope of this thesis, this appendix only focuses on the encounter geometry used in Chapter 5 and the reason behind it.

The Phi ( $\Phi$ ) manoeuvre was the encounter geometry used by the Remote Aerial Vehicles for ENvironment-monitoring (RAVEN) group at Memorial University until 2014 for the development of 4D simulation environments. Its original purpose was to provide a platform for the study of SAA coordinated techniques between Remotely Piloted Aircraft (RPA).

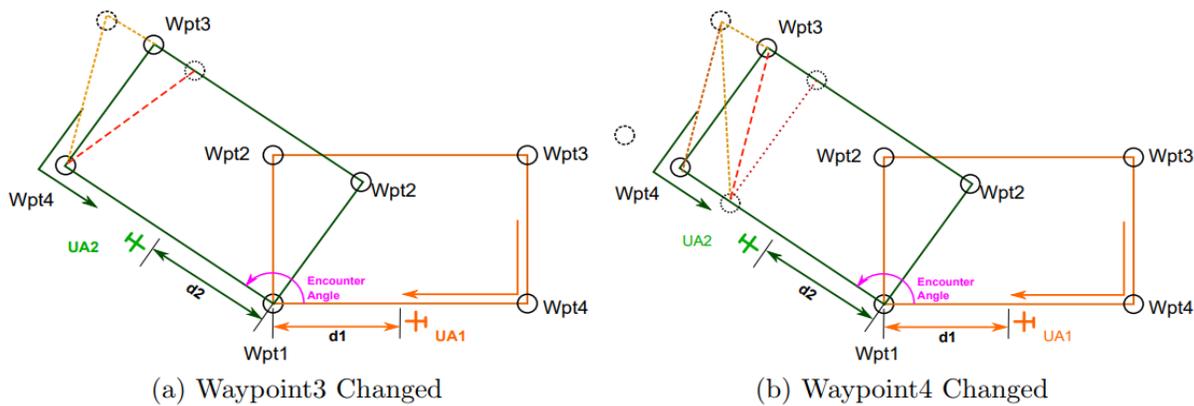
The initial geometry was called *opposing circuits* and as its name indicates, the procedure was defined by two rectangular paths that shared similar sections. In this architecture, the two paths had the same waypoints but the two aircraft flew in opposite directions (Figure D.1– red and blue). In a later version, the second aircraft was set to fly the same rectangular geometry but only sharing two waypoints out of the 4 that defined the rectangle (Figure D.1– green). Although encounters were possible, this geometry presented added difficulties to coordinate both aircraft.

An improved geometry focused on the synchronization of the aircraft by communicating the GS of both systems and modifying the flight path depending on an estimated reference point (Figure D.2). The goal was to minimize the distance error ( $d_1-d_2$ ) by adjusting the flight plan as the

aircraft are approached the encounter point. This alternative was dismissed since it was highly dependent on the flight conditions and the aircraft airspeed.

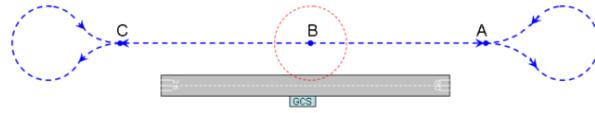


**Figure D.1.** 4D Opposing circuits. Ideal case (Cereceda & Stevenson, 2014)



**Figure D.2.** Waypoint adjustments to synchronize the time of arrival (Fang, 2014)

The proposed manoeuvre solves the main challenges presented by the above-mentioned geometries (Cereceda & Stevenson, 2014). The Phi ( $\Phi$ ) manoeuvre offers simplicity and the possibility of allowing more frequent encounters. This flight path evolved from the original idea as defined by opposing circuits (Figure D.1) to only 1 waypoint (B) for the first aircraft (Figure D.3- red path) and 2 (A and C) for the second (Figure D.3 – blue path). With an aircraft flying around a waypoint halfway between the returning waypoints of the other path, the synchronization problem is eliminated and the frequency of an encounter increases. This geometry also offers the study of the angle of the encounter by changing the radius of the orbital path in the first aircraft (out of the scope of this thesis).



**Figure D.3.** The Phi ( $\Phi$ ) manoeuvre (Cereceda & Stevenson, 2014)

# Appendix E

## *Code and Configuration Files*

### **D.1. General Aircraft Configuration Files**

#### *D.1.1. Cessna 172 Configuration File*

Available online on the JSBSim SourceForge website (“JSBSim Flight Dynamics Model - Code aircraft/c172x,” 2009).

#### *D.1.2. Twin Otter Configuration File*

Available online on the JSBSim SourceForge website (“JSBSim Flight Dynamics Model - Code aircraft/DHC6,” n.d.).

### **D.2. RPA Configuration Files**

#### *D.2.1. EPP FPV Aircraft Configuration File*

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl"
href="http://jsbsim.sourceforge.net/JSBSim.xsl"?>
<fdm_config name="EPP FPV" version="2.0" release="ALPHA"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://jsbsim.sourceforge.net/JSBSim.xsd">

  <fileheader>
    <author> Oihane Cereceda </author>
    <filecreationdate> 01-20-2016 </filecreationdate>
    <version> v4 </version>
    <description>
      EPP-FPV JSBSim model
```

```

v1: Files and values added from data
v2: Add some data from the generated values in Aeromatic
v3: Redimension of the CG
v4: Adjustments from flight data
</description>
</fileheader>

<metrics>
<wingarea unit="FT2"> 4.24 </wingarea>
<wingspan unit="FT"> 5.90551 </wingspan>
<wing_incidence> 3.00 </wing_incidence>
<chord unit="FT"> 0.7217848 </chord>
<htailarea unit="FT2"> 0.7 </htailarea>
<htailarm unit="FT"> 2.723097 </htailarm>
<vtailarea unit="FT2"> 0.23 </vtailarea>
<vtailarm unit="FT"> 2.559055 </vtailarm>
<location name="AERORP" unit="IN">
  <x> 13.9764 </x>
  <y> 0 </y>
  <z> 3.937 </z>
</location>
</metrics>

<mass_balance>
<ixx unit="SLUG*FT2"> 0.05034 </ixx>
<iyy unit="SLUG*FT2"> 0.01701 </iyy>
<izz unit="SLUG*FT2"> 0.12940 </izz>
<ixy unit="SLUG*FT2"> -0 </ixy>
<ixz unit="SLUG*FT2"> 0.00911 </ixz>
<iyz unit="SLUG*FT2"> -0 </iyz>
<emptywt unit="LBS"> 4.4 </emptywt>
<location name="CG" unit="IN">
  <x> 16.54 </x>
  <y> 0.00 </y>
  <z> 0.0 </z>
</location>
</mass_balance>

<ground_reactions>11

<contact type="STRUCTURE" name="LEFT_WING">
  <location unit="IN">
    <x> 11.81 </x>
    <y> -35.43 </y>
    <z> 0.0 </z>
  </location>
  <static_friction> 1.00 </static_friction>
  <dynamic_friction> 1.00 </dynamic_friction>
  <spring_coeff unit="LBS/FT"> 14.99 </spring_coeff>
  <damping_coeff unit="LBS/FT/SEC"> 14.99 </damping_coeff>
</contact>

```

---

<sup>11</sup> Note that the <ground\_reactions> section in the JSBSim aircraft configuration file should not be left blank regardless of the absence of a landing gear system such as in the glider case. The contact type STRUCTURE should include all contacts of the airframe with the ground such as wings and tail.

```

<contact type="STRUCTURE" name="RIGHT_WING">
  <location unit="IN">
    <x> 11.81 </x>
    <y> 35.43 </y>
    <z> 0.0 </z>
  </location>
  <static_friction> 1.00 </static_friction>
  <dynamic_friction> 1.00 </dynamic_friction>
  <spring_coeff unit="LBS/FT"> 14.99 </spring_coeff>
  <damping_coeff unit="LBS/FT/SEC"> 14.99 </damping_coeff>
</contact>

<contact type="STRUCTURE" name="TAIL">
  <location unit="IN">
    <x> 51.97 </x>
    <y> 0.0 </y>
    <z> -3.94 </z>
  </location>
  <static_friction> 1.00 </static_friction>
  <dynamic_friction> 1.00 </dynamic_friction>
  <spring_coeff unit="LBS/FT"> 14.99 </spring_coeff>
  <damping_coeff unit="LBS/FT/SEC"> 14.99 </damping_coeff>
</contact>

</ground_reactions>

<propulsion>
  <engine file="engEPPFPV">
    <location unit="IN">
      <x> 20.47244 </x>
      <y> 0.0 </y>
      <z> 0.0 </z>
    </location>
    <orient unit="DEG">
      <roll> 0.0 </roll>
      <pitch> 0 </pitch>
      <yaw> 0 </yaw>
    </orient>
    <feed>0</feed>
    <thruster file="propEPPFPV">
      <location unit="IN">
        <x> 20.47244 </x>
        <y> 0.0 </y>
        <z> 0.0 </z>
      </location>
      <orient unit="DEG">
        <roll> 0.0 </roll>
        <pitch> 0.0 </pitch>
        <yaw> 0.0 </yaw>
      </orient>
    </thruster>
  </engine>

</propulsion>

<!-- <autopilot file="EPPFPVap"/> -->

```

```

<!-- <system file="autothrottle"/> -->

<flight_control name="EPPFPV">

  <channel name="Pitch">
    <summer name="Pitch Trim Sum">
      <input>fcs/elevator-cmd-norm</input>
      <input>fcs/pitch-trim-cmd-norm</input>
      <clipto>
        <min>-1</min>
        <max>1</max>
      </clipto>
    </summer>

    <aerosurface_scale name="Elevator Control">
      <input>fcs/pitch-trim-sum</input>
      <range>
        <min>-1</min>
        <max>1</max>
      </range>
      <output>fcs/elevator-pos-rad</output>
    </aerosurface_scale>
  </channel>

  <channel name="Roll">
    <summer name="Roll Trim Sum">
      <input>fcs/aileron-cmd-norm</input>
      <input>fcs/roll-trim-cmd-norm</input>
      <clipto>
        <min>-1</min>
        <max>1</max>
      </clipto>
    </summer>

    <aerosurface_scale name="Aileron Control">
      <input>fcs/aileron-cmd-norm</input>
      <range>
        <min>-1</min>
        <max>1</max>
      </range>
      <output>fcs/left-aileron-pos-rad</output>
    </aerosurface_scale>
  </channel>

  <channel name="Yaw">
    <summer name="Yaw Trim Sum">
      <input>fcs/rudder-cmd-norm</input>
      <input>fcs/yaw-trim-cmd-norm</input>
      <clipto>
        <min>-1</min>
        <max>1</max>
      </clipto>
    </summer>

    <aerosurface_scale name="Rudder Control">
      <input>fcs/rudder-cmd-norm</input>

```

```

    <range>
      <min>-1</min>
      <max>1</max>
    </range>
    <output>fcs/rudder-pos-rad</output>
  </aerosurface_scale>
</channel>

</flight_control>

<aerodynamics>
  <axis name="DRAG">
    <function name="aero/coefficient/CDo">
      <description>Drag_at_zero_lift</description>
      <product>
        <property>aero/qbar-psf</property>
        <property>metrics/Sw-sqft</property>
        <value>0.0007</value>
      </product>
    </function>

    <function name="aero/coefficient/CDwbh">
      <description>Drag_due_to_alpha</description>
      <product>
        <property>aero/qbar-psf</property>
        <property>metrics/Sw-sqft</property>
        <table>
          <independentVar lookup="row">aero/alpha-rad</independentVar>
          <tableData>
            -0.0175  0.0100
            0.0000  0.0150
            0.0175  0.0200
            0.0349  0.0250
            0.0524  0.0300
            0.0698  0.0350
            0.0873  0.0400
            0.1745  0.1000
            0.2618  0.2300
            0.3491  0.3700
            0.4363  0.5000
            0.5236  0.6000
            0.6109  0.7600
            0.6981  0.8500
            0.7854  0.9600
            0.8727  1.0600
            0.9599  1.1400
            1.0472  1.2200
            1.2217  1.3800
            1.3963  1.5000
            1.5708  1.4600
          </tableData>
        </table>
      </product>
    </function>

    <function name="aero/coefficient/CDDe">

```

```

<description>Drag_due_to_Elevator_Deflection</description>
<product>
  <property>aero/qbar-psf</property>
  <property>metrics/Sw-sqft</property>
  <table>
    <independentVar>fcs/elevator-pos-rad</independentVar>
    <tableData>
      -1.0000  0.1140
      0.0000  0.0000
      1.0000  0.1140
    </tableData>
  </table>
</product>
</function>
</axis>

```

```

<axis name="SIDE">
  <function name="aero/coefficient/CYb">
    <description>Side_force_due_to_beta</description>
    <product>
      <property>aero/qbar-psf</property>
      <property>metrics/Sw-sqft</property>
      <property>aero/beta-rad</property>
      <value>-0.2850</value>
    </product>
  </function>

```

```

<function name="aero/coefficient/CYda">
  <description>Side_force_due_to_aileron</description>
  <product>
    <property>aero/qbar-psf</property>
    <property>metrics/Sw-sqft</property>
    <property>fcs/left-aileron-pos-rad</property>
    <value>-0.0456</value>
  </product>
</function>

```

```

<function name="aero/coefficient/CYdr">
  <description>Side_force_due_to_rudder</description>
  <product>
    <property>aero/qbar-psf</property>
    <property>metrics/Sw-sqft</property>
    <property>fcs/rudder-pos-rad</property>
    <value>0.1880</value>
  </product>
</function>
</axis>

```

```

<axis name="LIFT">
  <function name="aero/coefficient/CLwbh">
    <description>Lift_due_to_alpha</description>
    <product>
      <property>aero/qbar-psf</property>
      <property>metrics/Sw-sqft</property>
      <table>
        <independentVar>aero/alpha-rad</independentVar>
        <tableData>

```

-0.1571	0.0000
-0.1369	0.0600
-0.1222	0.1200
-0.0900	0.1900
-0.0524	0.2600
-0.0349	0.3300
-0.0175	0.4300
0.0000	0.4800
0.0175	0.5600
0.0349	0.6400
0.0524	0.7100
0.0698	0.7800
0.0873	0.8200
0.1047	0.9200
0.1222	0.9900
0.1369	1.0600
0.1571	1.1000
0.1745	1.1600
0.1920	1.2200
0.2094	1.2500
0.2269	1.2600
0.2444	1.2500
0.2618	1.2400
0.2793	1.2100
0.2967	1.1600
0.3142	1.1400
0.3316	1.1400
0.3491	1.0900
0.4363	0.9800
0.5236	0.8800
0.6109	0.8300
0.6981	0.8400
0.7854	0.8200
0.8727	0.7900
0.9599	0.7400
1.0472	0.6600
1.2217	0.4700
1.3963	0.2600
1.5708	0.0300

```

</tableData>
</table>
</product>
</function>

<function name="aero/coefficient/CLDe">
  <description>Lift_due_to_Elevator_Deflection</description>
  <product>
    <property>aero/qbar-psf</property>
    <property>metrics/Sw-sqft</property>
    <property>fcs/elevator-pos-rad</property>
    <value>0.3420</value>
  </product>
</function>
</axis>

<axis name="ROLL">
  <function name="aero/coefficient/Clo">

```

```

    <description>Roll_moment_due_to_vertical_tail_incidence</description>
    <product>
      <property>aero/qbar-psf</property>
      <property>metrics/Sw-sqft</property>
      <property>metrics/bw-ft</property>
      <value>-0.0000</value>
    </product>
  </function>

<function name="aero/coefficient/Clb">
  <description>Roll_moment_due_to_beta</description>
  <product>
    <property>aero/qbar-psf</property>
    <property>metrics/Sw-sqft</property>
    <property>metrics/bw-ft</property>
    <property>aero/beta-rad</property>
    <value>-0.0513</value>
  </product>
</function>

<function name="aero/coefficient/Clp">

<description>Roll_moment_due_to_roll_rate_(roll_damping)</description>
  <product>
    <property>aero/qbar-psf</property>
    <property>metrics/Sw-sqft</property>
    <property>metrics/bw-ft</property>
    <property>aero/bi2vel</property>
    <property>velocities/p-aero-rad_sec</property>
    <value>-0.4700</value>
  </product>
</function>

<function name="aero/coefficient/Clr">
  <description>Roll_moment_due_to_yaw_rate</description>
  <product>
    <property>aero/qbar-psf</property>
    <property>metrics/Sw-sqft</property>
    <property>metrics/bw-ft</property>
    <property>aero/bi2vel</property>
    <property>velocities/r-aero-rad_sec</property>
    <value>0.1500</value>
  </product>
</function>

<function name="aero/coefficient/Clda">
  <description>Roll_moment_due_to_aileron</description>
  <product>
    <property>aero/qbar-psf</property>
    <property>metrics/Sw-sqft</property>
    <property>metrics/bw-ft</property>
    <property>fcs/left-aileron-pos-rad</property>
    <value>0.2500</value>
  </product>
</function>

<function name="aero/coefficient/Cldr">

```

```

    <description>Roll_moment_due_to_rudder</description>
    <product>
      <property>aero/qbar-psf</property>
      <property>metrics/Sw-sqft</property>
      <property>metrics/bw-ft</property>
      <property>fcs/rudder-pos-rad</property>
      <value>0.0046</value>
    </product>
  </function>
</axis>

<axis name="PITCH">
  <function name="aero/coefficient/Cmalpha">
    <description>Pitch_moment_due_to_alpha</description>
    <product>
      <property>aero/qbar-psf</property>
      <property>metrics/Sw-sqft</property>
      <property>metrics/cbarw-ft</property>
      <property>aero/alpha-rad</property>
      <value>-0.5730</value>
    </product>
  </function>

  <function name="aero/coefficient/Cmq">
    <description>Pitch_moment_due_to_pitch_rate</description>
    <product>
      <property>aero/qbar-psf</property>
      <property>metrics/Sw-sqft</property>
      <property>metrics/cbarw-ft</property>
      <property>aero/ci2vel</property>
      <property>velocities/q-aero-rad_sec</property>
      <value>-9.0000</value>
    </product>
  </function>

  <function name="aero/coefficient/Cmadot">
    <description>Pitch_moment_due_to_alpha_rate</description>
    <product>
      <property>aero/qbar-psf</property>
      <property>metrics/Sw-sqft</property>
      <property>metrics/cbarw-ft</property>
      <property>aero/ci2vel</property>
      <property>aero/alphadot-rad_sec</property>
      <value>-5.2000</value>
    </product>
  </function>

  <function name="aero/coefficient/Cmo">
    <description>Pitching_moment_at_zero_alpha</description>
    <product>
      <property>aero/qbar-psf</property>
      <property>metrics/Sw-sqft</property>
      <property>metrics/cbarw-ft</property>
      <value>0.0000</value>
    </product>
  </function>

```

```

<function name="aero/coefficient/Cmde">
  <description>Pitch_moment_due_to_elevator_deflection</description>
  <product>
    <property>aero/qbar-psf</property>
    <property>metrics/Sw-sqft</property>
    <property>metrics/cbarw-ft</property>
    <property>fcs/elevator-pos-rad</property>
    <value>-1.2610</value>
  </product>
</function>
</axis>

<axis name="YAW">
  <function name="aero/coefficient/Cnb">
    <description>Yaw_moment_due_to_beta</description>
    <product>
      <property>aero/qbar-psf</property>
      <property>metrics/Sw-sqft</property>
      <property>metrics/bw-ft</property>
      <property>aero/beta-rad</property>
      <value>0.0170</value>
    </product>
  </function>

  <function name="aero/coefficient/Cnp">
    <description>Yaw_moment_due_to_roll_rate</description>
    <product>
      <property>aero/qbar-psf</property>
      <property>metrics/Sw-sqft</property>
      <property>metrics/bw-ft</property>
      <property>aero/bi2vel</property>
      <property>velocities/p-aero-rad_sec</property>
      <value>-0.1800</value>
    </product>
  </function>

  <function name="aero/coefficient/Cnr">
    <description>Yaw_moment_due_to_yaw_rate</description>
    <product>
      <property>aero/qbar-psf</property>
      <property>metrics/Sw-sqft</property>
      <property>metrics/bw-ft</property>
      <property>aero/bi2vel</property>
      <property>velocities/r-aero-rad_sec</property>
      <value>-0.0250</value>
    </product>
  </function>

  <function name="aero/coefficient/Cnda">
    <description>Yaw_moment_due_to_aileron</description>
    <product>
      <property>aero/qbar-psf</property>
      <property>metrics/Sw-sqft</property>
      <property>metrics/bw-ft</property>
      <property>fcs/left-aileron-pos-rad</property>
      <value>0.0115</value>
    </product>
  </function>

```

```

</function>

<function name="aero/coefficient/Cndr">
  <description>Yaw_moment_due_to_rudder</description>
  <product>
    <property>aero/qbar-psf</property>
    <property>metrics/Sw-sqft</property>
    <property>metrics/bw-ft</property>
    <property>fcs/rudder-pos-rad</property>
    <value>-0.0370</value>
  </product>
</function>
</axis>
</aerodynamics>

<output name="localhost" type="FLIGHTGEAR" port="5500" protocol="UDP"
rate="10"> </output>

<output name="EPPFPV_Out.csv" type="CSV" rate="100">

  <simulation>      ON </simulation>
  <atmosphere>      OFF </atmosphere>
  <massprops>       OFF </massprops>
  <aerosurfaces>    OFF </aerosurfaces>
  <rates>           ON </rates>
  <velocities>      ON </velocities>
  <forces>          OFF </forces>
  <moments>         OFF </moments>
  <position>        ON </position>
  <coefficients>    OFF </coefficients>
  <ground_reactions> OFF </ground_reactions>
  <fcs>            ON </fcs>
  <propulsion>      OFF </propulsion>
</output>

</fdm_config>

```

### ***D.2.2. Giant Big Stik Aircraft Configuration File***

```

<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl"
href="http://jsbsim.sourceforge.net/JSBSim.xsl"?>
<fdm_config name="GBS_4" version="2.0" release="ALPHA"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://jsbsim.sourceforge.net/JSBSim.xsd">

  <fileheader>
    <author> Oihane Cereceda </author>
    <filecreationdate> 2015-11-11 </filecreationdate>
    <version> v4 </version>
    <description> Giant Big Stik JSBSim model
    v1: Files and values created from Aeromatic v0.82
    v2: Files modified according to more specific data

```

v3: Validate model using scripts and plotting data  
v4: Output to FlightGear to visualize the performance </description>  
</fileheader>

```

<metrics>
  <wingarea unit="FT2"> 10.538 </wingarea>
  <wingspan unit="FT"> 6.709 </wingspan>
  <wing_incidence> 2.00 </wing_incidence>
  <chord unit="FT"> 1.148 </chord>
  <htailarea unit="FT2"> 1.69 </htailarea>
  <htailarm unit="FT"> 2.36 </htailarm>
  <vtailarea unit="FT2"> 1.05 </vtailarea>
  <vtailarm unit="FT"> 2.27 </vtailarm>
  <location name="AERORP" unit="FT">
    <x> -1.541 </x>
    <y> 0 </y>
    <z> 0.213 </z>
  </location>
  <location name="EYEPOINT" unit="IN">
    <x> 7.09 </x>
    <y> 0.0 </y>
    <z> 45.00 </z>
  </location>
  <location name="VRP" unit="IN">
    <x> 0.0 </x>
    <y> 0.0 </y>
    <z> 0.0 </z>
  </location>
</metrics>

<mass_balance>
  <ixx unit="SLUG*FT2"> 0.305 </ixx>
  <iyy unit="SLUG*FT2"> 0.476 </iyy>
  <izz unit="SLUG*FT2"> 0.704 </izz>
  <ixy unit="SLUG*FT2"> 0 </ixy>
  <ixz unit="SLUG*FT2"> 0.095 </ixz>
  <iyz unit="SLUG*FT2"> 0 </iyz>
  <emptywt unit="LBS"> 14.771 </emptywt>
  <location name="CG" unit="FT">
    <x> -1.207 </x>
    <y> 0.0 </y>
    <z> 0.0 </z>
  </location>
</mass_balance>

<ground_reactions>

  <contact type="BOGEY" name="LEFT_MAIN">
    <location unit="FT">
      <x> -0.82 </x>
      <y> -0.853 </y>
      <z> 1.058 </z>
    </location>
    <static_friction> 0.8 </static_friction>
    <dynamic_friction> 0.9 </dynamic_friction>
    <rolling_friction> 0.02 </rolling_friction>
    <spring_coeff unit="LBS/FT"> 14.99 </spring_coeff>
  </contact>

```

```

    <damping_coeff unit="LBS/FT/SEC"> 7.50 </damping_coeff>
    <max_steer unit="DEG"> 0.0 </max_steer>
    <brake_group> NONE </brake_group>
    <retractable>0</retractable>
</contact>

<contact type="BOGEY" name="RIGHT_MAIN">
  <location unit="FT">
    <x> -0.25 </x>
    <y> 0.26 </y>
    <z> 1.058 </z>
  </location>
  <static_friction> 0.8 </static_friction>
  <dynamic_friction> 0.9 </dynamic_friction>
  <rolling_friction> 0.02 </rolling_friction>
  <spring_coeff unit="LBS/FT"> 14.99 </spring_coeff>
  <damping_coeff unit="LBS/FT/SEC"> 7.50 </damping_coeff>
  <max_steer unit="DEG"> 0.0 </max_steer>
  <brake_group> NONE </brake_group>
  <retractable>0</retractable>
</contact>

<contact type="BOGEY" name="NOSE">
  <location unit="FT">
    <x> -4.4 </x>
    <y> 0 </y>
    <z> 0.42 </z>
  </location>
  <static_friction> 0.8 </static_friction>
  <dynamic_friction> 0.9 </dynamic_friction>
  <rolling_friction> 0.02 </rolling_friction>
  <spring_coeff unit="LBS/FT"> 4.50 </spring_coeff>
  <damping_coeff unit="LBS/FT/SEC"> 7.50 </damping_coeff>
  <max_steer unit="DEG"> 10 </max_steer>
  <brake_group> NONE </brake_group>
  <retractable>0</retractable>
</contact>

<contact type="STRUCTURE" name="LEFT_WING">
  <location unit="FT">
    <x> -1.541 </x>
    <y> -3.355 </y>
    <z> -0.213 </z>
  </location>
  <static_friction> 1.00 </static_friction>
  <dynamic_friction> 1.00 </dynamic_friction>
  <spring_coeff unit="LBS/FT"> 14.99 </spring_coeff>
  <damping_coeff unit="LBS/FT/SEC"> 14.99 </damping_coeff>
</contact>

<contact type="STRUCTURE" name="RIGHT_WING">
  <location unit="FT">
    <x> -1.541 </x>
    <y> 3.355 </y>
    <z> -0.213 </z>
  </location>
  <static_friction> 1.00 </static_friction>

```

```

        <dynamic_friction> 1.00 </dynamic_friction>
        <spring_coeff unit="LBS/FT"> 14.99 </spring_coeff>
        <damping_coeff unit="LBS/FT/SEC"> 14.99 </damping_coeff>
    </contact>

</ground_reactions>

<propulsion>
    <engine file="Zenoah_G-26A">
        <location unit="IN">
            <x> -2.5984 </x>
            <y> -0.2244 </y>
            <z> -0.2244 </z>
        </location>
        <orient unit="DEG">
            <roll> 0.0 </roll>
            <pitch> 0 </pitch>
            <yaw> 0 </yaw>
        </orient>
        <feed>0</feed>
        <thruster file="propGBS_3">
            <location unit="IN">
                <x> -5.4724 </x>
                <y> -0.4724 </y>
                <z> -0.4724 </z>
            </location>
            <orient unit="DEG">
                <roll> 0.0 </roll>
                <pitch> 0.0 </pitch>
                <yaw> 0.0 </yaw>
            </orient>
            <sense> 1 </sense>
        </thruster>
    </engine>
    <tank type="FUEL">
        <!-- Tank number 0 -->
        <location unit="IN">
            <x> 4.7244 </x>
            <y> 0 </y>
            <z> 0 </z>
        </location>
        <capacity unit="LBS"> 1.1 </capacity>
        <contents unit="LBS"> 0.55 </contents>
    </tank>

</propulsion>

<system file="GNCUtilities_GBS"/>
<autopilot file="GBSap"/>
    <flight_control name="FCS: unnamed">

        <channel name="Pitch">

            <summer name="fcs/pitch-trim-sum">
                <input>ap/elevator_cmd</input>
                <input>fcs/elevator-cmd-norm</input>
                <input>fcs/pitch-trim-cmd-norm</input>
            </summer>
        </channel>
    </flight_control>

```

```

        <clipto>
            <min>-1</min>
            <max>1</max>
        </clipto>
    </summer>

<aerosurface_scale name="Elevator Control">
    <input>fcs/pitch-trim-sum</input>
    <gain>1.0</gain>
    <range>
        <min> -0.4643 </min>
        <max>  0.4643 </max>
    </range>
    <output>fcs/elevator-pos-rad</output>
</aerosurface_scale>

<aerosurface_scale name="elevator normalization">
    <input>fcs/elevator-pos-rad</input>
    <domain>
        <min> -1 </min>
        <max>  1 </max>
    </domain>
    <range>
        <min> -1 </min>
        <max>  1 </max>
    </range>
    <output>fcs/elevator-pos-norm</output>
</aerosurface_scale>

</channel>

<channel name="Roll">

    <summer name="Roll Trim Sum">
        <input>ap/aileron_cmd</input>
        <input>fcs/aileron-cmd-norm</input>
        <input>fcs/roll-trim-cmd-norm</input>
        <clipto>
            <min> -1 </min>
            <max>  1 </max>
        </clipto>
    </summer>

    <aerosurface_scale name="Left Aileron Control">
        <input>fcs/roll-trim-sum</input>
        <gain>1.0</gain>
        <range>
            <min> -0.569 </min>
            <max>  0.569 </max>
        </range>
        <output>fcs/left-aileron-pos-rad</output>
    </aerosurface_scale>

    <aerosurface_scale name="left aileron normalization">
        <input>fcs/left-aileron-pos-rad</input>
        <domain>
            <min> -1 </min>

```

```

        <max> 1 </max>
    </domain>
    <range>
        <min> -1 </min>
        <max> 1 </max>
    </range>
    <output>fcs/left-aileron-pos-norm</output>
</aerosurface_scale>

<aerosurface_scale name="Right Aileron Control">
    <input>-fcs/roll-trim-sum</input>
    <gain>1.0</gain>
    <range>
        <min> -0.569 </min>
        <max> 0.569 </max>
    </range>
    <output>fcs/right-aileron-pos-rad</output>
</aerosurface_scale>

<aerosurface_scale name="right aileron normalization">
    <input>fcs/right-aileron-pos-rad</input>
    <domain>
        <min> -1 </min>
        <max> 1 </max>
    </domain>
    <range>
        <min> -1 </min>
        <max> 1 </max>
    </range>
    <output>fcs/right-aileron-pos-norm</output>
</aerosurface_scale>

</channel>

<channel name="Yaw">

    <summer name="Rudder Command Sum">
        <input>fcs/rudder-cmd-norm</input>
        <input>fcs/yaw-trim-cmd-norm</input>
        <clipto>
            <min> -1 </min>
            <max> 1 </max>
        </clipto>
    </summer>

    <aerosurface_scale name="Rudder Control">
        <input>fcs/rudder-command-sum</input>
        <gain>1.0</gain>
        <range>
            <min> -1 </min>
            <max> 1 </max>
        </range>
        <output>fcs/rudder-pos-rad</output>
    </aerosurface_scale>

    <aerosurface_scale name="rudder normalization">
        <input>fcs/rudder-pos-rad</input>

```

```

    <domain>
      <min> -1 </min>
      <max> 1 </max>
    </domain>
    <range>
      <min> -1 </min>
      <max> 1 </max>
    </range>
    <output>fcs/rudder-pos-norm</output>
  </aerosurface_scale>

</channel>

<channel name="Landing Gear">
</channel>

</flight_control>

<aerodynamics>

  <axis name="LIFT">
    <function name="aero/coefficient/CLwbh">
      <description>Lift_due_to_alpha</description>
      <product>
        <property>aero/qbar-psf</property>
        <property>metrics/Sw-sqft</property>
        <property>aero/alpha-rad</property>
        <value>5.32</value>
      </product>
    </function>

    <function name="aero/coefficient/CLadot">
      <description>Lift_due_to_alpha_rate</description>
      <product>
        <property>aero/qbar-psf</property>
        <property>metrics/Sw-sqft</property>
        <property>aero/alphadot-rad_sec</property>
        <property>aero/ci2vel</property>
        <value>1.7</value>
      </product>
    </function>

    <function name="aero/coefficient/CLq">
      <description>Lift_due_to_pitch_rate</description>
      <product>
        <property>aero/qbar-psf</property>
        <property>metrics/Sw-sqft</property>
        <property>velocities/q-aero-rad_sec</property>
        <property>aero/ci2vel</property>
        <value>3.9</value>
      </product>
    </function>

    <function name="aero/coefficient/CLDe">
      <description>Lift_due_to_Elevator_Deflection</description>
      <product>

```

```

    <property>aero/qbar-psf</property>
    <property>metrics/Sw-sqft</property>
    <property>fcs/elevator-pos-rad</property>
    <value>-5.0</value>
  </product>
</function>
</axis>

<axis name="DRAG">
  <function name="aero/coefficient/CDo">
    <description>Drag_at_zero_lift</description>
    <product>
      <property>aero/qbar-psf</property>
      <property>metrics/Sw-sqft</property>
      <value>0.1</value>
    </product>
  </function>

  <function name="aero/force/Drag_induced">
    <description>Induced drag</description>
    <product>
      <property>aero/qbar-psf</property>
      <property>metrics/Sw-sqft</property>
      <property>aero/cl-squared</property>
      <value>0.0877</value>
    </product>
  </function>

  <function name="aero/coefficient/CDDe">
    <description>Drag_due_to_Elevator_Deflection</description>
    <product>
      <property>aero/qbar-psf</property>
      <property>metrics/Sw-sqft</property>
      <property>fcs/elevator-pos-rad</property>
      <value>0.0135</value>
    </product>
  </function>

  <function name="aero/coefficient/CDDa">
    <description>Drag_due_to_Aileron_Deflection</description>
    <product>
      <property>aero/qbar-psf</property>
      <property>metrics/Sw-sqft</property>
      <property>fcs/left-aileron-pos-rad</property>
      <value>0.0302</value>
    </product>
  </function>

  <function name="aero/coefficient/CDDa">
    <description>Drag_due_to_Rudder_Deflection</description>
    <product>
      <property>aero/qbar-psf</property>
      <property>metrics/Sw-sqft</property>
      <property>fcs/rudder-pos-rad</property>
      <value>0.0303</value>
    </product>
  </function>

```

```

</function>

</axis>

<axis name="SIDE">
  <function name="aero/coefficient/CYb">
    <description>Side_force_due_to_beta</description>
    <product>
      <property>aero/qbar-psf</property>
      <property>metrics/Sw-sqft</property>
      <property>aero/beta-rad</property>
      <value>-0.83</value>
    </product>
  </function>

  <function name="aero/coefficient/CYda">
    <description>Side_force_due_to_aileron</description>
    <product>
      <property>aero/qbar-psf</property>
      <property>metrics/Sw-sqft</property>
      <property>fcs/left-aileron-pos-rad</property>
      <value>-0.075</value>
    </product>
  </function>

  <function name="aero/coefficient/CYdr">
    <description>Side_force_due_to_rudder</description>
    <product>
      <property>aero/qbar-psf</property>
      <property>metrics/Sw-sqft</property>
      <property>fcs/rudder-pos-rad</property>
      <value>0.1914</value>
    </product>
  </function>

</axis>

<axis name="ROLL">
  <function name="aero/coefficient/Clb">
    <description>Roll_moment_due_to_beta</description>
    <product>
      <property>aero/qbar-psf</property>
      <property>metrics/Sw-sqft</property>
      <property>metrics/bw-ft</property>
      <property>aero/beta-rad</property>
      <value>-0.074</value>
    </product>
  </function>

  <function name="aero/coefficient/Clp">
    <description>Roll_moment_due_to_roll_rate_(roll_damping)</description>
    <product>
      <property>aero/qbar-psf</property>
      <property>metrics/Sw-sqft</property>
      <property>metrics/bw-ft</property>

```

```

        <property>aero/bi2vel</property>
        <property>velocities/p-aero-rad_sec</property>
        <value>-0.41</value>
    </product>
</function>

<function name="aero/coefficient/Clr">
    <description>Roll_moment_due_to_yaw_rate</description>
    <product>
        <property>aero/qbar-psf</property>
        <property>metrics/Sw-sqft</property>
        <property>metrics/bw-ft</property>
        <property>aero/bi2vel</property>
        <property>velocities/r-aero-rad_sec</property>
        <value>0.107</value>
    </product>
</function>

<function name="aero/coefficient/ClDa">
    <description>Roll_moment_due_to_aileron</description>
    <product>
        <property>aero/qbar-psf</property>
        <property>metrics/Sw-sqft</property>
        <property>metrics/bw-ft</property>
        <property>fcs/left-aileron-pos-rad</property>
        <value>0.2</value>
    </product>
</function>

<function name="aero/coefficient/ClDr">
    <description>Roll_moment_due_to_rudder</description>
    <product>
        <property>aero/qbar-psf</property>
        <property>metrics/Sw-sqft</property>
        <property>metrics/bw-ft</property>
        <property>fcs/rudder-pos-rad</property>
        <value>-0.107</value>
    </product>
</function>
</axis>

<axis name="PITCH">
    <function name="aero/coefficient/Cmo">
        <description>Pitching_moment_at_zero_alpha</description>
        <product>
            <property>aero/qbar-psf</property>
            <property>metrics/Sw-sqft</property>
            <property>metrics/cbarw-ft</property>
            <value>0.15</value>
        </product>
    </function>

    <function name="aero/coefficient/Cmalph">
        <description>Pitch_moment_due_to_alpha</description>
        <product>
            <property>aero/qbar-psf</property>
            <property>metrics/Sw-sqft</property>

```

```

        <property>metrics/cbarw-ft</property>
        <property>aero/alpha-rad</property>
        <value>-1.8</value>
    </product>
</function>

<function name="aero/coefficient/Cmde">
<description>Pitch_moment_due_to_elevator_deflection</description>
    <product>
        <property>aero/qbar-psf</property>
        <property>metrics/Sw-sqft</property>
        <property>metrics/cbarw-ft</property>
        <property>fcs/elevator-pos-rad</property>
        <value>-0.458</value>
    </product>
</function>

<function name="aero/coefficient/Cmq">
    <description>Pitch_moment_due_to_pitch_rate</description>
    <product>
        <property>aero/qbar-psf</property>
        <property>metrics/Sw-sqft</property>
        <property>metrics/cbarw-ft</property>
        <property>aero/ci2vel</property>
        <property>velocities/q-aero-rad_sec</property>
        <value>-6.813</value>
    </product>
</function>

<function name="aero/coefficient/Cmadot">
    <description>Pitch_moment_due_to_alpha_rate</description>
    <product>
        <property>aero/qbar-psf</property>
        <property>metrics/Sw-sqft</property>
        <property>metrics/cbarw-ft</property>
        <property>aero/ci2vel</property>
        <property>aero/alphadot-rad_sec</property>
        <value>-3.5</value>
    </product>
</function>

</axis>

<axis name="YAW">
    <function name="aero/coefficient/Cnb">
        <description>Yaw_moment_due_to_beta</description>
        <product>
            <property>aero/qbar-psf</property>
            <property>metrics/Sw-sqft</property>
            <property>metrics/bw-ft</property>
            <property>aero/beta-rad</property>
            <value>0.071</value>
        </product>
    </function>

<function name="aero/coefficient/Cnr">

```

```

    <description>Yaw_moment_due_to_yaw_rate</description>
    <product>
      <property>aero/qbar-psf</property>
      <property>metrics/Sw-sqft</property>
      <property>metrics/bw-ft</property>
      <property>aero/bi2vel</property>
      <property>velocities/r-aero-rad_sec</property>
      <value>-0.12032</value>
    </product>
  </function>

  <function name="aero/coefficient/Cndr">
    <description>Yaw_moment_due_to_rudder</description>
    <product>
      <property>aero/qbar-psf</property>
      <property>metrics/Sw-sqft</property>
      <property>metrics/bw-ft</property>
      <property>fcs/rudder-pos-rad</property>
      <value>-0.062</value>
    </product>
  </function>

  <function name="aero/coefficient/Cnda">
    <description>Yaw_moment_due_to_aileron. Adverse_Yaw</description>
    <product>
      <property>aero/qbar-psf</property>
      <property>metrics/Sw-sqft</property>
      <property>metrics/bw-ft</property>
      <property>fcs/left-aileron-pos-rad</property>
      <value>0.0108</value>
    </product>
  </function>

  <function name="aero/coefficient/Cnp">
    <description>Yaw_moment_due_to_roll_rate</description>
    <product>
      <property>aero/qbar-psf</property>
      <property>metrics/Sw-sqft</property>
      <property>metrics/bw-ft</property>
      <property>aero/bi2vel</property>
      <property>velocities/p-aero-rad_sec</property>
      <value>-0.0575</value>
    </product>
  </function>

</axis>
</aerodynamics>

<output name ="localhost" type="FLIGHTGEAR" port="5500" protocol="UDP"
rate="10"> </output>

<output name="GBS_40Out.csv" type="CSV" rate="100">

  <simulation> OFF </simulation>
  <atmosphere> ON </atmosphere>
  <massprops> OFF </massprops>
  <aerosurfaces> ON </aerosurfaces>

```

```

    <rates> ON </rates>
    <velocities> ON </velocities>
    <forces> OFF </forces>
    <moments> OFF </moments>
    <position> ON </position>
    <coefficients> OFF </coefficients>
    <ground_reactions> OFF </ground_reactions>
    <fcs> ON </fcs>
    <propulsion> ON </propulsion>
      <property> fcs/throttle-cmd-norm </property>
      <property> ap/aileron_cmd </property>
      <property> fcs/wing-leveler-ap-on-off </property>
      <property> fcs/roll-ap-error-pid </property>
      <property> fcs/roll-ap-autoswitch </property>
      <property> fcs/roll-command-selector </property>
      <property> position/lat-geod-deg </property>
  </output>
</fdm_config>

```

## D.3. RPA Engine Files

### D.3.1. EPP FPV Engine File

```

<?xml version="1.0"?>
<electric_engine name="engEPPFPV">
  <power unit="WATTS"> 370.0 </power>
</electric_engine>

```

### D.3.2. GiantBig Stik Engine File

```

<?xml version="1.0"?>
<piston_engine name="Zenoah 26A">
  <minmp unit="INHG"> 6.0 </minmp>
  <maxmp unit="INHG"> 28.5 </maxmp>
  <displacement unit="IN3"> 1.55 </displacement>
  <maxhp> 2.96 </maxhp>
  <cycles> 4.0 </cycles>
  <idlerpm> 700.0 </idlerpm>
  <maxrpm> 2800.0 </maxrpm>
  <maxthrottle> 1.0 </maxthrottle>
  <minthrottle> 0.1 </minthrottle>

```

```
</piston_engine>
```

## D.4. RPA Propeller Files

### D.4.1. EPP FPV Propeller File

```
<?xml version="1.0"?>

<propeller name="propEPPFPV">
  <ixx> 0.001 </ixx>
  <diameter unit="IN"> 10.0 </diameter>
  <numblades> 2 </numblades>
  <gearratio> 0.93 </gearratio>
  <p_factor> 0.79 </p_factor>

  <table name="C_THRUST" type="internal">
    <tableData>
      0.0 0.0123
      0.1 0.0118
      0.2 0.0112
      0.3 0.0103
      0.4 0.0093
      0.5 0.0082
      0.6 0.0066
      0.7 0.0050
      0.8 0.0027
      1.0 -0.0009
      1.2 -0.0048
      1.4 -0.0087
    </tableData>
  </table>

  <table name="C_POWER" type="internal">
    <tableData>
      0.0 0.0082
      0.1 0.0082
      0.2 0.0080
      0.3 0.0078
      0.4 0.0074
      0.5 0.0068
      0.6 0.0062
      0.7 0.0052
      0.8 0.0043
      1.0 0.0015
      1.2 -0.0024
      1.4 -0.0073
      1.6 -0.0124
    </tableData>
  </table>
```

```

<table name="CT_MACH" type="internal">
  <tableData>
    0.85  1.0
    1.05  0.8
  </tableData>2.4
</table>

<table name="CP_MACH" type="internal">
  <tableData>
    0.85  1.0
    1.05  1.8
    2.00  1.4
  </tableData>
</table>

</propeller>

```

#### ***D.4.2. GiantBig Stik Propeller File***

```

<?xml version="1.0"?>

<propeller name="Fixed-Pitch 16-inch Two-Blade Propeller">
  <ixx> 0.001 </ixx>
  <diameter unit="IN"> 16.0 </diameter>
  <numblades> 2 </numblades>
  <gearratio> 0.62 </gearratio>
  <p_factor> 1.26 </p_factor>

  <table name="C_THRUST" type="internal">
    <tableData>
      0.0  0.0156
      0.1  0.0149
      0.2  0.0143
      0.3  0.0132
      0.4  0.0118
      0.5  0.0104
      0.6  0.0084
      0.7  0.0064
      0.8  0.0035
      1.0 -0.0012
      1.2 -0.0061
      1.4 -0.0110
    </tableData>
  </table>

  <table name="C_POWER" type="internal">
    <tableData>
      0.0  0.0105
      0.1  0.0105
      0.2  0.0102
      0.3  0.0100
      0.4  0.0094
      0.5  0.0086
    </tableData>
  </table>

```

```

    0.6    0.0079
    0.7    0.0066
    0.8    0.0054
    1.0    0.0019
    1.2   -0.0031
    1.4   -0.0093
    1.6   -0.0158
  </tableData>
</table>

<table name="CT_MACH" type="internal">
  <tableData>
    0.85   1.0
    1.05   0.8
  </tableData>2.4
</table>

<table name="CP_MACH" type="internal">
  <tableData>
    0.85   1.0
    1.05   1.8
    2.00   1.4
  </tableData>
</table>

</propeller>

```