ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL

DEVELOPMENT OF CERTIFICATION-COMPLIANT SAFETY-CRITICAL FLIGHT CONTROL SOFTWARE USING A MODEL-BASED DESIGN APPROACH

M.Sc. THESIS

Hakan BALLI

Department of Control and Automation Engineering

Control and Automation Engineering Programme

JULY 2024

ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL

DEVELOPMENT OF CERTIFICATION-COMPLIANT SAFETY-CRITICAL FLIGHT CONTROL SOFTWARE USING A MODEL-BASED DESIGN APPROACH

M.Sc. THESIS

Hakan BALLI
(504201136)

Department of Control and Automation Engineering

Control and Automation Engineering Programme

Thesis Advisor: Prof. Dr. Yaprak YALÇIN

JULY 2024

ISTANBUL TEKNİK ÜNİVERSİTESİ ★ LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

MODEL TABANLI TASARIM YAKLAŞIMI KULLANILARAK SERTİFİKASYON UYUMLULUĞUNA SAHİP EMNİYET KRİTİK UÇUŞ KONTROL YAZILIMININ GELİŞTİRİLMESİ

YÜKSEK LİSANS TEZİ

Hakan BALLI
(504201136)

Kontrol ve Otomasyon Mühendisliği Anabilim Dalı

Kontrol ve Otomasyon Mühendisliği Programı

Tez Danışmanı: Prof. Dr. Yaprak YALÇIN

TEMMUZ 2024

Hakan BALLI, a M.Sc. student of ITU Graduate School student ID 504201136, successfully defended the thesis entitled “DEVELOPMENT OF CERTIFICATION-COMPLIANT SAFETY-CRITICAL FLIGHT CONTROL SOFTWARE USING A MODEL-BASED DESIGN APPROACH”, which he prepared after fulfilling the requirements specified in the associated legislations, before the jury whose signatures are below.

Thesis Advisor :	**Prof. Dr. Yaprak YALÇIN** Istanbul Technical University	.....
Jury Members :	**Prof. Dr. Afife Leyla GÖREN** Istanbul Technical University	.....
	Asst. Prof. Mümin Tolga EMİRLER Yıldız Technical University	.....

Date of Submission : 24 May 2024
Date of Defense : 17 July 2024

To my family and love,

FOREWORD

First and foremost, I am grateful to my beloved family for their love, encouragement, patience, and unwavering support throughout my life.

I wish to extend my sincere thanks to my colleagues from KAAN / TF-X National Fighter Aircraft Development Program for their friendship, which made the work hours enjoyable. The long working hours with extra shifts, the survival attempts inside the company campus, the regular coffee breaks, and the lunch gym sessions will remain unique and unforgettable experiences in my life.

I would like to extend special thanks to Assoc. Prof. Dr. Tayfun Çimen, my lead engineer, for his invaluable contributions to flight dynamics, modeling, and simulation. His guidance has significantly expanded my knowledge, perspective, and understanding in these areas.

I am grateful to Dr. Murat Millidere for leading me into modeling and simulation area with sharing his knowledge and motivating me his support, technical assistance and suggestions.

I would like to extend my appreciation to Dr. Onur Albostan for sharing his expertise on flight control law development and HÜRJET Flight Control Laws team for their assistance.

I want to thank my thesis advisor Assoc. Prof. Yaprak Yalçın, for her guidance and patience throughout this thesis study.

Last but not least, I want to thank my girlfriend, Nilsu Gizem Gülcan, for supporting me throughout my entire master's degree, both while I spent countless hours in front of my computer and in daily life as my better half.

May 2024 Hakan BALLI

(Flight Dynamics, Simulation and Control Engineer)

TABLE OF CONTENTS

Page

FOREWORD ..... **ix**
TABLE OF CONTENTS ..... **xi**
ABBREVIATIONS ..... **xv**
SYMBOLS ..... **xvii**
LIST OF TABLES ..... **xix**
LIST OF FIGURES ..... **xxi**
SUMMARY ..... **xxiii**
ÖZET ..... **xxv**
1. INTRODUCTION **1**
1.1 Motivation and Purpose of Thesis 1
1.2 Literature Review 2
1.2.1 Safety-critical software development and DO-178C 2
1.2.2 Model based design 4
1.2.3 Air vehicle flight dynamics and flight control system design 5
1.3 Thesis Outline 6
2. SAFETY-CRITICAL SOFTWARE DEVELOPMENT PROCESS **7**
2.1 Airworthiness Certification 7
2.1.1 Flight safety and airworthiness 7
2.1.2 The ICAO and the civil aviation authorities 8
2.1.2.1 The ICAO (international civil aviation organization) 9
2.1.2.2 The civil aviation authorities 9
2.1.3 Airworthiness requirements 11
2.1.4 Type certification 12
2.1.5 Certificates of airworthiness 12
2.1.6 Continued airworthiness 13
2.1.7 Airworthiness and software certification relation 14
2.2 Safety-Critical Software Development 14
2.2.1 Software within the system framework 15
2.2.1.1 System development overview 15
2.2.1.2 System requirements 17
2.2.2 Software within the system safety assessment framework 18
2.2.2.1 Aircraft and system safety assessment process overview 18
2.2.2.2 Development assurance 20
2.2.3 Safety process and software relevance 22
2.3 DO-178C and Supporting Documents 24
2.3.1 DO-178C core document 25
2.3.2 DO-178C technology supplements 27
2.3.2.1 DO-331: model-based development supplement 28
2.3.2.2 DO-332: object-oriented technology supplement 28
2.3.2.3 DO-333: formal methods supplement 29
2.3.3 DO-330: software tool qualification considerations 29

2.4 DO-178C Software Development Lifecycle 30
2.4.1 Software planning process 30
2.4.1.1 Plan for software aspects of certification 31
2.4.1.2 Software development plan 31
2.4.1.3 Software verification plan 31
2.4.1.4 Software configuration management plan 32
2.4.1.5 Software quality assurance plan 32
2.4.1.6 Development standards identified by DO-178C 32
2.4.2 Software development process 33
2.4.2.1 Software requirements process 34
2.4.2.2 Software design process 35
2.4.2.3 Software coding and integration process 35
2.4.3 Software verification process 36
2.4.4 Software configuration management process 36
2.4.5 Software quality assurance process 39
2.4.6 Certification liaison process 39
3. MODEL BASED DESIGN 41
3.1 Model Based Design Overview 41
3.1.1 History of MBD 42
3.1.2 Advantages of MBD 43
3.1.3 Challenges of MBD 44
3.1.4 Model-based systems engineering 44
3.2 Model-Based Development Process 45
3.2.1 Essential elements of model-based design 46
3.2.1.1 Product concept 46
3.2.1.2 System requirements 46
3.2.1.3 System architecture 47
3.2.1.4 System design 47
3.2.1.5 Model simulation and implementation 48
3.2.1.6 Auto code generation and automation 48
3.2.1.7 Verification & validation 49
3.2.1.8 Knowledge capture and management 51
3.3 DO-331 Model Based Development and Verification Supplement 52
4. AIR VEHICLE MATHEMATICAL SIMULATION MODEL 55
4.1 Air Vehicle Flight Dynamics 56
4.1.1 Assumptions, notations, and conventions 56
4.1.2 Reference frames and coordinate systems 58
4.1.3 Kinematic relations 58
4.1.3.1 Coordinate transformations 59
4.1.3.2 Translational rates 61
4.1.3.3 Angular velocities and angular rates 61
4.1.3.4 Incidence angles 62
4.1.3.5 Flight path angles 63
4.1.4 Dynamic relations 63
4.1.4.1 Translational motion 63
4.1.4.2 Rotational motion 64
4.1.5 Forces and moments 66
4.1.5.1 Aerodynamic forces and moments 67
4.1.5.2 Propulsive forces 69
4.1.5.3 Gravitational force 69

4.1.6 Auxiliary equations 70
4.1.6.1 Accelerations 70
4.1.6.2 Airspeed dynamics 70
4.1.6.3 Incidence angle dynamics 71
4.1.6.4 Bank angle dynamics 72
4.1.6.5 Flight path angle dynamics 73
4.1.6.6 Altitude dynamics 73
4.2 Development of Air Vehicle Simulation Model 74
4.2.1 Plant model 75
4.2.1.1 Aerodynamic model 75
4.2.1.2 Propulsion model 78
4.2.1.3 Flight environment (atmosphere) model 80
4.2.2 Effector models 82
4.2.3 Sensor models 82
4.3 Development of Trim and Linear Aircraft Models 83
4.3.1 Representation of dynamic systems 83
4.3.2 Trim algorithm and linearization 86
4.3.3 Steady-state flight conditions 86
4.3.3.1 Straight and level flight 86
4.3.3.2 Steady climb or descent 87
4.3.3.3 Steady turn (coordinated turn) 88
5. FLIGHT CONTROL DESIGN AND ARCHITECTURE 89
5.1 Flight Control Algorithm Design Requirements 91
5.2 Stability and Control Augmentation 93
5.2.1 Longitudinal stability augmentation 94
5.2.2 Longitudinal control augmentation 95
5.2.3 Lateral-directional stability augmentation 95
5.2.4 Lateral-directional control augmentation 96
5.3 Nonlinear Dynamic Inversion 97
5.3.1 Nonlinear feedback linearization overview 99
5.3.2 Dynamic inverse controller design 100
5.3.3 Two-time scale method 102
5.3.4 Desired dynamics 103
5.3.5 Stabilization controller design 105
5.3.5.1 Inner loop design 107
5.3.5.2 Outer loop design 110
6. FLIGHT CONTROL SOFTWARE DESIGN USING DO-178C 113
6.1 FLCS System and Algorithm Development Overview 114
6.2 Software Planning Overview 114
6.3 Software Development Overview 116
6.3.1 Software development process 117
6.3.2 System requirements allocation to software 119
6.3.3 Software requirement process 120
6.3.3.1 High level software requirements (HLRs) development 120
6.3.4 Software design process 121
6.3.4.1 Low level software requirements (LLRs) development 121
6.3.5 Software coding process 122
6.3.6 Integration process 123
6.3.7 Software development process traceability 123
6.3.7.1 System to software HLR 123

6.3.7.2 Software HLR to LLR 123
6.3.7.3 Software Derived LLR 124
6.3.7.4 Software LLR to Source Code 124
6.4 Software Verification Overview 124
6.4.1 Reviews and analyses of software HLR 125
6.4.2 Reviews and analyses of LLR and software architecture 125
6.4.3 Reviews and analyses of source code 129
6.4.4 Software testing 131
6.4.5 Test coverage analysis 132
7. CONCLUSIONS AND RECOMMENDATIONS 135
REFERENCES 137
APPENDICES 143
CURRICULUM VITAE 145

ABBREVIATIONS

AoA : Angle of Attack
ARP : Aerospace Recommended Practices from SAE
CCA : Common Cause Analysis
CM : Center of Mass
CofA : Certificate of Airworthiness
DAL : Design Assurance Level
DI : Dynamic Inversion
DO : Document of RTCA
DoF : Degree of Freedom
EASA : European Union Aviation Safety Agency
EoM : Equations of Motion
EUROCAE : European Organization for Civil Aviation Equipment
FAA : US Federal Aviation Administration
FBW : Fly-by-Wire
FCC : Flight Control Computer
FCL : Flight Control Law
FDAL : Functional Development Assurance Level
FHA : Functional Hazard Assessment
FLCS : Flight Control System
HIL : Hardware-in-the-loop
HLR : Software high-level requirement according to glossary of DO-178C
HW : Hardware
IDAL : Item Development Assurance Level
LEF : Leading Edge Flap
LLR : Software low-level requirement according to glossary of DO-178C
LTI : Linear Time Invariant
MAAB : MathWorks Automotive Advisory Board
MBD : Model Based Design
MBSE : Model Based Systems Engineering
MCDC : Modified Condition/Decision Coverage

MISRA **:** Motor Industry Software Reliability Association

NFL **:** Nonlinear Feedback Linearization

NDI **:** Nonlinear Dynamic Inversion

NR **:** Newton-Raphson

OOT **:** Object-oriented Technology

PASA **:** Preliminary Aircraft Safety Assessment

PIL **:** Processor-in-the-loop

PID **:** Proportional-integral-derivative

PSAC **:** Plan for Software Aspects for Certification

PSSA **:** Preliminary System Safety Assessment

RTCA **:** Radio Technical Commission for Aeronautics

SAE **:** Society of Automotive Engineers

SCI **:** Software Configuration Item

SCM **:** Software Configuration Management

SCMP **:** Software Configuration Management Plan

SCS **:** Software Coding Standards

SDP **:** Software Development Plan

SDS **:** Software Design Standards

SOI **:** Stage of Involvement

SQAP **:** Software Quality Assurance Plan

SSA **:** System Safety Assessment

SIL **:** Software-in-the-loop

SW **:** Software

TC **:** Type Certificate

TQL **:** Tool Qualification Level

V&V **:** Verification and Validation

w.r.t. **:** With Respect To

SYMBOLS

$\boldsymbol{D^{(\cdot)}}$	**:** Rotational Tensor Derivative Operator w.r.t. Frame $\mathcal{F}_{(\cdot)}$; $\mathcal{D}^{(\cdot)}$
$\boldsymbol{B}$	**:** Origin of Vehicle Body-Fixed Frame $\mathcal{F}_B$, w.r.t. CM of the Vehicle
$\boldsymbol{E}$	**:** Origin of Earth frame $\mathcal{F}_E$, w.r.t. CM of the vehicle
$\boldsymbol{N}$	**:** Tangent Point on Earth's Surface, w.r.t Local Navigation Frame $\mathcal{F}_N$
$\boldsymbol{W}$	**:** Origin of Wind Frame $\mathcal{F}_W$, w.r.t. CM of the Vehicle
$\boldsymbol{\mathcal{F}_B}$	**:** Vehicle Body-Fixed Frame Having Origin B
$\boldsymbol{\mathcal{F}_N}$	**:** Local Geodetic Frame Having Origin N
$\boldsymbol{\mathcal{F}_W}$	**:** Wind Frame Having Origin W
$\boldsymbol{L, D, T, C}$	**:** Lift, Drag, Thrust, and Side Forces
$\boldsymbol{f_x, f_y, f_z}$	**:** Body Axis Forces
$\boldsymbol{l, m, n}$	**:** Aircraft Body Axis (roll, pitch, and yaw) Moments
$\boldsymbol{C_L, C_D}$	**:** Lift and Drag Coefficients in Wind Axis
$\boldsymbol{C_T, C_C}$	**:** Thrust and Side Force Coefficients in Wind Axis
$\boldsymbol{C_X, C_Y, C_Z}$	**:** Body Axis Force Coefficients
$\boldsymbol{C_l, C_m, C_n}$	**:** Body Axis (roll, pitch, and yaw) Moment Coefficients
$\boldsymbol{\bar{q}}$	**:** Dynamic Pressure
$\boldsymbol{p_s}$	**:** Static Pressure
$\boldsymbol{m}$	**:** Total Mass of the Air Vehicle
$\boldsymbol{c}$	**:** Wing Mean Aerodynamic Chord
$\boldsymbol{b}$	**:** Wing Span
$\boldsymbol{S}$	**:** Wing Area
$\boldsymbol{x_{cg_{ref}}}$	**:** Reference Center-of-Gravity Location for Aerodynamic Data
$\boldsymbol{I_X, I_Y, I_Z}$	**:** Moments of Inertia about X, Y, and Z Body Axes
$\boldsymbol{I_{XZ}}$	**:** Product of Inertia with respect to X and Z body Axes
$\boldsymbol{u, v, w}$	**:** Components of $\vec{v}_{B/N}^{N}$ Resolved in Body Coordinates
$\boldsymbol{u_w, v_w, w_w}$	**:** Components of Wind Velocity Resolved in Body Coordinates
$\boldsymbol{p, q, r}$	**:** Components of $\vec{\omega}^{B/N}$ Resolved in Body Coordinates
$\boldsymbol{\phi, \theta, \psi}$	**:** Euler roll, pitch, and yaw angles
$\boldsymbol{p_N, p_E, p_D}$	**:** X-axis, Y-axis, and Z-axis North-East-Down Positions

V_T : True Airspeed

M : Mach number

n_z, n_y : Normal and Lateral Acceleration

g : Acceleration due to gravity

α : Angle of Attack

β : Angle of Sideslip

γ : Vertical Flight Path Angle

χ : Heading Angle

μ : Bank Angle

$h, -p_D$: Altitude

$\delta_e, \delta_a, \delta_r$: Control Surface Deflections (elevator, aileron, and rudder)

δ_{th} : Throttle Deflection

τ_T : Engine Thrust Time Constant

$s_{(\cdot)}$: Shorthand notation for $\sin(\cdot)$

$c_{(\cdot)}$: Shorthand notation for $\cos(\cdot)$

$t_{(\cdot)}$: Shorthand notation for $\tan(\cdot)$

ω : Angular Velocity

LIST OF TABLES

Page

Table 2.1 : Consideration examples for system requirements. 17
Table 2.2 : Failure condition probabilities, and levels. 20
Table 2.3 : Summary of DO-178C objectives [1] [37]. 23
Table 2.4 : Summary of DO-178C Annex-A tables [1]. 27
Table 2.5 : Software level objectives in DO-178C. 27
Table 2.6 : Configuration status report example. 38
Table 4.1 : Kinematic relationships of transformations. 59
Table 4.2 : Air vehicle mass and dimensional parameters. 75
Table 4.3 : Aerodynamic data breakpoints 77
Table 4.4 : Tabular form of maximum thrust values [3]. 79
Table 4.5 : ISA standard conditions. 80
Table 4.6 : Effector actual position and rate limits [53]. 82
Table 5.1 : Functions of a flight control system. 89
Table 5.2 : Aircraft classes. 91
Table 5.3 : Flight phase categories. 92
Table 5.4 : Flying quality levels. 92
Table 6.1 : Model-based usage examples table from DO-331 [2]. 118
Table 6.2 : Simulation and test case aims. 126

LIST OF FIGURES

Page

Figure 2.1 : Overall system development process in ARP4754A [36]. 16
Figure 2.2 : System and safety assessment procedures overview [37]. 19
Figure 2.3 : The V-model for system development [37]..... 21
Figure 2.4 : Safety and software life cycle processes relevance. 22
Figure 2.5 : Relationship of DO-178C and its supporting documents [37]. 25
Figure 2.6 : Overview of DO-178C [1] [37]. 26
Figure 2.7 : DO-178C software development lifecycle process [42]. 30
Figure 2.8 : Requirements development activities [37]. 34
Figure 3.1 : Model-based design workflow [44]. 41
Figure 3.2 : V-model. 46
Figure 3.3 : Systematic use of models throughout the MBD process [44]. 47
Figure 4.1 : Air vehicle simulation model components [48]. 55
Figure 4.2 : Air vehicle frames and coordinate systems [50]. 57
Figure 4.3 : Orientation angles [23]. 58
Figure 4.4 : Pitch moment coefficient graph [54]. 78
Figure 4.5 : Simplified engine model [50]. 78
Figure 4.6 : Linearization procedure [48]. 85
Figure 4.7 : Straight and level flight [48]. 87
Figure 4.8 : Steady climb and descent, left and right, respectively [48]. 87
Figure 4.9 : Steady level turning flight [48]..... 88
Figure 5.1 : Fly-by-wire flight control SCAS systems [63]. 93
Figure 5.2 : Longitudinal-axis stability augmentation [59]. 94
Figure 5.3 : Longitudinal-axis control augmentation [59]. 95
Figure 5.4 : Lateral-directional stability augmentation [59]. 96
Figure 5.5 : Lateral-directional control augmentation [59]..... 97
Figure 5.6 : Dynamic inversion process [26]. 101
Figure 5.7 : Flowchart for the desired dynamics [25]. 104
Figure 5.8 : Closed-loop transfer function calculation [26]. 104
Figure 5.9: Stabilization controller [26]. 107

Figure 5.10 : Inner loop: slow dynamics [63]. 110
Figure 5.11 : Fast dynamics scheme [54]. 111
Figure 5.12 : NDI control block diagram [54]. 111
Figure 6.1 : MATLAB project structure with Git source control. 113
Figure 6.2 : System and item design levels. 114
Figure 6.3 : FCL software development overview. 114
Figure 6.4 : MATLAB Project software planning structure. 115
Figure 6.5 : FLCS software development overview. 116
Figure 6.6 : The V-model V&V process. 117
Figure 6.7 : Air vehicle simulation model in Simulink. 118
Figure 6.8 : System requirements allocated to software using Simulink. 119
Figure 6.9 : Software HLRs implementation using Simulink. 121
Figure 6.10 : Requirements implementation to Simulink model. 122
Figure 6.11 : Test procedure and simulation case example. 126
Figure 6.12 : Simulink model advisor report for conformance. 126
Figure 6.13 : Traceability links for system requirements in Simulink. 127
Figure 6.14 : Test cases and test procedures for IMU voting. 128
Figure 6.15 : HLRs linkage to test and simulation cases example. 129
Figure 6.16 : HLRs verification by test and simulation cases. 129
Figure 6.17 : Code generation for FCC model. 130

DEVELOPMENT OF CERTIFICATION-COMPLIANT SAFETY-CRITICAL FLIGHT CONTROL SOFTWARE USING A MODEL-BASED DESIGN APPROACH

SUMMARY

With the rapid pace of technological advancements, software is becoming increasingly complex and is being used in safety-critical areas such as aircraft. This situation further emphasizes the importance of software safety levels. The development of safety-critical software not only increases the complexity of implementation due to the critical nature of the operational environments of aircraft but also faces increasing challenges due to customer expectations for shorter product development cycles and demands for lower production costs. This situation necessitates that software developers meet high safety standards while also providing solutions optimized for efficiency and cost-effectiveness.

In this context, avionics and flight control software in aircraft must fully perform the functions expected of them in the operational environment and must not adversely affect other functions while executing their tasks. These two factors clearly highlight the need for engineering methods aimed at reducing development complexities and supporting certification efforts.

International aviation certification authorities have defined these two fundamental rules in their aviation regulations and have published the DO-178C software guidance document to ensure compliance of aviation software with these rules. Additionally, by referencing the DO-331 supplement, which explains the model-based design method used in software development processes, they aim to ensure the safe development of safety-critical software.

The advancement of software, directives from authorities, and the competitive environment among manufacturers have increased the variety of products and services in the aviation sector. This situation has enabled software development processes under the DO-178C/DO-331 guidance documents using the model-based design method, allowing for software development with superior aspects such as shorter product development cycles and cost advantages.

In this study, the processes followed throughout the software development lifecycle, the activities that need to be completed, and the outcome that need to be created and delivered during the development of a flight control software using the model-based design method are examined in detail. Additionally, the certification process for flight control software is thoroughly analyzed.

In this study, MathWorks' MATLAB/Simulink model-based design tool is used in a software development project for the F-16 fighter aircraft. The aircraft and its components together with the flight control system were modeled rigorously within the Simulink tool. The flight control algorithm is constructed using the nonlinear dynamic inversion method, and source code for the flight control software is generated

from the flight control algorithm model created using the model-based design approach. Additionally, this approach is applied throughout the software development process using the DO-331 guidance document.

During the flight control software development process, system requirements allocated to the software (SRAT) were defined corresponding to the software lifecycle procedures, and from these, high-level requirements (HLR) were derived. Bidirectional traceability links between SRATS and HLR were established using the MATLAB development tool, adhering to software lifecycle procedures. After the HLR were established, the flight control system model, which represents the low-level requirements, is developed using the model-based design approach. Prior to generating source code, requirement-based functional tests were performed on the model, considering the DO-331 model simulation activity, to detect any errors. Additionally, model coverage analyses were conducted alongside these tests. Performance metrics were obtained based on the results of both the requirement-based tests and the model coverage analyses. Similarly, a static standards compliance test conducted on the model is evaluated as a performance metric, leading to the source code generation phase. It is generated using the MATLAB development tool, reviewed using MATLAB's tools, and performance metrics were gathered. Subsequently, the executable object code is created, and the performance metrics were evaluated through in-loop software simulation testing.

Finally, throughout the software development process, relevant process reports were generated at each step from requirements to executable object code, configuration tracking is conducted, and the software quality assurance and certification processes were prepared for submission to the authority. As a result, a flight control software compliant with the certification requirements under the DO-178C/DO-331 guidance documents were successfully developed.

MODEL TABANLI TASARIM YAKLAŞIMI KULLANILARAK SERTİFİKASYON UYUMLULUĞUNA SAHİP EMNİYET KRİTİK UÇUŞ KONTROL YAZILIMI GELİŞTİRİLMESİ

ÖZET

Teknolojik ilerlemelerin hızla devam etmesiyle birlikte, yazılımlar giderek daha karmaşık hale gelmekte ve özellikle hava araçları gibi emniyet açısından kritik alanlarda kullanılmaktadır. Bu durum, yazılımların emniyet seviyesinin önemini daha da artırmaktadır. Emniyet-kritik yazılımların geliştirilmesi, bir yandan hava araçlarının operasyonel ortamlarının güvenlik açısından kritik doğası nedeniyle uygulama karmaşıklığını artırırken, diğer yandan müşterilerin daha kısa ürün geliştirme döngüsü beklentisi ve daha düşük maliyetlerle üretim talepleri doğrultusunda giderek artan zorluklarla karşı karşıya kalmaktadır. Bu durum, yazılım geliştiricilerin hem yüksek güvenlik standartlarını karşılamasını hem de verimlilik ve maliyet etkinliği açısından optimize edilmiş çözümler sunmasını zorunlu kılmaktadır.

Bu bağlamda, hava araçlarında aviyonik ve uçuş kontrol yazılımlarının, operasyonel ortamda kendisinden beklenen fonksiyonları eksiksiz yerine getirmesi ve görevlerini icra ederken diğer fonksiyonlara olumsuz etki etmemesi gerekmektedir. Bu iki faktör, geliştirme karmaşıklıklarını azaltmaya ve sertifikasyon çabalarını desteklemeye yönelik mühendislik yöntemlerine olan ihtiyacı açıkça ortaya koymaktadır.

Uluslararası havacılık sertifikasyon otoriteleri, bu iki temel kuralı kendi havacılık düzenlemelerinde tanımlamış ve havacılık yazılımlarının bu kurallara uygunluğunu denetlemek amacıyla DO-178C yazılım rehber dokümanını yayınlamışlardır. Ayrıca, yazılım geliştirme süreçlerinde kullanılan model tabanlı tasarım yöntemini açıklayan DO-331 ekini de referans alarak, emniyet kritik yazılımların güvenli bir şekilde geliştirilmesini sağlamayı hedeflemişlerdir.

Gelişen yazılımlar, otoritelerin direktifleri ve üreticiler arası rekabet ortamı, havacılık sektöründeki ürün ve hizmet alternatiflerin artmasını sağlamıştır. Bu durum, DO-178C/DO-331 rehber dokümanları kapsamında model tabanlı tasarım yöntemi kullanılarak yazılım geliştirme süreçlerinin, daha kısa ürün geliştirme döngüsü ve maliyet avantajı gibi üstün yönleri ile yazlım geliştirilmesine olanak sağlamıştır.

Bu çalışmada, model tabanlı tasarım yöntemi kullanılarak bir uçuş kontrol yazılımının geliştirilme sürecinde yazılım geliştirme yaşam döngüsü boyunca takip edilen süreçler, tamamlanması gereken faaliyetler ve üretilmesi gereken veriler ile uçuş kontrol yazılımı sertifikasyon süreci detaylı bir şekilde incelenmiştir.

Öncelikle, havacılıkta emniyet ve uçuşa elverişlilik tanımları yapılıp sertifikasyon konuları ile havacılık otoriteleri ele alınmış, bu otoritelerin regülasyonları ve bu düzenlemelere uyumluluk konularına değinilmiştir. Uçuşa elverişlilikten yazılıma kadar olan ilişki incelendikten sonra da emniyet kritik yazılımın detaylandırılmasına geçilmiştir.

Emniyet kritik yazılım, ilgili hava aracı sistemi ile ilişkili olup belirli rehber dokümanlar referansında öncelikle sistemin, ardından da ilgili sistem yazılımının emniyet seviyeleri belirli emniyet değerlendirmeleri sonucu belirlenmektedir. Eğer yazılım uçuş kontrol sistemi gibi kritik seviye ise rehber dokümanlar uyum gösterimi için DO-178C yazılım dokümanını gerekli kılmaktadır. Böylece, uçuş kontrol sistemi yazlımı geliştirilmesinde emniyet kritik kavramları ile DO-178C rehber dokümanının takip edilmesi ve ilgili yazlım geliştirme süreçleri ayrıntılı bir şekilde ele alınmıştır.

Son yıllarda, hava aracı sistemlerinin efektif ve sistematik bir şekilde geliştirilmesi ve test edilmesi aşamalarında model tabanlı tasarım yöntemi popüler bir yöntem olarak kullanılmaktadır. Bu yöntem ile birlikte hava aracı yazılımlarında DO-178C yardımcı dokümanı olan DO-331 ek dokümanı model tabanlı tasarım süreçlerinin ilerletilmesinde rehber doküman olarak kullanılmakta olup ayrıntılı şekilde ele alınmıştır.

Çalışmada, hava aracı olarak F-16 savaş uçağı modeli kullanılmış olup, bu model için oluşturulan bir yazılım geliştirme projesinde MathWorks şirketinin Matlab/Simulink model tabanlı tasarım aracı kullanılmıştır. Hava aracı ve bileşenleri, ayrıca uçuş kontrol sistemi Simulink ortamında detaylı bir şekilde modellenmiştir. Uçuş kontrol algoritması olarak hem kararlılık arttırma sistemi hem de komut artırma sistemi ile yüksek hücum açısı kontrolü için doğrusal olmayan dinamik tersleme yöntemi algoritmaları tasarlanmış, uçuş kontrol yazılımı için ise model tabanlı tasarım ile oluşturulan uçuş kontrol algoritması modelinden kaynak kodu üretimi gerçekleştirilmiştir Ayrıca, yazılım geliştirme süreci boyunca DO-331 rehber dokümanı kullanılarak model tabanlı tasarım yöntemi uygulanmıştır.

Uçuş kontrol yazılımı geliştirme süreci yazılım planlama, yazılım geliştirme ve yazlım geçerlemesi olarak üç ana başlıkta ele alınmıştır. İlk aşamada belirtilen planlara değinilmiş, süreç boyunca oluşturulacak planlar şekillendirilmiştir. Ardından, yazılım geliştirme süreci boyunca yazılım yaşam döngüsü prosedürlerine uygun olarak yazılıma atanan sistem gereksinimleri (YASG) belirlenmiş ve bu gereksinimlerden üst seviye gereksinimler (ÜSG) oluşturulmuştur. Yazılım yaşam döngüsü prosedürlerine göre, YASG ile ÜSG arasında çift yönlü izlenebilirlik bağlantıları, Matlab geliştirme aracı kullanılarak sağlanmıştır. ÜSG oluşturulduktan sonra, model tabanlı tasarım yöntemi kullanılarak alt seviye gereksinimleri içerik olarak barındıran tasarım modeli olan uçuş kontrol sistem modeli geliştirilmiştir. Burada, önceki paragrafta belirtilen uçuş kontrol algoritması Simulink modeli kullanılarak tün üst seviye gereksinimlerden türetilen alt seviye gereksinimler modele entegre edilmiş olup kaynak kod oluşturulmadan önce, DO-331 model benzetim aktivitesi göz önünde bulundurularak model üzerinde gereksinim tabanlı fonksiyonel testler gerçekleştirilmiş ve hata tespiti yapılmıştır. Ayrıca, testlerle birlikte model kapsama analizleri de yapılmıştır. Hem gereksinim bazlı testlerin hem de model kapsama analizlerinin sonuçlarına göre performans ölçütleri elde edilmiştir. Fonksiyonel testlerle benzer şekilde, model üzerinde statik olarak yapılan standart uyumluluk testi de performans ölçütü olarak değerlendirilmiş ve bu değerlendirmelerin ardından kaynak kod oluşturma aşamasına geçilmiştir. Kaynak kod, Matlab geliştirme aracı kullanılarak oluşturulmuş, yine Matlab araçlarından faydalanılarak üretilen kaynak kod incelenmiş ve başarım ölçütleri alınmıştır. Ardından, çalıştırılabilir nesne kodu oluşturulmuş ve döngüde yazılımsal benzetim testi ile başarım ölçütü değerlendirilmiştir. Özellikle yazılım geliştirme sürecinde MathWorks araçlarından Simulink Test, Requirements Editor, Simulink Design Verifier gibi özel kütüphaneler kullanılmıştır.

Son olarak, yazılım geliştirme sürecinde gereksinimlerden çalıştırılabilir nesne koduna kadar her adımda ilgili proses raporları oluşturulmuş, konfigürasyon takibi yapılmış ve otoriteye sunulmak üzere yazılım kalite teminatı ve sertifikasyon süreçleri hazır hale getirilmiştir. Sonuç olarak, DO-178C/DO-331 rehber dokümanı kapsamında sertifikasyon gereksinimlerine uygun bir uçuş kontrol yazılımı geliştirilmiştir.

1. INTRODUCTION

In recent decades, there has been a remarkable expansion in the scale and complexity of safety-critical software applications, accompanied by advancements in development standards. These applications have emerged as pivotal forces shaping cutting-edge development procedures and advancements within the aerospace industry. Despite the complexity involved in engineering such software applications, the transition towards software serving as the primary control mechanism for safety-critical systems has played a crucial role in enabling this progress.

In this thesis study, an aircraft flight control software is developed using a model-based design approach as an alternative to the traditional software development methods. The software development lifecycle processes, activities to be carried out in these processes, and the software lifecycle data to be produced were addressed during the software development process.

1.1 Motivation and Purpose of Thesis

The purpose is a developing a flight control system and flight control software compatible with the DO-178C/DO-331 guidelines using a model-based design approach for a fighter aircraft [1] [2].

The thesis delineates the development of a mathematical simulation model using model-based tools for an air vehicle, flight control algorithms, and model-based software, as per the author's understanding. The process outline is based on extensive experience gained during the development of mathematical simulation models and the design of flight control algorithms for the F-16 fighter aircraft [3].

The primary focus of this thesis revolves around the delineation of system development to software processes such as software requirements allocated to the software (SRATS), followed by the generation of high-level requirements (HLR) that encompass more detailed design requirements and performance criteria essential for the development of the flight control system. Based on the generated HLRs, the

relevant control algorithm is developed, and a design model representing the low-level requirements (LLR) is created, leading to the development of the flight control system and its software.

1.2 Literature Review

In scope of this thesis, a detailed review of national and international literature on airworthiness certification, safety critical software development and related guidelines, model-based design, flight control algorithm and flight control software development have been investigated.

1.2.1 Safety-critical software development and DO-178C

In [4], theoretical principles and engineering methodology of airworthiness is analyzed, airworthiness concept is given with the system safety and integrity perspective.

In [5], the importance of balancing safety certification requirements with software maintenance and economic considerations are emphasizes. It argues for addressing both certification and quality issues early to minimize long-term maintenance costs and ensure the software's economic viability throughout its lifecycle.

This article [6] provides an overview of the new DO-178C standard and its supplements, aimed at helping readers unfamiliar with DO-178C understand its extensive content. It contrasts DO-178B and DO-178C, detailing their similarities, differences, and practical implications for organizations. The article emphasizes that while it cannot cover every detail, its comprehensive information aims to assist applicants in reducing compliance issues, lowering development costs, and ensuring the safety of avionic systems.

The paper [7] presents a streamlined, automated software development workflow tailored for prototype and experimental airborne systems, focusing on selected DO-178C and DO-331 objectives. By leveraging extensive automation and a qualified model-based toolchain, along with custom tools and continuous integration, manual development efforts were significantly reduced while maintaining high software quality. The approach demonstrated potential for future certification under DO-178C/DO-331 with minimal data rework, achieving a 50% reduction in development effort compared to industry averages for DO-178C level C software. Future work aims

to bridge gaps for level A software development and explore scalability to complex systems.

The paper [8] explains how Model-Based Development and Verification transforms models into implementations. It outlines how various SDLCs (Waterfall, V, Spiral, Incremental) can be organized under Model-Driven Design (MDD) to meet DO-178C/DO-331 Level A software requirements effectively, despite differing in process sequence and granularity.

In [9], four approaches are presented to software development and verification for military aircraft systems, highlighting their respective pros and cons. It discusses strategies for reconciling civil and military standards, suggests stakeholders consider these options, and emphasizes ongoing dialogue to optimize safety, cost, schedule, and aircraft performance.

The thesis [10] aims to establish a model-based software development process and tooling that meets industry standards, supports scalability like agile methods and DevOps. The article introduces a novel, automated software development process using a model-based design approach compliant with DO-178C/DO-331 standards, with Siemens' web-based application Polarion employed for requirements management.

In [11], the study focuses on the investigation of MDE for supporting the use of different modelling languages for describing the same system with DO-178C regulatory objectives.

The dissertation [12] aims to achieve two primary goals: firstly, to advocate for the integration of Agile development methods enhanced by Lean principles for the development of safety-critical flight software; and secondly, to illustrate the adoption of Agile methodologies by proposing a framework aligned with the objectives of DO-178C.

Rapita Systems, in its recent report [13], has analyzed the safety-critical software development process in detail and discussed the software lifecycle from the perspective of the stage of involvement (SOI) processes with the authority.

1.2.2 Model based design

In [14], it is highlighted that the updated commercial aviation standards and their effects on developers that uses Model-Based Design.

[15] outlines a design methodology that utilizes mathematical modeling of physical dynamics using Model-Based Design (MBD) highlights its critical role in device construction, hardware selection.

In [16], MathWorks presented the model-based design method using qualified MATLAB tools for DO-178B. It is noted the significant impact of model usage in the development of flight control software. Therefore, it is important to emphasize the significance of this source for 2009.

In [17], it is stated that MathWorks tools offer robust mechanisms not just for requirements definition and design, but particularly for the diverse verification stages, significantly easing compliance with DO-178C objectives with reduced effort compared to traditional methods.

Study [18] highlights two critical issues of Model-Based Design (MBD) technology impacting the certification and safety of software in safety-critical systems. Firstly, it addresses model integrity during transfers between tools. Secondly, it examines the comparison between Structural Coverage Analysis and Model Coverage Analysis methods.

To address increasing costs and schedule delays in flight critical software development, an affordable Verification and Validation (V&V) process has been developed in [19]. It includes three components: model validation tools, software verification tools, and a system bench for real-time simulation (hardware-in-the-loop) to evaluate flight critical software during runtime. This approach aims to mitigate technical challenges in V&V, ensuring cost-effectiveness and efficiency.

In [20], the implementation of DO-178C and DO-331 guidelines for designing and deploying control algorithms in a Model-Based Design framework, initially for BLDC motor control and with future plans for turbo-propeller engine applications.

In [21], a Model-Based Agile process is presented in order to use of agile methodologies in the development of safety-critical systems.

1.2.3 Air vehicle flight dynamics and flight control system design

The book [22] consolidates all aspects of flight dynamics for developing aerospace vehicle simulations, offering comprehensive tools for building, programming, and executing simulations. Unique to this book is its use of tensors for modeling flight dynamics, ensuring invariance under coordinate transformations.

In [23], it is derived and utilized full nonlinear and coupled 6-DOF equations of motion for missile dynamics in all three control loops. State-dependent nonlinear control is applied in each loop without any simplifications to the equations.

[24] details the use of nonlinear dynamic inversion for designing a flight control system for a supermaneuverable aircraft. By separating dynamics into fast (angular rates) and slow variables (angle of attack, sideslip angle, bank angle), and applying control laws to each, the system outperformed conventional gain-scheduled systems in lateral acceleration, sideslip, and control deflections.

[25] originates from a research project started in February 1999 by the X-38 Flight Controls Branch at NASA Johnson Space Center, and aimed to develop design guidelines for flight control systems of vehicles with highly nonlinear dynamics, operating across broad flight regimes. These guidelines focus on reentry vehicles, like the X-38, capable of transitioning from space reentry to controlled touchdown on Earth, using parachutes, paragliders, or automatic/human-controlled landings similar to the space shuttle Orbiter.

[26] implements modern control techniques to achieve stable recovery from high angle of attack (stall) and agile maneuvering in air combat or defense missions. TVC is combined with conventional aerodynamic controls. The controller design uses NDI methodologies, with stability and robustness analyzed through robust performance techniques. The control architecture supports both stall recovery (stabilization controller) and agile maneuvering (attitude controller).

[27] designs an NDI autopilot for an air defense missile with aerodynamic and thrust vector control. A two-loop cascaded structure using a second-order reference model and PI controller is employed, with the reference model adapting to flight conditions. The autopilot maintains stability and tracks commands within acceptable deviations despite uncertainties. A guided scenario, using an IMU model without aerodynamic angles, demonstrates the missile hitting the target following guidance commands.

1.3 Thesis Outline

After the introduction, the fundamental concepts and general information related to the aircraft certification process have been provided and its history and challenges are discussed in Chapter 2.1.

Chapter 2 describes all processes and process activities in the DO-178C guideline document while developing a flight control software and the DO-331 model-based design supplement document in detail.

Chapter 3 presents the definition and overview of Model Based Development (MBD) technology and the advantages it provides to user. The differences between traditional software lifecycle and MBD technology, and the types of MBD lifecycle are presented. In addition, the additional activities specific to MBD technology that need to be incorporated into the software development lifecycle are detailed on a process-by-process basis.

Chapter 4 explains the development of a mathematical simulation model for air vehicle flight dynamics, detailing the acquiring of the equations of motion to capture the aircraft's behavior in flight. In addition, airframe specific data such as geometry, weight and balance, aerodynamics as well as atmosphere calculations and subsystem models such as propulsion, actuator and sensor models are given. The input vector, state vectors and output vector of the aircraft model are determined and then the algorithms for trim and linearization are presented.

Chapter 5 defines the mathematical background of the control method and presents the control problem. The choice of controlled parameter of the aircraft is selected and related control architecture is demonstrated.

Chapter 6 provides the application details of this thesis study realized through the model-based design approach. A nonlinear flight control system for the F-16 fighter aircraft has been developed by implementing all processes specified in the DO-178C software guideline and its model-based design supplement DO-331. Traceability, verification, model simulation, model coverage analysis, and activities compliant with the DO-331 standard have been conducted.

Chapter 7 interprets the results of this thesis study and details possible further studies based on the subjects addressed in the scope of this thesis study.

2. SAFETY-CRITICAL SOFTWARE DEVELOPMENT PROCESS

The safety-critical software development process in the context of airworthiness certification is governed by stringent standards to ensure the highest levels of reliability and safety. A key standard, DO-178C, provides guidelines for the software lifecycle, including planning, development, verification, and validation activities. Compliance with DO-178C is mandatory for obtaining airworthiness certification from aviation authorities, ensuring that software functions correctly and safely under all operating conditions. The lifecycle defined by DO-178C ensures rigorous documentation, testing, and review processes to mitigate risks associated with software failures in critical aviation systems. This chapter details the topics summarized above.

2.1 Airworthiness Certification

Airworthiness certification is a crucial process in the aviation industry that ensures aircraft, parts, and equipment meet established safety standards set by regulatory authorities. To ensure that an aircraft can safely operate within its operational envelope, it's imperative to prove that its design, development and maintenance adhere to relevant regulations. The responsibility for verifying this compliance lies with the competent authorities which will be covered in this chapter.

2.1.1 Flight safety and airworthiness

Safety can be defined as a state of being shielded from, or with minimal risk of, harm, danger, or injury [28]. It involves the implementation of measures and practices designed to prevent accidents, harm, and damage, thereby ensuring the well-being and security of individuals, property, and the environment. It is governed by guidelines, regulations, and standards aimed at mitigating potential hazards and promoting a secure environment.

Flight safety refers to the set of practices, regulations, and technologies designed to prevent accidents and incidents in aviation, ensuring the well-being of passengers, crew, and aircraft. It encompasses various components, including the meticulous

design and air vehicle maintenance, rigorous training and continuous assessment of pilots, and the efficient management of air traffic control. Factors affecting flight safety include weather conditions, operational procedures, human factors, and adherence to international safety standards and regulations established by authorities like the International Civil Aviation Organization (ICAO), Federal Aviation Administration (FAA) in USA, and European Aviation Safety Agency) (EASA) in EU. Continuous advancements for safety related management systems and aviation technology improvements are also essential in enhancing flight safety.

Airworthiness is the state of an aircraft or other aviation product that meets the requirements for safe flight within its limits. This definition highlights three fundamental elements such as safe situations, requirements, and the limits [29]. *Safe situations* ensure that the aircraft can perform its intended functions without posing any undue risk to passengers, crew, or property, involving rigorous testing and validation to confirm that all systems and components function correctly under various operational scenarios. *Requirements* refers to the adherence to design, construction, and maintenance standards governed by aviation regulatory authorities for instance EASA and FAA, which are managing the regulations in Europe and USA, and adhering these requirements is essential for obtaining and maintaining an airworthiness certificate. *Limits* involve ensuring that all aspects of the aircraft's performance and operational capabilities remain within specified boundaries, including weight and balance, structural integrity, engine performance, and environmental conditions, ensuring that the aircraft can perform safely and efficiently without exceeding its designed capacity. In summary, airworthiness is a comprehensive concept that ensures an aircraft or its parts meet all necessary safety and regulatory standards, are maintained in optimal condition, and operate within defined performance limits, which is crucial for maintaining the reliability and the safety of any flight or air transportation [29].

2.1.2 The ICAO and the civil aviation authorities

It is crucial to establish global standards and regulations for aviation safety, security and environmental protection with authorities such as ICAO and numerous civil aviation organizations for global aviation system.

2.1.2.1 The ICAO (international civil aviation organization)

The International Civil Aviation Organization (ICAO) manages the administration and governance of the Convention on International Civil Aviation which is called Chicago Convention. Founded in 1944, ICAO was formally launched in 1947 to facilitate the safe and structured growth of global civil aviation. Its creation aimed to support diplomacy and cooperation among countries, ensuring the harmonization and standardization of regulations and practices [30].

ICAO aims to develop principles and techniques of international air navigation and promote international air transport [31].

ICAO develops and adopts standards and recommended aerospace related practices (SARPs) to ensure uniformity and safety in international aviation. These SARPs are detailed in Annexes to the Chicago Convention, each addressing specific aspects of aviation like flight operations, environmental protection, and safety management. There are currently 18 Annexes, the most utilized ones are related with the personnel licenses, aircraft operations and its safe flight which is airworthiness and ground facilities such as aerodromes [32].

These Annexes are critical for maintaining global aviation standards and ensuring the interoperability and international flight safety. The efforts of ICAO in standardization and recommended practices facilitate international air travel, making it safer and more efficient.

2.1.2.2 The civil aviation authorities

The origins of aviation authorities can be linked back to the beginning of 20th century, when governments initialized establishing regulatory bodies to oversee the growing field of aviation. The United States took the lead from the Air Commerce Act in early 20s, creating the Aeronautics Branch to ensure safety and reliability in air travel. Other nations followed suit, developing their own authorities to license pilots, certify aircraft, and regulate air traffic. The creation of international regulatory organizations like the ICAO in 1944 further facilitated cooperation and the development of global aviation standards. Today, civil aviation authorities worldwide continue to fulfil a critical role in guaranteeing the safety, efficiency, and security of air transportation within their respective jurisdictions [29].

Aviation authorities are responsible for several key tasks such as establishing airworthiness requirements and procedures, informing stakeholders by publishing standards and regulations, overseeing aeronautical materials, design processes, manufacturing organizations, and aircraft operations, and certifying aeronautical organizations and its materials.

i. The Joint Aviation Authorities (JAA)

JAA is a subsidiary organization of the European Civil Aviation Conference (ECAC). It encompasses the civil aviation regulatory authorities of multiple European sides that collaborate on the development and the implementation of unified safety standards and procedures [29].

The European Aviation Safety Agency (EASA) was established as the successor for the JAA in 2002. EASA assumed responsibility for aviation safety regulation and oversight within the member states of European Union (EU), taking over many of the functions previously carried out by the JAA. EASA continues to work toward harmonizing aviation regulations and standards across. Europe while also representing the EU in international aviation forums.

ii. The European Aviation Safety Agency (EASA)

The European Union Aviation Safety Agency (EASA) which is launched in 2002 and responsible for overseeing safety of civil aviation and environmental protection within European Union's member states. EASA operates under EU law and performs a range of regulatory and oversight functions to maintain high standards of safety across the aviation sector.

EASA's primary responsibilities include the certification of aircraft and aircraft components. It issues type certificates for new aircraft designs and approves modifications and repairs to existing aircraft. This certification process ensures that all aircraft operating in EU airspace meet rigorous safety and environmental standards, promoting confidence in the safety of air travel.

In addition to certification, EASA develops and maintains the regulatory framework for civil aviation safety in Europe. This framework encompasses airworthiness standards, operational rules, and requirements for aircraft maintenance. EASA conducts safety assessments and inspections to verify compliance with these

regulations, ensuring that airlines, maintenance organizations, and air traffic management systems operate safely and effectively [33].

iii. The Federal Aviation Administration (FAA)

The Federal Aviation Administration (FAA) is an administrative body within the United States air transportation that holds responsibility for the oversight and regulation for civil aviation operations along the US. Established in 1958, the FAA is tasked with ensuring for safety, efficiency, and environmental responsibility of US aviation ecosystem [34].

The responsibilities of the FAA encompass a wide range of activities across the aviation industry. One of its key functions is the certification of aircraft, pilots, and other aviation personnel. The FAA issues type certificates for new aircraft designs and approves modifications and repairs to existing aircraft. It also grants certificates to civil pilots, aircraft maintenance engineers, and also air traffic controllers ensuring that they meet stringent training and competency standards.

The FAA is responsible for establishing and upholding the regulatory structure that ensures the safety of civil aviation in the United States. This framework includes airworthiness standards, operational rules, and requirements for aircraft maintenance. The agency conducts regular inspections and audits of airlines, maintenance organizations, and air traffic management systems to verify compliance with these regulations, thereby ensuring the safety and reliability of air transportation.

2.1.3 Airworthiness requirements

Airworthiness requirements refer to the set of standards, criteria, and regulations that an aircraft or aircraft component must meet in order to be deemed airworthy. These requirements are established by aviation regulatory authorities like FAA, EASA, and other national civil aviation authorities worldwide.

The main objective of airworthiness requirements is guaranteeing an aircraft is to guarantee the safety and suitability of an aircraft for flight and can perform its intended functions without posing undue risk to passengers, crew, or the public. These requirements encompass different facets of the aircraft design, construction, maintenance, and operation. Some key areas addressed by airworthiness requirements

include structural integrity, systems and equipment, performance, flight characteristics, environmental compliance and maintenance.

Airworthiness requirements are documented in regulatory documents such as:

- Federal Aviation Regulations (FARs) such as Part 21 in the United States,
- EASA Implementing Rules such as Part-21, Part-145, Part-M in Europe,
- Civil Aviation Regulations (CARs) in many other countries.

2.1.4 Type certification

Type certification is the formal approval process conducted by regulatory authorities in aviation, such as the FAA or the EASA, which ensures that an aircraft design meets stringent airworthiness standards before it can be operated commercially. This certification process is crucial for establishing that the aircraft is safe, reliable, and capable of operating within specified environmental and operational limits.

The certification process itself involves extensive testing, analysis, and documentation. Manufacturers must submit detailed technical data, test reports, and compliance documentation to the regulatory authority. The authority reviews these materials, conducts inspections, audits, and flight tests as crucial to verify adherence with the specified requirements. Once satisfied that all requirements have been met, the regulatory authority issues a type certificate (TC), which permits the manufacturer to produce and sell the aircraft.

Type certification ensures that aircraft designs meet the highest safety and performance standards before they are introduced into commercial service. It is essential in maintaining the reliability and safety of operations for civil aviation worldwide, ensuring that passengers, crew, and the public are protected and that environmental impacts are minimized.

2.1.5 Certificates of airworthiness

A Certificate of Airworthiness (CofA) is an official document released by an aviation regulatory authority, like EASA or FAA, certifying that an aircraft meets specific safety standards and is fit to fly. It is issued after the aircraft manufacturer demonstrates compliance with rigorous airworthiness requirements during the type certification process. The CofA confirms that the aircraft is in a situation suitable for safe flight according to its design and operational limitations. It is valid for a set period

and must be renewed periodically, with the responsibility of operator for maintaining the aircraft's airworthiness through regular inspections, maintenance, and compliance with operational limitations. The CofA is crucial for guaranteeing the reliability and safety of operations in civil aviation worldwide [29].

2.1.6 Continued airworthiness

Continued airworthiness refers to the ongoing process of ensuring that an aircraft remains in a condition where it is safe to operate throughout its operational life. It involves maintenance, inspections, and other activities handled to provide that the aircraft proceed to address the requirements regulated by the specified by the aviation regulatory authority.

Continued airworthiness requires maintenance programs that outlines the scheduled maintenance tasks required to keep the aircraft in a condition suitable for safe flight. Conforming with mandatory directives like Airworthiness Directives (ADs) issued by the regulatory authority to address safety issues that may arise with specific aircraft models or components during continued airworthiness, also the aircraft's maintenance history, modifications, repairs, and compliance with airworthiness directives and service bulletins are kept. Component and system life limits are taken into account ensuring they are replaced or overhauled before exceeding their operational limits. In addition, periodic review conducted by a qualified person or organization to ensure that the aircraft and its components, including all modifications and renovations, meet the safety requirements and are deemed safe for continued operation.

Continued airworthiness is essential to ensure that the safe, reliable, and efficient of aircraft operations throughout their service life. It involves collaboration between aircraft operators, maintenance organizations, and regulatory authorities to ensure that all airworthiness requirements are fulfilled and any safety-related issues are promptly addressed. By maintaining airworthiness, operators ensure that their aircraft are safe for passengers, crew, and the public, and that they comply with all regulatory standards and requirements.

2.1.7 Airworthiness and software certification relation

DO-178C offers guidance for certifying software utilized in airborne systems such as aircraft flight control software and avionics. DO-178C fits into the broader context of airworthiness and certification processes as following [1]

DO-178C outlines the activities and objectives which must be fulfilled during software development lifecycle to ensure the software functions correctly and safely within the aircraft's avionics system.

Airworthiness certification, including the issuance of a Certificate of Airworthiness (CofA), requires assurance that all components of the aircraft, including software, meet certain safety and reliability standards. DO-178C is specifically referenced by the FAA and EASA, which are the aviation regulatory authorities in US and EU, respectively, as the standard for software assurance in airborne systems. Compliance with DO-178C is essential for demonstrating that software-related risks have been prevented or mitigated to an acceptable level, thereby contributing to the overall airworthiness of the aircraft. These makes DO-178C integration with airworthiness.

DO-178C is a critical standard for the development and airborne software systems' certification. This guideline ensures that software used in aircraft meets rigorous safety and reliability standards, thereby contributing to the overall airworthiness of the aircraft. Compliance with DO-178C is necessary to obtain airworthiness certification and is closely integrated into the broader airworthiness and certification processes defined by regulatory authorities worldwide.

2.2 Safety-Critical Software Development

Safety-critical software development is a specialized domain focused on creating software systems that must operate reliably and without failure in environments where errors could lead to catastrophic consequences, such as in aviation, medical devices, and nuclear power plants. This development process is characterized by stringent standards and rigorous methodologies, such as DO-178C in aviation, which govern the entire software lifecycle from initial planning through to deployment and maintenance. Ensuring both system functionality and safety, this process integrates comprehensive risk management strategies to identify, analyze, and mitigate potential hazards, thereby ensuring the highest levels of operational safety and reliability.

2.2.1 Software within the system framework

Software within the system framework focuses on the integration and interaction of software components with hardware and other system elements to achieve overall system functionality. This involves designing software architecture, interfaces, and communication protocols to ensure that the software can effectively support system operations. Key considerations include performance, scalability, interoperability, and compliance with system requirements. The goal is to ensure that the software components work seamlessly within the broader system to deliver the intended functionalities efficiently and effectively.

2.2.1.1 System development overview

Understanding the context of software role in the overall system is crucial for the successful implementation of safe software. Briefly, software operates in the context of a system, and safety is a property of the overall system. Software plays a crucial role in safety across diverse systems like aircraft, automotive, nuclear, and medical. To ensure software enhances safety, it's vital to grasp the system it operates in and the overall safety process.

An air vehicle represents a complex system comprised of numerous subsystems, essentially embodying a system of systems concept These subsystems encompass a wide array of functionalities such as structures, avionics, landing gear, flight control, actuators, sensors, environmental control, electrical power, engine control, ground steering, fuel systems, air data systems, and more [35]. Each system within the aircraft contributes to its overall operation, with varying degrees of impact on safety. Consequently, civil aviation regulations and supporting advisory materials mentioned in Chapter 2.1, classify failure categories based on their respective risk levels will be discussed in Chapter 2.2.2. Every function within each system, as well as their collective combination, undergoes assessment to determine their overall impact on ensuring safe aircraft operation. Aircraft operation encompasses several phases, including taxiing, take-off, climb, cruise, descent, and landing. Each phase is critical for the safe and efficient movement of an aircraft from departure to arrival at its destination. System safety is assessed for each mentioned phase, recognizing that the probable effects of failure conditions can vary significantly depending on the specific phase.

Developing a safe system usually demands years of experience in the particular type of system, coupled with a comprehensive understanding of aircraft operation, system functionalities, and their interactions with both each other and human operators, such as pilots. Safety-critical systems require significant time, resources, and integrity; hence, developers are encouraged to use SAE's Aerospace Recommended Practice (ARP) 4754 revision A [36] [37].

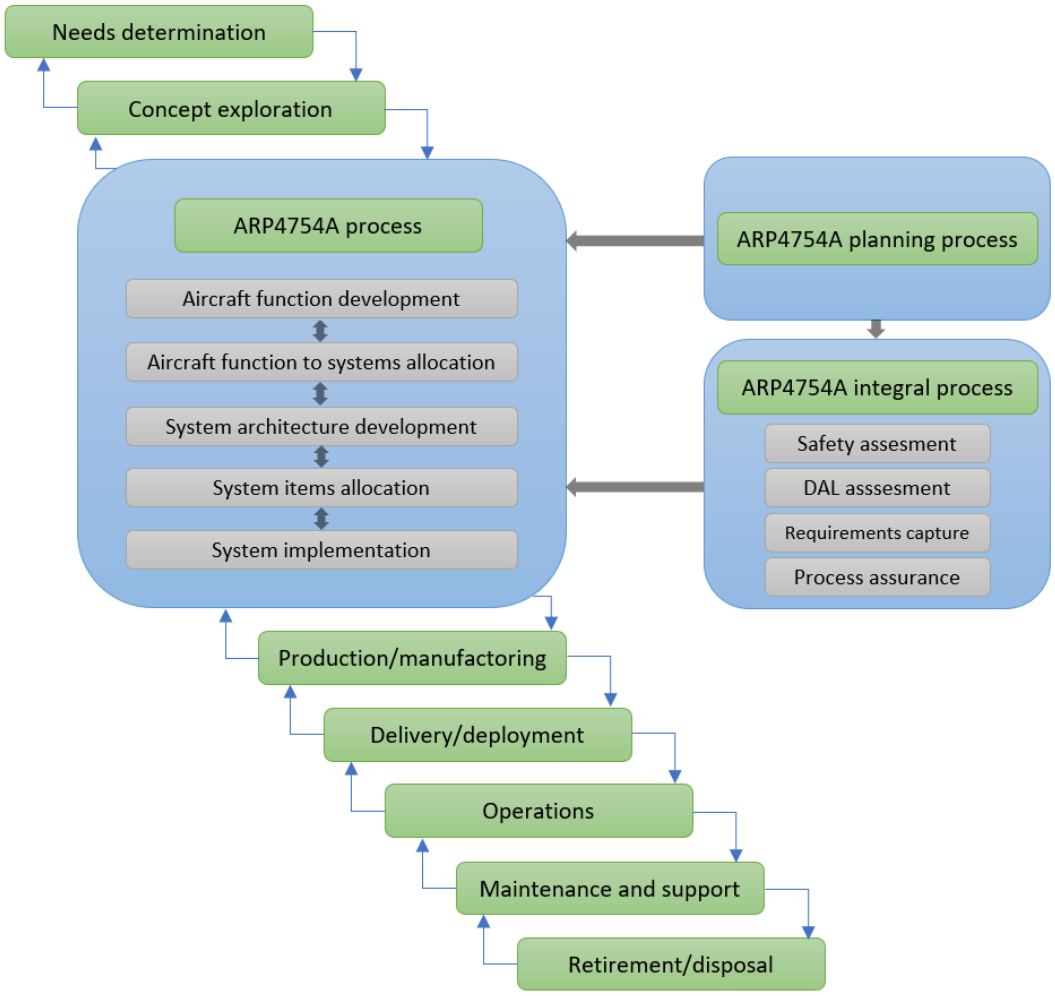


Figure 2.1 : Overall system development process in ARP4754A [36].

ARP4754A's scope encompasses system development from inception through certification completion. However, in addition to the system phases outlined in ARP4754A, system development involves activities both before product launch and after certificate issuance. Figure 2.1 illustrates the overall system development framework with the ARP4754A processes.

2.2.1.2 System requirements

Software development often arises from inadequately defined system requirements and architecture. Many challenges in software development arise from inadequately defined system requirements. To mitigate these issues, most companies should prioritize allocating additional effort towards defining, validating, and verifying system requirements. This emphasis is particularly crucial given the escalating complexity and criticality of systems, ensuring the continued development of safe, reliable, and dependable systems. ARP4754A recommends that the following types of requirements be developed:

- Safety requirements, identify functions that either enhance or directly impact the safety of the aircraft
- Functional requirements, identify functionality of the system to obtain desired performance
- Other requirements, include regulatory requirements and derived requirements

Requirements authors and reviewers incorporate the characteristics of good system requirements such as atomic, complete, traceable, etc.

System requirements considerations can be specified numerous functional items. Fundamental concepts to consider when developing systems requirements for software-intensive and safety-critical systems can be found in following Table 2.1.

Table 2.1 : Consideration examples for system requirements.

Integrity and Availability Considerations	Other Considerations
Integrity	Safety requirements
Availability	Error detection
	Graceful degradation
	Error handling
	Fault avoidance
	Functional limitations
	Critical functionality
	Partitioning
	Diversity
	Mode and state transitions

When developing the systems requirements and design, assumptions need to be documented. An assumption is a statement, principle, or premise that is not certain at the time the requirements are written.

The end result documentation of the software requirements process is the Software Requirements Document (SWRD). The SWRD explains the software functionality and limitations comprehensively and not leaves room for assumptions.

Typically, there exist multiple hierarchical levels of system requirements, including aircraft, system, and subsystem levels. These lower levels are derived from the higher ones, with increasingly detailed specifications defined at each lower tier. The most granular level of system requirements is then allocated to either software or hardware for implementation. In ARP4754A, software and hardware are referred to as items. RTCA DO-178C is then applied to the software items and RTCA DO-254 is applied to the electronic hardware items [36] [1] [38].

2.2.2 Software within the system safety assessment framework

Software within the system safety framework emphasizes the identification, analysis, and mitigation of risks associated with software failures that could impact system safety. This framework involves the application of safety engineering principles and standards, such as DO-178C in aviation, to ensure that software does not contribute to hazardous conditions. It includes rigorous processes for hazard analysis, safety requirements allocation, fault tolerance, and safety verification and validation. The primary objective is to ensure that the software operates safely under all conditions, preventing and mitigating potential safety risks that could lead to accidents or incidents.

2.2.2.1 Aircraft and system safety assessment process overview

As outlined in Chapter 2.2.1, safety assessment progresses concurrently with the development of both the aircraft and its systems. The process for safety assessment in civil aviation will be discussed in this chapter and elucidates the integration of software within the system and safety framework. Figure 2.2 illustrates the iterative progression system advances for system and safety development, and identifying and addressing the safety aspects through design measures.

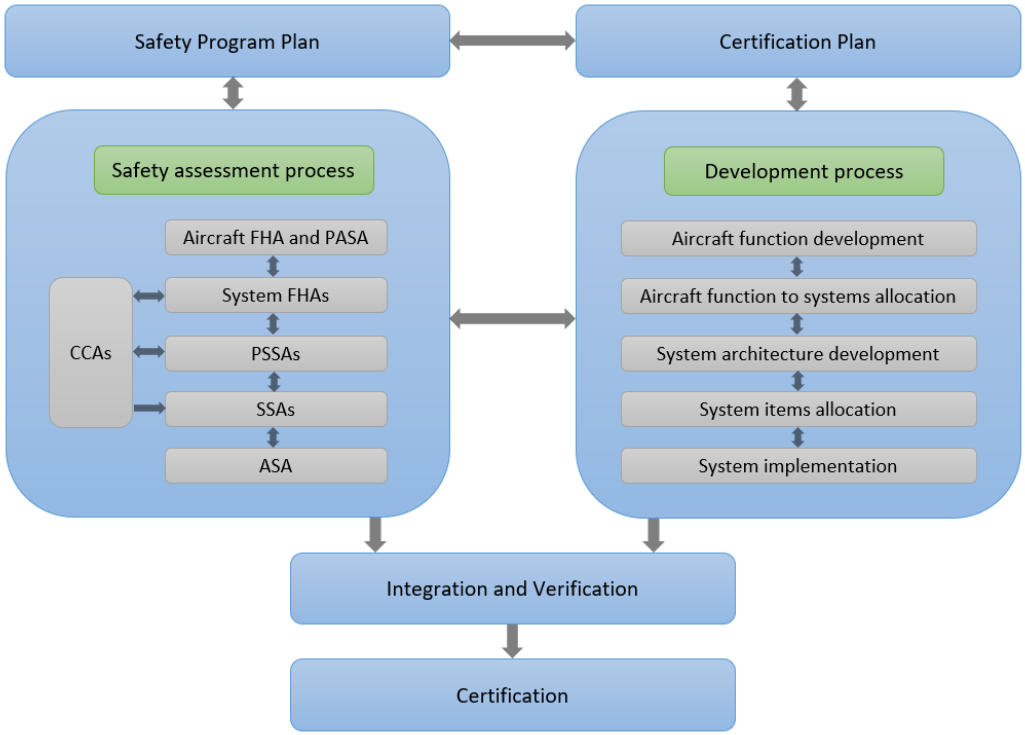


Figure 2.2 : System and safety assessment procedures overview [37].

SAE's ARP4761 offers an intricate delineation of the requirements' safety for civil aviation [28].

The development of the *Safety Program Plan* is typically the initial task carried out in the overall assessment of aircraft safety. The plan delineates the methods by which aircraft safety will be guaranteed, the process for identifying safety-related requirements, the application of safety standards, data generation, methodology implementation, task assignment, key milestones, and coordination mechanisms. These measures are aimed at ensuring correctness, completeness, and consistency throughout the process.

The Functional Hazard Assessment (FHA) is created once the fundamental aircraft functionality and conceptual design are documented. The FHA identifies and categorizes failure conditions linked to aircraft functions and their combinations. Table 2.2 shows the severity classifications along with the failure condition effects and required probabilities for any aircraft.

The System Functional Hazard Assessment (SFHA) is akin to the FHA but delves deeper into the analysis of system architecture. It aims to identify and classify potential failure conditions and their combinations within each aircraft system.

Table 2.2 : Failure condition probabilities, and levels.

Severity Classification	Likelihood of Occurrence	Assurance Level
Catastrophic	Extremely improbable	A
Hazardous	Extremely remote	B
Major	Remote	C
Minor	Reasonably probable	D
No safety effect	Probable	E

The Preliminary Aircraft Safety Assessment (PASA) entails a structured evaluation of proposed architectures to ascertain the potential impact of failures in causing the Failure Conditions described by the FHA [28].

The Preliminary System Safety Assessment (PSSA) is a thorough analysis of how failures can lead to functional hazards identified in the FHA and SFHA. The PASA and PSSA are evaluations that typically assign development assurance levels for the system (functional development assurance levels, FDALs) and for software and electronic hardware (item development assurance levels, IDALs). [37].

The Common Cause Analysis (CCA) assesses system architecture and aircraft design to determine vulnerability to common cause events.

The Aircraft and System Safety Assessments (ASA) and *System Safety Assessments* (SSA) ensure that the performed design addresses the safety requirements outlined in the previous assessment procedures. Unlike the PASA and PSSA, which evaluate proposed designs, the ASA and SSA are conducted on the final, as-built design, providing the final proof of compliance with regulatory safety standards.

2.2.2.2 Development assurance

In intricate and deeply integrated electronic systems featuring hardware and/or software, testing all conceivable inputs and output combinations to ascertain failure probability is impractical. This constraint, coupled with the fact that computer programs do not degrade or fail over time like physical components, necessitates the adoption of a concept known as *development assurance*. Development assurance is employed to instill confidence in the process used to develop the system and its components. Development assurance operates on the premise that a more particular process is better equipped to detect and rectify errors before the product delivery

compared to a less meticulous process. Again, for most aircraft and engines, the assurance levels shown in Figure 2.3 apply.

ARP4754A delineates two phases of system development: the *function development phase* and the *item development phase* [36]. The function development phase encompasses the development, validation, verification, managing the configuration, assurance of the processes, and coordination with the certification authorities for the system. At the lowest level, system requirements are allocated to software or hardware, referred to as *items*. The software or hardware items undergo their own development phases.

ARP4754A guidance applies to the system development phase; DO-178B/C applies to the software development phase; and DO-254 applies to the electronic hardware development phase.

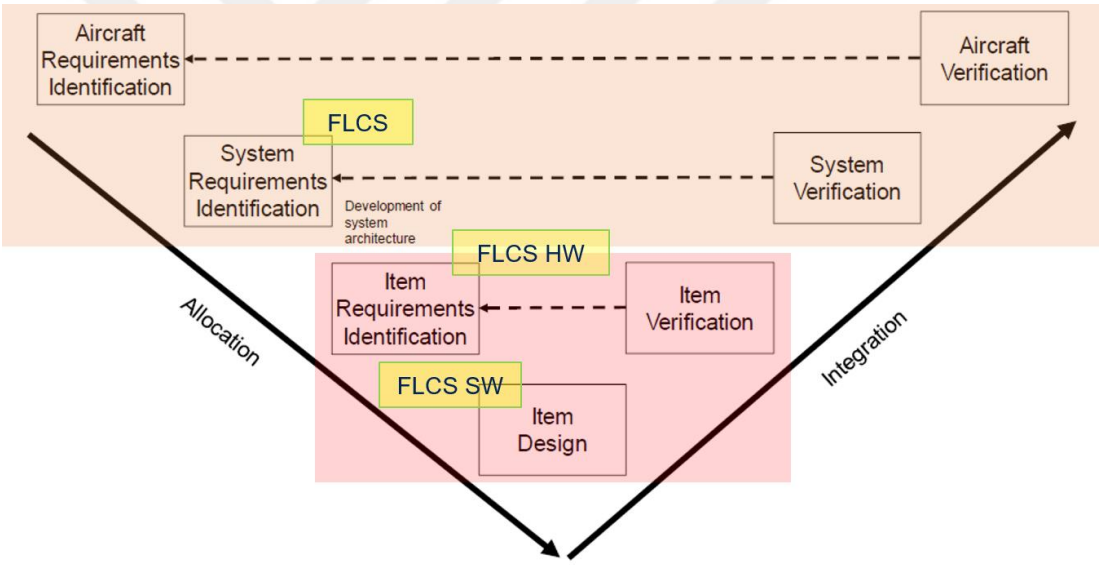


Figure 2.3 : The V-model for system development [37].

The traditional V-model illustrated in Figure 2.3 shows the function and item development phases. At the function level, an FDAL is assigned based on the system's potential impact on safety. The software and electronic hardware are assigned IDALs.

The FDAL determines the amount of rigor required at the system level such as the extent of requirements reviews, testing, and independence required. The IDAL specifies the level of rigor necessary for the development of the specific item, whether it be software or electronic hardware. Depending on the architecture, the IDAL may be lower than the FDAL. In essence, the FDAL governs which objectives of

ARP4754A are applicable at the system level, while the IDALs dictate which objectives of DO-178C (or its supplements) apply to software, or DO-254 objectives apply to electronic hardware.

2.2.3 Safety process and software relevance

As software has become more widespread, there has been a corresponding evolution in system and safety processes. Conventional analog and mechanical system failure analysis and reliability techniques do not apply to software because of its complexity and unique characteristics. These processes are summarized in Figure 2.4.

Due to the challenge of applying reliability models accurately to software, development assurance concept is commonly employed across industries which are safety-focused such as aviation. For the aviation industry, DO-178C identify five levels of software considering the safety process. Table 2.3 shows the relationship between the software levels with respect to condition of failures. Level A is the most stringent category and pertains to software functionality capable of causing or contributing to a catastrophic event at the aircraft level. For level A, all of the objectives of DO-178C apply. Level E software has no safety implications; therefore, none of the DO-178C objectives are required.

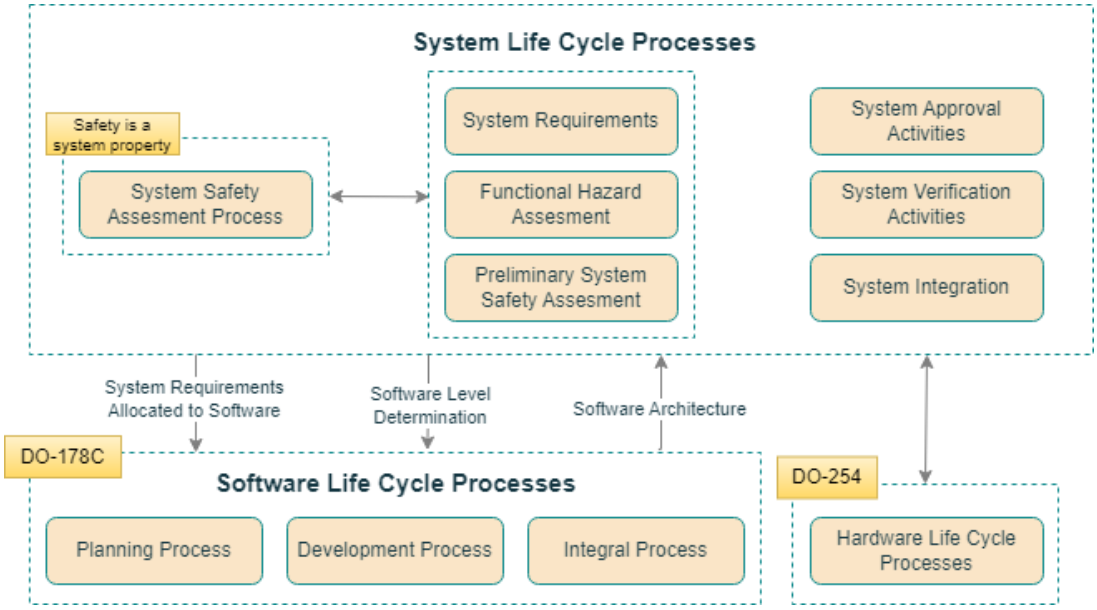


Figure 2.4 : Safety and software life cycle processes relevance.

Table 2.3 provides a visual representation of how the level of software development and verification rigor increases as the software level rises along with the number of

objectives. As the level increases, more activities and more independence are required. It's important to understand that the higher the software level, the more verification is performed and the more independence is required, and the more errors are identified and removed.

Table 2.3 : Summary of DO-178C objectives [1] [37].

Level	Obj Count	Obj w/Ind	Summary of Objectives
E	0	0	No activities required
D	26	2	Plans (five plans) High-level requirements developed Architecture developed Executable object code developed Only review or analyze architecture if using partitioning Normal and robustness testing of HLRs Requirements coverage of HLRs Testing to verify target compatibility Configuration management Quality assurance (compliance to plans and conformity review) Accomplishment summary and configuration index
C	62	5	Level D activities Development standards (three standards) Low-level requirements developed Additional review and analysis of HLRs Some review and analysis of LLRs Some review and analysis of architecture Review and analysis of some source code Verification of parameter data item files Normal and robustness testing of LLRs Data and control coupling analysis Additional quality assurance
B	69	18	Level C and D activities Additional review and analysis of HLRs Additional review and analysis of LLRs Decision coverage
A	71	30	Level B, C, and D activities Modified condition and decision coverage analysis Source to object code traceability verification

2.3 DO-178C and Supporting Documents

DO-178C was established as a consensus-based set of recommendations by the international community under the sponsorship of RTCA and EUROCAE [39]. It was published on December 13, 2011. DO-178C fulfills a guidance for developing airborne software; however, most of the guidance is also applicable to other safety-critical domains.

RTCA is an association for both government and industry aviation organizations and European Organization for Civil Aviation Equipment (EUROCAE) is an international organization open to membership from European manufacturers and users of aeronautical equipment, national civil aviation administrations, and trade associations.

DO-178 and its EUROCAE equivalent (ED-12) have a 30-year history. In 1982, the initial version of DO-178 was released. This document was concise and offered only a broad framework for software development. DO-178A was published in 1985. Many systems that are still in existence today used DO-178A as their means of compliance. DO-178 and DO-178A preceded the development of ARP4754 and ARP4761, which put the system and safety assessment framework into place.

DO-178B was published in late 1992. It was developed in parallel with ARP4754 and ARP4761, it was developed concurrently with ARP4754 and ARP4761, which were published shortly thereafter. DO-178B provided more comprehensive detail compared to DO-178A and subsequently became the established standard for software development in civil aviation.

In 1999–2001, RTCA published DO-248, DO-248A, and DO-248B. DO-248 initially began as an annual report aimed at elucidating some of the more intricate aspects of DO-178B. The final version of the report was published in 2001. DO-248B was not designated as guidance material; rather, it served to clarify guidance already present in DO-178B. It functioned as an informational and educational document intended to assist users in comprehending and implementing DO-178B more effectively.

In 2005, RTCA and EUROCAE established a joint committee with the objective of revising DO-178B to offer a framework for accommodating changes in software technology. RTCA published the following documents as stated in Figure 2.5.

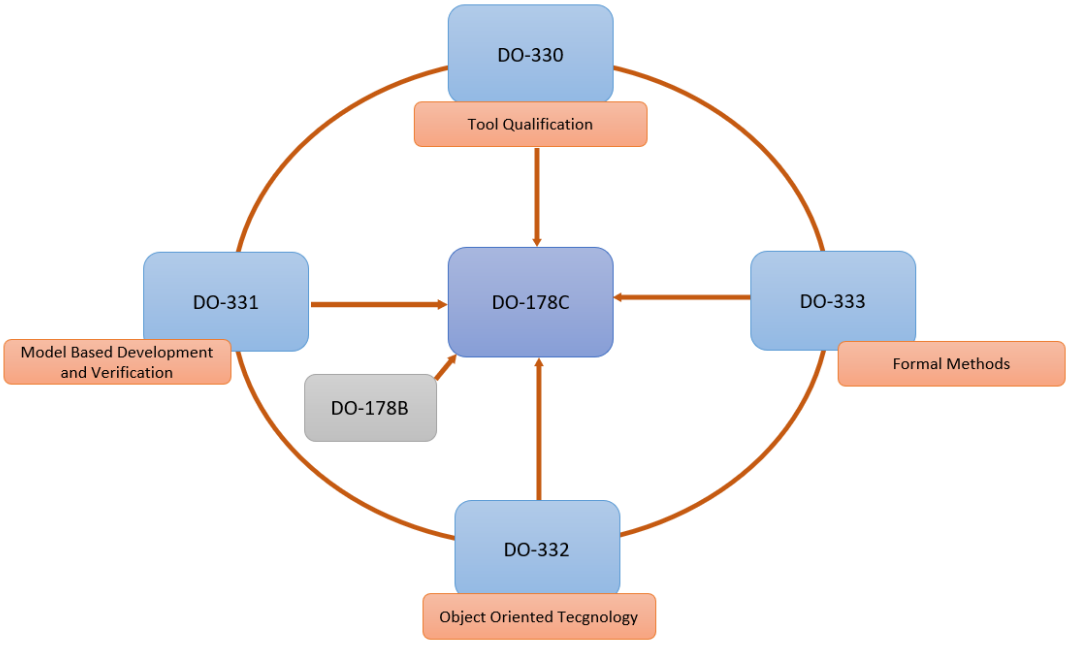


Figure 2.5 : Relationship of DO-178C and its supporting documents [37].

2.3.1 DO-178C core document

DO-178C and DO-278A are commonly referred to as the core documents because they lay down the fundamental concepts and structure upon which other documents are constructed. These core documents aim to remain as technology-independent as feasible. DO-248C and the supplements, on the other hand, technology-specific detail-oriented supplements. The connections between the sections of DO-178C are shown in Figure 2.6 explained below. [1]

- *DO-178C section 1* introduces the document
- *DO-178C section 2* includes the system framework which is aligned with ARP4754A.
- *DO-178C section 3* provides an information about software life cycle processes briefly.
- *DO-178C section 4* explains objectives and activities for the planning process
- *DO-178C section 5* elucidates the development processes' objectives, the activities include the requirements, design, code, and integration phases.
- *DO-178C section 6* details the process of verification. Verification is an integral process that commences with verifying the plans and extends to the reporting and review of verification outcomes. Verification includes reviews, analyses, and tests

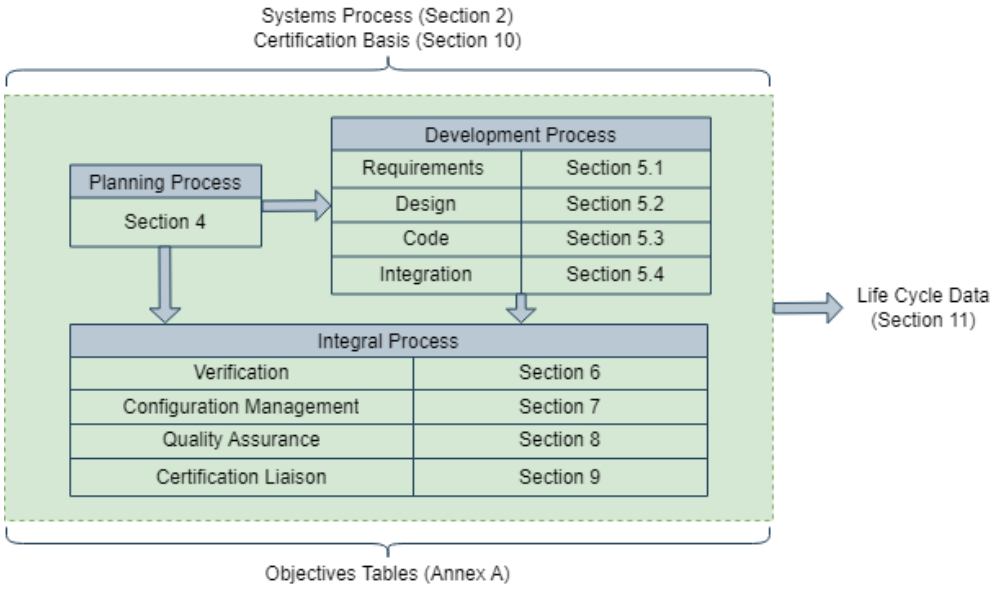


Figure 2.6 : Overview of DO-178C [1] [37].

- *DO-178C appendix A* provides background information, mentions the differences briefly from DO-178B to DO-178C, and the committee's terms of reference
- *DO-178C appendix B* lists the committee membership.

It is often said that DO-178C is best read from the back forward. DO-178C Annex A summarizes the objectives, applicable sections, and required data from DO-178C sections 4 through 11. The DO-178C Annex A tables essentially provide a roadmap for software level objective's applications. Table 2.4 provides the title of the ten DO-178C Annex-A tables, as well as the number of objectives included in each table.

DO-178C Annex A tables are essential tools that outline specific objectives and activities required for each phase of the software development lifecycle in safety-critical systems. These tables serve as comprehensive checklists for ensuring compliance with DO-178C standards, detailing the necessary processes and documentation for planning, requirements capture, design, coding, integration, verification, and configuration management. They provide a clear framework for developers to follow, ensuring systematic and thorough validation of software against safety criteria. By adhering to these guidelines, developers can demonstrate that all aspects of the software meet stringent safety and reliability requirements, ultimately supporting the successful attainment of airworthiness certification.

Table 2.4 : Summary of DO-178C Annex-A tables [1].

Table Number	Number of Objectives	DO-178C Annex-A Table Title
A-1	7	Software planning process
A-2	7	Software development process
A-3	7	Verification of outputs of software requirements process
A-4	13	Verification of outputs of software design process
A-5	9	Verification of outputs of software coding and integration process
A-6	5	Testing of outputs of integration process
A-7	9	Verification of verification process results
A-8	6	Software configuration management process
A-9	5	Software quality assurance process
A-10	3	Certification liaison process

Table 2.5 shows the number of objectives applicable by the level of software.

Table 2.5 : Software level objectives in DO-178C.

Software level	A	B	C	D	E
Number of objectives	71	69	62	26	0

The planning process drives the other processes, while the configuration management, quality assurance, and certification liaison processes occur throughout the entire project.

2.3.2 DO-178C technology supplements

To create a flexible approach capable of adapting to evolving technology and trends, the RTCA and EUROCAE committee chose to develop technology supplements. The vision is for the DO-178C main structure to remain unaltered while technology supplements can be changed or initiated as necessary in response to technological advancements.

Three technology supplements were released concurrently with DO-178C. Each supplement, organized around objectives, elaborates on the objectives outlined in the

DO-178C core for its particular technology. These supplements clarify how objectives from the DO-178C core in terms of applicability, modification, or replacement point. When multiple supplements are utilized simultaneously, the Plan for Software Aspects of Certification (PSAC) must delineate which supplement and objectives pertain to the software life segments cycle and the resultant data.

2.3.2.1 DO-331: model-based development supplement

DO-331, MBD Supplement to DO-178C [2] which define model as an abstract model representing various software aspects of a system, employed to facilitate the software development or verification process. This supplement pertains to models that exhibit the following characteristics [2]:

- The model is fully described using a clearly defined modeling notation, which may be graphical, textual, or both.
- The model includes software requirements and/or software architecture definition.
- The model is structured and categorized for direct analysis or behavioral evaluation, as supported by the software development or verification processes.

DO-331 builds upon the DO-178C core by introducing modifications or additional guidance. This supplement covers both model-based development and model-based verification. DO-331 provides guidance for *specification models and design models*, as well as analysis for model coverage and model simulation. Model-based development will be discussed more in Chapter 3.

2.3.2.2 DO-332: object-oriented technology supplement

DO-332 explains object-oriented technology using the software based on objects. There are concepts and techniques commonly found in object-oriented languages that necessary to consider when evaluating software safety [40]. DO-332 highlights the most significant concerns related to object-oriented technology (OOT) and offers guidance on how to address them. Moreover, DO-332 covers six associated techniques commonly linked with, though not confined to, object-oriented technology (OOT): parametric polymorphism, overloading, type conversion, exception management, dynamic memory management, and virtualization.

2.3.2.3 DO-333: formal methods supplement

DO-333 is related with the formal methods defines formal methods [41]. DO-178B identified formal methods as an alternative method; however, the guidance was vague and not widely applied. Since DO-178B's publication advances have been made in the area of formal methods, and tools and techniques have been improved to make the technology more feasible. Therefore, the formal methods section has been removed from DO-178C and addressed in a supplement (DO-333). DO-333 uses the DO-178C core as the foundation and supplements the core with formal methods-specific guidance.

2.3.3 DO-330: software tool qualification considerations

DO-178C a tool which is a computer program utilized to assist in the development, testing, analysis, production, or modification of another program or its documentation. Examples include automated design tools, compilers, test tools, and modification tools [1].

Due to the growing number of tools in use and the proliferation of third-party tool manufacturers, guidance on tool qualification was expanded and separated from the DO-178C core into a standalone document, DO-330. DO-178C section 12.2 explains determining the appropriate application of a qualified tool and assessing the required level of qualification involves. There are five TQLs defined; TQL-1 is the most rigorous one.

DO-330 outlines the steps required to qualify a tool. DO-330 is organized similar to DO-178C and has its own set of objectives. DO-330 is a stand-alone document that allows tool developers to develop qualifiable tools without having detailed knowledge of DO-178C itself. DO-330 was designed with the aim of being as domain-independent as feasible, enabling its application in other domains such as aeronautical database development, electronic hardware development, system safety assessment tool development, and system development and it applies to tolls implemented in software.

2.4 DO-178C Software Development Lifecycle

The DO-178C software development lifecycle outlines a rigorous process for developing safety-critical aviation software, covering planning, requirements, design, coding, integration, and verification. This standard ensures thorough documentation, testing, and reviews to meet strict safety and reliability criteria necessary for airworthiness certification.

2.4.1 Software planning process

The DO-178C planning process involves the development of project-specific plans and standards. Compliance with DO-178C requires that plans are both documented and followed. If the plans are written to address the applicable DO-178C objectives, then following them will ensure compliance with the objectives. Therefore, the success of the planning effort is crucial for DO-178C compliance and certification authority approval.

DO-178C identifies five process related plans and three life cycle related standards and explains the expected contents of each individual document as seen in Figure 2.7. While most organizations follow the five-plans-and-three-standards suggestion, it is acceptable to package the plans and standards as best fits the needs of the organization.

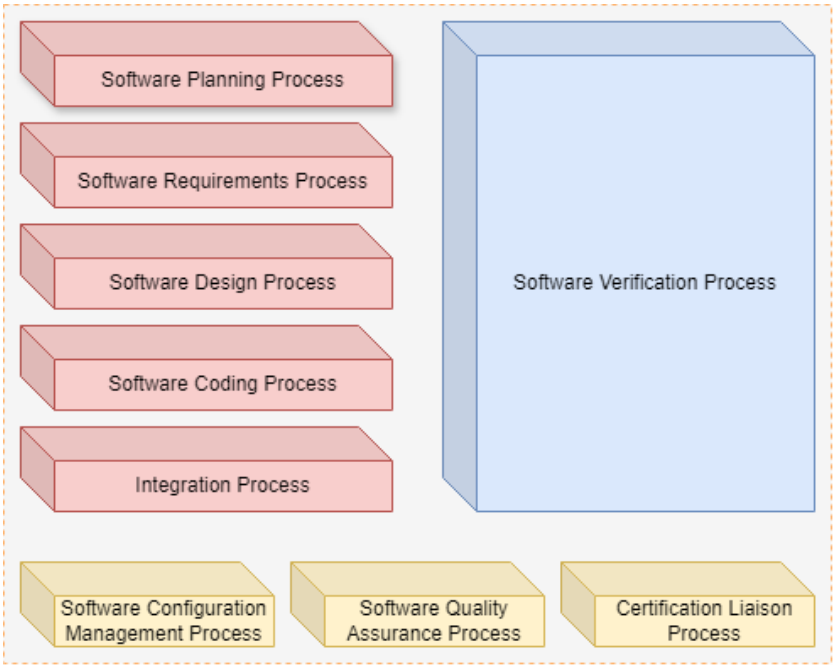


Figure 2.7 : DO-178C software development lifecycle process [42].

2.4.1.1 Plan for software aspects of certification

The Plan for Software Aspects of Certification (PSAC) is the one plan that is consistently served to the authority which is responsible for certification. The PSAC provides a high-level description of the overall project and explains how the DO-178C (and applicable supplements) objectives will be satisfied. It also provides a summary of the other four plans. DO-178C section 11.1 provides a summary of the expected PSAC contents and is often used as the outline for the PSAC.

2.4.1.2 Software development plan

The Software Development Plan (SDP) briefly explains the software development process while covering it in the planning aspect The SDP is written for the developers who generate requirements, make design, and generate code and perform the integration activities. SDP plays a crucial role in guiding developers toward successful implementation. It should strike a delicate balance: being detailed enough to offer clear direction yet not overly prescriptive to the extent that it stifles their capacity to exercise engineering judgment. Achieving this balance is essential as it empowers developers to make informed decisions while ensuring alignment with project objectives and compliance requirements. Therefore, the SDP serves as a guiding document, facilitating effective development practices and enabling teams to navigate complexities inherent in software development processes

DO-178C section 11.2 identifies the preferred contents of the SDP. The SDP includes a description of three major items:

- The standards used for development (occasionally, the standards are even included in the plan)
- The software life cycle with an explanation of each of the phases and criteria for transitioning between phases, and
- The development environment includes the methods for hardware platforms

2.4.1.3 Software verification plan

Software Verification Plan (SVP) recipients are the team members responsible for executing verification activities, which encompass testing among other tasks. The SVP maintains a close relationship with the Software Development Plan (SDP) as the verification process involves assessing data generated throughout the development

phases. The Software Development Plan (SDP) frequently offers a condensed overview of various development phases, including requirements, design, code, and integration verification, which may encompass activities like peer reviews.

DO-178C verification comprises reviews, analyses, and tests. The Software Verification Plan (SVP) elucidates the methodology for conducting these reviews, analyses, and tests. Additionally, any checklists utilized in conjunction with the verification process are either integrated within the SVP or referenced from it.

2.4.1.4 Software configuration management plan

The Software Configuration Management Plan (SCMP) outlines the procedures for managing the configuration of lifecycle data throughout the software development and verification process. Software configuration management (SCM) initiates during the planning phase and persists throughout the entirety of the software lifecycle, encompassing deployment, maintenance, and retirement phases of the software.

DO-178C section 11.4 provides a synopsis of the essential components to be included in the Software Configuration Management Plan (SCMP). The SCMP delineates the procedures, tools, and methodologies for both developmental SCM (employed by engineering before formal baselines or releases) and formal SCM. Furthermore, it outlines the transition criteria for the SCM process.

2.4.1.5 Software quality assurance plan

The Software Quality Assurance Plan (SQAP) articulates the software quality team's strategy for ensuring compliance of the software with approved plans, standards, and DO-178C objectives. It encompasses the organization of the Software Quality Assurance (SQA) team within the broader company structure and underscores their independence in executing quality assurance activities.

2.4.1.6 Development standards identified by DO-178C

In addition to the five plans, DO-178C identifies the need for three standards as following

- Requirements standard
- Design standard, and
- Coding standard

Industrywide standards (e.g., Institute of Electrical and Electronic Engineers standards) may serve as input to the project-specific standards, but each project has specific needs that are not dealt with in the industrywide standards. Most standards consist of both mandatory and advisory components. The mandatory elements are typically referred to as *rules*, while the advisory elements are termed *guidelines*. In many instances, the applicability of these rules and guidelines varies depending on the software level. For example, an item may be advisory for level C but mandatory for levels A and B.

i. Software requirements standards

The Software Requirements Standards (SRS) define methodology, tools, guidelines, and constraints for the software requirements development. SRS usually apply to high-level requirements, although some projects extend the application of the requirements standards to low-level requirements as well. Overall, the requirements standards serve as a guide for the team responsible for drafting the requirements.

ii. Software design standards

The Software Design Standards (SDS) define methodology, tools, guidelines, and constraints for developing the designing a software. The design standard includes the LLRs and the software architecture [1].

iii. Software coding standards

Similar to requirements and design standards, coding standards offer guidance to developers. They elucidate the proper usage of a specific language, restrict certain language constructs that may not be suitable for the safety-critical domain, define naming conventions, address the utilization of global data, and promote the creation of readable and maintainable code. Coding standards are prevalent in software development, and industry-wide resources such as MISRA-C from the Motor Industry Software Reliability Association can offer valuable guidance and input when formulating C-coding standards.

2.4.2 Software development process

The software development process involves a structured sequence of stages including planning, design, coding, testing, deployment, and maintenance. Each stage ensures

that the software meets user requirements, performs reliably, and can be maintained and updated efficiently.

2.4.2.1 Software requirements process

Software requirements serve as the foundation for DO-178C. The success or failure of a project depends on the quality of the requirements. The DO-178C glossary defines a requirement as software requirement, related with the system as high level, more detailed like low level and derived requirements.

In safety-critical software development, the significance of requirements cannot be overstated, with five key reasons with the "good requirements" underscoring their vital role as stated below

- Software development essentials
- Save time and money
- Safety
- Meet the customer needs
- Testing

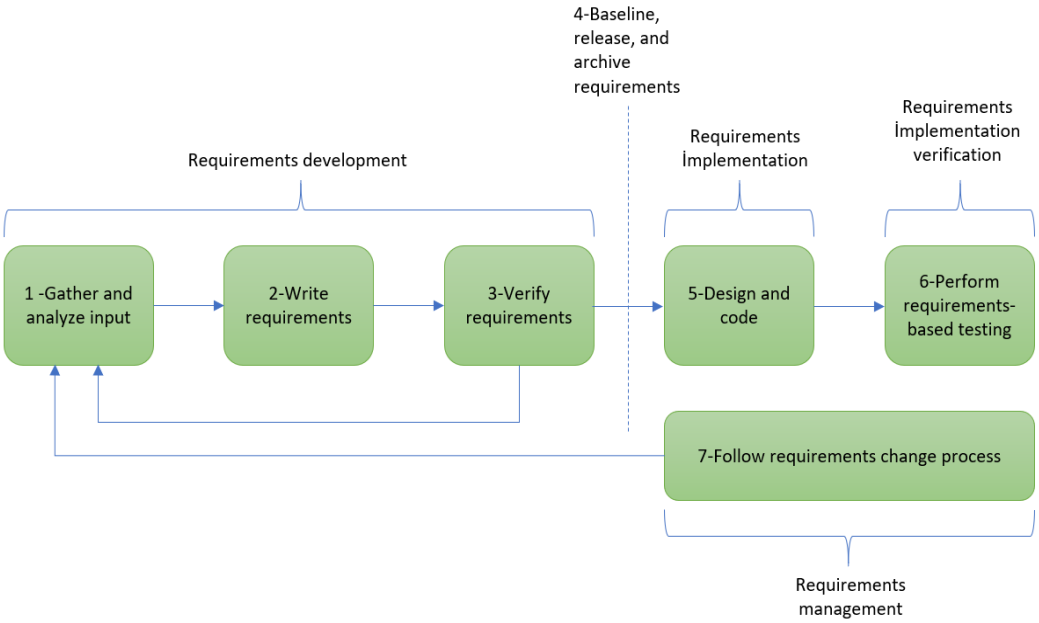


Figure 2.8 : Requirements development activities [37].

The requirements development effort can be organized into seven activities illustrates in Figure 2.8 can be stated as following:

- Gather and analyze input
- Write requirements
- Review requirements
- Baseline, release, and archive requirements
- Implement requirements
- Test requirements
- Change requirements using the change management process

In the software requirement process, each requirement should trace to one or more parent requirements, which are the higher-level requirements from which it was decomposed. The requirements team guarantees that each system requirement which is allocated to software is properly implemented from the software requirements which trace back to the system requirement. This process involves establishing a clear traceability path from system requirements to software requirements, ensuring that every aspect of the system's functionality is properly translated and implemented in the software. This traceability is crucial for verifying that the software meets all specified system-level requirements and is compliant with safety and regulatory standards such as DO-178C.

2.4.2.2 Software design process

During the software design process, LLR and software architecture are developed based on the defined software HLRs. This process will be detailed in Chapter 6.3.4

2.4.2.3 Software coding and integration process

Software source code is developed from the software design data, including software architecture and lower-level software requirements. As previously mentioned, in the aviation sector, qualified tools are often used to automatically generate code from design models.

The integration process covers two stages: software/software integration, software/hardware integration. Integration of software/software requires correct functioning of software components, whose verification process has been completed, is verified when they are combined together. In software/hardware integration, the integrated software is verified on the target hardware. Initially, integration of the software components developed within the software/software integration scope is

performed. Subsequently, the software is loaded onto the target hardware within the software/hardware integration scope.

2.4.3 Software verification process

Verification is a crucial process implemented throughout the entire SLC. It starts in the planning phase and goes all the way through production release and even into maintenance. Verification is the technical evaluation of outputs generated during the planning and development processes. The primary objective of the verification process is to identify errors in the data developed up to that stage such as source code and requirements. The verification guidance in DO-178C includes a combination of reviews, analyses, and tests can be summarized as:

- Reviews
- Analyses
- Testing
- Verification of verification
- Problem reporting (PR)

Reviews and analyses assess the accuracy, completeness, and verifiability of the outputs of each of the life cycle phases including planning, requirements, design, code/integration, test development, and test execution. Testing involves the methodical evaluation of a system or its components to ascertain that they meet predefined requirements and to uncover any potential errors.

2.4.4 Software configuration management process

Software Configuration Management (SCM) is an integral process that spans the entire software life cycle, impacting all aspects and processes. Configuration management involves the identification, organization, and control of software modifications, aiming to enhance productivity by minimizing errors [43].

SCM extends beyond source code, contrary to common belief, encompassing all data throughout the software life cycle. This includes documentation used in software production, verification, and demonstrating compliance, all of which necessitate some form of configuration management. In essence, all software life cycle data outlined in DO-178C section 11 calls for SCM.

Effective SCM mitigates various issues such as missing source code modules, difficulty in locating the latest file version, recurrence of resolved errors, absent requirements, ambiguity regarding changes and their timing, conflicts arising from simultaneous file updates by multiple programmers, and numerous other challenges. It provides a means to manage change by:

- Identifying data items likely to change,
- Defining the relationship between the data items,
- Identifying the approach for controlling revisions of data
- Controlling changes implemented, and
- Reporting any changes made

To fulfill the DO-178C objectives which are provide to ensure integrity, accountability, reproducibility, visibility, coordination, and control of the software life cycle data as it evolves for the SCM process, several activities are necessary, as explained:

i. Configuration identification

It serves as the beginning for SCM activities, enables the identification of system components throughout their life cycle and trace between the SW and its life cycle data.

ii. Baselining and Traceability

In software development, a baseline refers to the software and its associated life cycle data at a particular moment on time. It functions as foundation for subsequent development. Once a baseline is set, any alterations should only occur through a formal change control process.

iii. Problem reporting

Problem reporting stands out as one of the most crucial SCM activities. PRs serve to pinpoint issues concerning baselined data, address non-compliances in processes, and rectify any anomalous behavior observed in the software.

iv. Change control and review

Change may occur to add or modify functionality, fix a bug, improve performance, update hardware and the supporting interfaces, improve processes, change the

environment at any phase of the software development process and is actually a fundamental characteristic of software.

v. Status accounting

Configuration status accounting encompasses recording the necessary information to reduce effort. Configuration status reports should be scheduled early in the project so as to properly capture the data and updated at the defined frequency as depicted in Table 2.6.

Table 2.6 : Configuration status report example.

Configuration item	Model Version	Issue date	Source code version	Configuration identification model
Plant design model	V1.0.0	07.03.2021	V1.0.0	Plant_DM_v100
Plant design model	V2.0.0	21.07.2022	V2.0.0	Plant_DM_v200
Plant design model	V3.0.0	20.02.2024	V3.0.0	Plant_DM_v300
Plant design model	V3.1.1	28.04.2024	V3.1.1	Plant_DM_v311
FLCS design model	V1.0.0	16.05.2021	V1.0.0	FLCS_DM_v100
FLCS design model	V2.0.0	10.05.2024	V2.0.0	FLCS_DM_v200

vi. Release

Once a data item is mature, it is released into the formal configuration management system (typically an enterprise-wide library like PLM system). DO-178C defines release as: “The act of formally making available and authorizing the use of a retrievable configuration item” [1].

vii. Load control, and environment control

Controlled software load process and environment data includes the methods, tools, procedures, programming languages, and hardware that will be utilized to develop, verify, control, and produce the software life cycle data and software product is controlled [34].

2.4.5 Software quality assurance process

The software quality assurance process purpose is to ensuring the software development processes are conducted following approved plans and standards. At the onset of a project, when the software development team is formed, it is imperative to appoint a software quality responsible individual. The software quality responsible individual must be organizationally independent from the development team in organizations involved in aviation software development.

In companies developing aircraft software, the software quality responsible individual has a unique task compared to other sectors, which is to conduct a "Compliance Review" activity correctly at the end of the software development phase. The purpose of the Compliance Review is to ensure that all planned software lifecycle processes have been completed, all software lifecycle data has been generated, executable object code is under configuration control, and the same code can be produced consistently each time. This review, conducted by the quality assurance responsible individual, serves as a form of audit.

2.4.6 Certification liaison process

Certification liaison involves continuous communication and coordination between the applicant company and the certification authority. It is an essential process for successful certification. The purpose of the certification process is to create an interaction between the software-developing company and the responsible from the authority of certification for issuing the type certificate for the aircraft in which the software is used. It involves sharing the documents designated as software compliance documents with the authority to ensure mutual agreement. The certification authority reviews the aircraft's software compliance documents and conducts detailed software audits during the planning, development, verification, and closure software lifecycle stages to examine the developed software and data. Based on these assessments, the certification authority decides whether to grant or deny flight clearance.

3. MODEL BASED DESIGN

3.1 Model Based Design Overview

Model-based design (MBD) is a model-centric method for designing systems that turning the authorized disciplines from traditional documentation procedures to digital models directed within a large information area. MBD necessitates the creation of a centralized model that encompasses all relevant components of system behavior-such as mathematical models, control algorithms, physical members. This approach integrates all stages of development, requirements generation, design and code generation, implementation to the HW, verification and validation, and dilation into a unified framework.

Model-based design (MBD) facilitates system or component level design and simulation, generation of an automatic source code, ongoing testing, and lastly verification processes. It tackles against the difficulties involved in designing intricate systems across diverse sectors such as aerospace and defense, automotive and robotics. MBD offers notable advantages, including enhanced engineering efficiency, cost savings, and shorter development schedules.

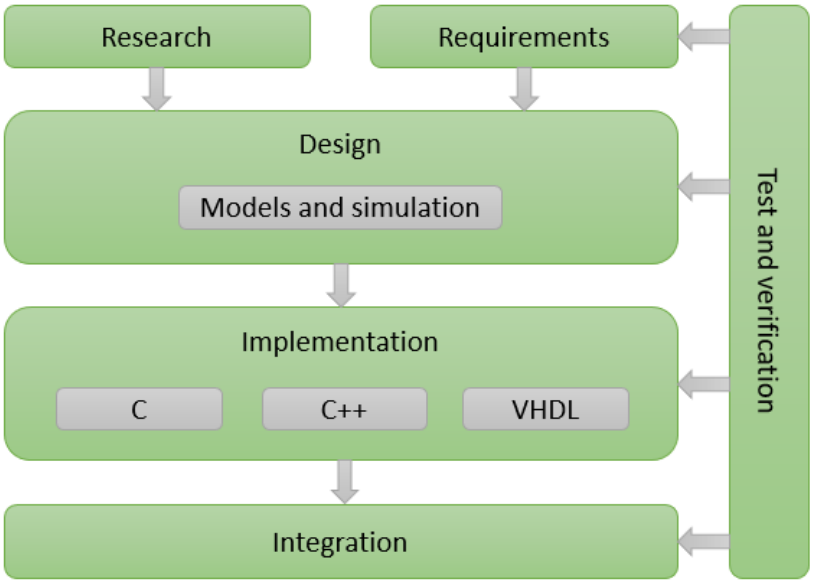


Figure 3.1 : Model-based design workflow [44].

3.1.1 History of MBD

Traditionally, the development of new products was divided into different systems, with tasks assigned to separate teams. For instance, in automotive design, one team focused on the suspension system development, another one developing ADAS, and yet another detailed on the braking systems. The culmination of these efforts happened at the final assembly stage, where issues often surfaced due to the lack of early integration and testing. As systems grew in complexity, this approach became increasingly impractical. Modern products are not merely standalone entities; they are interconnected systems of systems that must function harmoniously. This complexity led to the involvement of numerous teams and suppliers, with thousands of engineers collaborating across different organizational stages.

The major challenge with this decentralized approach was the intricate interdependencies among systems, subsystems, and components. Even minor changes could lead to significant and sometimes unpredictable impacts elsewhere in the design. Communication across such a vast and dispersed landscape became cumbersome, and the management of paper-based documentation became impractical. Model-based development (MBD) emerged as a solution to address these challenges in multi-team, multi-organizational communication. MBD shifts the focus from traditional documents to digital models, serving as the central artifact that represents every aspect of the system under development.

In MBD, models are intended to capture and communicate the system's requirements, design concepts, and implementation details. All engineering activities are conducted within a single, integrated environment, ensuring that changes made to the model propagate seamlessly to all team members simultaneously. Requirements are no longer written on paper but are instead captured as conceptual models, which provide a higher level of abstraction. These models enable shared understanding, facilitate discussions, revisions, and implementations, and guarantee that entire verification and validation activities are guided with respect to the requirements and carried out with complete traceability. [45]

Today, model-based development has been widely adopted across various industries, including technology companies and original equipment manufacturers (OEMs). MBD and model-based systems engineering (MBSE) practices involve sharing

requirements and design concepts across all project stakeholders, creating and simulating models in an integrated development environment, and generating code automatically or manually before deploying it to the target controller. These practices have significantly improved collaboration, reduced errors, and accelerated the development of complex systems.

3.1.2 Advantages of MBD

Most companies that adopt model-based design (MBD) experience a significant reduction in development time, typically ranging from 30 percent to 50 percent. These savings are primarily attributed to enhanced communication across different groups and the early detection of errors during the system design phase. By catching errors early, MBD minimizes the time and financial impact associated with making necessary changes.

In terms of time and cost efficiency alone, MBD surpasses the traditional waterfall development method. However, there are additional advantages that accrue to MBD practitioners. For instance, MBD promotes collaboration among diverse teams by providing a centralized platform where engineers can work concurrently on the same model. This collaborative environment helps to streamline decision-making processes and ensures that all team members have access to the latest design updates in real-time.

Moreover, MBD facilitates improved system integration by providing a holistic view of the entire system architecture. This enables engineers to identify and resolve potential conflicts or dependencies between different subsystems early in the design process. Additionally, MBD supports automated code generation from models, which helps to reduce manual coding errors and ensures consistency between the design model and the implemented system.

Another significant advantage of MBD is its ability to support iterative design and rapid prototyping. Engineers can easily iterate on design concepts and perform virtual simulations to validate system performance before committing to physical prototypes. This iterative approach accelerates the development cycle and allows for more robust and optimized designs.

Furthermore, MBD enhances traceability and compliance by maintaining end-to-end traceability of requirements, design decisions, and verification activities. This traceability is crucial for meeting regulatory standards and for auditing purposes,

guaranteeing the final product applies all specified requirements and these requirements thoroughly validated.

In conclusion, model-based design not only improves time and cost efficiency but also fosters collaboration, supports iterative design, enhances system integration, and ensures traceability and compliance. These advantages make MBD a superior approach to system design compared to traditional waterfall methods, particularly in industries requiring the development of complex and interconnected systems.

3.1.3 Challenges of MBD

Model-based design (MBD) has revolutionized from complex system development, offering significant advantages over traditional methods. However, several challenges remain for companies looking to adopt MBD. Firstly, there is a high upfront investment required. Companies must ensure that every team follows the MBD process for benefits to materialize, necessitating upfront expenditure on tooling and processes. Secondly, the increasing complexity of modern systems, such as autonomous vehicles, with numerous subsystems and components, poses a challenge. Integrating all these elements into a single model can be daunting. Thirdly, MBD tools often offer limited integration options and can be cumbersome to use, requiring substantial time to set up and maintain, which can result in high costs. Overcoming these challenges is crucial for organizations aiming to fully leverage MBD's benefits, including improved collaboration, reduced development time, and enhanced system performance.

3.1.4 Model-based systems engineering

Model-Based Systems Engineering (MBSE) covers all facets of model-based development concerning systems engineering, including all the subsets of product life cycle process. The process of systems engineering commences during the concept phase, where system requirements are defined and relationships between various elements are established. The primary objective is to guarantee that the requirements are achievable, appropriately restrict the design, and maintain consistency from the overall product down to the individual components and subsystems.

Model-Based Systems Engineering (MBSE) is a comprehensive approach which covers all essential activities for planning, developing, designing, and deploying complex systems. MBSE's primary advantage lies in its ability to establish a clear

organizational structure and a centrally-shared model that ensures uniform expression and interpretation of all subsystems, units, and information. This contrasts with traditional document-based approaches, which are prone to misinterpretation and inconsistencies across teams. Without MBSE, managing such complexities would be costly, time-consuming, and challenging for individuals without a holistic view of the entire program.

Effectively implemented MBSE offers several key benefits to organizations:

- The end system can be visualized without the need for full construction. It allows for understanding of how all components fit together and anticipation of potential issues at the beginning of the development process.
- Improved communication along the development lifecycle is facilitated. Improved communication results in greater alignment, shared understanding, realistic expectations, resulting in higher-quality products.
- Enhanced coordination across multiple teams and tasks is achieved. Specifications can be standardized and maintained by systems engineers, progress can be effectively tracked by project managers, and success can be measured against requirements by product teams.
- Insight into the progress of different parts of the system is gained without requiring deep technical knowledge of every component's complexity by engineering leaders.

In summary, MBSE enables organizations to manage complexity more effectively, improve communication and coordination among teams, and gain clearer insight into system development progress. This approach helps streamline development processes, enhance product quality, and ultimately deliver more successful outcomes.

3.2 Model-Based Development Process

Model-based development (MBD) is organized around the V-model development lifecycle, which is an advancement of the waterfall system model. Each stage is closely integrated with testing, enhancing its role in verification and validation through iterative testing processes in model-based development. Figure 3.2 shows the main stages of the V Model [45].

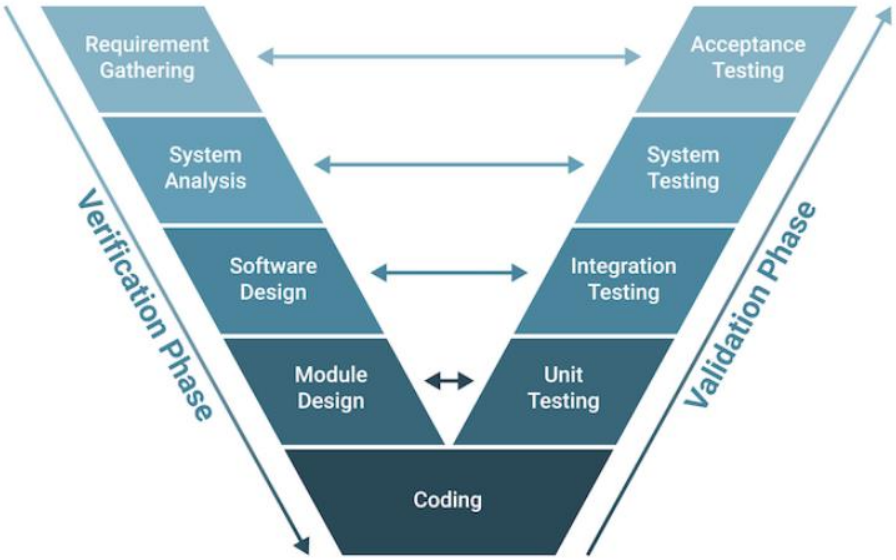


Figure 3.2 : V-model.

The process typically commences with product requirements positioned in the upper left quadrant, followed by the establishment of system architecture and subsequent system design. From there, the process advances to system implementation. The process then reverses course, progressing towards the upper right quadrant, where verification and validation activities take place, ultimately culminating in the system's production phase. The essence lies not only in the series of cases but also in the methodology which uses information environment data that interconnects all system elements and advancing development through ongoing testing procedures.

3.2.1 Essential elements of model-based design

3.2.1.1 Product concept

The product concept, a primary focus during the concept and idea of the product phase, entails a comprehensive product description under development. This concept delineates the customer issues to be addressed, articulates the perception for the product that is designed and provides everything in detail from its objectives to the scope of its applications.

3.2.1.2 System requirements

In Chapter 2.2.1.2, System Requirement Document (SRD) define the analysis of the system requirements, in addition these requirements provide a comprehensive outline and statements of all essential fundamentals necessary for the system’ distribution such

as it outlines the necessary hardware components and specifies the software environments required for the system to operate effectively.

In model-based development, requirements are maintained in a directory where individual requirements are derived immediate from the product specifications. These requirements cascade down from the level of product to level of component and these requirements must be sufficiently detailed to align with the followed approaches. It is vital to clarify which specific requirements affect which element or vice versa which elements are linked to which requirement [45].

Since the process of product development advances, the collective system understanding deepens, allowing for iterative refinement of system requirements. However, it is crucial to consistently maintain and update all connections to ensure alignment across development teams.

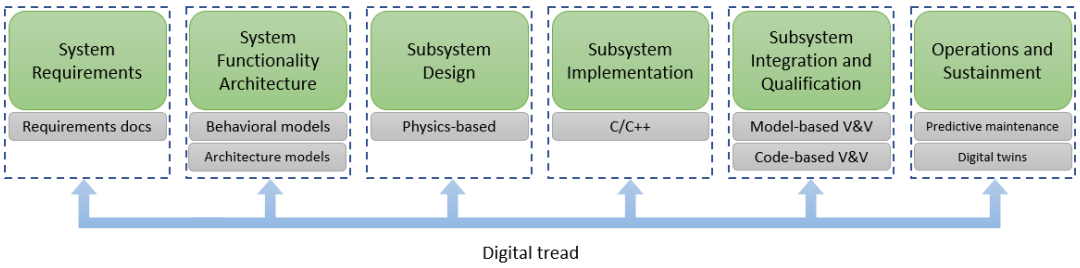


Figure 3.3 : Systematic use of models throughout the MBD process [44].

3.2.1.3 System architecture

The system architecture provides a comprehensive functional bounding of both HW and SW elements, alongside the structure of subsystems in MBD. This can be depicted hierarchically, event-driven, or in layers, illustrating the flow of interactions inside the system. System architecture utilizes modeling languages which support clarity and precision throughout the development of systems.

In model-based development, a well-defined system architecture provides seamless integration of systems and its components in order to achieve desired performance. This approach helps avoid time lost in managing interfaces with other teams.

3.2.1.4 System design

In model-based development (MBD), system design covers a process from proposed idea that initializes the development to the smallest unit of a proposed system. This

process focuses on the performance aspects that provides a comprehensive system overview with an abstraction to evaluate critical risks. During this phase, system model representations and physical simulation tools are utilized.

Although guidelines exist, there are no rigid procedures for designing a system using model-based tools in general. Preferences vary by organization but some effective methods contribute to excellent system design process which are interaction, continuous improvement, frequent documentation, and simplicity.

3.2.1.5 Model simulation and implementation

During this simulation and implementation, simplified models which comprising tens or hundreds of blocks that represent the system's functions. Designers progressively develop more intricate models through testing and redesign to closely simulate the system during the operational conditions, also using the detailed models can be used while evaluating and optimizing the design. This entails identifying the best design options, determining the most crucial variables for constructing the final product, and balancing trade-offs. During model simulations, modifications are done based on the results of evaluations are conducted simulations to assess the results until the satisfactory level.

Optimizing design and analyzing uncertainties are effectively conducted using High-Performance Computing (HPC) with the help of technology. This allows engineers to execute a design problem in numerous times, identifying the optimal solutions that enhance the system performance. Leading companies utilize these applications in their simulations in order to discover unforeseen issues [2].

3.2.1.6 Auto code generation and automation

Automatic code generation is an essential step in the effective implementation of Model-Based Design, as it equips users with advanced verification and validation methods to generate high-quality code. This approach reduces the iterations in a standard industry product development cycle and eliminates errors that can arise from manual coding. By automatically translating models into executable code, automatic code generation ensures consistency between the design model and the implementation, leading to more reliable and efficient software development [46].

Embedded software for embedded systems is typically developed using programming languages suited for such purposes. C is often the preferred choice due to its processor independence, portability, speed, and direct access to memory management. Instead of manually coding in C or other languages, many companies opt for automatic code generation tools in order to transform their models into executable code. The primary advantage of automatic code generation is the ability to automatically update and deploy code to hardware whenever there are changes to the model, irrespective of the system or algorithm complexity. These tools help ensure consistency between the design model and the implementation, reduce the likelihood of errors, and accelerate the development cycle. Automatic code generation tools also enable engineers to focus more on system design and verification rather than low-level coding tasks.

3.2.1.7 Verification & validation

Verification and validation are essential steps in software development. Verification ensures generated code operates correctly inside of the intended platform, while validation confirms whether the actual system performance meets the specified requirements. Together, these processes cover all procedures, including real-world tests and direct applications, to make sure that the related software is designed and developed in line with defined requirements [36].

Comprehensive requirements review initiate verification and validation. This stage checks the individual requirements are accompanied by a verification plan which outlining how it will be tested to ensure it has been fulfilled and meets its intended purpose. This verification ensures that the software development complies with the requirements specified and that the final product achieves its intended objectives.

Loop-based model testing (x-in-the-loop) is a crucial part which covers conducting pass/fail checks on various system components to confirm they meet requirements and are functionally safe. MBD testing typically utilizes two main types of models: the control model and the plant model. These models are integrated and tested together to evaluate system behavior and performance. [45]

i. Model-in-the-Loop (MIL)

It is an initial phase of the V&V activities and aims to create a structure between controller model and the plant model. It allows to forecast the attitude of the control algorithm and real conditions. The test procedure is performed at the beginning while

the model. Later, the model is enhanced with details and executed with more fidelity to achieve precise representation rather than an approximation.

Highly uncertain and risky elements of the system should be given priority during MIL testing, aiming to operate them at the highest fidelity level possible. This approach targets the early resolution of edge cases within the overall system during the development cycle, when the cost of making changes is lower. Once results from the MIL simulation is met defined system requirements, the development process advances to the next step

ii. Software-in-the-loop (SIL)

SIL test are conducted with after the code generation based on the plant or control model. The primary aim is identifying the differences between source code and related source of it.

During SIL testing, source code is executed on the operating system which can be Windows or Linux based in order to assess the actual model outputs at the same time. If discrepancies occur between the results from SIL and MIL, it indicates there may be errors in the generated code that comes from model or model may has problems that require review and resolution.

After adjusting and confirming alignment between two sides, the development process guarantee that the software meets the defined requirements and functions correctly in simulated environments

iii. Processor-in-the-loop (PIL)

PIL testing is conducted when the executable object code is generated from source code, and this code embedded into the hardware which has a specific processor that stimulates this code.

Evaluation is done with the hardware outputs are compared with the outputs of the plant while it is executing the control logic and performing all the required tasks. If any discrepancies occur between the results, it indicates potential issues with the model, code, or processor that require review and adjustment.

Once adjustments are done and the results from previous testing align, demonstrating consistent behavior across all stages, the development process can proceed to the next

stage. This iterative testing process helps guarantee that developed software functions on the aimed hardware and meets the specified requirements.

iv. Hardware-in-the-loop (HIL)

HIL test is the final phase presence of the integration of the system being developed. During HIL testing, the generated code runs on a real-time system and interacts with the plant model.

Throughout HIL testing, the outcomes are checked and evaluated with those from PIL test results. Discrepancies indicate the model dependent issues, disordered source code incorrect identification of the processor, network protocol errors, necessitating further review and adjustment.

Once adjustments are made and the results from previous testing converge, demonstrating consistent behavior across all stages, the development process can proceed to production. This rigorous testing approach ensures that the software meets the required specifications and performs reliably in real-world conditions

v. Production

During production, there is usually tests are conducted in an actual environment with a human setting. The objective of production is assessing the complete functionality of the system that driven by a human which being developed and prepared to intervene if a safety issue arises. At this phase, companies evaluate how well the model performs compared to the actual product in real-world conditions.

If improvements are needed for the plant model, these adjustments are fulfilled using the Model-Based Design. If the updates are required for the controller, the model will be refreshed, simulations will be run, and verification and validation will be conducted using all historical information.

3.2.1.8 Knowledge capture and management

Models act as the primary repository of project information in Model-Based Design. System design specifications which belong to the system being development are encapsulated within the models, as well as product knowledge, the design expertise of team members, past experiences, and best practices in design.

The models act as a universal medium for sharing information among individuals. As these models can is used along these individuals, their content grows alongside the system's understanding. MBD improves communication, protects rights about the product, promotes a knowledge sharing culture, and reinforces project interactions.

3.3 DO-331 Model Based Development and Verification Supplement

DO-331 supplement offers comprehensive direction on the implementation of model-based development and verification techniques for the software of aviation related systems and equipment. It is directly related to DO-178C, which is the essential standard for software development and certification in the aerospace industry. DO-178C sets the guidelines for the certification of software used in aviation to ensure safe and reliable operation.

The DO-331 supplement extends DO-178C by specifically addressing how model-based methods can be incorporated into the software development lifecycle. DO-331 details the use of models to specify, design, implement, and verify software requirements, ensuring that they comply with DO-178C objectives. This includes processes for model-based requirements capture, architectural design, verification, and validation.

In terms of certification, DO-331 provides requirements and objectives for demonstrating that model-based methods meet the safety and reliability requirements stipulated in DO-178C. It outlines specific activities and artifacts necessary to ensure that the development and verification using MBD processes are rigorous as well as meet the safety and performance criteria required for certification.

Certification under DO-331 involves showing traceability from the models to the final software implementation, ensuring that models accurately represent the software requirements and design. It requires a comprehensive verification and validation process, including simulation and testing of the models, to ensure that they accurately represent the behavior of the software in the actual airborne system. The model serves as a reflection of the actual software code or a process section of the verification while including the requirements and/or the defining of the software architecture, allowing for direct analysis of software behavior as outlined in DO-331 [47].

Overall, DO-331 ensures that model-based development and verification techniques are applied in a manner consistent with the goals of DO-178C, thereby enabling software developers to leverage advanced modeling methods while meeting the stringent safety and certification standards required in the aerospace industry.

4. AIR VEHICLE MATHEMATICAL SIMULATION MODEL

Air-vehicle flight using six degrees of freedom (6-DoF) involves a set of three translational movements and three rotational movements along and around its reference frame. In computational implementations aimed at analyzing the equations of motion for air vehicles, the reference frame commonly employed is the coordinate system aligned with the body-fixed reference frame such that all forces and moments exerted on the air vehicle are converted into the body coordinates. Although it is typical to employ the body fixed reference frame as the primary frame of reference for denotating vector components in body coordinate system during computational modeling, this study does not restrict itself solely to the body reference frame. The wind frame may also be selected for analyses based on aircraft flight path and nonlinear controller design.

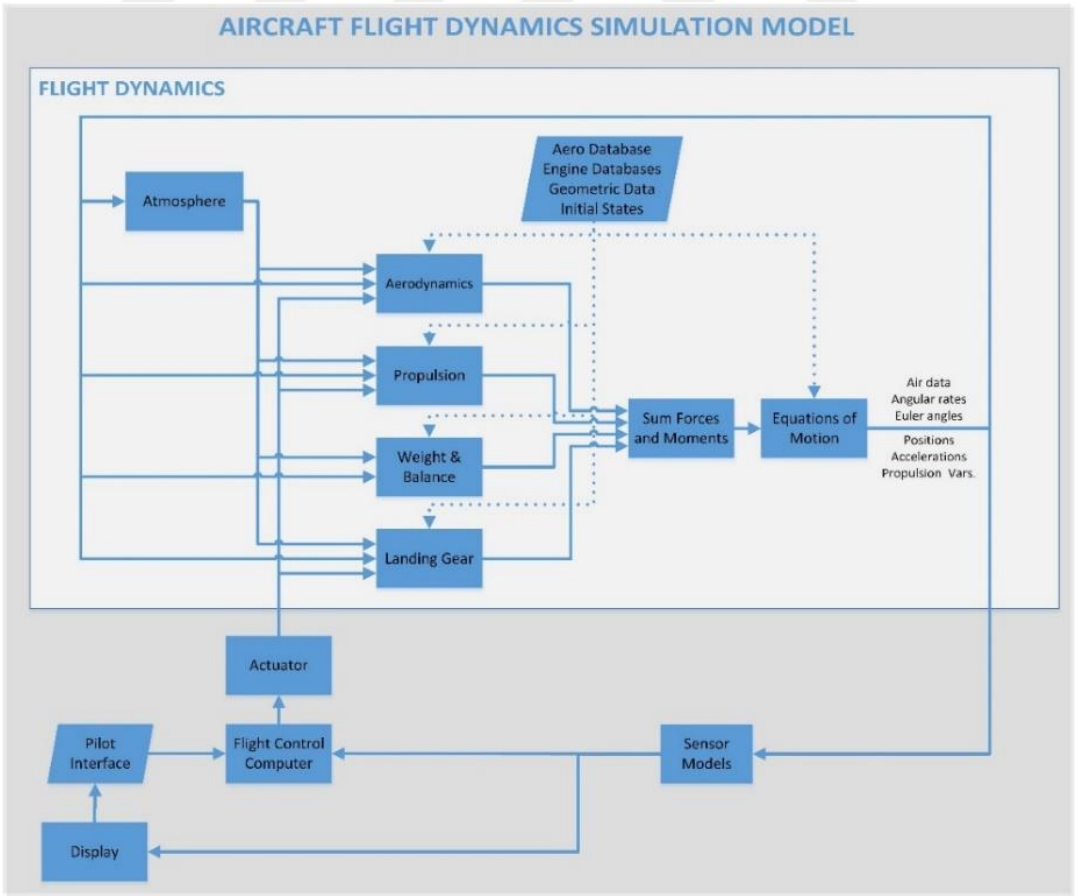


Figure 4.1 : Air vehicle simulation model components [48].

In this study, the wind frame serves as the reference frame for the equations of motion. In conclusion, equations of motion are derived, incorporating both translational and rotational motions along with kinematic relationships, which depend on variable states. This process aims to establish comprehensive state equations for the air vehicle. The terms mentioned above will be further elaborated upon in subsequent chapters [22].

Air vehicle flight simulation involves the integration of numerous subsystem models such as equations of motion, aerodynamics, as well as flight control system, actuators, and sensor models [49]. While this list is not exhaustive, it encompasses critical components. A visual representation of this process is provided in the Figure 4.1.

4.1 Air Vehicle Flight Dynamics

Points and frames are mathematical abstracts that hold physical significance, where;

- *A point*, describes the location of a particle (an air vehicle center-of-mass),
- *A frame*, consisting of at least three noncollinear and mutually-fixed points

represents the model of a physical object in three-dimensional space.

The modeling of any problem in flight dynamics can be formulated independently of coordinate systems, which have no physical significance, using the two physical concepts of points and frames. This is mathematically described by tensors in their simple Cartesian form. In this way, all physical laws and quantities are represented in tensor form. These tensor equations are then expressed in coordinate-dependent component form resolved in a particular coordinate system, arranged as matrices, for computational implementation and numerical analysis. This necessitates a clear distinction between frames and coordinate systems [22].

4.1.1 Assumptions, notations, and conventions

Air vehicle equations of motion (EoM) can be formulated using Newton's Second Law for translational motion and Euler's equations for rotational motion. For conventional air vehicles, derivation of EoM is carried out under the following *assumptions*:

1. The influence of structural deflections, such as aeroelasticity, and the control surfaces' relative motion dynamics on the overall air vehicle dynamics are

neglected. Hence, the dynamics of the air vehicle are determined exclusively by the equations of motion governing a single rigid body.

2. The stationary and flat-Earth approximation is applicable to in-atmosphere scenarios, disregarding the Earth's rotation. This is achieved by unwrapping the curved surface of the Earth into a flat plane tangential to the initial point
3. The air vehicle center of mass (CM) coincides with the center of gravity (CG) under the assumption that gravity acts through the center of mass of the air vehicle.

In flight dynamics, a *tensor* is a mathematical object that represents physical quantities like forces, moments, velocities, and accelerations which are often described in terms of their components along different axes in three-dimensional space.

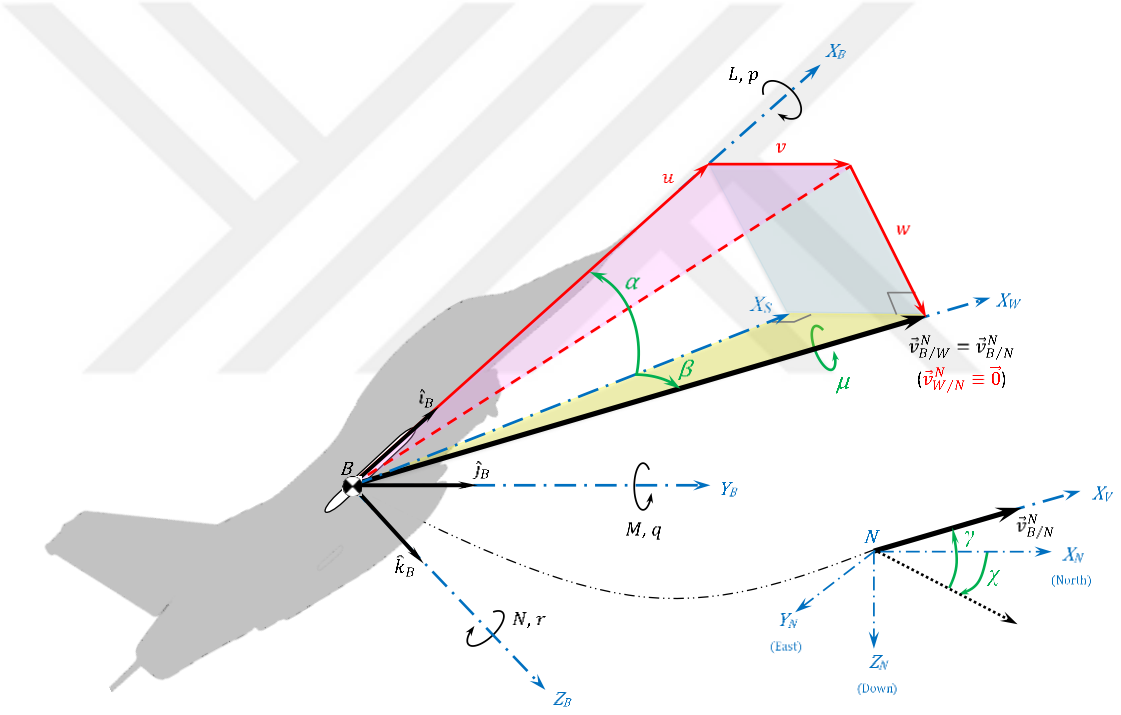


Figure 4.2 : Air vehicle frames and coordinate systems [50].

The rigid body, six degrees of freedom equations of motion typically comprises set of differential equations which describe the motion of the air vehicle in terms of its translational and rotational degrees of freedom. These equations can be derived using principles of classical mechanics, such as Newton's laws of motion and Euler's equations of motion [51].

Tensors play a crucial role in this derivation process by providing provide robust mathematical framework for analyzing the dynamics of aircraft motion and are

essential for developing accurate models of flight dynamics in a compact and systematically.

In addition, a *frame* with *origin A* and triad of orthonormal base vectors $\hat{\imath}_A, \hat{\jmath}_A, \hat{k}_A$ is represented as $\mathcal{F}_A\{A; \hat{\imath}_A, \hat{\jmath}_A, \hat{k}_A\}$ or, in short, $\mathcal{F}_A$.

4.1.2 Reference frames and coordinate systems

It is essential to introduce five frames in order to adopt a suitable notation system and represent a rigid-body air vehicle motion. These frames are explained in Figure 4.2 are explained in following chapters [50] [23].

- *Local Geodetic North-East-Down* frame, $\mathcal{F}_N\{N; x_N y_N z_N\}$, considered as Earth frame.
- *Vehicle body-fixed frame*, $\mathcal{F}_B\{B; x_B y_B z_B\}$, related with the CM of the air vehicle.
- *Wind frame*, $\mathcal{F}_W\{W; x_W y_W z_W\}$, velocity vector related frame
- *Velocity frame*, $\mathcal{F}_V\{V; x_V y_V z_V\}$, velocity vector-around related frame
- *Propulsion frame*, $\mathcal{F}_P\{P; x_P y_P z_P\}$, thrust frame.

4.1.3 Kinematic relations

Coordinate systems associated with the frames can be expressed using orthogonal transformations, which will be explained further within this chapter. These transformations are indicated in Figure 4.3.

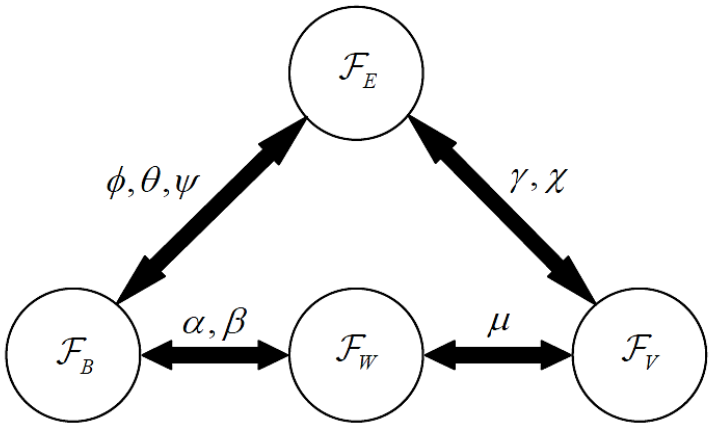


Figure 4.3 : Orientation angles [23].

Referring to Figure 4.3, instantaneous angular orientations of the frames (or their preferred coordinate systems) are described by the following kinematic relationships

specifying the angle and unit base vector of rotation between successive rotations as depicted in Table 4.1.

Table 4.1 : Kinematic relationships of transformations.

Transformation Direction	Kinematic Relationships
NED to Vehicle Body	$N \xrightarrow[\psi]{\hat{k}_N=\hat{k}_{N_1}} N_1 \xrightarrow[\theta]{\hat{\jmath}_{N_1}=\hat{\jmath}_{N_2}} N_2 \xrightarrow[\phi]{\hat{\imath}_{N_2}=\hat{\imath}_B} B$
Vehicle Body to Wind	$W \xrightarrow[-\beta_S]{\hat{k}_W=\hat{k}_S} S \xrightarrow[\alpha]{\hat{\jmath}_S=\hat{\jmath}_B} B$
Wind to Velocity	$V \xrightarrow[\mu]{\hat{\imath}_V=\hat{\imath}_W} W$

4.1.3.1 Coordinate transformations

i. Vehicle Body to Earth Transformation

The transformation of the body fixed frame to the earth fixed frame is performed with a 3-2-1 Euler sequence rotating. The consecutive rotations around z_B, y_B and x_B are introduced as Euler roll, pitch and yaw angles (ϕ, θ, ψ) respectively, and the rotations convert from the Earth-fixed frame, through intermediate frames, to the body frame $\mathcal{F}_B$, respectively. Thus, this transformation can be can be expressed in the matrix form as:

$$\breve{T}^{(N,B)} = \breve{T}^{(N,N_1)}\breve{T}^{(N_1,N_2)}\breve{T}^{(N_2,B)} \tag{4.1}$$

$$\breve{T}^{(N,B)} = \begin{bmatrix} c_\psi & -s_\psi & 0 \\ s_\psi & c_\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} c_\theta & 0 & s_\theta \\ 0 & 1 & 0 \\ -s_\theta & 0 & c_\theta \end{bmatrix}\begin{bmatrix} 1 & 0 & 0 \\ 0 & c_\phi & -s_\phi \\ 0 & s_\phi & c_\phi \end{bmatrix} \tag{4.2}$$

$$\breve{T}^{(N,B)} = \begin{bmatrix} c_\psi c_\theta & -s_\psi c_\phi + c_\psi s_\theta s_\phi & s_\psi s_\phi + c_\psi s_\theta c_\phi \\ s_\psi c_\theta & c_\psi c_\phi + s_\psi s_\theta s_\phi & -c_\psi s_\phi + s_\psi s_\theta c_\phi \\ -s_\theta & c_\theta s_\phi & c_\theta c_\phi \end{bmatrix} \tag{4.3}$$

ii. Vehicle Body to Wind Axis Transformation

The orientation of body coordinates with wind coordinates is determined by the angle of attack α and angle of sideslip β. The transformation matrix from body-fixed frame to wind-axes frame, the transformation matrices are concatenated with matrix-multiplication from body-fixed to stability frame and stability to wind axes frame, is represented as:

$$\breve{T}^{(W,B)} = \breve{T}^{(W,S)}\breve{T}^{(S,B)}$$

$$\breve{T}^{(W,B)} = \begin{bmatrix} \cos\alpha & 0 & \sin\alpha \\ 0 & 1 & 0 \\ \sin\alpha & 0 & \cos\alpha \end{bmatrix}\begin{bmatrix} \cos\beta & \sin\beta & 0 \\ -\sin\beta & \cos\beta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.4}$$

$$\breve{T}^{(W,B)} = \begin{bmatrix} \cos\alpha\cos\beta & \sin\beta & \sin\alpha\cos\beta \\ -\cos\alpha\sin\beta & \cos\beta & -\sin\alpha\sin\beta \\ -\sin\alpha & 0 & \cos\alpha \end{bmatrix}$$

Notations $s_{(\cdot)}$ and $c_{(\cdot)}$ are the shorthand notations of trigonometric functions $\sin(\cdot)$ and $\cos(\cdot)$, then,

$$\breve{T}^{(W,B)} = \begin{bmatrix} c_\alpha c_\beta & s_\beta & s_\alpha c_\beta \\ -c_\alpha s_\beta & c_\beta & -s_\alpha s_\beta \\ -s_\alpha & 0 & c_\alpha \end{bmatrix} \tag{4.5}$$

iii. Wind to Velocity Axis Transformation

The bank angle μ relates the orientation of wind coordinates relative to velocity coordinates. The transformation matrix from wind frame with respect to velocity frame is represented as

$$\breve{T}^{(V,W)} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_\mu & -s_\mu \\ 0 & s_\mu & c_\mu \end{bmatrix} \tag{4.6}$$

iv. Velocity to Earth Axis Transformation

The orientation of velocity coordinates with respect to Earth coordinates is decided by the heading angle χ and the vertical flight path angle γ. The transformation matrix of velocity coordinates relative to the Earth coordinates, giving

$$\breve{T}^{(N,V)} = \breve{T}^{(N,X)}\breve{T}^{(X,V)} \tag{4.7}$$

$$\breve{T}^{(N,V)} = \begin{bmatrix} c_\chi & -s_\chi & 0 \\ s_\chi & c_\chi & 0 \\ 0 & 0 & 1 \end{bmatrix}\begin{bmatrix} c_\gamma & 0 & s_\gamma \\ 0 & 1 & 0 \\ -s_\gamma & 0 & c_\gamma \end{bmatrix}$$

$$\breve{T}^{(N,V)} = \begin{bmatrix} c_\chi c_\gamma & -s_\chi & c_\chi s_\gamma \\ s_\chi c_\gamma & c_\chi & s_\chi s_\gamma \\ -s_\gamma & 0 & c_\gamma \end{bmatrix} \tag{4.8}$$

4.1.3.2 Translational rates

The kinematic expression describing translational rates is as follows:

$$\left\{\vec{v}_{B/N}^{N}\right\}^{(N)} = \breve{T}^{(N,B)}\left\{\vec{v}_{B/N}^{N}\right\}^{(B)} \tag{4.9}$$

expanding (4.9) in component form as following

$$\begin{bmatrix}\dot{p}_N\\ \dot{p}_E\\ \dot{p}_D\end{bmatrix} = \begin{bmatrix}\mathrm{c}_\theta\mathrm{c}_\psi & -\mathrm{c}_\phi\mathrm{s}_\psi + \mathrm{s}_\phi\mathrm{s}_\theta\mathrm{c}_\psi & \mathrm{s}_\phi\mathrm{s}_\psi + \mathrm{c}_\phi\mathrm{s}_\theta\mathrm{c}_\psi\\ \mathrm{c}_\theta\mathrm{s}_\psi & \mathrm{c}_\phi\mathrm{c}_\psi + \mathrm{s}_\phi\mathrm{s}_\theta\mathrm{s}_\psi & -\mathrm{s}_\phi\mathrm{c}_\psi + \mathrm{c}_\phi\mathrm{s}_\theta\mathrm{s}_\psi\\ -\mathrm{s}_\theta & \mathrm{s}_\phi\mathrm{c}_\theta & \mathrm{c}_\phi\mathrm{c}_\theta\end{bmatrix}\begin{bmatrix}u\\ v\\ w\end{bmatrix} \tag{4.10}$$

Taking $\dot{p}_D$ the component in (4.17), altitude rate is calculated as:

$$\begin{aligned}\dot{p}_D &= -\mathrm{s}_\theta u + \mathrm{s}_\phi\mathrm{c}_\theta v + \mathrm{c}_\phi\mathrm{c}_\theta w\\ \dot{h} = -\dot{p}_D &= +\mathrm{s}_\theta u - \mathrm{s}_\phi\mathrm{c}_\theta v - \mathrm{c}_\phi\mathrm{c}_\theta w\\ \dot{h} &= +\mathrm{s}_\theta u - \mathrm{s}_\phi\mathrm{c}_\theta v - \mathrm{c}_\phi\mathrm{c}_\theta w\end{aligned} \tag{4.11}$$

4.1.3.3 Angular velocities and angular rates

Angular velocity vectors, which connect one frame to another, are derived from the vector addition of angular rates multiplied by their respective unit vectors. The following equations between angular rates of the body and Euler angles are then derived using the previous kinematic relationships can be found as:

$$\left\{\vec{\omega}^{B/N}\right\}^{(B)} = \dot{\phi}\{\hat{\imath}_B\}^{(B)} + \dot{\theta}\breve{T}^{(B,N_2)}\left\{\hat{\jmath}_{N_2}\right\}^{(N_2)} + \dot{\psi}\breve{T}^{(B,N_2)}\breve{T}^{(N_2,N_1)}\left\{\hat{k}_{N_1}\right\}^{(N_1)} \tag{4.12}$$

$$\left\{\vec{\omega}^{B/N}\right\}^{(B)} = \begin{bmatrix}p\\ q\\ r\end{bmatrix} = \begin{bmatrix}1 & 0 & -\mathrm{s}_\theta\\ 0 & \mathrm{c}_\phi & \mathrm{s}_\phi\mathrm{c}_\theta\\ 0 & -\mathrm{s}_\phi & \mathrm{c}_\phi\mathrm{c}_\theta\end{bmatrix}\begin{bmatrix}\dot{\phi}\\ \dot{\theta}\\ \dot{\psi}\end{bmatrix} \tag{4.13}$$

These expressions govern the rates of Euler angles by taking the inverse as:

$$\begin{bmatrix}\dot{\phi}\\ \dot{\theta}\\ \dot{\psi}\end{bmatrix} = \begin{bmatrix}1 & \mathrm{s}_\phi\mathrm{t}_\theta & \mathrm{c}_\phi\mathrm{t}_\theta\\ 0 & \mathrm{c}_\phi & -\mathrm{s}_\phi\\ 0 & \mathrm{s}_\phi/\mathrm{c}_\theta & \mathrm{c}_\phi/\mathrm{c}_\theta\end{bmatrix}\begin{bmatrix}p\\ q\\ r\end{bmatrix} \tag{4.14}$$

4.1.3.4 Incidence angles

The velocity vector of the air vehicle from relative wind coordinates to body coordinates is derived from

$$\left\{\vec{v}_{B/N}^{N}\right\}^{(B)} = \breve{T}^{(W,B)}\left\{\vec{v}_{B/N}^{E}\right\}^{(W)} \tag{4.15}$$

$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} \mathrm{c}_\alpha \mathrm{c}_\beta & \mathrm{s}_\beta & \mathrm{s}_\alpha \mathrm{c}_\beta \\ -\mathrm{c}_\alpha \mathrm{s}_\beta & \mathrm{c}_\beta & -\mathrm{s}_\alpha \mathrm{s}_\beta \\ -\mathrm{s}_\alpha & 0 & \mathrm{c}_\alpha \end{bmatrix}^T \begin{bmatrix} V_T \\ 0 \\ 0 \end{bmatrix} \tag{4.16}$$

where V_T represents the airspeed given by the magnitude of the air vehicle's velocity vector such that

$$V_T = \sqrt{u^2 + v^2 + w^2} \tag{4.17}$$

Using (4.16), the components of the total velocity vector in body coordinates are expressed using α and β.

$$\begin{aligned} u &= V_T \mathrm{c}_\alpha \mathrm{c}_\beta \\ v &= V_T \mathrm{s}_\beta \\ w &= V_T \mathrm{s}_\alpha \mathrm{c}_\beta \end{aligned} \tag{4.18}$$

Incidence Angles with Presence of Wind

The relations for Cartesian wind angles in the presence of wind are derived from

$$\left[\breve{T}^{(W,B)}\right]^T \left\{\vec{v}_{B/W}^{E}\right\}^{(W)} = \left\{\vec{v}_{B/N}^{N}\right\}^{(B)} - \left\{\vec{v}_{W/N}^{N}\right\}^{(B)} \tag{4.19}$$

Expanding (4.19) into its components as following

$$\begin{bmatrix} \mathrm{c}_\alpha \mathrm{c}_\beta & \mathrm{s}_\beta & \mathrm{s}_\alpha \mathrm{c}_\beta \\ -\mathrm{c}_\alpha \mathrm{s}_\beta & \mathrm{c}_\beta & -\mathrm{s}_\alpha \mathrm{s}_\beta \\ -\mathrm{s}_\alpha & 0 & \mathrm{c}_\alpha \end{bmatrix}^T \begin{bmatrix} V_{TAS} \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} u \\ v \\ w \end{bmatrix} - \begin{bmatrix} u_W \\ v_W \\ w_W \end{bmatrix} \tag{4.20}$$

Resulting for the angle of attack α and angle of sideslip β in the following trigonometric relationships, respectively:

$$\alpha = \arctan_2\left(\frac{w - w_w}{u - u_w}\right), \qquad -\pi < \alpha \leq \pi \tag{4.21}$$

$$\beta = \arcsin\left(\frac{v - v_W}{V_{TAS}}\right) = \arctan\left(\frac{v - v_W}{\sqrt{(u - u_W)^2 + (w - w_W)^2}}\right), -\frac{\pi}{2} < \beta < \frac{\pi}{2} \tag{4.22}$$

4.1.3.5 Flight path angles

Flight path angle (γ) and heading angle (χ) are formulated using the trigonometric relations and components of $\{\vec{v}_{B/N}^{N}\}^{(N)}$ as following

$$\gamma = \arcsin\left(\frac{\dot{h}}{\left|\vec{v}_{B/N}^{N}\right|}\right) = \arctan\left(\frac{\dot{h}}{\sqrt{\dot{p}_N^2 + \dot{p}_E^2}}\right), \qquad -\frac{\pi}{2} < \gamma < \frac{\pi}{2} \tag{4.23}$$

$$\chi = \arctan_2[\ \dot{p}_E \quad , \quad \dot{p}_N\ ], \qquad -\pi < \chi \leq \pi \tag{4.24}$$

4.1.4 Dynamic relations

Dynamic relations which consists of the three translational degrees of freedom that are governed by Newton's second law and the three rotational degrees of freedom that are governed by Euler's law are detailed following subchapters.

4.1.4.1 Translational motion

Newton's second law governs the three translational degrees of freedom. Considering the flat Earth assumption in 4.1.1, the air vehicle translational equations of motion over a nonrotating and flat Earth can be expressed in tensor form as

$$\frac{1}{m}\sum \vec{f}_B = \vec{a}_{B/N}^{N} = \mathcal{D}^N \vec{v}_{B/N}^{N} = \mathcal{D}^B \vec{v}_{B/N}^{N} + \vec{\omega}^{B/N} \times \vec{v}_{B/N}^{N} \tag{4.25}$$

demonstrating the vector components in body coordinates are represented

$$\{\mathcal{D}^B \vec{v}_{B/N}^{N}\}^{(B)} = \begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix}, \qquad \{\vec{f}\}^{(B)} = \begin{bmatrix} f_x \\ f_y \\ f_z \end{bmatrix}, \tag{4.26}$$

$$\left\{\vec{v}_{B/N}^{N}\right\}^{(B)} = \begin{bmatrix} u \\ v \\ w \end{bmatrix}, \qquad \left\{\vec{\omega}^{B/N}\right\}^{(B)} = \begin{bmatrix} p \\ q \\ r \end{bmatrix}$$

with the coordinate-dependent component form of translational motion equations in body coordinates and the body-axes accelerations yield in the scalar equations are expressed

$$\begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix} = \frac{1}{m}\begin{bmatrix} f_x \\ f_y \\ f_z \end{bmatrix} - \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times \begin{bmatrix} u \\ v \\ w \end{bmatrix}$$

$$\begin{aligned} \dot{u} &= f_x/m - qw + rv \\ \dot{v} &= f_y/m - ru + pw \\ \dot{w} &= f_z/m - pv + qu \end{aligned} \tag{4.27}$$

resultant force expression when the landing gear contact forces are excluded, can be defined as follows

$$\vec{f} = \vec{f}_a + \vec{f}_g + \vec{f}_p \tag{4.28}$$

where $\vec{f}_a, \vec{f}_p$ and $\vec{f}_g$ are the resultant forces due to aerodynamics, propulsion, and thrust, respectively.

The three first-order differential equations in (4.27) provides the translational motions of an air vehicle in 3-DoF dynamics expressed in the body frame with respect to the Earth. These equations are nonlinear and coupled through the body angular velocities $\left\{\vec{\omega}^{B/N}\right\}^{(B)} = [p \quad q \quad r]^T$. The nonlinearity arises from the incidence angles involved in the aerodynamic force and moment calculations [50].

4.1.4.2 Rotational motion

The three rotational degrees of freedom are governed by Euler's law. The rotational equations of motion of an air vehicle with respect to a nonrotating and flat Earth in 4.1.1 are expressed by the vector differential equation.

$$\sum \vec{m}_B = \mathcal{D}^N \vec{l}_B^{B/N} = \mathcal{D}^B \vec{l}_B^{B/N} + \vec{\omega}^{B/N} \times \vec{l}_B^{B/N}, \qquad \vec{l}_B^{B/N} = \breve{I}_B^B \vec{\omega}^{B/N} \tag{4.29}$$

$$\sum \vec{m}_B = \mathcal{D}^B\left(\breve{I}_B^B \vec{\omega}^{B/N}\right) + \left(\vec{\omega}^{B/N} \times \breve{I}_B^B \vec{\omega}^{B/N}\right)$$

$$\sum \vec{m}_B \approx \breve{I}_B^B\left(\mathcal{D}^B \vec{\omega}^{B/N}\right) + \left(\vec{\omega}^{B/N} \times \breve{I}_B^B \vec{\omega}^{B/N}\right)$$

with components in body coordinates represented as

$$\left\{\mathcal{D}^B \vec{\omega}^{B/N}\right\}^{(B)} = \begin{bmatrix}\dot{p}\\ \dot{q}\\ \dot{r}\end{bmatrix}, \qquad \{\vec{m}_B\}^{(B)} = \begin{bmatrix} l\\ m\\ n\end{bmatrix},$$
$$\left\{\vec{\omega}^{B/N}\right\}^{(B)} = \begin{bmatrix} p\\ q\\ r\end{bmatrix}, \qquad \left\{\breve{I}_B^B\right\}^{(B)} = \begin{bmatrix} I_{XX} & -I_{XY} & -I_{XZ}\\ -I_{XY} & I_{YY} & -I_{YZ}\\ -I_{XZ} & -I_{YZ} & I_{ZZ}\end{bmatrix} \tag{4.30}$$

provides the coordinate-dependent form which consists of the components in body coordinates as following

$$\begin{bmatrix}\dot{p}\\ \dot{q}\\ \dot{r}\end{bmatrix} = \begin{bmatrix} I_{XX} & -I_{XY} & -I_{XZ}\\ -I_{XY} & I_{YY} & -I_{YZ}\\ -I_{XZ} & -I_{YZ} & I_{ZZ}\end{bmatrix}^{-1}\left\{\begin{bmatrix} l\\ m\\ n\end{bmatrix} - \begin{bmatrix} p\\ q\\ r\end{bmatrix} \times \begin{bmatrix} I_{XX} & -I_{XY} & -I_{XZ}\\ -I_{XY} & I_{YY} & -I_{YZ}\\ -I_{XZ} & -I_{YZ} & I_{ZZ}\end{bmatrix}\begin{bmatrix} p\\ q\\ r\end{bmatrix}\right\} \tag{4.31}$$

Assuming the air vehicle is symmetric about the $X_B Z_B$ plane, then $I_{XY} = I_{YZ} = 0$.

Equation (4.31) now simplifies to

$$\begin{bmatrix}\dot{p}\\ \dot{q}\\ \dot{r}\end{bmatrix} = \begin{bmatrix} I_{XX} & 0 & -I_{XZ}\\ 0 & I_{YY} & 0\\ -I_{XZ} & 0 & I_{ZZ}\end{bmatrix}^{-1}\left\{\begin{bmatrix} l\\ m\\ n\end{bmatrix} - \begin{bmatrix} p\\ q\\ r\end{bmatrix} \times \begin{bmatrix} I_{XX} & 0 & -I_{XZ}\\ 0 & I_{YY} & 0\\ -I_{XZ} & 0 & I_{ZZ}\end{bmatrix}\begin{bmatrix} p\\ q\\ r\end{bmatrix}\right\} \tag{4.32}$$

The inverse of inertia tensor can be expressed in compact from as follows

$$\begin{bmatrix} I_{XX} & 0 & -I_{XZ}\\ 0 & I_{YY} & 0\\ -I_{XZ} & 0 & I_{ZZ}\end{bmatrix}^{-1} = \frac{1}{\Gamma}\begin{bmatrix} I_{ZZ} & 0 & I_{XZ}\\ 0 & \Gamma/I_{YY} & 0\\ I_{XZ} & 0 & I_{XX}\end{bmatrix} \tag{4.33}$$

where $\Gamma = I_{XX}I_{ZZ} - I_{XZ}^2$

The body-axes angular accelerations are obtained as by expanding out each component in Eq.(4.32)

$$\dot{p} = \frac{I_{XZ}(I_{XX} - I_{YY} + I_{ZZ})pq - (I_{ZZ}^2 - I_{YY}I_{ZZ} + I_{XZ}^2)qr + I_{ZZ}l + I_{XZ}n}{I_{XX}I_{ZZ} - I_{XZ}^2} \tag{4.34}$$

$$\dot{q} = \frac{(I_{ZZ} - I_{XX})pr - I_{XZ}(p^2 - r^2) + m}{I_{YY}}$$

$$\dot{r} = \frac{(I_{XX}^2 - I_{XX}I_{YY} + I_{XZ}^2)pq - I_{XZ}(I_{XX} - I_{YY} + I_{ZZ})qr + I_{XZ}l + I_{XX}n}{I_{XX}I_{ZZ} - I_{XZ}^2}$$

The three first-order differential equations (4.34) creates the rotational motions of a vehicle with respect to the inertial frame expressed in body axes govern the dynamics of the these 3-DoF.

These equations are nonlinear and couple with the translational equations in (4.27) solely through the aerodynamic moments included in the total moments $\{\vec{m}_B\}^{(B)} = [l \quad m \quad n]^T$. Integrating these equations provides the angular velocity of the body with respect to the inertial frame, expressed in body coordinates, the body rates $\left\{\vec{\omega}^{B/N}\right\}^{(B)} = [p \quad q \quad r]^T$.

4.1.5 Forces and moments

The translational and rotational equations of motion in body coordinates can be represented from the equations (4.27) and (4.34). With the exception rigid body and gravity in 4.1.1, these equations are fully general, indicating that no simplifying assumptions have been made in their derivation.

The solution to these equations of motion requires knowledge of the total forces and moments exerted on the air vehicle. These forces and moments originate from gravity, aerodynamics, and propulsion. Because gravity acts uniformly through the center of mass (CM), it does not generate a gravitational moment on the air vehicle. Therefore, gravity affects the air vehicle only as an external force acting along its body axes. Hence, the forces and moments consist of following components

- Aerodynamic forces and moments
- Propulsive forces
- Gravitational forces

The resultant force and moment are can be stated as

$$\begin{aligned} \vec{f} &= \vec{f}_a + \vec{f}_p + \vec{f}_g \\ \vec{m}_B &= \vec{m}_{B,a} + \vec{m}_{B,p} \end{aligned} \tag{4.35}$$

where $\vec{f}_a$, $\vec{f}_p$ and $\vec{f}_g$ are the *aerodynamic*, *propulsive*, and *gravitational* resultant forces. It's crucial to note that the force of gravity does not produce any external moment around the center of mass. Thus, the entirety of the external moment is solely attributed to the combined aerodynamic and propulsive forces.

4.1.5.1 Aerodynamic forces and moments

The aerodynamic forces and moments on the aircraft arise from the relative motion between the aircraft and the air, which is influenced by air density. These forces and moments are expressed in body coordinates using nondimensional aerodynamic coefficients for axial, side, and normal forces (C_X, C_Y, C_Z) such that

$$\left\{\vec{f}_a\right\}^{(B)} = \begin{bmatrix} X_a \\ Y_a \\ Z_a \end{bmatrix} = \bar{q}S \begin{bmatrix} C_X \\ C_Y \\ C_Z \end{bmatrix} = \frac{1}{2}\rho V_T^2 \begin{bmatrix} C_X \\ C_Y \\ C_Z \end{bmatrix} \tag{4.36}$$

And nondimensional aerodynamic coefficients in body axis for rolling, pitching, and yawing moments (C_l, C_m, C_z) acting about the CG are also considered as following

$$\left\{\vec{m}_{B,a}\right\}^{(B)} = \begin{bmatrix} l_a \\ m_a \\ n_a \end{bmatrix} = \bar{q}S \begin{bmatrix} bC_l \\ \bar{c}C_m \\ bC_n \end{bmatrix} \tag{4.37}$$

where $\bar{q}$ states dynamic pressure, V_T states true airspeed magnitude, ρ states air density, S states the wing reference area, b states the wingspan, and $\bar{c}$ is the mean aerodynamic chord.

The is calculation of dynamic pressure $\bar{q}$ as a function of airspeed V_T and air density ρ, is expressed as follows

$$\bar{q} = \frac{1}{2}\rho {V_T}^2 \tag{4.38}$$

The non-dimensional force components resolved in the wind axis can be obtained using the transformation matrix which consists of wind axis flight parameters in (4.5) and written as follows

$$\begin{bmatrix} -C_D \\ -C_C \\ -C_L \end{bmatrix} = \begin{bmatrix} \mathrm{c}_\alpha \mathrm{c}_\beta & \mathrm{s}_\beta & \mathrm{s}_\alpha \mathrm{c}_\beta \\ -\mathrm{c}_\alpha \mathrm{s}_\beta & \mathrm{c}_\beta & -\mathrm{s}_\alpha \mathrm{s}_\beta \\ -\mathrm{s}_\alpha & 0 & \mathrm{c}_\alpha \end{bmatrix} \begin{bmatrix} C_X \\ C_Y \\ C_Z \end{bmatrix} \tag{4.39}$$

Typically, non-dimensional force and moment coefficients exhibit nonlinear dependencies on several factors. These include the translational and angular velocity components of the aircraft, control surface deflections, and possibly their time derivatives. Additional factors such as Mach number and Reynolds number may also impact these coefficients.

The aerodynamic force and moment coefficients are intricately linked to various flight conditions, including Mach number, altitude, incidence angles, and their respective rates. These coefficients are also influenced by body angular rates and effective aerodynamic control surface deflections. The relationships governing these coefficients are established through rigorous analysis and empirical data, reflecting the complex interplay of aerodynamic forces and moments with the aircraft's dynamic state. This comprehensive understanding is crucial for accurate modeling and simulation and can be expressed as following [52].

$$C_a = C_a\left(M, \alpha, \beta, p, q, r, \delta_e, \delta_a, \delta_r, \delta_{lef}, \delta_{sb}\right) \tag{4.40}$$

where a stands for X, Y, Z, l, m, n

When examining the aerodynamic data of the air vehicle under investigation in this study;

The dependency can be expressed as in terms of C_a for the longitudinal coefficients

$$\begin{aligned} C_a = {} & C_a(\alpha) + \Delta C_a(\beta, \alpha) + \Delta C_a(\delta_e, \alpha) + \Delta C_a\left(\delta_{lef}, \alpha, \beta\right) \\ & + \frac{q\bar{c}}{2V_T}\left\{\left[C_{a_q}(\alpha) + \Delta C_{a_q}\left(\alpha, \delta_{lef}\right)\right]\right\} \end{aligned} \tag{4.41}$$

where a stands for X, Z, m.

The dependency also can be expressed as in terms of C_a for the lateral coefficients as

$$\begin{aligned} C_a = {} & C_a(\beta) + \Delta C_a(\beta, \alpha) + \Delta C_a\left(\delta_{lef}, \alpha, \beta\right) \\ & + \left[\Delta C_a(\delta_a, \alpha, \beta) + \Delta C_a\left(\delta_a, \alpha, \beta, \delta_{lef}\right)\right] + \left[\Delta C_a(\delta_r, \alpha, \beta)\right] \\ & + \frac{b}{2V_T}\left\{\left[C_{a_p}(\alpha) + \Delta C_{a_p}\left(\alpha, \delta_{lef}\right)\right] p \right. \\ & \left. + \left[C_{a_r}(\alpha) + \Delta C_{a_r}\left(\alpha, \delta_{lef}\right)\right] r\right\} \end{aligned} \tag{4.42}$$

where a stands for Y, l, m.

It is important to emphasize that within the expansive range of the aircraft's flight envelope; these intricate aerodynamic coefficients demonstrate nonlinear relationships with their corresponding dependent variables. These coefficients are integrated into the equations of motion to accurately characterize the aerodynamics of the air vehicle.

4.1.5.2 Propulsive forces

The propulsion forces are expressed assuming that the thrust from the propulsion systems acts along the longitudinal body axis X_B, with an inclination angle of the engines. This can be formulated as follows:

$$\overrightarrow{f_p} = \begin{bmatrix} T_p \cos \alpha_T \\ 0 \\ -T_p \sin \alpha_T \end{bmatrix} \tag{4.43}$$

$$\vec{m}_{B,p} = \overrightarrow{f_p} \times \vec{r}_{EN/B} = \begin{bmatrix} 0 \\ T_p \cos \alpha_T \, z_{EN/B} + T_p \sin \alpha_T \, x_{EN/B} \\ 0 \end{bmatrix} \tag{4.44}$$

where $\vec{r}_{EN/B}$ $(x_{EN/B}, y_{EN/B}, z_{EN/B})$ is the engine position vector with respect to center of mass in the body-fixed coordinate axes.

4.1.5.3 Gravitational force

The flat-Earth model assumes that the gravitational force always points along the Z_E axis in the inertial coordinate system, represented as $\{\overrightarrow{f_g}\}^{(B)} = [0 \quad 0 \quad mg]^T$. The gravitational force acting on the body can be expressed using the transformation as

$$\{\overrightarrow{f_g}\}^{(B)} = \begin{bmatrix} X_g \\ Y_g \\ Z_g \end{bmatrix} = \bar{T}^{(N,B)} \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} = \begin{bmatrix} -mg\mathrm{s}_\phi \\ mg\mathrm{s}_\phi \mathrm{c}_\theta \\ mg\mathrm{c}_\phi \mathrm{c}_\theta \end{bmatrix} \tag{4.45}$$

The total forces and moments acting on the body axis can be obtained from the equation in (4.36)-(4.37), and (4.43)-(4.45) and stated in body coordinates as following

$$\{\vec{f}\}^{(B)} = \begin{bmatrix} f_{X_B} \\ f_{Y_B} \\ f_{B_B} \end{bmatrix} = \bar{q}S \begin{bmatrix} C_X \\ C_Y \\ C_Z \end{bmatrix} + mg \begin{bmatrix} -\mathrm{s}_\theta \\ \mathrm{s}_\phi \mathrm{c}_\theta \\ \mathrm{c}_\phi \mathrm{c}_\theta \end{bmatrix} + T_p \begin{bmatrix} \mathrm{c}_{\alpha_T} \\ 0 \\ -\mathrm{s}_{\alpha_T} \end{bmatrix} \tag{4.46}$$

4.1.6 Auxiliary equations

4.1.6.1 Accelerations

Accelerometers measure the specific accelerations. The body-axes components of specific linear accelerations at the vehicle center of mass, $\overrightarrow{a_{s_{B/N}}^{N}}$, can be determined with subtracting the gravitational force, $\overrightarrow{f_g}$ and can be expressed in this way

$$\overrightarrow{a_{s_{B/N}}^{N}} = \frac{1}{m}\left(\vec{f} - \overrightarrow{f_g}\right) \tag{4.47}$$

with separating this acceleration into the its body coordinate vector components as following

$$\left\{\overrightarrow{a_{s_{B/N}}^{N}}\right\}^{(B)} = \begin{bmatrix} a_{s_x} \\ a_{s_y} \\ a_{s_z} \end{bmatrix}, \qquad \left\{\overrightarrow{f_a}\right\}^{(B)} = \begin{bmatrix} X_a \\ Y_a \\ Z_a \end{bmatrix}, \qquad \left\{\overrightarrow{f_P}\right\}^{(B)} = \begin{bmatrix} X_p \\ Y_p \\ Z_p \end{bmatrix} \tag{4.48}$$

Expanding (4.47) in using the vector components and writing out the specific accelerations yields

$$\begin{bmatrix} a_{s_x} \\ a_{s_y} \\ a_{s_z} \end{bmatrix} = \frac{1}{m}\left(\begin{bmatrix} X_a + X_p \\ Y_a + Y_p \\ Z_a + Z_p \end{bmatrix}\right) \tag{4.49}$$

4.1.6.2 Airspeed dynamics

The Mach number M determines the functional dependence of aerodynamic coefficients, rather than the body axes linear velocity components u, v, w or the airspeed V_T. Thus, an equation describing the dynamics of M can be easily expressed as following

$$M = \frac{V_T}{a} \tag{4.50}$$

where a states that the speed of sound which is a function of the altitude h in standard atmosphere. Thus, taking the derivation of (4.50) gives

$$\dot{M} = \frac{\dot{V}_T a - V_T \dot{a}}{a^2} \tag{4.51}$$

Inside the lowest portion of the Earth's atmosphere, the speed of sound changes is negligibly small. Therefore, in the derivation of the differential equation for Mach number, it can be stated that even though both h and a are varying, the time rate of change of a is so small, so considering negligibility (4.52) can be stated

$$\dot{M} = \frac{\dot{V}_T}{a} \tag{4.52}$$

Also, $\dot{V}_T$ equation can also be is obtained by differentiating (4.17), and substituting (4.27) gives

$$\dot{V}_T = \frac{1}{V}[u\dot{u} + v\dot{v} + w\dot{w}]$$

$$\dot{V}_T = \frac{1}{V}\big[u(f_x/m - qw + rv) + v\big(f_y/m - ru + pw\big) + w(f_z/m - pv + qu)\big] \tag{4.53}$$

The differential equation for airspeed can be derived by substituting (4.18) into this equation yields

$$\dot{V}_T = a_{X_B}\mathrm{c}_\alpha\mathrm{c}_\beta + a_{Y_B}\mathrm{s}_\beta + a_{Z_B}\mathrm{s}_\alpha\mathrm{c}_\beta \tag{4.54}$$

Hence, for Mach number, substituting (4.54) into (4.52) yields

$$\dot{M} = \frac{1}{a}\big[a_{X_B}\mathrm{c}_\alpha\mathrm{c}_\beta + a_{Y_B}\mathrm{s}_\beta + a_{Z_B}\mathrm{s}_\alpha\mathrm{c}_\beta\big] \tag{4.55}$$

4.1.6.3 Incidence angle dynamics

The translational equations of motion (4.27) are calculated for the rates of change of the components u , v and w of the use of total velocity vector V_T using body coordinates. Their integration and interaction yield body linear velocities, which are used to calculate α and β. Using of the wind coordinate system accounted for the rate of change of the magnitude of the velocity vector ($\dot{V}_T$) which also to lie along the X_W wind axis, and induces for explicit equations for the rates of change of α and β.

$$\dot{\alpha} = \frac{u\dot{w} - w\dot{u}}{u^2 + w^2} = \frac{\dot{w}c_a - \dot{u}s_a}{Vc_\beta} \tag{4.56}$$

$$\dot{\beta} = \frac{V_t \dot{v} - v\dot{V}_T}{V_T \sqrt{{V_T}^2 - v^2}} = \frac{\dot{v} - \dot{V}_T s_\beta}{V c_\beta}$$

Substituting (4.27) and (4.54) into these equations yields

$$\begin{aligned} \dot{\alpha} &= q - (p\mathrm{c}_\alpha + r\mathrm{s}_\alpha)\mathrm{t}_\beta - \frac{1}{V_T \mathrm{c}_\beta}\left(a_{X_B}\mathrm{s}_\alpha - a_{Z_B}\mathrm{c}_\alpha\right) \\ \dot{\beta} &= p\mathrm{s}_\alpha - r\mathrm{c}_\alpha - \frac{1}{V_T}\left(a_{X_B}\mathrm{c}_\alpha\mathrm{s}_\beta - a_{Y_B}\mathrm{c}_\beta + a_{Z_B}\mathrm{s}_\alpha\mathrm{s}_\beta\right) \end{aligned} \tag{4.57}$$

4.1.6.4 Bank angle dynamics

The angular velocities $\vec{\omega}^{B/E}$ and $\vec{\omega}^{E/B}$ of the wind frame $\mathcal{F}_W$ and the body frame $\mathcal{F}_B$ with respect to the Earth frame $\mathcal{F}_E$ are expressed as

$$\vec{\omega}^{B/W} = \vec{\omega}^{B/S} + \vec{\omega}^{S/W} = \dot{\alpha}\hat{\jmath}_B - \dot{\beta}\hat{k}_S \tag{4.58}$$

Solving this equation in body coordinates produces following relations

$$\begin{aligned} \left\{\vec{\omega}^{B/W}\right\}^{(B)} &= \dot{\alpha}\{\hat{\jmath}_B\}^{(B)} - \dot{\beta}\breve{T}^{(B,S)}\{\hat{k}_S\}^{(S)} = \begin{bmatrix} \dot{\beta}\mathrm{s}_\alpha \\ \dot{\alpha} \\ -\dot{\beta}\mathrm{c}_\alpha \end{bmatrix} \\ \vec{\omega}^{W/N} &= \vec{\omega}^{B/N} - \vec{\omega}^{B/W} \\ \left\{\vec{\omega}^{W/N}\right\}^{(B)} &= \left\{\vec{\omega}^{B/N}\right\}^{(B)} - \left\{\vec{\omega}^{B/W}\right\}^{(B)} = \begin{bmatrix} p - \dot{\beta}\mathrm{s}_\alpha \\ q - \dot{\alpha} \\ r + \dot{\beta}\mathrm{c}_\alpha \end{bmatrix} \end{aligned} \tag{4.59}$$

Using a sequence of rotations from fixed-Earth coordinates to wind coordinates with $\chi \to \gamma \to \mu$ sequence, a relationship between the angular rates $\dot{\mu}$, $\dot{\gamma}$, $\dot{\chi}$ and the wind-fixed angular rates $\left\{\vec{\omega}^{W/N}\right\}^{(W)}$ can also be derived as

$$\begin{aligned} \left\{\vec{\omega}^{W/N}\right\}^{(W)} &= \dot{\mu}\{\hat{\imath}_W\}^{(W)} + \dot{\gamma}\breve{T}^{(W,V)}\left\{\vec{\omega}^{V/N}\right\}^{(V)} \\ \begin{bmatrix} \mathrm{c}_\alpha\mathrm{c}_\beta & \mathrm{s}_\beta & \mathrm{s}_\alpha\mathrm{c}_\beta \\ -\mathrm{c}_\alpha\mathrm{s}_\beta & \mathrm{c}_\beta & -\mathrm{s}_\alpha\mathrm{s}_\beta \\ -\mathrm{s}_\alpha & 0 & \mathrm{c}_\alpha \end{bmatrix} \begin{bmatrix} p - \dot{\beta}\mathrm{s}_\alpha \\ q - \dot{\alpha} \\ r + \dot{\beta}\mathrm{c}_\alpha \end{bmatrix} &= \begin{bmatrix} 1 & 0 & -\mathrm{s}_\gamma \\ 0 & \mathrm{c}_\mu & \mathrm{s}_\mu\mathrm{c}_\gamma \\ 0 & -\mathrm{s}_\mu & \mathrm{c}_\mu\mathrm{c}_\gamma \end{bmatrix} \begin{bmatrix} \dot{\mu} \\ \dot{\gamma} \\ \dot{\chi} \end{bmatrix} \end{aligned} \tag{4.60}$$

$$\begin{bmatrix}\dot{\mu}\\ \dot{\gamma}\\ \dot{\chi}\end{bmatrix}=\begin{bmatrix}1 & 0 & -\mathrm{s}_\gamma\\ 0 & \mathrm{c}_\mu & \mathrm{s}_\mu\mathrm{c}_\gamma\\ 0 & -\mathrm{s}_\mu & \mathrm{c}_\mu\mathrm{c}_\gamma\end{bmatrix}^{-1}\begin{bmatrix}\mathrm{c}_\alpha\mathrm{c}_\beta & \mathrm{s}_\beta & \mathrm{s}_\alpha\mathrm{c}_\beta\\ -\mathrm{c}_\alpha\mathrm{s}_\beta & \mathrm{c}_\beta & -\mathrm{s}_\alpha\mathrm{s}_\beta\\ -\mathrm{s}_\alpha & 0 & \mathrm{c}_\alpha\end{bmatrix}\begin{bmatrix}p-\dot{\beta}\mathrm{s}_\alpha\\ q-\dot{\alpha}\\ r+\dot{\beta}\mathrm{c}_\alpha\end{bmatrix}$$

The bank angle dynamics $\dot{\mu}$ is obtained from the above relation by expanding it and substituting (4.56) for $\dot{\alpha}$ and $\dot{\beta}$, respectively, as follows

$$\begin{aligned}\dot{\mu}=\left(p\mathrm{c}_\alpha\mathrm{c}_\beta+q\mathrm{s}_\beta+r\mathrm{s}_\alpha\mathrm{c}_\beta-\dot{\alpha}\mathrm{s}_\beta\right)-\mathrm{s}_\mu\mathrm{t}_\gamma\left(p\mathrm{c}_\alpha\mathrm{s}_\beta-q\mathrm{c}_\beta+r\mathrm{s}_\alpha\mathrm{s}_\beta+\dot{\alpha}\mathrm{c}_\beta\right)\\ -\mathrm{c}_\mu\mathrm{t}_\gamma\left(p\mathrm{s}_\alpha-r\mathrm{c}_\alpha-\dot{\beta}\right)\end{aligned}\tag{4.61}$$

4.1.6.5 Flight path angle dynamics

The angular velocity $\vec{\omega}^{N/V}$ of the velocity frame $\mathcal{F}_V$ with respect to the Earth frame $\mathcal{F}_E$ can be found by vector addition of the flight path angular rates $\dot{\chi}$ and $\dot{\gamma}$ multiplied by their respective unit vectors. The following combined coordinate equation for $\dot{\gamma}$ is obtained by resolving $\vec{\omega}^{N/V}$ in velocity coordinates for expressing the velocity-fixed angular rates $\{\vec{\omega}^{N/V}\}^{(V)}$ in component form yields

$$\begin{aligned}\vec{\omega}^{V/N}&=\vec{\omega}^{V/X}+\vec{\omega}^{X/N}=\dot{\gamma}\hat{\jmath}_V+\dot{\chi}\hat{k}_X\\ \{\vec{\omega}^{V/N}\}^{(V)}&=\dot{\gamma}\{\hat{\jmath}_V\}^{(V)}+\dot{\chi}\breve{T}^{(V,X)}\{\hat{k}_X\}^{(X)}=\begin{bmatrix}-\dot{\chi}\mathrm{s}_\gamma\\ \dot{\gamma}\\ \dot{\chi}\mathrm{c}_\gamma\end{bmatrix}\\ \dot{\gamma}&=\begin{bmatrix}-\dot{\chi}\mathrm{s}_\gamma\\ \dot{\gamma}\\ \dot{\chi}\mathrm{c}_\gamma\end{bmatrix}\end{aligned}\tag{4.62}$$

4.1.6.6 Altitude dynamics

The component transformation of $\{\vec{v}_{B/N}^{N}\}^{(V)}$ to the fixed-Earth coordinate system provides following equations

$$\begin{aligned}\breve{T}^{(N,V)}&=\breve{R}_{\hat{k}}(\chi)\breve{R}_{\hat{\jmath}}(\gamma)=\begin{bmatrix}\mathrm{c}_\chi & -\mathrm{s}_\chi & 0\\ \mathrm{s}_\chi & \mathrm{c}_\chi & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}\mathrm{c}_\gamma & 0 & \mathrm{s}_\gamma\\ 0 & 1 & 0\\ -\mathrm{s}_\gamma & 0 & \mathrm{c}_\gamma\end{bmatrix}\\ \{\vec{v}_{B/N}^{N}\}^{(N)}&=\begin{bmatrix}\mathrm{c}_\chi\mathrm{c}_\gamma & -\mathrm{s}_\chi & \mathrm{c}_\chi\mathrm{s}_\gamma\\ \mathrm{s}_\chi\mathrm{c}_\gamma & \mathrm{c}_\chi & \mathrm{s}_\chi\mathrm{s}_\gamma\\ -\mathrm{s}_\gamma & 0 & \mathrm{c}_\gamma\end{bmatrix}\begin{bmatrix}V_T\\ 0\\ 0\end{bmatrix}\end{aligned}\tag{4.63}$$

$$\left\{\vec{v}_{B/N}^{N}\right\}^{(N)} = V\begin{bmatrix} c_\chi c_\gamma \\ s_\chi c_\gamma \\ -s_\gamma \end{bmatrix}$$

Notice that, altitude dynamics $\dot{h}$ can be derived from the negative component of $\left\{\vec{v}_{B/N}^{N}\right\}^{(N)}$, so that results in following

$$\dot{h} = V s_\gamma \tag{4.64}$$

4.2 Development of Air Vehicle Simulation Model

After deriving the equations of motion for an air vehicle across a flat Earth, the subsequent stage in the development process involves constructing a mathematical simulation model of the air vehicle using industry-standard model-based design tools. This step is crucial within the context of this thesis, which emphasizes model-based design governed by the DO-331 supplement within the scope of DO-178C.

As discussed in Chapter 4, the simulation model of the air vehicle involves the integration of various plant and subsystem models. These models are simulated separately and include the plant model comprising aerodynamic, propulsion, and environmental models, as well as effector and sensor models. The plant model encapsulates the physical characteristics and behaviors of the air vehicle, such as its aerodynamic properties, propulsion system dynamics, and environmental influences like wind and atmospheric conditions. Effector models represent the control surfaces and other secondary actuation mechanisms, while sensor models simulate the inputs received from various sensors on the air vehicle, such as inertial measurement units (IMUs), airspeed indicators, and altitude sensors.

The development of these models is essential for conducting comprehensive simulations of the air vehicle in flight. By separately modeling each aspect of the vehicle's dynamics and environment, it can be evaluated point of its performance under various conditions and scenarios. This approach supports the iterative refinement of the air vehicle's design, guaranteeing that it meets the required performance specifications and safety standards stipulated by DO-178C and DO-331. Moreover, the use of industry-standard tools facilitates the verification and validation of the simulation models, ensuring their accuracy and fidelity to the real-world air vehicle.

4.2.1 Plant model

The plant model encompasses calculations pertaining to the aerodynamic database, the propulsion database, and their respective dynamics. It also incorporates computations of total force and moments derived from these databases, alongside integration with the six degrees of freedom (6-DoF) equations of motion model. Furthermore, Table 4.2 provides mass and dimensional information pertaining to the plant model.

Table 4.2 : Air vehicle mass and dimensional parameters.

Mass properties		Dimensions	
$m\ [kg]$	9295.44	$b\ [m]$	9.144
$I_X\ [kgm^2]$	9496	$S\ [m^2]$	27.87
$I_Y\ [kgm^2]$	75674	$c\ [m]$	3.45
$I_Z\ [kgm^2]$	85552	$X_{xg_{ref}}$	0.35c
$I_{XZ}\ [kgm^2]$	1331		

4.2.1.1 Aerodynamic model

The development of an aerodynamic model relies heavily on a comprehensive database derived from wind tunnel tests conducted by NASA Langley and Ames Research Center [3]. This database is essential for accurately predicting aerodynamic forces and moments across a wide range of flight conditions. It encompasses various parameters such as angle of attack, which spans from negative values to high positive angles, angle of sideslip, ranging from left to right, and true airspeed, which varies from low to high values. The aerodynamic model's accuracy and fidelity are critical for supporting stability, control, and performance assessments of air vehicles.

Each total aerodynamic coefficient is modeled as the sum of a baseline component in addition to incremental terms since the complicated functional dependence of the aerodynamic coefficients represented in (4.40). The baseline components are essentially are functions of α, β, δ_e. Total nondimensional aerodynamic coefficients for F-16 fighter aircraft are expressed in the body axis of the airframe as seen in the equations from (4.65) to (4.70).

$$C_X = C_X(\alpha, \beta, \delta_e) + \frac{c}{2V}\left[C_{Xq}(\alpha)q + \delta_{C_{Xq_{lef}}}\delta_{lef_{norm}}\right] + \delta C_{x_{lef}}\delta_{lef_{norm}} \qquad (4.65)$$

$$C_Y = C_y(\alpha,\beta) + \delta_{C_{Y_{lef}}}\delta_{lef_{norm}} + \left[\delta_{C_{Y_{\delta_{a20}}}} + \delta_{C_{Y_{\delta_{a20_{lef}}}}}\delta_{lef_{norm}}\right]\delta_{a_{norm}} + \delta_{C_{Y_{\delta_{r30}}}}\delta_{r_{norm}} + \frac{b}{2V}\left[C_{Yr}(\alpha) + \delta_{C_{Yr_{lef}}}\delta_{lef_{norm}}\right]r + \frac{b}{2V}\left[C_{Yp}(\alpha) + \delta_{C_{Yp_{lef}}}\delta_{lef_{norm}}\right]p \quad (4.66)$$

$$C_Z = C_z(\alpha,\beta,\delta_e) + \delta_{C_{Z_{lef}}}\delta_{lef_{norm}} + \frac{c}{2V}\left[C_{Zq}(\alpha) + \delta_{C_{Zq_{lef}}}\right]q \quad (4.67)$$

$$C_l = C_l(\alpha,\beta,\delta_e) + \delta_{C_{l_{lef}}}\delta_{lef_{norm}} + \left[\delta_{C_{l_{\delta_{a20}}}} + \delta_{C_{l_{\delta_{a20_{lef}}}}}\delta_{lef_{norm}}\right]\delta_{a_{norm}} + \delta_{C_{l_{\delta_{r30}}}}\delta_{r_{norm}} + \frac{b}{2V}\left[C_{l_r}(\alpha) + \delta_{C_{l_{r_{lef}}}}\delta_{lef_{norm}}\right]r + \frac{b}{2V}\left[C_{l_p}(\alpha) + \delta_{C_{l_{p_{lef}}}}\delta_{lef_{norm}}\right]p + \delta_{C_{l\beta}}(\alpha)\beta \quad (4.68)$$

$$C_m = C_m(\alpha_d,\delta_E)\Pi_e + C_Z\left(X_{cg_{ref}} - X_{cg}\right) + \delta_{C_{m_{lef}}}\delta_{lef_{norm}} + \frac{c}{2V}\left[C_{m_q}(\alpha) + \delta_{C_{mq_{lef}}}(\alpha)\delta_{lef_{norm}}\right]q + \delta_{C_m}(\alpha) + \delta_{C_{m_{ds}}}(\alpha,\delta_e) \quad (4.69)$$

$$C_n(\alpha,\beta,\delta_e) + \delta_{C_{n_{lef}}}\delta_{lef_{norm}} - C_Y\left(X_{cg_{ref}} - X_{cg}\right) + \left[\delta_{C_{n_{\delta_{a20}}}} + \delta_{C_{n_{\delta_{a20_{lef}}}}}\delta_{lef_{norm}}\right]\delta_{a_{norm}} + \frac{b}{2V}\left[C_{n_r}(\alpha) + \delta_{C_{nr_{lef}}}(\alpha)\delta_{lef_{norm}}\right]r + \frac{b}{2V}\left[C_{n_p}(\alpha) + \delta_{C_{n_{p_{lef}}}}(\alpha)\delta_{lef_{norm}}\right]p + \delta_{C_{n_{\delta_{r30}}}}\delta_{r_{norm}} + \delta_{C_{n_\beta}}(\alpha)\beta \quad (4.70)$$

The essential parts of the aerodynamic coefficients are the primary terms on the right-hand side of above equations. They provide a foundational representation of the coefficients under standard conditions. Following the baseline components are incremental terms, ordered by their importance, which include variables such as angular rates. These incremental terms capture deviations from the baseline due to changes in flight conditions, such as variations in angles of attack, sideslip, and other

aerodynamic parameters. Noting that angular rate variables are nondimensional, and aerodynamic database from [3] embedded using the look-up table form.

During the generation of aerodynamic coefficients, six sets of breakpoints are defined and depicted in Table 4.3 [53] [54].

Table 4.3 : Aerodynamic data breakpoints.

Breakpoint parameter	Notation	Breakpoint range (deg)
Angle of attack	α_{bp_1}	$[-20,-15,-10,-5,0,5,10,15,20,25,30,35,40,45,50,55,60,70,80,90]$
	α_{bp_2}	$[-20,-15,-10,-5,0,5,10,15,20,25,30,35,40,45]$
Angle of sideslip	β_{bp_1}	$[-30,-25,-20,-15,-10,-8,-6,-4,-2,0,2,4,6,8,10,15,20,25,30]$
HT deflection	$\delta_{e_{bp_1}}$	$[-25,-10,0,10,25]$
	$\delta_{e_{bp_2}}$	$[-25,0,25]$
	$\delta_{e_{bp_3}}$	$[-25,-10,0,10,15,20,25]$

These breakpoint sets are defined during the generation of aerodynamic coefficients, and the components and related breakpoints of the aerodynamic dataset can be listed as (4.71).

The breakpoints used for elevator deflection consist of three sets, each with different resolutions within their upper and lower ranges. Similarly, the angle of attack parameter has two sets of breakpoints, while the angle of sideslip has only one, each within their upper and lower range of breakpoints.

$$\begin{aligned} C_X &= f\left(\alpha(\alpha_{bp_1}), \beta(\beta_{bp_1}), \delta e(\delta_{e_{bp_1}})\right) \\ C_Y &= f\left(\alpha(\alpha_{bp_1}), \beta(\beta_{bp_2})\right) \\ C_Z &= f\left(\alpha(\alpha_{bp_1}), \beta(\beta_{bp_1}), \delta e(\delta_{e_{bp_1}})\right) \\ C_l &= f\left(\alpha(\alpha_{bp_1}), \beta(\beta_{bp_1}), \delta e(\delta_{e_{bp_2}})\right) \\ C_m &= f\left(\alpha(\alpha_{bp_1}), \beta(\beta_{bp_1}), \delta e(\delta_{e_{bp_1}})\right) \\ C_n &= f\left(\alpha(\alpha_{bp_1}), \beta(\beta_{bp_1}), \delta e(\delta_{e_{bp_2}})\right) \end{aligned} \tag{4.71}$$

In Figure 4.4, the pitch moment coefficient C_m in (4.69), which is illustrated graphically. The 3-D plot in the Figure 4.4 implies that the C_m depends on the angles of attack and

angles of sideslip, as well as elevator deflection as $f\left(\alpha(\alpha_{bp_1}), \beta(\beta_{bp_1}), \delta e(\delta_{e_{bp_1}} = 0)\right)$. This graphical representation allows for a comprehensive understanding of how these parameters interact to influence the pitch moment coefficient in aerodynamic analysis.

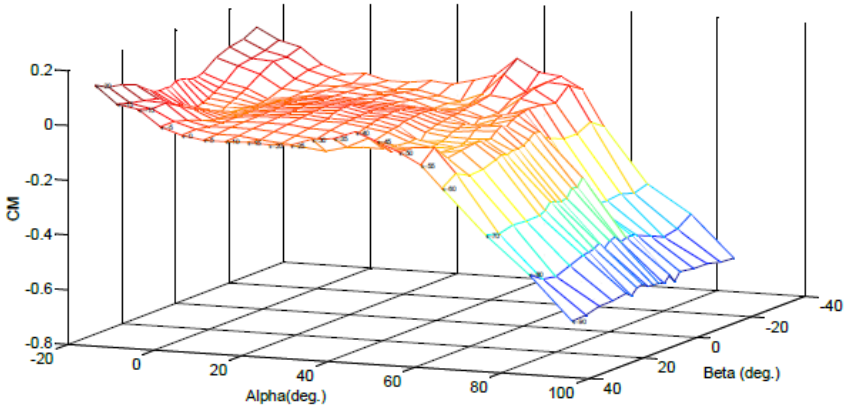


Figure 4.4 : Pitch moment coefficient graph [54].

The aerodynamic model F-16 also includes a LEF that schedules according to the values of α, $\bar{q}$ and p_s. The formula for the calculation of deflection of the LEF can be seen in (4.72).

$$\delta_{lef} = 1.38\left(\frac{2s + 7.25}{s + 7.25}\right)\alpha - 9.05\left(\frac{\bar{q}}{p_s}\right) + 1.45 \tag{4.72}$$

4.2.1.2 Propulsion model

This chapter provides a mathematical model of the propulsion which is engine, enabling estimation of its thrust force across various flight conditions.

The engine model is split into two sub-models: one detailing the static engine state and the other addressing the engine dynamics, encompassing transient states. These sub-model representations are depicted in Figure 4.5. The subsequent chapters will present each sub-model individually.

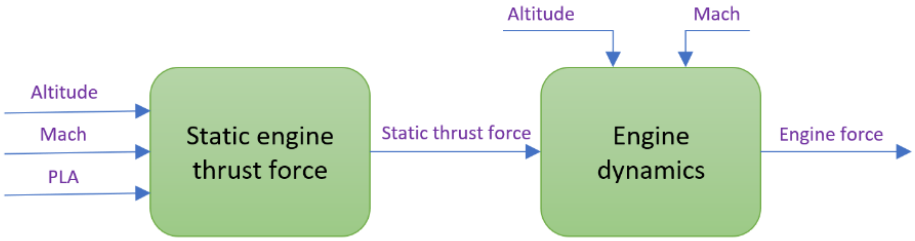


Figure 4.5 : Simplified engine model [50].

i. Static engine model

Thrust variation is influenced by three main external parameters like air density, Mach number, and altitude. Additionally, the position of the throttle level affects internal variables. The engine static model is described as follows [55] :

$$\frac{T(h, M, \delta_{th})}{T_0} = \bar{\sigma}(h, M) \times \tilde{T}(h, M, \delta_{th}) \tag{4.73}$$

here $\tilde{T}(h, M, \delta_{TLA})$ is the nondimensional thrust coefficient and $\bar{\sigma}$ is the relative air density ratio, and $\tilde{T}(h, M, \delta_{TLA})$ is a polynomial is used calculate the static thrust which is can be observable in Table 4.4. Engine thrust is found by utilizing the equation (4.72).

Table 4.4 : Tabular form of maximum thrust values [3].

Mach	Thrust values at an altitude, ft, of -					
	0	10000	20000	30000	40000	50000
			T_{idle}			
0.2	635	425	690	1010	1330	1700
0.4	60	25	345	755	1130	1525
0.6	-1020	-710	-300	350	910	1360
0.8	-2700	-1900	-1300	-247	600	1100
1	-3600	-1400	-595	-342	-200	700
			T_{mil}			
0.2	12680	9150	6313	4040	2470	1400
0.4	12610	9312	6610	4290	2600	1560
0.6	12640	9839	1090	4660	2840	1660
0.8	12390	10176	7750	5320	3250	1930
1	11680	9848	8050	6100	3800	2310
			T_{max}			
0.2	21420	15700	11255	7323	4425	2600
0.4	22700	16860	12250	8154	5000	2835
0.6	24240	18910	13760	9285	5700	3215
0.8	26070	21075	15975	11115	6860	3950
1	28886	23319	18300	13484	8612	5057

ii. Transient (Dynamic) engine model

The transient engine dynamic response for a turbofan engine is represented by a state equation, modeling a first-order lag from the actual thrust to the commanded thrust, as shown below [50], [51], [46], [47] and [48].

$$T_a = \frac{1}{\tau_{eng}}(T - T_a) \tag{4.74}$$

where τ_{eng} states the engine time constant, T_a states the actual thrust, and T denotes the steady-state thrust at specified altitude, Mach number, and throttle position. τ_{eng} varies according to the flight condition.

The engine model created in this study follows the approach of Huo [56] and is valid for airspeeds below $Mach = 1$ and altitudes below 15240 . As stated in (4.74), the actual power of the engine is defined by a first-order differential equation that accounts for its dynamic response. This equation captures how the engine responds to changes in the throttle input or other operating conditions. In addition to the dynamic response, the thrust levels are classified into three distinct categories: idle, military, and maximum power. These categories define the thrust output of the engine under different operational scenarios.

4.2.1.3 Flight environment (atmosphere) model

The International Standard Atmosphere (ISA) model plays a crucial role as a reference for atmospheric conditions when developing air vehicle mathematical environment model. The ISA model is corresponding to average atmospheric conditions and defines the temperature variations, density, pressure and other parameters with altitude. Table 4.5. shows that the standard conditions according to the ISA model at sea level.

Table 4.5 : ISA standard conditions.

Standard Condition	Value	Unit
Pressure	1013.25	hPa
Temperature	15	$°C$
Density	1.225	kg/m^3

In ISA model, the temperature tends to decline with altitude at roughly $6.5°C$ per kilometer up to an altitude of $11\ km$ on the layer of tropopause, where it becomes constant. Above this altitude, the temperature starts to increase with altitude. The ISA model provides a standardized set of conditions such as dynamic pressure and atmospheric gravity that are used as a baseline for developing air vehicle mathematical models, enabling consistent performance predictions and comparisons across different aircraft and atmospheric conditions.

Atmospheric gravity is calculated as following:

$$g_{atm} = g_0 \left(\frac{R_e^2}{R_e^2 + h} \right) \tag{4.75}$$

where R_e is the radius of the earth the pressure altitude h is used.

The calculation of Mach number as follows

$$M = \frac{V_T}{\sqrt{\gamma_e RT}} \tag{4.76}$$

here, γ_e denotes the isentropic expansion factor of air and R denotes the gas constant of air.

Temperature is calculated as

$$T = \begin{cases} P_0 \left(\frac{\tau}{\tau_0} \right)^{\frac{-g_0}{RT_{grad}} \left(10^{\left(\frac{-g_0 (h - 11000))}{2.3026RT} \right)} \right)} & if\ h \geq 11000\ m \\ P_0 \left(\frac{\tau}{\tau_0} \right)^{\frac{-g_0}{RT_{grad}}} & if\ h \geq 11000\ m \end{cases} \tag{4.77}$$

where; T_{grad} states the gradient of temperature, g_0 states gravity constant at sea level and P_0 denotes the sea level pressure as stated in Table 4.5.

Air density is calculated in following equation

$$\rho = \begin{cases} \rho_0 \left(\frac{T}{T_0} \right)^{\frac{-g_0}{RT_{grad}}} & if\ h \geq 11000\ m \\ \frac{P}{RT} & if\ h \geq 11000\ m \end{cases} \tag{4.78}$$

where; ρ_0 is the air density, and T_0 temperature at sea level as stated in Table 4.5.

Dynamic pressure is calculated as given

$$\bar{q} = \frac{1}{2} \rho V_T^2 \tag{4.79}$$

4.2.2 Effector models

Effectors play a pivotal role in the flight control system by supplying the necessary motive power to maneuver the air vehicle's flight control surfaces. Due to their inherent dynamics, effectors wield considerable influence on aircraft performance and therefore warrant careful consideration in flight control system design. Primary control effectors and secondary flight control functions such as leading-edge flaps (LEF) are generally represented as first-order transfer functions are given as

$$\frac{\delta_e(s)}{\delta_{e_{com}}(s)} = \frac{60}{s+60} \tag{4.80}$$

$\delta_{(\cdot)_{com}}(s)$ represents the reference commands sent for related effector to achieve in transfer function stated in (4.80) and the effector position and rate limits are specified in Table 4.6.

Table 4.6 : Effector actual position and rate limits [53].

Effector command	Position	Rate	τ
δ_e	$[-25°, 25°]$	$60\ deg/s$	$0.0495s$
δ_a	$[21{,}5°, 21{,}5°]$	$80\ deg/s$	$0.0495s$
δ_r	$[-30°, 30°]$	$120\ deg/s$	$0.0495s$
δ_{lef}	$[0°, 25°]$	$25\ deg/s$	$0.136s$

4.2.3 Sensor models

The dynamics of sensors, alongside the filters they incorporate, calculation delays, and analog-to-digital converters, can introduce a phase delay into the controlled system, necessitating careful consideration. This delay needs must be considered, highlighting the importance of mathematically modeling sensors to accurately represent their behavior. The sensor models addressed in the study can be listed as follows

- Inertial measurement unit (IMU) that senses body angular rates (p, q, r) and translational accelerations (a_x, a_y, a_Z)
- Attitude sensor that computes body Euler rates (ϕ, θ, ψ)
- Air data sensor that computes α and β
- Deflection sensor that computes effector deltas $\delta_{(\cdot)}$

The sensors are implemented as first-order transfer functions as specified

$$\frac{\alpha_m(s)}{\alpha(s)} = \frac{1}{0.02s + 1} \tag{4.81}$$

4.3 Development of Trim and Linear Aircraft Models

After the development of the air vehicle mathematical simulation model, it is crucial to investigate the trim and linearization of the bare airframe models.

4.3.1 Representation of dynamic systems

The mathematical equation of the nonlinear dynamic system can be reflected in state-space form as follows

$$\begin{aligned} \dot{\vec{x}} &= f(\vec{x}, \vec{u}), \;\; \vec{x}(t_0) = \vec{x}_0 \\ \vec{y} &= g(\vec{x}, \vec{u}) \end{aligned} \tag{4.82}$$

In (4.82), state space form elements are

- $\vec{x}$, denotes state variables, $n_u \times 1$
- $\vec{u}$, denotes the control input, $n_u \times 1$
- $\vec{y}$, denotes the output variables, $n_y \times 1$

Considering simple linear system representations in (4.82), these linear models are valid with small perturbations around the equilibrium points. By linearizing the dynamic system equations around a suitable operating point as following

$$\begin{aligned} \dot{\vec{x}} &= A\vec{x} + B\vec{u}, \;\; x(t_0) = x_0 \\ y &= C\vec{x} + D\vec{u} \end{aligned} \tag{4.83}$$

A, B, C, and D represent the system matrices and denoted by [57]

$$\begin{aligned} \mathrm{A}_{ij} &= \frac{f_i(\mathrm{x} + \epsilon e_j, u) - f_i(\mathrm{x} - \epsilon e_j, u)}{2\epsilon}; i = 1,2,\cdots,n_{\dot{x}};\; j = 1,2,\cdots,n_x \\ \mathrm{B}_{ij} &= \frac{f_i(\mathrm{x}, u + \epsilon e_j) - f_i(\mathrm{x}, u - \epsilon e_j)}{2\epsilon}; i = 1,2,\cdots,n_{\dot{x}};\; j = 1,2,\cdots,n_u \end{aligned} \tag{4.84}$$

$$\mathrm{C}_{ij} = \frac{g_i(\mathrm{x} + \epsilon e_j, u) - g_i(\mathrm{x} - \epsilon e_j, u)}{2\epsilon}; i = 1,2,\cdots,n_y;\ j = 1,2,\cdots,n_x$$

$$\mathrm{D}_{ij} = \frac{g_i(\mathrm{x}, u + \epsilon e_j) - g_i(\mathrm{x}, u - \epsilon e_j)}{2\epsilon}; i = 1,2,\cdots,n_y;\ j = 1,2,\cdots,n_u$$

where ϵ is a very small, e_j is unit vector of jth, and n_u, n_x, $n_{\dot{x}}$, and n_y represent the element number in the input vector in u, x, $\dot{x}$ and y, respectively. In detail A, B, C, D matrices are:

- A is the $n_x \times n_x$ square matrix represents state matrix
- B is the $n_x \times n_u$ column vector represents output matrix
- C is the $n_y \times n_x$ matrix represents output matrix
- D is the $n_y \times n_u$ matrix represents output matrix

Solving nonlinear equations is necessary to determine the air vehicle's trim condition. To adequately describe the motion of an air vehicle, at least 12 states are necessary. These states encompass translational, rotational, kinematic, and navigation equations. In the context of system matrices A, B, C, and D, these represent critical components of the state-space representation used in the mathematical modeling of dynamic systems. Here, the columns of matrices A, B, C, and D denote input and state vectors, while the rows correspond to state derivative and output vectors. This approach allows for a comprehensive modeling of the vehicle's behavior, encompassing its dynamic response and interaction with the environment [57].

The input vector for an air vehicle depends on its type and typically includes elevator, aileron, and rudder deflections, as well as throttle setting. These inputs are crucial for controlling the vehicle's orientation, motion, and overall flight dynamics.

The air vehicle dynamics in terms of input, state and output terms can be defined as

$$\begin{aligned} \vec{x} &= [V_T, \alpha, \beta, \phi, \theta, \psi, p, q, r, p_N, p_E, h]^T \\ \vec{y} &= \left[a_z, a_y, \gamma, p_v, q_v, r_v\right]^T \\ \vec{u} &= [\delta_{th}, \delta_e, \delta_a, \delta_r]^T \end{aligned} \tag{4.85}$$

The trim point is equivalent to the equilibrium point of the nonlinear system which is stated as x_{eq}. In linear domain, using state space representation, this mentioned equilibrium point, in other words, trim condition, is defined as state variables are

equals to zero $\dot{x} = 0$, with the control input arrangement $u_{eq} = constant$ making steady-state equilibrium possible. Finding a trim condition solution for an aircraft generally requires numerical methods due to the complexity of the equations involved.

In this thesis, the Newton-Raphson method is used initially employed to determine the trim condition. Linearization is required to apply the Newton-Raphson method efficiently; so that the linearization process is illustrated in the form of a flow chart in Figure 4.6 [58] [59] [60] [61]. The small perturbations are chosen as fractions of the equilibrium values and can be defined as predefined values with an equilibrium value which is equal to zero [57].

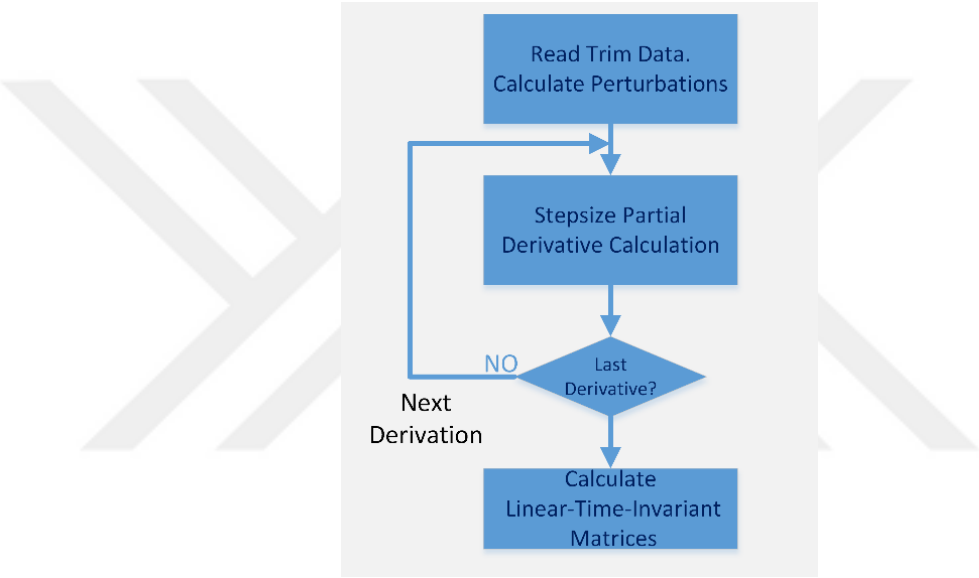


Figure 4.6 : Linearization procedure [48].

The linearized steady-state equations of motion are described in matrix form using the Jacobian matrix can de denoted as follows

$$\begin{bmatrix} \dot{V}_T \\ \dot{\alpha} \\ \dot{\beta} \\ \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \\ \dot{p} \\ \dot{q} \\ \dot{r} \\ \dot{p_N} \\ \dot{p_E} \\ \dot{h} \end{bmatrix} = \begin{bmatrix} \frac{\partial \dot{V}_T}{\partial V_T} & \frac{\partial \dot{V}_T}{\partial \alpha} & \cdots & \frac{\partial \dot{V}_T}{\partial h} \\ \frac{\partial \dot{\alpha}}{\partial V_T} & \frac{\partial \dot{\alpha}}{\partial \alpha} & \cdots & \frac{\partial \dot{\alpha}}{\partial h} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \dot{h}}{\partial V_T} & \frac{\partial \dot{h}}{\partial \alpha} & \cdots & \frac{\partial \dot{h}}{\partial h} \end{bmatrix} \begin{bmatrix} V_T \\ \alpha \\ \beta \\ \phi \\ \theta \\ \psi \\ p \\ q \\ r \\ p_N \\ p_E \\ h \end{bmatrix} + \begin{bmatrix} \frac{\partial \dot{V}_T}{\partial \delta_e} & \cdots & \frac{\partial \dot{V}_T}{\partial \delta_{th}} \\ \frac{\partial \dot{\alpha}}{\partial \delta_e} & \cdots & \frac{\partial \dot{\alpha}}{\partial \delta_{th}} \\ \vdots & \ddots & \vdots \\ \frac{\partial \dot{h}}{\partial \delta_e} & \cdots & \frac{\partial \dot{h}}{\partial \delta_{th}} \end{bmatrix} \begin{bmatrix} \delta_e \\ \delta_a \\ \delta_r \\ \delta_{th} \end{bmatrix} \quad (4.86)$$

4.3.2 Trim algorithm and linearization

The Newton-Raphson method stands as a robust and widely used numerical technique for solving equations based on the principle of linear approximation, can be expressed in general form:

$$\vec{F}\left(\vec{\Theta}\right) = 0 \tag{4.87}$$

4.3.3 Steady-state flight conditions

The steady-state flight is important since it is the starting condition for flight simulation and a flight condition in which the aircraft is linearized. In the case of flight dynamics, steady-state flight is defined as a state in which all force and moment components in the body-fixed coordinate system are constant or zero. In addition, the aerodynamic angles and angular velocity components must be constant with zero derivative.

4.3.3.1 Straight and level flight

For straight and level flight is a fundamental flight condition where an aircraft maintains a constant altitude and heading as well as moving along a horizontal path without climbing, descending, or turning as seen in Figure 4.7.

In this flight condition, the lift force exactly balances the weight of the aircraft, preventing any vertical acceleration. Similarly, the thrust from the engines matches the drag force, resulting in a constant speed, while the aircraft maintains a steady heading. This is achieved with the constraints in angular rates $p, q, r \equiv 0$, which states that Euler angles $\dot{\phi}, \dot{\theta}, \dot{\psi} \equiv 0$. Also, steady heading meaning that $N_y = 0$ [57].

Then, the problem of calculating the trim variables are outlined below

$$\dot{V_T}, \dot{\alpha}, \dot{\beta}, \dot{p}, \dot{q}, \dot{r}, N_y, \gamma \equiv 0 \tag{4.88}$$

In order to find the independent variables, which must be adjusted during the trim calculation problem is adjusted as

$$\delta_e, \delta_a, \delta_r, \delta_{th}, \alpha, \beta, \phi, \theta \tag{4.89}$$

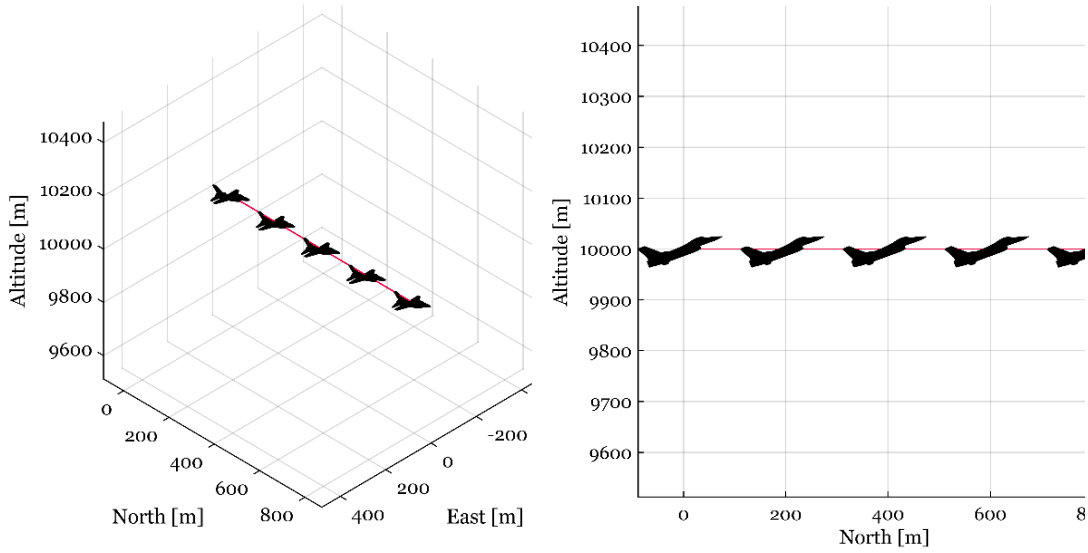


Figure 4.7 : Straight and level flight [48].

4.3.3.2 Steady climb or descent

Trim condition for climb or descent is achieved during the air vehicle is flying steady-state, which the aircraft sustains a constant flight-path angle. For climb conditions, the vertical flight-path angle tends to be positive ($\gamma > 0$) direction, indicating an upward trajectory, while for descent conditions, the vertical flight-path angle is set to be negative ($\gamma < 0$), indicating a downward trajectory as depicted in Figure 4.8.

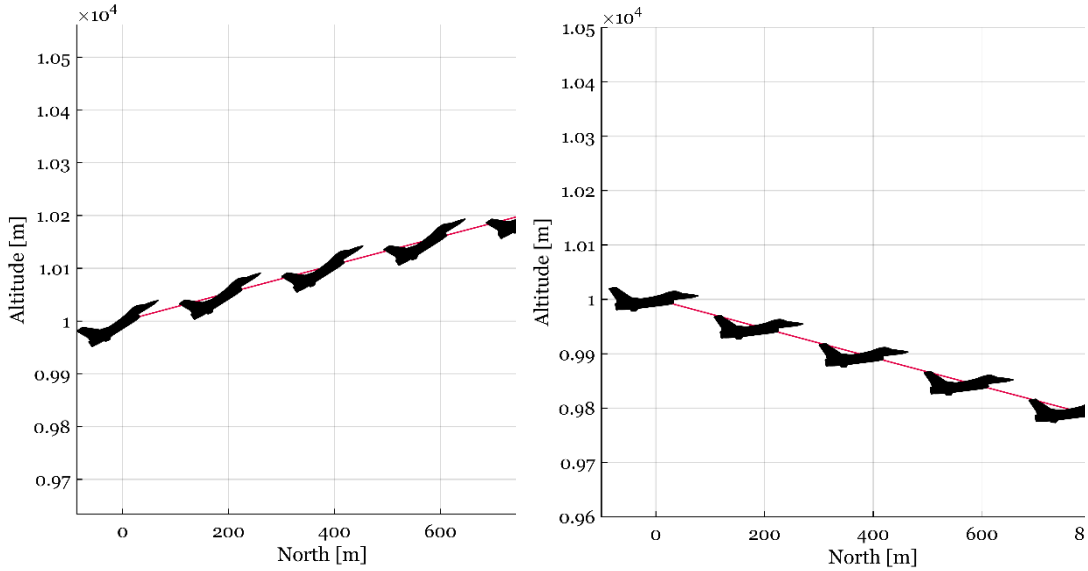


Figure 4.8 : Steady climb and descent, left and right, respectively [48].

4.3.3.3 Steady turn (coordinated turn)

Steady-state turning refers a maneuver where an aircraft changes its direction while maintaining a constant altitude and speed, without climbing or descending. Coordinated turn implies that the turn is performed such that the aircraft's flight path remains a smooth, $\gamma = 0$, coordinated curve, without any skidding or slipping as can be seen in Figure 4.9.

During a coordinated steady turn, several key principles of aerodynamics and flight dynamics come into play. The aircraft's bank angle ϕ is adjusted using the ailerons δ_a, which causes the aircraft to roll to one side. This rolling motion tilts the lift vector of the wings, creating a horizontal component of lift that pulls the aircraft into the turning state. The elevator is then used in order to adjust the pitch and maintain a constant altitude during the turn [57].

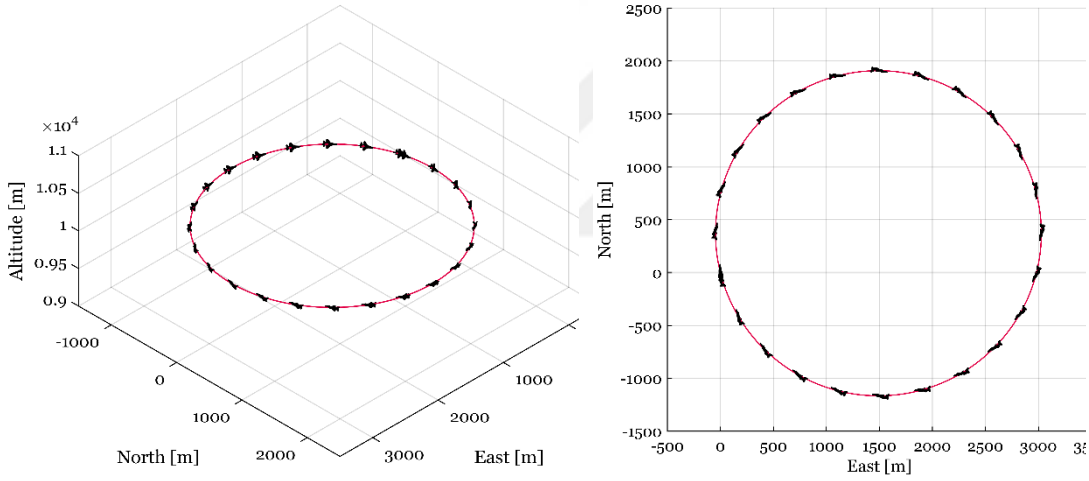


Figure 4.9 : Steady level turning flight [48].

5. FLIGHT CONTROL DESIGN AND ARCHITECTURE

The Flight Control System (FLCS) is a complex network of equipment and software that allows the pilot to maneuver the aircraft safely and effectively for aircraft. FLCS is responsible for stability and control of aircraft through the use of control surfaces (ailerons, elevators, rudder), trim devices, and other actuators such as leading or trailing edge flaps. Its functions can be can be divided into three essential categories primary, secondary flight controls and autopilot functions as depicted in Table 5.1.

Table 5.1 : Functions of a flight control system.

Category	Function	Control Surface
Primary Flight Controls	Controls the aircraft in its pitch, roll, and yaw axis	Ailerons, Elevators, Rudder
Secondary Flight Controls	Manages additional stability and performance	Flaps, Slats, and Spoilers
Autopilot	Provides automatic control based on predefined flight parameters	Dedicated or combined primary and/or secondary flight controls

The Flight Control System (FLCS) consists of several key components from both a hardware and functional perspective. These components include flight control actuators, sensors, and the flight control computer. Flight control system effectors or mainly actuators are responsible for translating the commands from the flight control system into physical movements of the aircraft control surfaces. Flight control sensors such as inertial measurement unit (IMU) play a critical role in providing real-time data on the aircraft's attitude, rate and acceleration, speed, and other sensitive parameters to the flight control system It processes the measurement data from sensors and commands to calculate the appropriate control outputs for maintaining stable flight and executing pilot inputs. Together, these components form a crucial part of the aircraft's overall flight control system, ensuring safe and effective operation during all phases of flight. FLCS generally designed as redundant systems in order to provide safe and effective operation of any failure event.

Flight Control Law (FCL) algorithms are the heart of the flight control system. They determine how the control surfaces and other actuators move in response to pilot commands and environmental conditions based on its architecture such as proportional-integral-derivative (PID), linear state-space, or nonlinear controller design. FCL development Requirements Analysis The development process of Flight Control Laws (FCL) involves several stages. First, requirements analysis defines the desired behavior of the aircraft across different flight conditions. Mathematical models are then created to represent the aircraft's dynamics, followed by simulation to validate and refine these control laws. Once validated, the control laws are implemented into software code for execution by flight control computers. Extensive testing and validation are conducted to ensure the correct and safe operation of the control laws under various conditions. Also, safety, reliability, and certification standards are essential considerations during development of flight control laws in order to assure that the developed software is safe, reliable, resist the failures and can be traced back to the initial requirements.

Flight control modes, including manual (direct) mode for direct pilot control, assisted mode with automation for stability and parameter maintenance, and autopilot mode for full automation based on predefined plans, are essential for aviation safety and efficiency. Manual mode allows pilots precise control during critical phases, while assisted mode reduces workload and improves stability. Autopilot mode enhances safety by following precise flight plans. Together, these modes ensure pilots can effectively manage varying flight conditions, contributing to safe and efficient flight operations.

Embedded software in flight control systems provides real-time processing, safety and reliability and certification compliance. This part will be detailed in Chapter 6.

Developing flight control systems and flight control law algorithms is a rigorous process that requires expertise in control theory, software engineering, and aviation safety. These systems are vital for maintaining the safe and efficient operation of aircraft, and their development must hold to stringent certification standards to provide airworthiness and safety. In order to achieve required stability and control performance, FCL algorithm is chosen to Nonlinear Dynamic Inversion in this study and will be investigated latter chapters.

5.1 Flight Control Algorithm Design Requirements

In flight control system design, after specifying design requirements and performance objectives, the design is expected to meet the desired dynamics. These dynamics are determined according to both military and civil aviation standards. Also, the importance of evaluating their flying qualities has grown with the fly-by-wire flight control systems. These flying qualities determine whether the aircraft can successfully perform defined mission tasks. All these design and performance requirements are specified within the military standard MIL-F-8785C (Flying Qualities of Piloted Airplanes) [62]. The following subsections outline these requirements and the corresponding design standards.

To evaluate the performance of an aircraft, the mission tasks it will perform during flights need to be determined. Different aircraft classes are designed for specific purposes. Some aircraft are intended to carry payloads, while others are designed for combat, bomb delivery, or as fuel tankers to support other aircraft. These specific purposes define the requirements for flight control algorithms. Consequently, each type of aircraft has its own evaluation criteria that must be met. There are four main aircraft classes, each with common purposes related to their mission tasks during flights, as shown in the Table 5.2.

Table 5.2 : Aircraft classes.

Airplane Class	Specification
Class I	Small, light airplanes
Class II	Medium weight, low-to-medium-maneuverability airplanes
Class III	Large, heavy, low-to-medium-maneuverability airplanes
Class IV	High-maneuverability airplanes

Class I aircrafts are defined as small, light airplanes, including light utility, primary trainer, and light observation planes. Class II aircrafts are medium-weight, low-to-medium-maneuverability airplanes, such as antisubmarine, reconnaissance, tactical bomber, and heavy attack aircraft. Class III airplanes are large, heavy, low-to-medium maneuverability aircraft, including heavy bombers, heavy transport, cargo, tankers, and trainers for Class III. Class IV aircraft are high-maneuverability aircraft, such as fighter-interceptors, attack, and observation planes [62].

In this study, Class IV aircraft standards which are suitable for control law design for the F-16 aircraft are carried out.

Flight phase refers to the description of flight handling quality levels. An aircraft performs various tasks during its mission, from take-off to landing, each requiring different desired characteristics and corresponding definitions of satisfactory response. To evaluate an aircraft's performance during different mission tasks, numerous flight phases are categorized into three main flight phase categories based on their similar flight characteristics, as shown in the Table 5.3.

Table 5.3 : Flight phase categories.

Flight Phase Category		Specification
Cat A	High Maneuvering	Phases involving high maneuvering, such as air combat, aerobatics, or other high agility operations.
Cat B	Medium Maneuvering	Phases involving moderate maneuvering, including formation flying, tactical maneuvers, and some air-to-air and air-to-ground operations
Cat C	Low Maneuvering	Phases that involve low maneuvering, like approach, landing, takeoff, and low-level navigation.

Three flying quality levels are specified in order to ensure that military aircraft meet essential handling characteristics across various operational conditions. Level 1 qualities are the most desirable, ensuring that the aircraft can perform its intended tasks with minimal pilot effort. Level 2 qualities are still acceptable but may result in higher pilot workload, especially under challenging conditions. Level 3 qualities represent the minimum acceptable standards, requiring significant pilot skill and concentration, suitable only for extreme operational scenarios that can be seen in Table 5.4.

Table 5.4 : Flying quality levels.

Flying Quality Level		Specification
Level 1	Adequate Performance	Aircraft handling qualities are adequate for the assigned tasks without requiring exceptional skill or concentration from the pilot. These are the desired flying qualities for normal operations
Level 2	Satisfactory Performance	Aircraft handling qualities are satisfactory, but may require moderate pilot skill and concentration. These qualities are acceptable but not optimal and may cause increased workload during demanding conditions
Level 3	Minimum Performance	Aircraft handling qualities are marginally acceptable and require considerable pilot skill and concentration. These are the lowest acceptable qualities for safe operation under extreme conditions.

These levels above help ensure that military aircraft maintain safety and performance standards across diverse missions and conditions. Based on the above information, it is possible to classify F-16 fighter aircraft as Class IV, Category A, and Level 1. This classification helps determine the required boundaries for the aircraft's dynamics.

5.2 Stability and Control Augmentation

Stability and Control Augmentation Systems (SCAS) are essential for fly-by-wire (FBW) aircraft for several reasons. FBW aircraft, especially modern high-performance ones, are often designed to be aerodynamically unstable to achieve better maneuverability and efficiency. SCAS helps maintain the necessary stability for safe flight by automatically adjusting control surfaces to counteract instability by designed separately for the longitudinal and the lateral-directional axes decoupling the aircraft dynamics in most flight conditions.

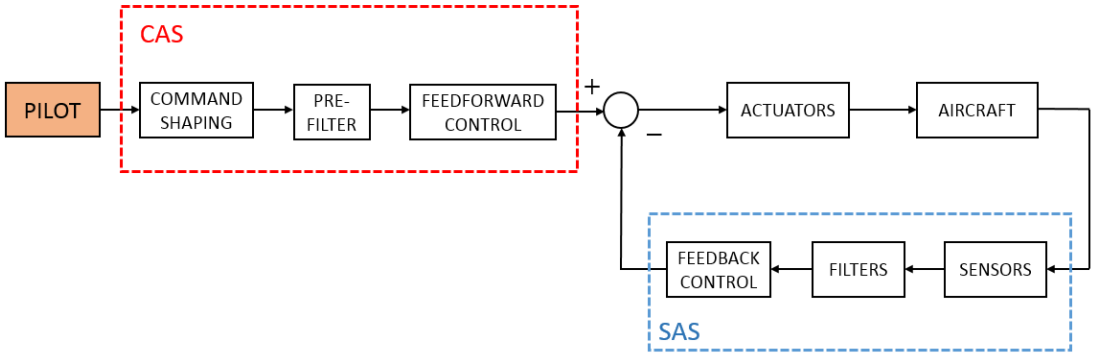


Figure 5.1 : Fly-by-wire flight control SCAS systems [63].

SCAS reduces the pilot's workload by providing automatic adjustments and corrections, allowing pilots to focus more on mission objectives and less on maintaining stable flight, which is particularly important in complex or high-stress situations. SCAS ensures consistent and predictable handling characteristics across a wide range of flight conditions. This results in smoother and more precise control, enhancing the overall flying experience and operational effectiveness as depicted in Figure 5.1.

It provides critical support in preventing dangerous flight conditions, such as stalls or spins, by making continuous, rapid adjustments that are beyond human reaction times. This contributes significantly to the overall safety of the aircraft and its occupants. Aircraft can operate closer to its performance limits with greater safety with SCAS,

the allowing for enhanced agility and responsiveness that are crucial for modern military and commercial operations.

In essence, SCAS is vital for FBW aircraft to comply with the stringent stability, control, workload, handling quality, and safety requirements established by MIL-STD-8785. These systems help transform the inherent design characteristics of FBW aircraft into operational realities that meet the standard's comprehensive criteria for flying qualities.

5.2.1 Longitudinal stability augmentation

Pitch-axis SAS enhances the longitudinal stability of aircraft by providing satisfactory natural frequency and damping particularly for the short-period mode. Figure 5.2 shows that this augmentation system typically employs sensors to detect deviations from desired flight attitudes, which are angle of attack and pitch rate, and actuates control surfaces or adjusts control laws by feedback to counteract these deviations promptly in order to modify the frequency and damping. By providing automatic corrections, pitch-axis stability augmentation systems contribute significantly to the overall stability and controllability of the aircraft.

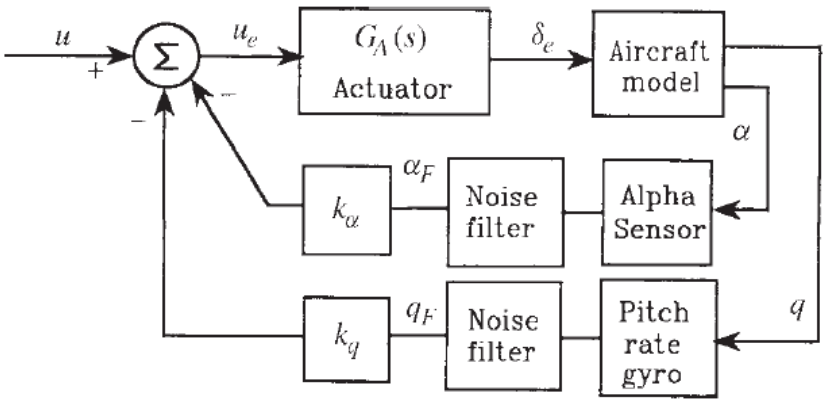


Figure 5.2 : Longitudinal-axis stability augmentation [59].

In longitudinal SAS, pitch-rate feedback is used if the short-period mode is lightly damped but otherwise adequate. However, if both the frequency and damping are unsatisfactory or if the mode is unstable, alpha feedback becomes necessary [59].

5.2.2 Longitudinal control augmentation

Pitch-axis control augmentation is an advanced flight control system designed to improve the precision and responsiveness of an aircraft's pitch movements. This system utilizes sophisticated sensors and algorithms to monitor the aircraft's pitch attitude and make continuous, real-time adjustments to the control surfaces, such as the elevators. By enhancing the pilot's ability to control the pitch axis with greater accuracy and less effort, pitch-axis control augmentation systems contribute to improved handling qualities, smoother flight paths, and reduced pilot workload. These systems are particularly crucial in modern fly-by-wire (FBW) aircraft, where they ensure that the aircraft meets rigorous performance and safety standards by providing consistent and reliable control responses under varying flight conditions.

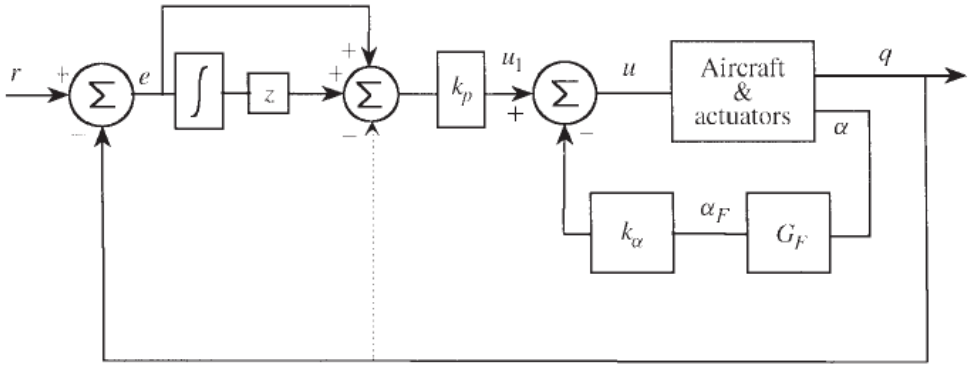


Figure 5.3 : Longitudinal-axis control augmentation [59].

An example structure of the pitch rate control augmentation system is shown in Figure 5.3. Using the SAS, which includes angle of attack feedback in the inner loop, the aircraft dynamics are stabilized, and then a sequential loop is created with pitch rate feedback, which is fed to the elevator control surface actuator. A PI (Proportional-Integral) type controller structure is used to generate the pitch rate command to be applied.

5.2.3 Lateral-directional stability augmentation

Lateral-directional stability augmentation uses body-axis roll rate feedback to the ailerons to modify the roll subsidence mode, and rudder feedback for yaw rate in order to modify the Dutch Roll mode as seen in the Figure 5.4.

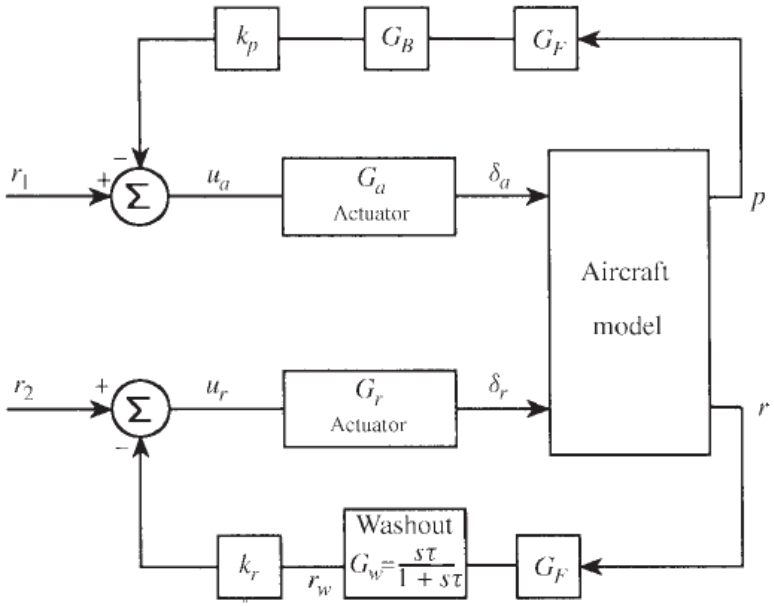


Figure 5.4 : Lateral-directional stability augmentation [59].

This system integrates advanced sensors, actuators, and control algorithms to continuously monitor and correct deviations in lateral and directional stability. The primary objective of designing the lateral-longitudinal axis stability augmentation system is to ensure that the damping and frequency of the Dutch Roll are adjusted as desired and to prevent any loss in the aircraft's roll performance. Aileron effectiveness changes significantly, especially at high angles of attack. This leads to a decrease in roll performance at high angles of attack. The roll rate (p) feedback structure is created to prevent potential losses in the aircraft's roll performance under varying flight conditions as well as the yaw rate (r) feedback loop is created to generate a yaw moment with a damping effect to counteract any yaw rate resulting from the Dutch Roll (DR) mode.

5.2.4 Lateral-directional control augmentation

At high angles of attack, undesirable effects such as kinematic and inertial coupling arise from the aircraft rotating axis, significantly degrading the aircraft's performance.

Kinematic coupling occurs when, during a rapid roll maneuver at a high angle of attack, this angle of attack transforms into sideslip. The sideslip resulting from kinematic coupling is referred to as adverse sideslip, and this effect opposes the roll. Due to the inability to maintain the angle of attack, the aircraft will face an inefficient attempt when it needs to make a new turn. Therefore, the designed control system aims

to maintain the initial angle of attack of the aircraft. Additionally, the large amount of sideslip resulting from kinematic coupling significantly reduces the effectiveness of the control surfaces. This is another reason why it is desirable to eliminate kinematic coupling. Another undesirable effect encountered during aircraft roll is inertia coupling. This effect causes a pitching moment to occur simultaneously during rapid roll maneuvers at high angles of attack.

As mentioned earlier, control surface effectiveness tends to significantly decrease at high angles of attack since the roll responses of aircraft are often deemed unacceptable. Control systems typically limit the commanded roll rate to prevent departure in the pitching axis as seen in Figure 5.5.

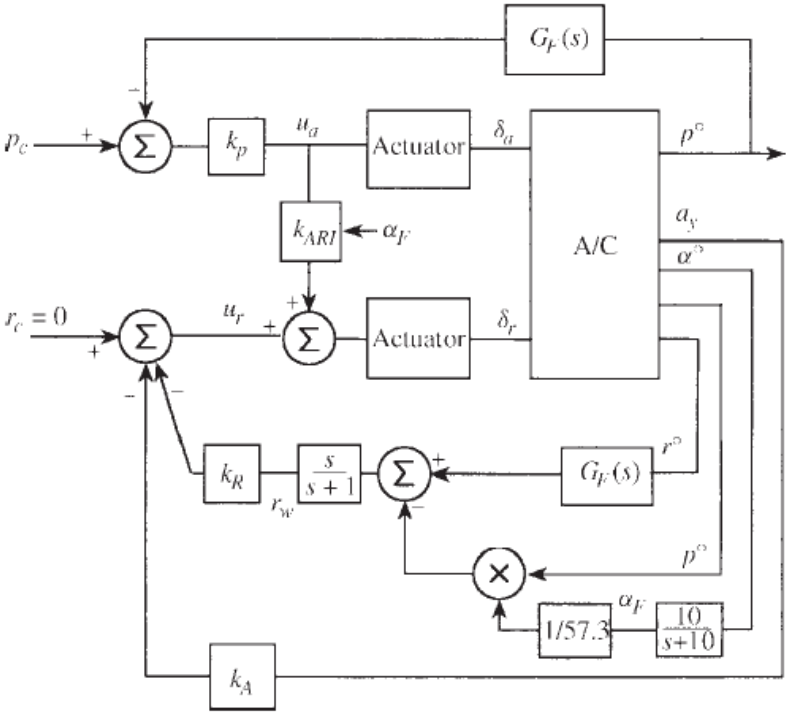


Figure 5.5 : Lateral-directional control augmentation [59].

At the high angle of attack regions, nonlinear dynamic inversion control law is applied in this study in order to control the aircraft.

5.3 Nonlinear Dynamic Inversion

Nonlinear Dynamic Inversion (NDI) is an advanced control method used in the field of aerospace engineering and other applications involving complex nonlinear systems such as mechanical system design, robotics and vehicle control. The fundamental concept underlying the NDI is to simplify the control of nonlinear systems by

effectively inverting their nonlinear dynamics into a linear dynamic. NDI is used to manage the behavior of nonlinear systems by algebraically canceling out the nonlinearities in the system dynamics. This enables the application of linear control methods to achieve desired performance and stability, particularly in complex, nonlinear scenarios such flight maneuvers at high angles of attack. These maneuvers demand accurate predictions of aircraft dynamics and rely on advanced control mechanisms guided by complex algorithms to counterbalance the reduced stability of the airframe. NDI control method follows basic steps including modeling the system dynamics, inverting the dynamics linear control design and if required sensitivity and robustness analysis.

The Nonlinear Dynamic Inversion (NDI) approach differs from conventional linearization methods, such as those based on the Jacobian approach. NDI algebraically converts the dynamics of the nonlinear system into a linear dynamic form, facilitating the utilization of linear control methods. This transformation is achieved through feedback linearization, where the state transformation is exactly taken into account. The objective is transforming the original nonlinear system model into a corresponding equivalent model into a simplified and linear model, facilitating the application of linear control laws. This approach allows for more effective handling of complex, nonlinear dynamics in systems like aircraft, where precise control is essential for stability and performance.

The NDI controller necessitates precise understanding of the system dynamics to achieve optimal performance, making robustness crucial during the design process. However, in real-world applications, precise knowledge of the dynamics of the system is often unavailable due to parameter uncertainties and unmodeled dynamics. The presence of unmodeled dynamics is particularly significant, as the exact model of the system is never fully attainable in practice. Furthermore, the impact of modeling errors can be considerable, especially when the linearizing transformation is not well-conditioned. These factors highlight the importance of robust control strategies within the NDI framework to ensure stability and effective performance across varying operational conditions.

5.3.1 Nonlinear feedback linearization overview

As stated in 0, the nonlinear feedback linearization (NFL) strategy aims to simplify a nonlinear system by canceling out its nonlinear terms, thereby rendering it more manageable. If the fundamentals of control systems are revisited, in linear control theory dynamical systems have different state-space representations. The controllable canonical form as a dynamical system can be shown as

$$\dot{x} = Ax + Bu$$

$$\xi = \xi(x)$$

$$[\dot{\xi}] = \begin{bmatrix} 0 & 1 & \dots & 0 \\ 0 & 0 & \ddots & \vdots \\ 0 & 0 & \dots & 1 \\ -a_0 & -a_1 & \dots & -a_{n-1} \end{bmatrix} [\xi] + \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix} [u] \tag{5.1}$$

where is x is state vector with n-dimension, u is control input, *A* and *B* are LTI state and control matrices, ξ is transformation matrix and $a_0, a_1 \dots a_n$ are characteristic polynomial coefficients of the system. The transformation yields

$$\xi = [\xi_1 \quad \xi_2 \quad \cdots \quad \xi_n]^T = \left[\xi_1 \quad \dot{\xi}_1 \quad \dots \quad {\xi_1}^{(n-1)}\right]^T \tag{5.2}$$

In addition, the relation between input vector u and transformation vector ξ

$${\xi_1}^{(n)} + a_{n-1}{\xi_1}^{(n-1)} + \cdots + a_0\xi_1 = u \tag{5.3}$$

It is observed that the controllable canonical form results in a system representation where the state vector is described exclusively by the first component of the transformation vector. ξ_1 and n^{th} order state equation is substituted by a single differential equation in (5.3) that allows the calculation of the associated control input for the desired state.

NFL extends this idea to nonlinear systems following

$$\dot{x} = f(x) + g(x)u$$

$$\xi^{(n)} = f(\xi \quad \dot{\xi} \quad \cdots \quad \xi^{(n-1)}, t) + g(\xi \quad \dot{\xi} \quad \cdots \quad \xi^{(n-1)}, t)u \tag{5.4}$$

$$\xi^{(n)} \triangleq \vartheta$$

A controllable canonical form of a system, such that the equations contain state derivatives without the input derivatives. It can be represented as in (6.1), nonlinear system with a single input is aforehand and vector transformation is performed in (5.4).

If the right-hand side of the entire nonlinear equation (5.4) is combined into a single variable, the state equation will be linearized, enabling the application of linear control theory. However, achieving the input u would require knowledge of f and g. Finally, ϑ rely on the controller design of this of this linearized system in (5.5) and nonlinear state equation are linear with respect to input u, so feedback is linearized.

$$\begin{aligned} \xi^{(n)} &= \vartheta \\ u &= g^{-1}(\vartheta - f) \end{aligned} \tag{5.5}$$

5.3.2 Dynamic inverse controller design

Nonlinear feedback linearization deviates from the traditional linearization method by achieving a quantitatively identical dynamic system representation, as opposed to an approximation. In general, the aircraft dynamics are expressed by

$$\begin{aligned} \dot{x} &= f(x) + g(x)u \\ x &= \left[x \ \dot{x} \ \cdots \ x^{(n-1)}\right]^T \\ y &= h(x) \end{aligned} \tag{5.6}$$

where $x \in \Re^n$ state vector, $u \in \Re^m$ is control vector, y $\in \Re^m$ is output vector, $r \leq n$ which implies that the relative degree of a system could be equal to or less than the full order of the system, $f(x)$ and $g(x)$ are nonlinear state dependent vectors.

If the $g(x)$ for state x values, the DI control structure is derived by subtracting $f(x)$ from state x and multiplying $g(x)^{-1}$ as follows

$$u = g(x)^{-1}(\dot{x} - f(x)) \tag{5.7}$$

After deriving the control law, the subsequent step involves directing the air vehicle to the specified states. Rather than immediately specifying the desired states, the rates of the desired states are determined. as $\dot{x}$. By substituting state derivatives in equation (5.6) with the commanded state values $\dot{x}_c$, the dynamic inversion control law is finalized and derived as follows:

$$u_c = g(x)^{-1}(\dot{x}_c - f(x)) \tag{5.8}$$

Figure 5.6 depicts the block diagram representation of the DI process.

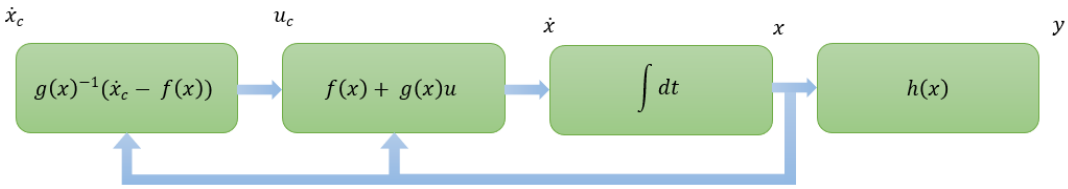


Figure 5.6 : Dynamic inversion process [26].

The direct relationship between the output y and the input u is obtained by differentiating the output y until the control input u is seen in the output expression explicitly. In the case of investigated aircraft, a single differentiation is sufficient to achieve this explicit relationship as following

$$y = h(x)$$

$$\dot{y} = \frac{\partial h(x)}{\partial x}\dot{x} = \frac{\partial h(x)}{\partial x}f(x) + \underbrace{\frac{\partial h(x)}{\partial x}g(x)}_{\Delta}u \tag{5.9}$$

If Δ is invertible for all values of states x , then the linearization using the inverse dynamics by applying the following transformation

$$u = \left(\frac{\partial h(x)}{\partial x}g(x)\right)^{-1}\left[r - \frac{\partial h(x)}{\partial x}f(x)\right] \tag{5.10}$$

This transformation converts equation (3.3) into the following simple and linear form

$$\dot{y} = r \tag{5.11}$$

Here, r represents the control vector, which is applied as the command vector n on $\dot{y}$ o ensure that the output y tracks a desired output vector y_d.

The control vector r can be provided to achieve the desired motion for the aircraft by proportional (P) control laws as following

$$r = \omega_d(y_d - y) \tag{5.12}$$

where, ω is the proportional of the closed loop dynamics.

After determining the control vector r, the state vector x must be completely available, and Δ in (5.9) should be invertible for all values of x to provide the actual control u considering the equation (5.10)

Although the process of dynamic inversion is straightforward, there are several key topics that require emphasis. It is assumed that $g(x)$ is invertible for all values of x. However, it should be noted that if there are more states than controls, $g(x)$ is not invertible, the control inputs u may increase significantly, potentially resulting in saturation of actuators.

The dynamics of actuators and sensors in the feed-forward loop, as well as noise resulting from sensors in the feedback loop, are ignored during the control law development process.

The DI controller requires precise knowledge of the system and subsystem dynamics in order to acquire optimal performance. Consequently, robustness points are critically important in the design process. Generally, the DI controller is used as an inner loop controller in conjunction with an outer loop controller designed using advanced control design techniques to overcome these challenges [64].

5.3.3 Two-time scale method

A two-time-scale method has been devised and proven highly effective in resolving singularity issues encountered during the inversion of an ineffective control matrix $g(x)$. This method has also been demonstrated to be useful in inverting wind-related motion variables as well as body rates.

In aircraft control, the relatively limited effectiveness of control on the dynamics of variables like angle of attack and angle of sideslip, and body attitudes such as roll, pitch, and yaw are classified as slow dynamics variables. Conversely, the body angular rates (p, q, r) exhibit high control effectiveness and are considered as fast dynamics variables. The two-time-scale method defines a system with a pair of distinct and sequentially connected differential equations to address these dynamics.

$$\begin{aligned} \dot{x}_1 &= f_1(x_1) + g_1(x_1)x_2 \\ \dot{x}_2 &= f_2(x_1, x_2) + g_2(x_1, x_2)u \end{aligned} \tag{5.13}$$

The time rate of change $\dot{x}_{1_c}$ of x_1 is considered as commanded value and computed in s-domain as follows

$$s\dot{X}_{1_c}(s) = G_1(s)\left[X_{1_d}(s) - X_1(s)\right] \tag{5.14}$$

where $X_{1_c}(s)$, $X_{1_d}(s)$, and $X_1(s)$ are the Laplace transform of signals x_{1_c}, x_{1_d} and x_1

Then, the value of $\dot{x}_{1_c}$ is utilized to compute the commanded value of x_{2_c}

$$x_{2_c} = g_1^{-1}(x_1)\left[\dot{x}_{1_c} - f_1(x_1)\right] \tag{5.15}$$

Afterwards, like (5.14) the time rate of change $\dot{x}_{2_c}$ of x_2 , again considered as the commanded value, is calculated in s-domain

$$s\dot{X}_{2_c}(s) = G_2(s)\left[X_{2_d}(s) - X_2(s)\right] \tag{5.16}$$

where $X_{2_c}(s), X_{2_d}(s)$, and $X_2(s)$ are the Laplace transform of signals x_{2_c}, x_{2_d} and x_2.

Finally, $\dot{x}_{2_c}$ is calculated the commanded control deflections u which then serve as the input to the inherent dynamics as follows

$$u_c = g_2^{-1}(x_1, x_2)\left[\dot{x}_{2_c} - f_2(x_1, x_2)\right] \tag{5.17}$$

where $G_1(s)$ and $G_2(s)$ represent the transfer matrices of the controller for the first and second sets of differential equations.

5.3.4 Desired dynamics

Control applications typically use either displacements or rates as command states to regulate the system. However, DI control necessitates the use of acceleration terms can be seen in (5.17). The desired dynamics structure serves as a mapping function that relates the rate commands to the desired acceleration terms, which are essential for the DI equations. The desired dynamics block structure is illustrated in Figure 5.7. The desired dynamics can be formed as following:

- Proportional (P) controller
- Proportional integral (PI) controller
- Flying quality (FQ) controller
- Ride quality (RQ) dynamics

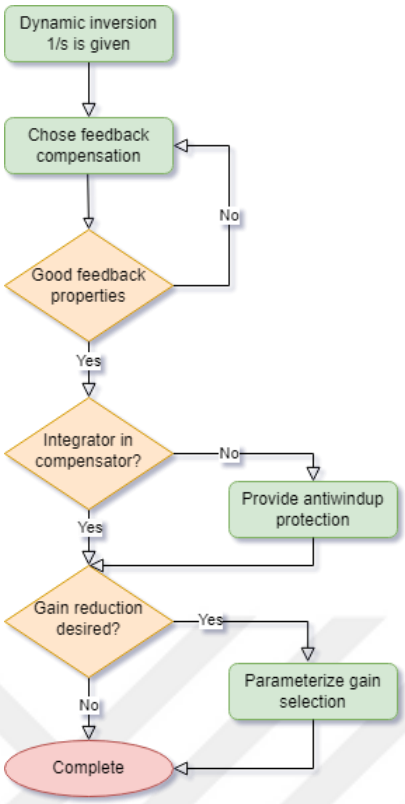


Figure 5.7 : Flowchart for the desired dynamics [25].

The simplest form of desired dynamics is the proportional or first-order dynamics case. In the proportional form, the closed-loop transfer function places a single pole $s = -k_p$ as depicted in (5.18). Constant k_p scales the error between the command of the control variable and its feedback term.

$$\frac{x(s)}{x_d(s)} = \frac{k_p}{s + k_p} \tag{5.18}$$

In addition, the bandwidth must be chosen to fulfill the assumptions for time-scale separation without structural mode excitation or causing the actuator saturation.

The desired dynamics block as seen in Figure 5.8 is not restricted to a first-order component. If the set of first-order equations in the desired dynamics fails to achieve acceptable handling qualities, then a higher-order system is employed.

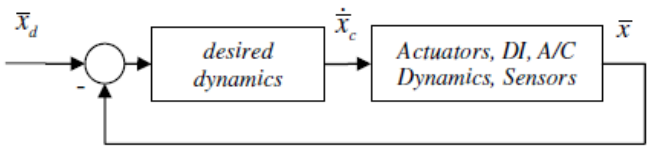


Figure 5.8 : Closed-loop transfer function calculation [26].

5.3.5 Stabilization controller design

There is a substantial difference in time scale between the slow and fast dynamics of the investigated air vehicle in this study. The nonlinear dynamic inversion control law can be implemented in two distinct steps, as explained in Chapter 5.3.3. Using the two-time scale method, the first step involves inverting the slow dynamics from flight path motion inputs α, β, μ to outputs of body rate p, q, r. The subsequent step inverts the fast dynamics from inputs as body rates p, q, r to control surface outputs $\delta_e, \delta_a, \delta_r$ which are then used to design a DI controller. In addition, air vehicle state vector in (5.6) consists of body rates (p, q, r), Euler attitudes such as (ϕ, θ, ψ), and the flight path parameters (α) and (β). The control vector is effector deflections $(\delta_e, \delta_a, \delta_r)$. The output vector is chosen from the state and input variables such as α, β, μ.

$$x_{slow} = \begin{bmatrix} \alpha \\ \beta \\ \mu \end{bmatrix}, \qquad x_{fast} = \begin{bmatrix} p \\ q \\ r \end{bmatrix}$$
$$u = \begin{bmatrix} \delta_e \\ \delta_a \\ \delta_r \end{bmatrix}, \qquad y = \begin{bmatrix} \alpha \\ \beta \\ \mu \end{bmatrix} \tag{5.19}$$

In this study, stabilization controller is designed using NDI controller in order to control the air vehicle during undesired high angle of attack (α) flight regimes using the aerodynamic control surfaces.

At high-alpha flight regimes, stabilization of the aircraft is crucial and it is necessary to first control the body angular acceleration components (p, q, r) to enhance and augment the stability of the air vehicle. After the stability augmentation the wind axis flight components α, β, μ of the air vehicle needs to be stabilized and returned to moderate angles of attack levels so that the aerodynamic controls can adequately stabilize the aircraft in that flight regime.

The stabilization controller is bifurcated into two parts due to the necessity for the desired accelerations that stabilize the aircraft to be directly linked with the body angular velocity and linear velocity components of the air vehicle. Therefore, the designed controller generates acceleration commands based on both the desired and actual body angular and linear velocities. Specifically, one part of the controller generates the commanded angular accelerations by utilizing the desired and actual angular velocities, while another part generates the commanded flight path parameters

using the desired and actual linear velocities along with kinematic relationships. Despite the different controller structures for control the linear dynamics in the literature, the proportional controller structure is used for both the flight path parameters controller and the angular acceleration controller such as stability augmentation in this study.

i. Angular acceleration (inner loop) controller

Using the desired angular velocity components p_d , q_d and r_d the error vector e represents the difference between the desired body angular velocities $(\cdot)_d$, and the actual body angular velocities, i.e.

$$e_{in} = \begin{bmatrix} p_d \\ q_d \\ r_d \end{bmatrix} - \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{5.20}$$

Implementing the Proportional controller, the commanded $(\cdot)_{com}$ angular accelerations $\dot{p}_{com}$, $\dot{q}_{com}$and $\dot{r}_{com}$ can be expressed as

$$\begin{bmatrix} \dot{p}_{com} \\ \dot{q}_{com} \\ r_{com} \end{bmatrix} = K_{in} e_{in} \tag{5.21}$$

ii. Flight path parameters (outer loop) controller

For the flight parameters controller, angles α and β can be directly measured by the air vehicle sensors and have first-hand relationship to pilot control inputs. In addition, bank angle μ is derived using the sensor angle measurements and kinematic relationships. Using these the error vector values, e_{out} is describes as the difference between the desired $(\cdot)_d$ and the actual flight path parameters values as follows

$$e_{out} = \begin{bmatrix} \alpha_d \\ \beta_d \\ \mu_d \end{bmatrix} - \begin{bmatrix} \alpha \\ \beta \\ \mu \end{bmatrix} \tag{5.22}$$

Implementing the Proportional controller, the commanded $(\cdot)_{com}$ flight path parameters $\dot{p}_{com}$, $\dot{q}_{com}$and $\dot{r}_{com}$ can be expressed as

$$\begin{bmatrix} \dot{\alpha}_{com} \\ \dot{\beta}_{com} \\ \dot{\mu}_{com} \end{bmatrix} = K_{out} e_{out} \tag{5.23}$$

The block diagram represents the stabilization controller which uses aerodynamic control surfaces is depicted in Figure 5.9.

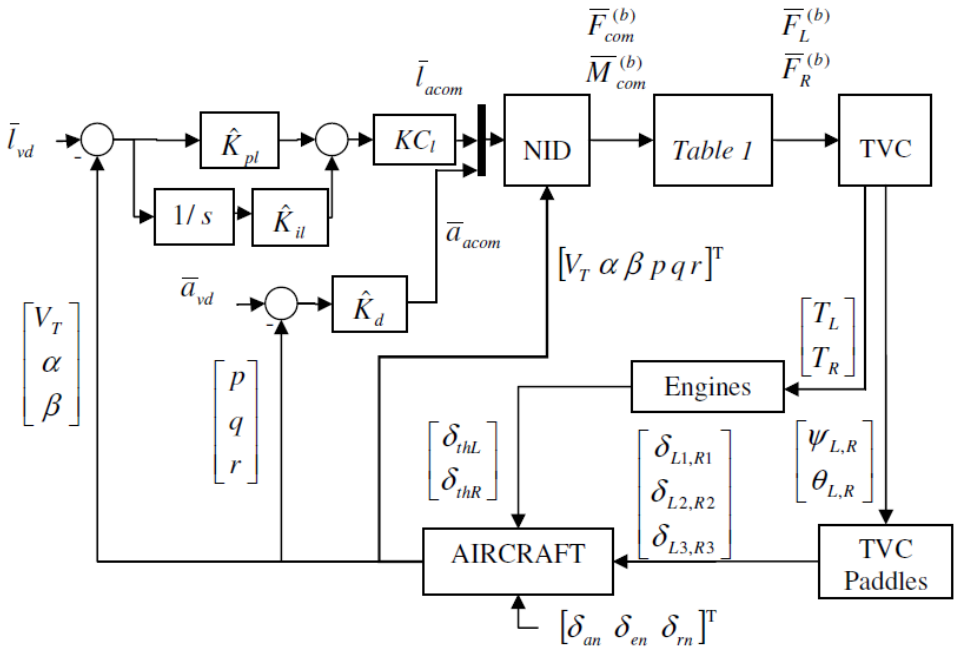


Figure 5.9: Stabilization controller [26].

5.3.5.1 Inner loop design

The inner loop consists of fast dynamic equations that describe the connection between the body rates and the control effector deflections. Noting the Equation (4.32) rotational dynamics equations that contains angular rates are already obtained as follows

$$\begin{bmatrix}\dot{p}\\ \dot{q}\\ \dot{r}\end{bmatrix} = \begin{bmatrix} I_{XX} & 0 & -I_{XZ} \\ 0 & I_{YY} & 0 \\ -I_{XZ} & 0 & I_{ZZ}\end{bmatrix}^{-1} \left\{ \begin{bmatrix} l \\ m \\ n \end{bmatrix} - \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times \begin{bmatrix} I_{XX} & 0 & -I_{XZ} \\ 0 & I_{YY} & 0 \\ -I_{XZ} & 0 & I_{ZZ}\end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \right\} \tag{5.24}$$

employing the dynamic inversion control in conjunction with equations (5.10) and (5.24), aerodynamic moment command values are computed for fast dynamics.

By utilizing the aerodynamic moment coefficients (C_l, C_m, C_n) which are described in Chapter 4.1.4.2 the aerodynamic moments in (4.37) are expressed as

$$\begin{bmatrix} l \\ m \\ n \end{bmatrix} = \bar{q}S \begin{bmatrix} bC_l \\ \bar{c}C_m \\ bC_n \end{bmatrix} \tag{5.25}$$

Writing the equation (5.24) following complex equations are obtained in the form of (5.6) in the following manner

$$\begin{aligned} \dot{x} &= f(x) + g(x)u \\ u &= g(x)^{-1}(\dot{x} - f(x)) \end{aligned} \tag{5.26}$$

Writing (5.24) in a component form separately as following

$$\dot{p} = \frac{1}{I_X I_Z - I_{XZ}^2}\{I_Z l + I_{XZ} n + [I_Z(I_Y - I_Z) - I_{XZ}^2]qr + I_{XZ}(I_X - I_Y + I_Z)pq\}$$

$$\dot{q} = \frac{1}{I_Y}[m + (I_Z - I_X)pr - I_{XZ}(p^2 - r^2)] \tag{5.27}$$

$$\dot{r} = \frac{1}{I_X I_Z - I_{XZ}^2}\left\{I_{XZ} l + I_X n + \left[I_X(I_X - I_Y) + I_{XZ}^2\right]pq - I_{XZ}(I_X - I_Y + I_Z)qr\right\}$$

As noted in 4.1.5.1 and 4.2.1.1, aerodynamic moment coefficients C_l , C_m and C_n are structured as

$$\begin{aligned} C_a = C_a(\alpha) + \Delta C_a(\beta, \alpha) + \Delta C_a(\delta_e, \alpha) + \Delta C_a(\delta_{lef}, \alpha, \beta) \\ + \frac{q\bar{c}}{2V_T}\left\{\left[C_{a_q}(\alpha) + \Delta C_{a_q}(\alpha, \delta_{lef})\right]\right\} \end{aligned} \tag{5.28}$$

$$\begin{aligned} C_a = C_a(\beta) + \Delta C_a(\beta, \alpha) + \Delta C_a(\delta_{lef}, \alpha, \beta) \\ + \left[\Delta C_a(\delta_a, \alpha, \beta) + \Delta C_a(\delta_a, \alpha, \beta, \delta_{lef})\right] + \left[\Delta C_a(\delta_r, \alpha, \beta)\right] \\ + \frac{b}{2V_T}\left\{\left[C_{a_p}(\alpha) + \Delta C_{a_p}(\alpha, \delta_{lef})\right]p \right. \\ \left. + \left[C_{a_r}(\alpha) + \Delta C_{a_r}(\alpha, \delta_{lef})\right]r\right\} \end{aligned} \tag{5.29}$$

The aerodynamic database is organized to separate the input effects and express them independently from flight parameters as follows

$$\begin{aligned} \dot{p} = \frac{1}{I_X I_Z - I_{XZ}^2}\left\{I_Z\left[C_l(\beta) + \Delta C_a(\delta_a, \alpha, \beta) + \cdots + \frac{b}{2V_T}\left[C_{a_p}(\alpha)p + C_{a_r}(\alpha)r\right]\right]\right. \\ + I_{XZ}\left[C_l(\beta) + \Delta C_a(\delta_a, \alpha, \beta) + \cdots + \frac{b}{2V_T}\left[C_{a_p}(\alpha)p + C_{a_r}(\alpha)r\right]\right] \\ \left. + [I_Z(I_Y - I_Z) - I_{XZ}^2]qr + I_{XZ}(I_X - I_Y + I_Z)pq\right\} \end{aligned} \tag{5.30}$$

$$\dot{q} = \frac{1}{I_Y}\left[\left[C_a(\alpha) + \Delta C_a(\delta_e, \alpha) + \cdots + \frac{q\bar{c}}{2V_T}\left\{\left[C_{a_q}(\alpha) + \Delta C_{a_q}(\alpha, \delta_{lef})\right]\right\}\right] + (I_Z - I_X)pr - I_{XZ}(p^2 - r^2)\right]$$

$$\dot{r} = \frac{1}{I_X I_Z - I_{XZ}^2}\left\{I_{XZ}\left[C_l(\beta) + \Delta C_a(\delta_a, \alpha, \beta) + \cdots + \frac{b}{2V_T}\left[C_{a_p}(\alpha)p + C_{a_r}(\alpha)r\right]\right] + I_X\left[C_l(\beta) + \Delta C_a(\delta_a, \alpha, \beta) + \cdots + \frac{b}{2V_T}\left[C_{a_p}(\alpha)p + C_{a_r}(\alpha)r\right]\right] + \left[I_X(I_X - I_Y) + I_{XZ}^2\right]pq - I_{XZ}(I_X - I_Y + I_Z)qr\right\}$$

In the equations (5.30), body angular rates of the air vehicle are expressed detail such that displaying entire terms related to the air vehicle's dynamic motion. These equations can be expressed in matrix format by calculating the aerodynamic moments as following as represented in (5.26) as follows

$$\underbrace{\begin{bmatrix}\dot{p}\\ \dot{q}\\ \dot{r}\end{bmatrix}}_{x} = \underbrace{\begin{bmatrix}f_p(x_1)\\ f_q(x_1)\\ f_r(x_1)\end{bmatrix}}_{f(x)} + \underbrace{\begin{bmatrix}g_{p_{\delta_a}}(x_2) & 0 & g_{p_{\delta_r}}(x_2)\\ 0 & g_{p_{\delta_e}}(x_2) & 0\\ g_{r_{\delta_a}}(x_2) & 0 & g_{r_{\delta_r}}(x_2)\end{bmatrix}}_{g(x)}\underbrace{\begin{bmatrix}\delta_a\\ \delta_e\\ \delta_r\end{bmatrix}}_{u} \tag{5.31}$$

The dynamic inversion technique calculates the desired control surface deflections as

$$\underbrace{\begin{bmatrix}\delta_{a_d}\\ \delta_{e_d}\\ \delta_{r_d}\end{bmatrix}}_{u} = \underbrace{\begin{bmatrix}g_{p_{\delta_a}}(x_2) & 0 & g_{p_{\delta_r}}(x_2)\\ 0 & g_{p_{\delta_e}}(x_2) & 0\\ g_{r_{\delta_a}}(x_2) & 0 & g_{r_{\delta_r}}(x_2)\end{bmatrix}}_{g(x)}^{-1}\left(\underbrace{\begin{bmatrix}\dot{p}_d\\ \dot{q}_d\\ \dot{r}_d\end{bmatrix}}_{u_d} - \underbrace{\begin{bmatrix}f_p(x_1)\\ f_q(x_1)\\ f_r(x_1)\end{bmatrix}}_{f(x)}\right) \tag{5.32}$$

$\dot{p}_d, \dot{q}_d$ and $\dot{r}_d$are computed with proportional controllers with respect to outer loop angular rate commands. Proportional gains (w_p, w_q, w_r) are set as mentioned in (5.21) and set to 5 or 10 rad/s in the inner loop as stated in chapter 5.3.4 [65] [24]. Block diagram of inner loop dynamic inversion control is given in Figure 5.10.

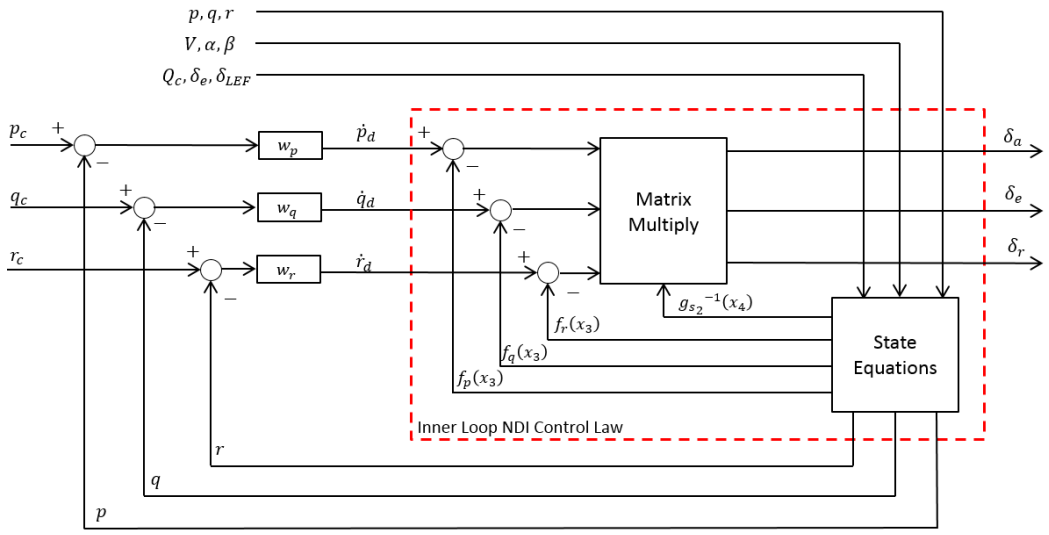


Figure 5.10 : Inner loop: slow dynamics [63].

5.3.5.2 Outer loop design

The outer loop generates the wind axis angle related commands and translates them into body angular rate commands for usage of the inner loop, enabling the aerodynamic control surfaces to achieve the desired control. Equations (4.60) and (4.61) can be represented in the matrix format as represented in (5.26) as

$$\underbrace{\begin{bmatrix}\dot{\alpha}\\ \dot{\beta}\\ \dot{\mu}\end{bmatrix}}_{x} = \underbrace{\begin{bmatrix}f_\alpha(V_T,\alpha,\beta,\mu,\gamma)\\ f_\beta(V_T,\alpha,\beta,\mu,\gamma)\\ f_\mu(V_T,\alpha,\beta,\mu,\gamma)\end{bmatrix}}_{f(x)} + \underbrace{\begin{bmatrix}-t_\beta c_\alpha & 1 & -t_\beta s_\alpha\\ s_\alpha & 0 & -c_\alpha\\ c_\alpha/c_\beta & 0 & s_\alpha/c_\beta\end{bmatrix}}_{g(x)} \underbrace{\begin{bmatrix}p\\ q\\ r\end{bmatrix}}_{u} \tag{5.33}$$

Desired angular rates are calculated by dynamic inversion technique as in the inner loop

$$\underbrace{\begin{bmatrix}p_d\\ q_d\\ r_d\end{bmatrix}}_{u_d} = \underbrace{\begin{bmatrix}-t_\beta c_\alpha & 1 & -t_\beta s_\alpha\\ s_\alpha & 0 & -c_\alpha\\ c_\alpha/c_\beta & 0 & s_\alpha/c_\beta\end{bmatrix}^{-1}}_{g(x)} \left(\underbrace{\begin{bmatrix}\dot{\alpha}_d\\ \dot{\beta}_d\\ \dot{\mu}_d\end{bmatrix}}_{x_d} - \underbrace{\begin{bmatrix}f_\alpha(V_T,\alpha,\beta,\mu,\gamma)\\ f_\beta(V_T,\alpha,\beta,\mu,\gamma)\\ f_\mu(V_T,\alpha,\beta,\mu,\gamma)\end{bmatrix}}_{f(x)} \right) \tag{5.34}$$

In a manner akin to that employed for the fast dynamics in inner loop, flight path parameters are chosen as virtual control inputs in (5.34) in the aim of linearizing the feedback. $\dot{\alpha}_d$, $\dot{\beta}_d$ and $\dot{\mu}_d$ are calculated with proportional controllers considering the pilot commands in the outer loop. Proportional gains $(\omega_\alpha, \omega_\beta, \omega_\mu)$ are adjusted in the fast dynamics outer loop as stated in chapter 5.3.4 [53]. Dynamic inversion control in fast dynamics outer loop is illustrated Figure 5.11.

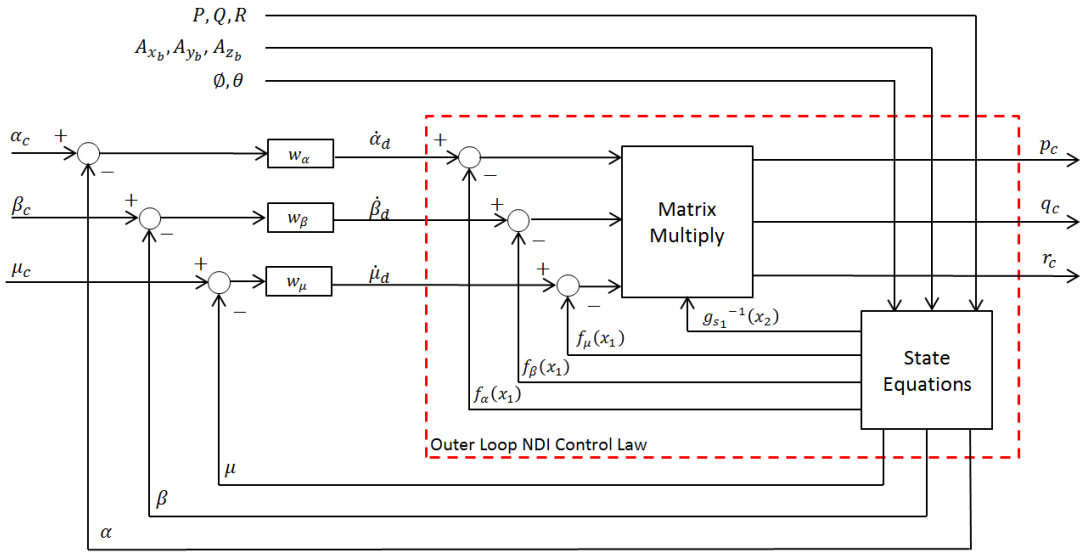


Figure 5.11 : Fast dynamics scheme [54].

The holistic framework of the dynamic inversion control is portrayed through the amalgamation of outer and inner loop controller configurations, as delineated in Figure 5.12. The dashed line represents the virtual control loop that generates the required commands from the pilot and it incorporates a low-pass filter in order to generate an appropriate pilot signal.

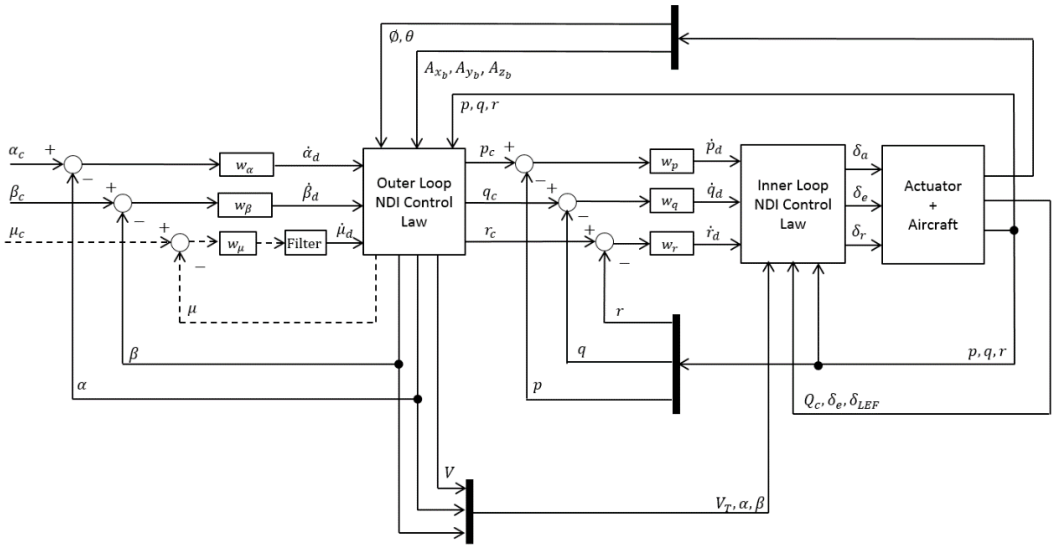


Figure 5.12 : NDI control block diagram [54].

6. FLIGHT CONTROL SOFTWARE DESIGN USING DO-178C

In this study, the design of a flight control system has been carried out in accordance with the DO-178C guidance document and the DO-331 model-based design approach, which are mandated by authorities to certify the software at Level A, as detailed in previous sections. Software development, verification, and traceability activities have been implemented through a model-based design approach.

In software development, it is important to define the boundaries between the system and software for the essential flight control system functions, such as control laws in this study. It provides top level guidance on how certain aspects of the FLCS software will be developed and verified using MBD in accordance with DO-178C and the associated MBD supplement DO-331. Information provided in this chapter creates a basis from which the software development plan (SDP) and software verification plan (SVP) for the software using abovementioned approach is developed besides the detailing the software life cycle processes such as planning, other aspects of integral processes including configuration management briefly. Software configuration item (SCI) defines the flight control law throughout in this chapter.

Within the scope of the software life cycle, the MATLAB Project tool is used to properly advance the software development phase, using a model-based design approach for the air vehicle under consideration as seen in Figure 6.1. Software configuration control is maintained by continuously tagging versions in the Git environment, ensuring configuration traceability from the beginning.

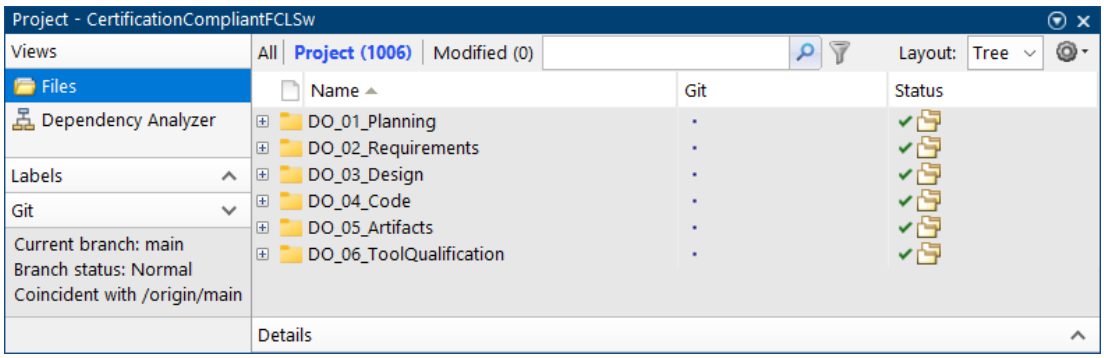


Figure 6.1 : MATLAB project structure with Git source control.

6.1 FLCS System and Algorithm Development Overview

The mapping between system requirements side and the design side can be seen in the Figure 6.2. Flight control functions system design and the requirement allocation to software and (also hardware) is fulfilled in the dashed line level in the diagram. Software items are then developed under DO-178C.

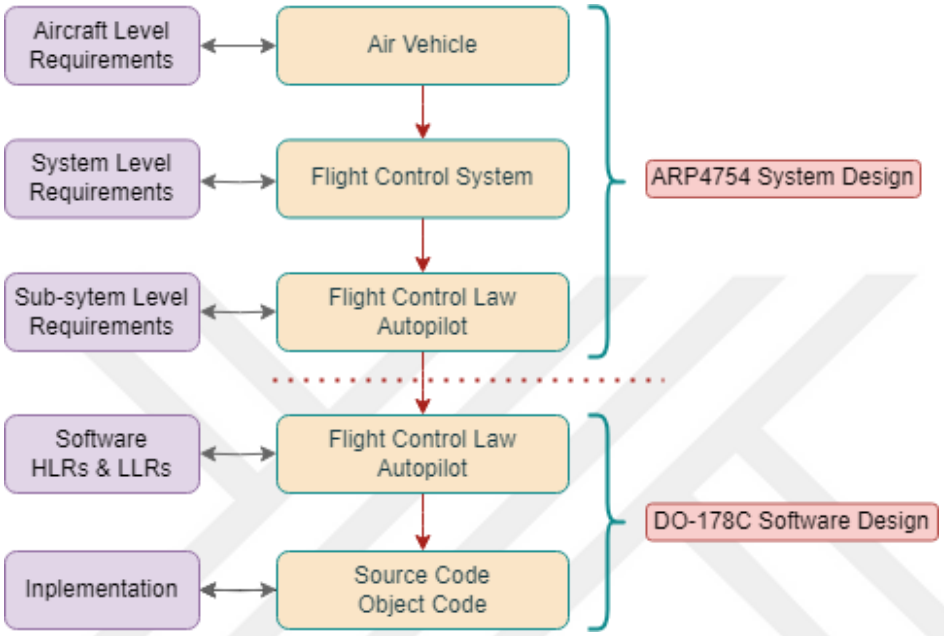


Figure 6.2 : System and item design levels.

6.2 Software Planning Overview

Software planning process is summarized in Chapter 2.4.1. As a reminder, software planning process gives five plans and three standards which is summarized as following in Figure 6.3.

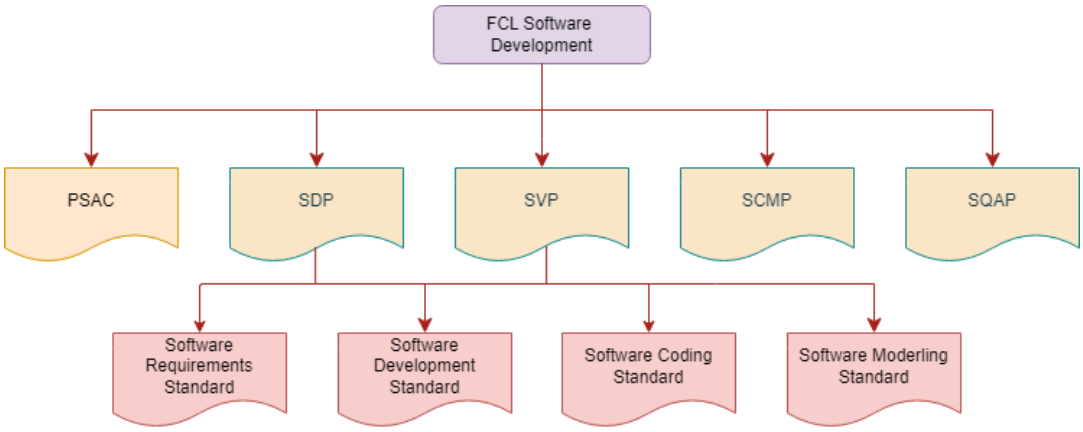


Figure 6.3 : FCL software development overview.

In this study, the plans and standards established during the flight control software planning process are briefly summarized, and the project structure created using MATLAB Project within the scope of the study is explained in Figure 6.4 below.

As introduced in 2.4.1.1, PSAC starts with a system overview by introducing the FLCS description, architecture, and flight control functions and modes. Then, it provides a software overview and explain software functions, operating systems, safety and software related details. It continues with outlining the software life cycle (from organization, processes to certification liaison) and software life cycle data. Towards the end of the document additional considerations such as alternative means of compliance, tool qualification, compiler, and programming languages, etc. is provided.

SDP starts with the development processes and activities including requirement, design, coding and integration processes as well as developing related standards for each process as introduced in 2.4.1.2. In addition, it outlines the design, modeling, and coding methods and languages that used in the software development environment.

SVP starts with the organizational responsibilities for verification of the developed software because of the verification independence. Also, verification methods for each life cycle step is introduced, verification environment is described and some software related issues are detailed in this plan, also detailed in 2.4.1.3.

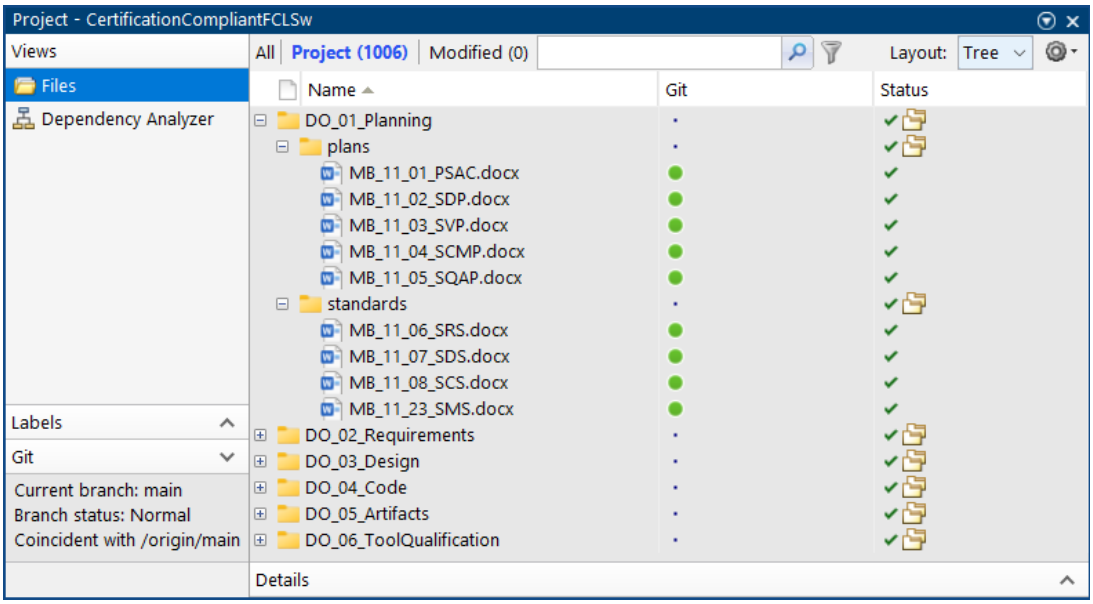


Figure 6.4 : MATLAB Project software planning structure.

SCMP provides the activities and some SCM data processes, SQAP provides problem reporting, conformance reviews and environment briefly.

6.3 Software Development Overview

The Software development process follows a traditional process with a Simulink model being developed from the HLRs and serving as the LLRs and design model for code generation as shown in Figure 6.5. In this process, source code is developed in accordance with DO-178C and DO-331 using code generation and automated verification tools from MathWorks products.

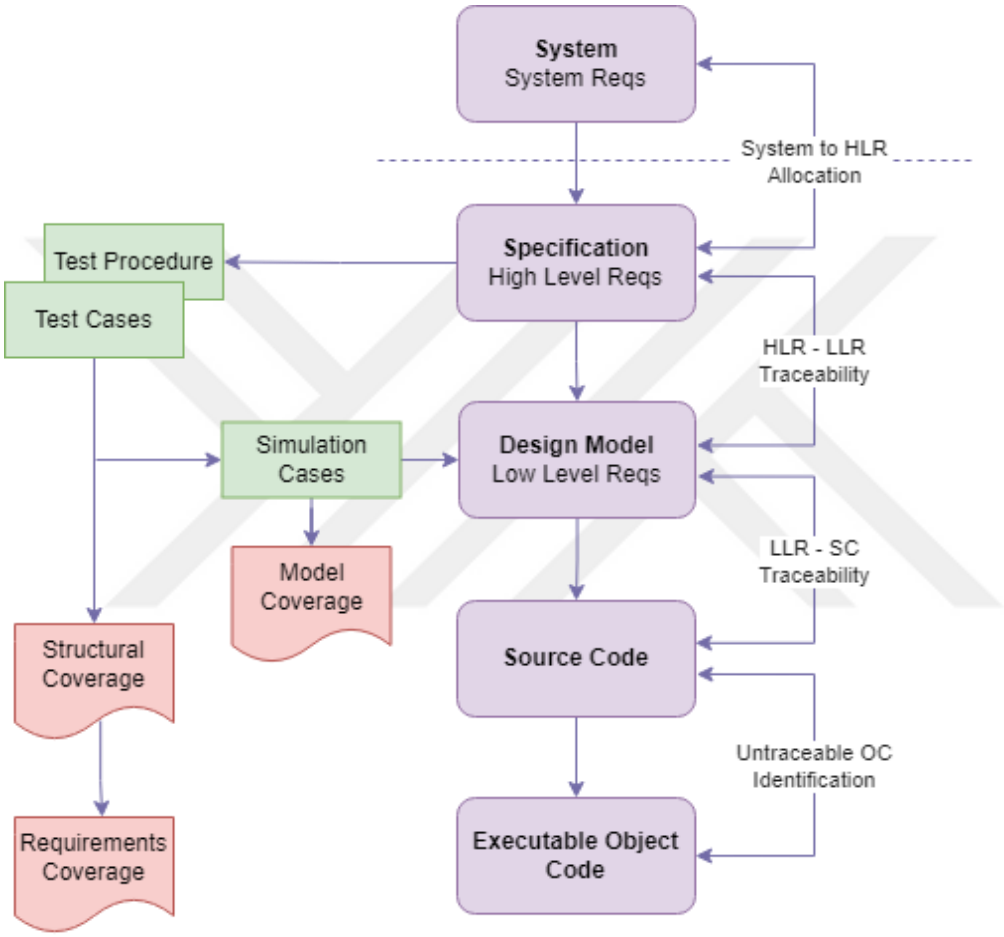


Figure 6.5 : FLCS software development overview.

This approach ensures that the code delivers the same behavior as dictated by the software requirements and design process. It also removes the breaks in the process, where manual coding and translation activities introduce bugs and, potentially, modify behavior unintentionally.

DO-178C formalizes usage of design models to gain credit for low-level requirement software design verification activities. Also, the benefits of test scenario repeatability across the software verification life cycle will help to bring software problem identification forward to the left-hand side of the V-model as seen in Figure 6.6. This

will tie testing and bug detection closely with the requirements development and software design phases, during which more scope exists in the program to modify and refine system and software requirements.

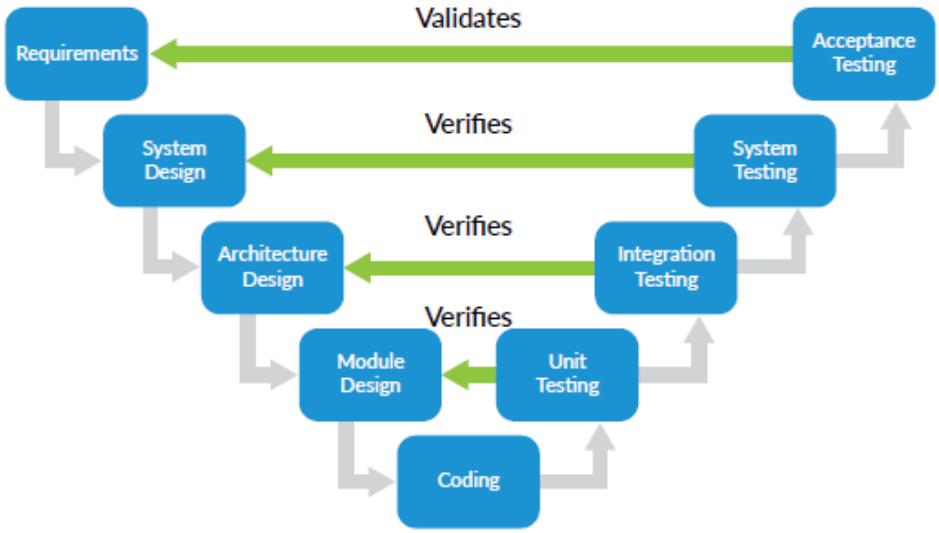


Figure 6.6 : The V-model V&V process.

At the core of the philosophy in Figure 6.5, a so-called *Design Model* replaces textual LLRs. The Design Model is an exact representation of the source code both structurally and in terms of functionality, software architecture and resulting behavior. Traceability makes effective use of integration between tool platforms by importing/exporting requirement tools. Based on recommendations outlined in DO-178C and DO-331, test cases derived from textual HLRs can executed on both the LLR Design Model as simulation cases and the final target processor as software integration tests. Conformance with software design and coding standards, identification of software defects and run-time errors, is achieved via analysis of the output of automated verification tools.

6.3.1 Software development process

To align software development activities with DO-178C, a hierarchical structure of requirements, design, and code should be used. This structure produces clear high-level functional requirements, from which detailed low-level requirements (LLRs) and Software Integration Tests can be derived. Additionally, it distinguishes between system design and software design, ensuring clear allocation to corresponding compliance methods: system testing and software integration testing. For this reason, MB Example 1 from DO-331 is followed as outlined in Table 6.1 [2]. This approach

ensures a clear separation between the system, HLRs, and LLRs, establishing a well-defined hierarchy that maps development activities to verification activities. Other examples, such as MB Example 5, can blur the distinction between system and software, complicating requirements definition. Additionally, flat hierarchies of requirements can lead to each requirement needing multiple compliance methods, including system tests and various levels of software integration testing. Dividing the left side of the V model into a hierarchy that closely aligns with the DO-178C standard allows for easy mapping to the established testing methods on the right side. This approach helps eliminate ambiguity in both development and verification activities.

Table 6.1 : Model-based usage examples table from DO-331 [2].

Process that generates the life-cycle data	MB Example 1	MB Example 2	MB Example 3	MB Example 4	MB Example 5
System requirements and system design processes	Requirements allocated to software	Requirements from which the model is developed	Requirements from which the model is developed	Requirements from which the model is developed	Requirements from which the model is developed
Software requirements and software design processes	Requirements from which the model is developed	Specification model	Specification model	Design model	Design model
	Design model	Design model	Textual description		
Software coding process	Source code	Source code	Source code	Source code	Source code

At the design side of the software development activities with DO-178C, the aircraft Simulink model has been created as follows. This Simulink model will be used throughout the entire process, from requirements generation to source code production, according to the chosen model-based approach.

Figure 6.7 : Air vehicle simulation model in Simulink.

6.3.2 System requirements allocation to software

System-level models encompass representations of both physical hardware and algorithms or logic. These models serve as executable specifications for systems to be installed on the aircraft, essential for validating and assessing system requirements. Briefly, air vehicle subsystem models are used for integration in the 6-DoF simulation model for stability, FQ and HQ assessments and code generation intended for MIL simulation or simulator activities and HIL test environments.

System models may include simplified versions of anticipated hardware and software implementations. The fidelity of these models needs only to be adequate for the intended use cases of assessment and system-level requirements validation. Additionally, system requirements are not bound by software architecture or guidelines like MISRA-C, which are specific to software. Therefore, a clear distinction is maintained between requirements and models meant for system-level and software item-level design purposes.

To facilitate the allocation of system requirements to the software, Simulink Requirements is employed. This toolbox allowed for the efficient management and tracking of all relevant requirements throughout the development process. The requirements were imported from an MS Word document into Simulink Requirements, ensuring that each requirement is accurately represented and linked to its corresponding software component. As depicted in the Figure 6.8, these imported requirements are systematically listed, providing a clear and organized framework for the subsequent stages of software development. This approach not only enhances traceability but also ensures compliance with the stringent standards.

SystemRequirements			
1	SystemRequirements	Imported from SystemRequirements.docx	Container
1.1	Introduction	Introduction	Informational
1.2	System Description	System Description	Informational
1.3	System Requirements	System Requirements	Container
1.3.1	SR_1	Pilot Input Signals	Functional
1.3.2	SR_2	Hydraulic Actuator Interfaces	Functional
1.3.3	SR_3	Hydraulic Actuator Signals	Functional
1.3.4	SR_4	Hydraulic Actuator Control Loop Performance	Functional
1.3.5	SR_5	Attitude Heading Reference System Interfaces	Functional
1.3.6	SR_6	Attitude Rate Tracking Performance	Functional
1.3.7	SR_7	Control Bandwidth	Functional
1.3.8	SR_8	Control Gain and Phase Margins	Functional
1.3.9	SR_9	Attitude Rate Authority Limiting	Functional
1.3.10	SR_10	Sensor Validation	Functional
1.3.11	SR_11	Sensor Voting	Functional

Figure 6.8 : System requirements allocated to software using Simulink.

For transitioning to software-level design models, it is crucial to identify the boundary around the Software Configuration Item (SCI), focusing on system requirements allocated to software which is FCL in this study. Additionally, software architecture and target processor (SIL in this study) are considered when constructing the Software Low-Level Requirements (LLR) Design Model because these factors can influence data types, functional structure, and execution order compared to system models.

6.3.3 Software requirement process

In the DO-178C software development lifecycle, the initial stage of the DO-331 model-based design process is requirements development. This process aims to consolidate and develop system requirements allocated to software (SRATS) for a specific Software Configuration Item (SCI) and to identify derived High-Level Requirements (HLRs). The HLRs offer a clear description of a function's behavior, serving as a foundation for software design and allowing for the independent development of software integration test procedures.

6.3.3.1 High level software requirements (HLRs) development

The software requirements are developed in accordance with DO-331 *MB Example 1* usage as depicted in Table 6.1. Therefore, textual requirements define the HLRs allocated to software from the system or subsystem design requirements as seen in the flowchart in Figure 6.2. Derived HLRs are identifiable in the MS Word document and can be updated in accordance with the SCMP.

The HLRs are developed from the system requirements allocated to the corresponding SCI which is an FCL. The HLRs include functional, performance, interface and safety related requirements and also response to abnormal conditions and inputs according to the DO-178C Section 5.1 [1].

HLRs do not describe LLRs/SC or dictate the implementation specifics such as software architecture. They focus on detailing functions, interfaces, and constraints, leaving exact implementation details to be specified during LLR software design activities as per DO-178C Section 5.1.2g [1]. It's important to highlight that processes lacking a clear distinction between requirements and design have been rejected by both the FAA and EASA in the past.

Following the implementation of SRATS, the HLRs were also addressed using Simulink. This phase involved the detailed modeling and validation of HLRs within the Simulink environment as seen in Figure 6.9, ensuring that all high-level functional and performance criteria were met. The use of Simulink for HLR implementation facilitates a seamless transition from system-level requirements to detailed design, maintaining consistency and traceability throughout the development lifecycle.

Software Requirements			
1	Software Requirements	Imported from Software Requirements.docx	Container
1.1	Introduction	Introduction	Informational
1.2	System Description	System Description	Informational
1.3	High-Level Software Requirements	High-Level Software Requirements	Container
1.3.1	HLR_1	Pilot Input Signal Processing	Functional
1.3.2	HLR_2	Hydraulic Actuator Feedback	Functional
1.3.3	HLR_3	Hydraulic Actuator Drive	Functional
1.3.4	HLR_4	Hydraulic Actuator Loop Control	Functional
1.3.5	HLR_5	Multi-Variable Inner Loop Control	Functional
1.3.6	HLR_6	Pitch Outer Loop Control	Functional
1.3.7	HLR_7	Roll Outer Loop Control	Functional
1.3.8	HLR_8	Yaw Outer Loop Control	Functional
1.3.9	HLR_9	IMU Validity Check	Functional
1.3.10	HLR_10	IMU Input Signal Processing	Functional
1.3.11	HLR_11	IMU Voting for Triple Sensors	Functional
1.3.12	HLR_12	IMU Voting for Dual Sensors	Functional
1.3.13	HLR_13	IMU Usage of Single Sensor	Functional

Figure 6.9 : Software HLRs implementation using Simulink.

Textual requirements must adhere to the SW HLR standards as stated in Chapter 2.4.1.6. It is recommended to keep textual requirements concise, avoiding the use of complex tools or constructs such as pseudocode in their development.

6.3.4 Software design process

The Software Design Process aims to translate and refine HLRs allocated to a SCIs into LLRs, including any derived LLRs. LLRs provide a detailed and structurally accurate description of the implementation and software architecture, guiding coding activities effectively.

6.3.4.1 Low level software requirements (LLRs) development

The software requirements are developed following the guidelines DO-331 *MB Example 1*. Consequently, a design model capable of generating code will be produced, encompassing both functional specifications and software architecture. The design model is based on Simulink modeling language and placed under Git source control. Modifications made are tracked in accordance with the SCMP.

LLRs expressed as a *Design Model* shall conform to the tool vendors modeling guidelines for DO-178C, MISRA-C and Simulink Code Inspector. Specifically, the Simulink Code Inspector modeling guidelines limit the use of Simulink language constructs to ensure that source code can be traced back to the model structure. These guidelines emphasize strong data typing principles and simplify the modeling language to elementary blocks. This requirement necessitates that designers implement functionality at a sufficiently detailed level to meet low-level requirements. This approach mandates that designers have a clear understanding of and control over their implementations, avoiding the use of supplier-provided functions that may yield unpredictable outcomes.

The design model, were developed within the Simulink environment as seen in Figure 6.10. Each requirement is linked to corresponding Simulink blocks, inputs, and mathematical operations using Simulink tools. This method ensures that all design elements are directly traceable to their respective requirements, facilitating thorough validation and verification throughout the development process.

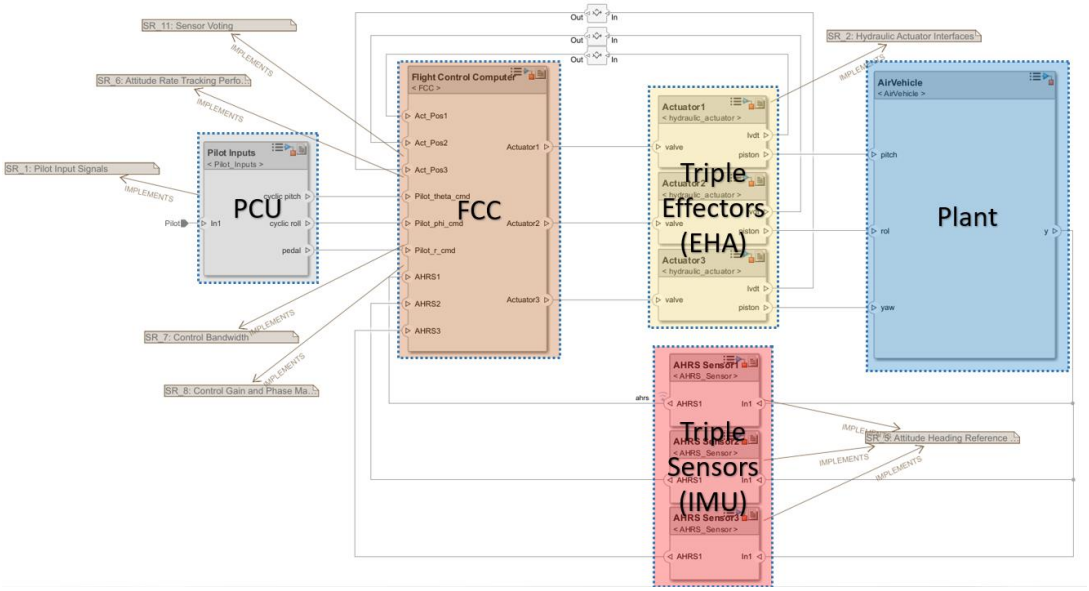


Figure 6.10 : Requirements implementation to Simulink model.

6.3.5 Software coding process

The software coding process delivers code from the LLRs in accordance with DO-178C Section 5.3.1a. For the FLCS, these are expressed as per the previous section as a *Design Model*. Source Code is integrated with Git configuration control, tracked in accordance with the SCMP.

The source code is auto-generated using MathWorks Embedded Coder, only from models compliant with the tool vendors guidelines specified in Chapter 6.3.4.

Adhering to MISRA modeling guidelines during the development of LLR Design Models ensures that the generated code complies with mandatory MISRA rules. Verification of this compliance, along with justifications for any non-mandatory deviations, is linked to the process of source code verification.

Simulation scenarios designed for validating the LLR Design Model can also be executed as SIL tests on a host computer, serving as ad-hoc testing. This approach enables designers to identify and address issues related to code generation settings, modeling techniques, and incorrect inputs early in the design phase.

6.3.6 Integration process

The purpose of the integration process is to compile and link all source code components into the integrated system and then load this onto the target computer. As the Software Configuration Items (SCIs) discussed in this document do not represent the entire system, only object code will be developed and provided to the broader software development process of the FCC as library. Delivering object code as libraries from various suppliers within an integrated system is a recognized approach based on previous experience for providing functions to the integration process.

6.3.7 Software development process traceability

In order to ensure compliance with DO-178C and correct allocation of system functions to software, trace data from system to SW LLRs must be developed. It is developed using Requirements toolbox using Simulink.

6.3.7.1 System to software HLR

Both the system requirements and HLRs exist as textual requirements within the MS Word in order to provide bi-directional traceability. This bi-directional traceability is provided using Requirements toolbox using Simulink.

6.3.7.2 Software HLR to LLR

HLRs for *MB Example 1* usage exist as textual requirements within MS Word. These textual requirements are imported to Simulink using Simulink Requirements and linked to LLR Design Models. The resulting trace data (HLR requirement ID to LLR

model path element) will be exported from Simulink Requirements as slreqx. This ensures that MS Word remains the master storage space for trace data across design levels, whilst providing developers of LLR Design Models access to both software requirements and design in the Simulink environment.

6.3.7.3 Software Derived LLR

As described in Chapter 6.3.4.1, additional modeling constructs that do not trace to HLRs are identified in a MS Word database of textual-derived LLRs. These are linked to the LLR Design Model via the same process as the HLRs, that is, using the slreqx export to Simulink Requirements approach. In this way, any resulting test cases derived for achieving LLR coverage can be clearly identified in the Design Model.

6.3.7.4 Software LLR to Source Code

Bi-directional LLR Design Model to auto-generated SC is auto generated via Simulink Code Inspector. A report is generated with detailed model-to-code and code-to-model trace data. Simulink Code Inspector will also be qualified as verification tool thereby automating the review of the trace data.

6.4 Software Verification Overview

This chapter outlines the methods for verifying the requirements, design, and code for the SCI, as introduced in Chapter 6. The key verification activities recommended in DO-178C should be integrated into the workflow as depicted in Figure 6.5 for the software verification process. This approach will utilize the Design Model to achieve verification credit for the LLRs through more effective functional execution of the Design Models.

While the verification process addresses all aspects of DO-178C, some elements are automated using qualified tools or integrated earlier in the verification lifecycle to facilitate early bug detection, as recommended in the DO-331 supplement. To fully harness the benefits of MBD, as demonstrated in other projects certified by EASA and FAA, it is necessary to apply verification process outputs earlier in the lifecycle. This includes deploying *test procedures and simulation cases* on the LLR SW Design Model.

To gain the full benefit of MBD capabilities, the FLCS software verification process will ensure key functional defects and run-time errors are detected during the LLR verification, prior to the SW coding process. Verification of the code itself will verify that the code is free from unintended or incorrect behavior and the software defects deemed removed during development of the Design Model. This also provides the necessary overlapping software verification activities for DAL-A software items as described in DO-178C Section 6 [1].

The following chapters provide an overview of how each verification lifecycle activity is conducted and outline the timing for coordinating outputs with specific development activities to identify issues as early as possible.

6.4.1 Reviews and analyses of software HLR

HLRs are purely textual and stored within MS Word document as per Section 7.1 in DO-178C. For this reason, all aspects of DO-178C Section 6.3.1 are addressed via manual reviews as with traditional workflows.

6.4.2 Reviews and analyses of LLR and software architecture

This element of the verification process marks the initial stage where MBD offers alternative methods to achieve these objectives. Traditionally, a textual-based LLR SW Design would be independently reviewed for DAL-A SCIs. However, these reviews are often ineffective, particularly in identifying compliance with HLRs, verifiability, and consistency of the software architecture. Textual requirements frequently lack the necessary detail at the LLR level to fully capture the software design. Conversely, Design Models accurately capture these aspects but require domain expertise in the specific language used for the Design Models. Manual reviews are inefficient in identifying significant design flaws, with issues often remaining undetected until Software Integration Testing.

To automate the manual reviews of LLRs, as specified in DO-331 Section 6.8.1, Simulation Cases derived from Test Cases outlined in Figure 6.11 should be employed. Table 6.2 summarizes the Simulation Cases which play a crucial role in verifying the software by performing essential steps typically conducted during manual reviews. By leveraging these Simulation Cases, the verification process becomes more efficient and thorough, ensuring that the LLRs are accurately and consistently evaluated against

the established criteria. This approach not only streamlines the verification process but also enhances the detection of potential issues early in the development lifecycle, contributing to a more robust and reliable software design.

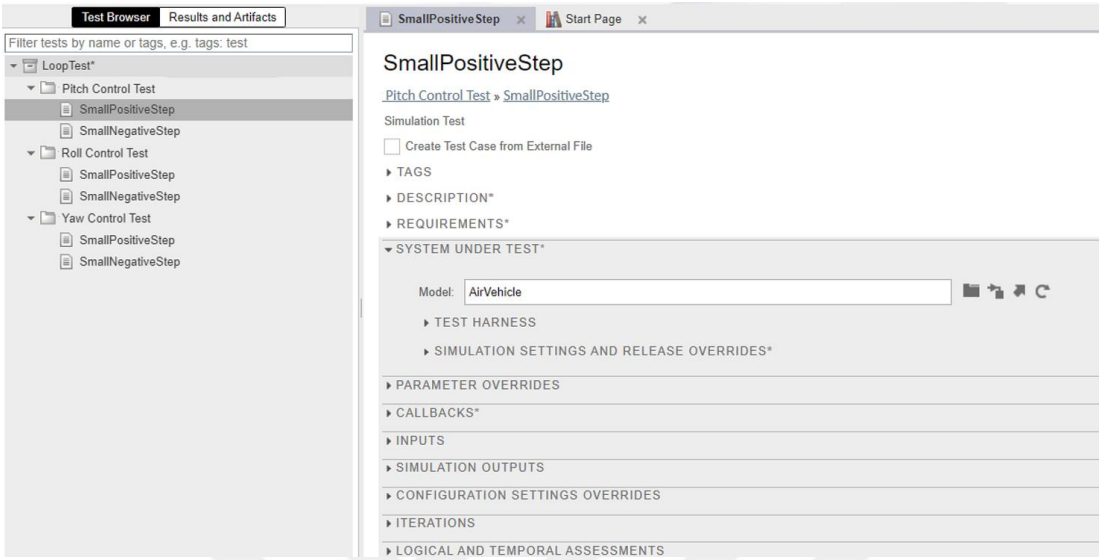


Figure 6.11 : Test procedure and simulation case example.

Table 6.2 : Simulation and test case aims.

Specific target	DO-178C Section	DO-331 Section
Compliance with SW HLRs	6.3.2a	MB 6.3.2a
Compatibility with SW Architecture	6.3.3a	MB 6.3.3a
Accuracy and Consistency	6.3.2b & 6.3.3b	MB 6.3.2b & MB 6.3.3b
Verifiability	6.3.2d & 6.3.3d	MB 6.3.2d & MB 6.3.3d
Algorithm Aspects	6.3.2g	MB 6.3.2g

Conformance to Software Design standards, as indicated in DO-178C sections 6.3.2e and 6.3.3e, as well as DO-331 sections MB 6.3.2e and MB 6.3.3e, is achieved through qualified model-level static analysis checks. These checks will be conducted using the Simulink Model Advisor as indicated in Figure 6.12 and will adhere to industry-recognized guidelines provided by the supplier.

Figure 6.12 : Simulink model advisor report for conformance.

Traceability, as required by DO-178C section 6.3.2f and DO-331 section MB 6.3.3f, is achieved through a traditional manual review process using MS Word and within the Simulink model. According to Section 7.7.2 of DO-178C, HLRs will be exported from Simulink Requirements to MS Word, where they will be imported and linked to the Simulink Design Models. The resulting traceability data will then be exported from Simulink and imported back into MS Word for comprehensive review and verification as seen in Figure 6.13.

Figure 6.13 : Traceability links for system requirements in Simulink.

Compatibility with the target computer, as outlined in DO-178C Sections 6.3.2c and 6.3.3c and DO-331 MB 6.3.2c and MB 6.3.3c, primarily involves manual review and analysis. While hardware-specific code generation settings can be verified using automated checks via Simulink Model Advisor, the evaluation of update rates and system response times relative to a specific target requires manual inspection. This inspection will be based on the system design description (SDD), which will be automatically generated using the Simulink Report Generator and the Design Model.

It is important to note that certification is not solely obtained through the execution of Simulation Cases, but also through the independent review and analysis of these cases and their results, as specified in DO-331 MB 6.8.3.2b & MB 6.8.3.2c. To accomplish this, test cases and procedures independently developed from the HLRs, from which the LLR Design Model was also derived, will be incorporated into the Simulink Test environment for the purpose of on-target Software Integration Testing. Both the automated Simulation Case procedures and the results from these executions will undergo a thorough software testing review. It is crucial to verify that the Simulation

Case test procedures outline the same conditions, inputs, and objectives as those specified in the Test Cases and Test Procedures as stated in Figure 6.14 using Simulink Test. Additionally, it must be ensured that all Simulation Cases meet the objectives in the generated results report as defined by the Test Case objectives.

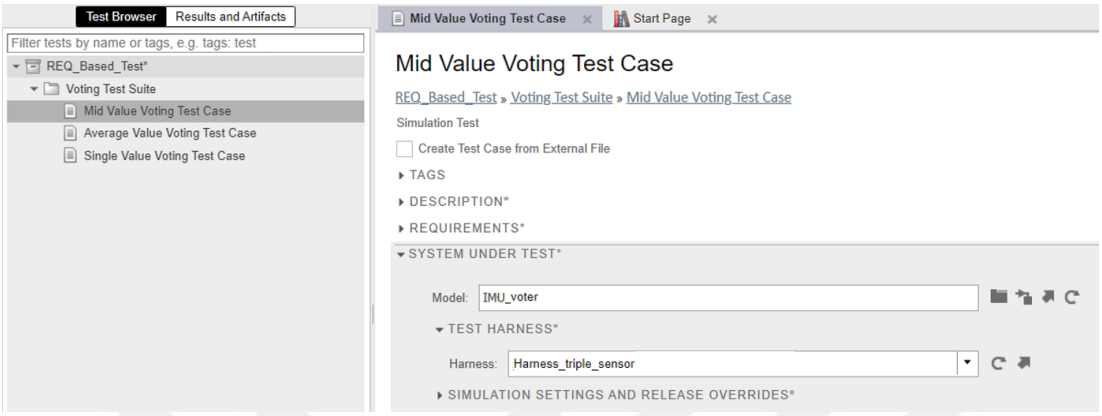


Figure 6.14 : Test cases and test procedures for IMU voting.

At this stage, the execution of Simulation Cases not only contributes to certification credit for LLR and SW Architecture Verification activities but also offers early detection of bugs and incorrect functional behavior. This feedback is provided to the Design Teams responsible for the SCIs identified in this document, ensuring timely and efficient resolution of issues.

The SVP must fully justify why the aforementioned objectives, achievable through Simulation Cases, are better suited to this method. It must also clarify how the simulation activity thoroughly satisfies the review or analysis objectives outlined in this section. For instance, achieving 100% model coverage through Simulation Cases provides detailed evidence that executable LLRs such as statements, transitions, etc. are reachable and verifiable. This level of certainty and accuracy cannot be achieved through manual review and analysis, which would be susceptible to human error.

Simulation Cases have been confirmed to exist for each HLR from which the LLR Design Model was developed. This confirmation is achieved by linking the HLRs imported into Simulink to the corresponding Simulation Cases. The Simulation Case Procedure, exported from Simulink Test, includes the corresponding HLRs along with the associated Simulation Cases. This setup allows for an independent review of the procedure's correctness in relation to the test cases and test procedures.

Test Cases derived from HLRs and implemented as Simulation Cases are expected to already satisfy the criteria of DO-178C Section 6.4.2 for normal range and robustness as seen in Figure 6.15 and Figure 6.16. This expectation includes the independent review of the developed test cases, as required. Therefore, when verifying that the Simulation Cases are correct and match the test cases, the requirement for normal range and robustness is also fulfilled for the Simulation Cases.

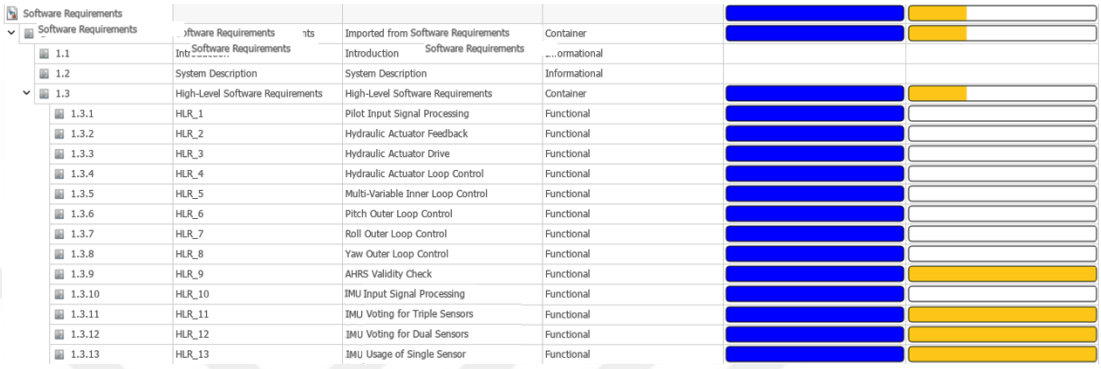


Figure 6.15 : HLRs linkage to test and simulation cases example.

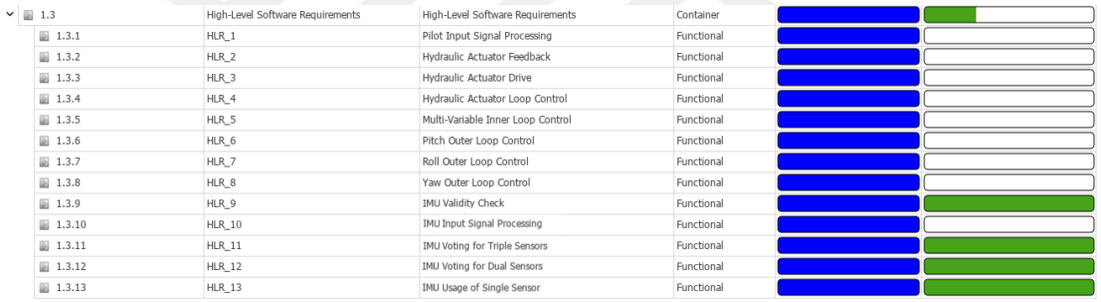


Figure 6.16 : HLRs verification by test and simulation cases.

6.4.3 Reviews and analyses of source code

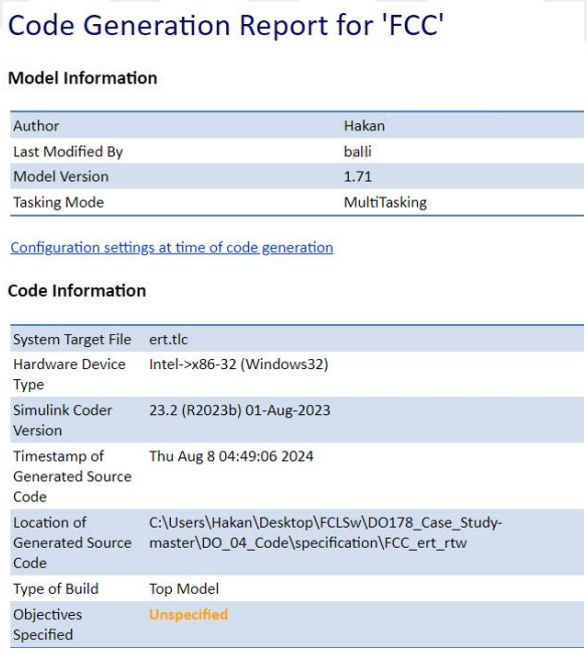
Verification activities associated with reviews and analysis of the SC as seen in Figure 6.17 is fully automated with qualified verification tools:

- Simulink Code Inspector
- Polyspace Bug Finder
- Polyspace Code Prover

This section identifies which tools will provide individually full and partial coverage to the verification activities required. Also, those verification activities that cannot be addressed are identified. Usually, activities that cannot be addressed rely on analysis of the Application SW SC together with the hardware, such as worst-case execution timing, and therefore cannot be addressed by SC analysis verification tools alone.

SC verification activities are performed via model-to-code traceability, SC static analysis for MISRA compliance, and Formal methods using abstract interpretation in detailed DO-333 FM 1.6.2(3) for proof of SC robustness and absence of defects ss shown in Figure 6.5.

Compliance of the Software Component (SC) with Low-Level Requirements (LLRs) and Software architecture, as specified in DO-178C Sections 6.3.4a and 6.3.4b, is ensured using Simulink Code Inspector. This verification process confirms that there are no alterations in algorithms during model-to-code translation and that the code structurally maps precisely to the model. Since the Design Model includes the software architecture, verifying that both algorithms and structure are maintained during code generation fulfills this requirement. Independence is necessary for these verification activities. Per DO-178C Section 6.2e, achieving human equivalence can be facilitated by a tool. In this case, EASA and FAA have validated that Simulink Code Inspector was developed by a team separate from the Embedded Coder, ensuring the verification tool is independent of the development tool. Thus, the verification of model-to-code translation is independent as required.



Code Generation Report for 'FCC'

Model Information

Author	Hakan
Last Modified By	balli
Model Version	1.71
Tasking Mode	MultiTasking

Configuration settings at time of code generation

Code Information

System Target File	ert.tlc
Hardware Device Type	Intel->x86-32 (Windows32)
Simulink Coder Version	23.2 (R2023b) 01-Aug-2023
Timestamp of Generated Source Code	Thu Aug 8 04:49:06 2024
Location of Generated Source Code	C:\Users\Hakan\Desktop\FCLSw\DO178_Case_Study-master\DO_04_Code\specification\FCC_ert_rtw
Type of Build	Top Model
Objectives Specified	Unspecified

Figure 6.17 : Code generation for FCC model.

Verifiability, as outlined in DO-178C Section 6.3.4c and DO-333 FM.6.3.4c of the Software Component (SC), is attained through the use of Formal Methods with Polyspace Code Prover. By employing abstract interpretation, it must be mathematically demonstrated that there is no dead logic or unreachable code, thereby

ensuring the verifiability of the SC. This aspect does not necessitate independence and is appropriate for verification of the Software Component Integration (SCI) application software, which is classified as Functional Certification Level (FCL).

It is considered that, since the Simulation Cases executed on the LLR model have already achieved 100% coverage, and the Simulink Code Inspector has verified the structural equivalence of the Design Model and SC, there will be no dead code in the SC. Consequently, the verification activity is simply to confirm that the SC verification tool's output report (from Polyspace) indicates there is no dead code. This activity requires minimal effort when the SC analyzed originates from a Design Model already verified to have no dead code or unreachable statements. Any identification of actual dead code was effectively addressed earlier in the development lifecycle during the verification of the LLRs on the Design Model.

Verification that the SC conforms to standards as stated in DO-178C Section 6.3.4d is conducted using Polyspace Bug Finder and Polyspace Code Prover. These tools perform code-level static analysis to ensure compliance with MISRA 2012 coding standards. Since independence is not required for this aspect, it is appropriate to verify this for the FCL. As part of the MISRA 2012 conformance analysis, the verification tool's output must be reviewed, and justifications must be provided for any non-mandatory MISRA non-compliances.

Verification of traceability between the SC and the LLR Design Model is achieved using Simulink Code Inspector. This verification tool generates detailed traceability reports for both "Design Model to SC" and "SC to Design Model." The granularity of these traceability reports has been previously accepted by both EASA and FAA on prior projects.

6.4.4 Software testing

FCL verification must align with DO-178C Section 6.4, ensuring that the software meets its requirements on the final target processor. To detect bugs early and secure certification credit for LLRs as outlined in 6.4.2, test cases shall also be executed early in the life cycle as Simulation Cases on the LLR Design Model.

The simulation cases are executed on the LLR Design Model and all verification results delivered to the test team for review of the behavior, model coverage and HLRs. Several iterations can be performed at this stage with model-level problem reporting

used to trigger requirements refinement and bug fixes. This process ensures that key functional deficiencies and improvements to HLRs take place at the earliest stage possible in the development process, as opposed to waiting until hardware/software integration is performed.

Prior to deployment of the HLR test cases, it is recommended that additional ad-hoc SIL testing is performed. This is not for the purpose of gaining certification credit, but more as an additional check prior to expensive software integration and on-target testing. The Test Cases already deployed as Simulation Cases will be re-deployed as SIL. The resulting behavior and first look at the resulting structural coverage is delivered to identify any potential issues in the code generation process.

Once the HLRs have reached sufficient maturity, the test cases that when first deployed as Simulation Cases, deliver full model coverage and second during SIL mode deliver adequate structural coverage, is deployed in PIL mode. As per DO-178C Section 6.4.1, deploying the SCIs in test environment such as PIL, whereby the Software component attributed to the HL SW function, such as CLAWs, is functionally isolated, the full FCC software integration allows for more precise control and monitoring of that function. Also, full-code instrumentation of this function can be achieved to deliver the necessary structural coverage during execution of the test cases. The output of this process will deliver both OC for the SCI, Test Case results and the structural coverage report for analysis.

The above approach will deliver the necessary outputs to ensure the correct behavior and the test coverage aspects of HLRs, LLRs and SC related to DO-178C Table A-7.

6.4.5 Test coverage analysis

Test coverage includes requirements review, model coverage, and structural coverage. However, structural coverage analysis objectives are not replaced or fulfilled through model coverage.

The model coverage provided by Simulink Coverage possesses the same properties (e.g., MC/DC) as the structural coverage delivered by Simulink Coverage for Source Code (SC) structural coverage. However, it is not necessary to replace the structural coverage activity with alternate means.

Test cases that achieve 100% MC/DC coverage at the model level when executed as simulation cases also provide a sufficiently high level of structural coverage.

The remaining missing structural coverage usually consists of known, justifiable omissions that occur as a result of achieving certain MISRA compliances.

Due to the ease of redeploying test cases as simulation cases and vice versa, it is more valuable to reuse test cases across lifecycle verification activities. This approach allows for the collection of evidence within the appropriate contexts. Specifically, structural coverage should be gathered during the execution of the code on the intended target processor.

This approach eliminates the need to justify how the semantics of model and source code coverage are representative.

Model coverage is collected in accordance with DO-331 Section M.B.6.7a. This requires the High-Level Requirement (HLR) test cases, developed independently by the Software Test Team, to be deployed as simulation cases on the Low-Level Requirement (LLR) design model.

The coverage results will be used to identify deficiencies in the test cases and High-Level Requirements (HLRs), as well as to develop any necessary Low-Level Requirement (LLR) tests for LLR-derived requirements.

The model coverage result will also contribute to verifying the verifiability of the Low-Level Requirements (LLRs), as described in Section 8.1.2 of DO-178C 6.3.2d & 6.3.3d / DO-331 MB 6.3.2d & MB 6.3.3d.

Structural coverage analysis will be collected in accordance with DO-178C Section 6.4.4 for the Software Configuration Items (SCIs) affected during the Processor-in-the-Loop (PIL) deployment of the test cases. There is a high level of confidence that the test cases and their High-Level Requirements (HLRs) already possess sufficient maturity to achieve structural coverage, minimizing the need for extensive analysis or justification activities. This confidence stems from the deployment capabilities of Simulink Test, which facilitates the transition from test cases to simulation cases, and the early lifecycle requirement refinement activities conducted during LLR behavior analysis and model coverage assessment.

7. CONCLUSIONS AND RECOMMENDATIONS

In this thesis, a flight control system and software compliant with certification requirements were developed using the model-based design approach, which offers numerous advantages over traditional software lifecycle processes and has seen increasing use in the aviation industry in recent years. The development adhered to the requirements of the DO-178C guidelines to ensure certification compliance.

From the perspective of flight dynamics and control design, the flight control system in this study has been modeled as comprising flight control actuators, sensors, and a flight control algorithm. Redundancy and voting features of the system have also been integrated into the model. Nonlinear dynamic inversion has been employed as the flight control algorithm. The flight control system has been translated into source code and prepared for integration into hardware.

Similarly, within the scope of the software development life cycle, SRATS is have been generated from aircraft-level system requirements. High-level requirements (HLR) have been derived based on military standards and SRATS. Using these HLR, a flight control algorithm has been developed, and redundancy and voting algorithms have been added to design the flight control system.

Then, one of the most critical processes of the model-based design approach, the validation of the model, is initiated. The validation of the flight control system against the HLR is conducted through model simulation tests developed using Simulink Test. The model simulation activity, as derived from the DO-331 guideline document, involves running simulations on the model to test for any potential errors before transitioning to code. The performance of the developed flight control system is evaluated based on the test results. During the tests, a Model Coverage Analysis (MCA) is performed to check if there were any elements in the flight control system that were not tested. The analysis provided performance criteria to investigate whether there is any incomplete coverage. The final activity conducted on the model is the

compliance check with standards, performed statically using the Model Advisor for DO-178C/DO-331 standards.

Following these evaluations, the process moved on to the generation of source code. The source code is created using the MATLAB Embedded Coder tool, and the generated code is reviewed using the MATLAB Simulink Code Prover tool to obtain performance metrics. Subsequently, executable object code is generated using a compiler, and using the MATLAB/Simulink tool, performance criteria were evaluated with software simulation tests in a loop. With this activity, the flight control software development process using the DO-331 model-based design approach, which offers many advantages over traditional document-based processes in terms of criteria such as time, cost, and effort, is completed.

Finally, throughout the software development process, software and system plans were written at each step from requirements to executable object code, relevant process reports were generated, configuration management is conducted, and software quality assurance and certification processes were prepared for submission to the authorities. As a result, activities required by the DO-178C guidance document were executed in conjunction with the DO-331 model-based design supplement, resulting in the development of a flight control software that complies with certification requirements.

In Model-Based Design (MBD) methodology, developing models at the system level allows for the retention of requirements as models, and enables model verification through simulations before any code is even written. This approach provides advantages through a common language, visualization, and notation established by all project stakeholders. Consequently, it eliminates difficulties such as documentation procedures used in traditional methods and challenges related to continuous communication between teams and technical terminology ambiguity. Another significant advantage is the early detection of potential errors through model simulation tests conducted in the initial stages of the project, enabling prompt correction of these errors and thereby providing significant gains in project schedule and budget.

In the next stages, the development can aim to verify the generated executable object code by running it on actual hardware through hardware-in-the-loop (HIL) simulation tests, and then embedding it into the flight computer.

REFERENCES

[1] **RTCA**, "DO-178C - Software Considerations in Airborne Systems and Equipment Certification," RTCA , Washington DC, December 2011.

[2] **RTCA**, "DO-331 - Model-Based Development and Verification Supplement to DO-178C and DO-278A," RTCA, 2011.

[3] **L. Nguyen, M. Ogburn, W. P. Gilbert, K. S. Kibler, P. W. Brown and P. L. Deal**, "Simulator Study of Stall/PostStall Characteristics of a Fighter Airplane with Relaxed Longitudinal Static Stability," NASA Technical Paper 1538, Hampton, VA, 1979.

[4] **J. Hu, L. Jun, L. Zheng, P. Wang and Z. Ma**, "A discussion of airworthiness in terms of safety and integrity," *Journal of Physics: Conference Series*, vol. 1053, no. 1, 2018.

[5] **I. Habli and I. Dodd**, "Safety certification of airborne software: An empirical study," *Reliability Engineering & System Safety,* vol. 98, no. 1, pp. 7-23, 2012.

[6] **W. K. Youn and S. B. Hong**, "Software certification of safety-critical avionic systems: DO-178C and its impacts," *IEEE Aerospace and Electronic Systems Magazine,* vol. 30.4, no. 4, pp. 4-13, 2015.

[7] **K. Dmitriev, S. A. Zafar and K. Schmiechen**, "A lean and highly-automated model-based software development process based on do-178c/do-331," in *AIAA/IEEE 39th Digital Avionics Systems Conference (DASC)*, 2020.

[8] **J. Marques and A. M. da Cunha**, "Tailoring traditional software life cycles to ensure compliance of rtca do-178c and do-331 with model-driven design," in *IEEE/AIAA 37th Digital Avionics Systems Conference (DASC)*, 2018.

[9] **S. P. Cook and G. Haverkamp**, "Challenges and Opportunities for Software Development and Verification on Military Aircraft Systems," in *AIAA SciTech Forum*, Orlando, FL, 2020.

[10] **M. T. Hochstrasser**, "Modular model-based development of safety-critical flight control software," PhD Thesis, 2020.

[11] **A. F. Paz Loboguerrero**, "A DO-178C-Compliant Model-Driven Approach to Support the Development and Certification of Safety-Critical Avionics Software," PhD Thesis, 2020.

[12] **D. L. Bacon**, "An Agile Framework to Develop Safety Critical Software for Aircraft," PhD Thesis, 2018.

[13] **Rapita Systems**, "Efficient verification through the DO-178C life cycle," Rapita Systems Ltd., 2024.

[14] **B. Potter**, "Complying with DO-178C and DO-331 using Model-Based Design," MathWorks, Natick, Massachusetts, USA, 2012.

[15] **J. C. Jensen**, "Elements of model-based design," University of California, Berkeley, 2010.

[16] **T. Erkinnen and B. Potter**, "Model-Based Design for DO-178B with Qualified Tools," in *AIAA Modeling and Simulation Technologies Conference and Exhibit*, Chicago, Illinois, 2009.

[17] **U. Eisemann**, "Applying Model-Based Techniques for Aerospace Projects in Accordance with DO-178C, DO-331, and DO-333," in *8th European Congress on Embedded Real Time Software and Systems (ERTS 2016)*, 2016.

[18] **T. Sarac**, "Certification Aspects of Model Based Development for Airborne Software," in *IEEE 2nd International Conference on Information and Computer Technologies*, 2019 .

[19] **L. C. Jaw and H. T. Van**, "Model-based Approach to Validation and Verification of Flight Critical Software," in *2008 IEEE Aerospace Conference*, 2008.

[20] **J. Krizan, L. Ertl, M. Bradac and M. Jasansk**, "Automatic Code Generation from MatlabSimulink for Critical Applications," in *CCECE 2014*, Toronto, Canada, 2014.

[21] **D. J. Coe and J. H. Kulick**, "A Model-Based Agile Process for DO-178C Certification," University of Alabama in Huntsville, Alabama, USA.

[22] **P. H. Zipfel**, Modeling and Simulation of Aerospace Vehicle Dynamics, vol. 2nd Edition, AIAA, 2007.

[23] **T. Cimen,** "A Generic Approach to Missile Autopilot Design using State-Dependent Nonlinear Control," *18th IFAC World Congress*, 2016.

[24] **S. Snell, D. F. Enns and L. Garrard**, "Nonlinear Inversion Flight Control for a Supermaneuverable Aircraft," *Journal of Guidance, Control and Dynamics*, 1992.

[25] **D. Ito, J. Georgie, J. Valasek and D. T. Ward**, "NASA TP 2002 - Reentry Vehicle Flight Controls Design Guidelines," NASA, USA, 2002.

[26] **Ö. Ateşoğlu**, "High Angle of Attack Maneuvering and Stabilization Control of Aircraft," PhD Thesis, 2007.

[27] **R. Bıyıklı, R. Tekin and İ. Yavrucuk**, "Nonlinear Dynamic Inversion Autopilot Design for Aerial Vehicles with Thrust Vector and Aerodynamic Control," in *EUCASS*, 2023.

[28] **SAE International**, "ARP-4761 - Guidelines And Methods For Conducting The Safety Assessment Process On Civil Airborne Systems And Equipment," SAE International Press, Washington DC, USA, December 1996.

[29] **F. De Florio**, Airworthiness: An Introduction to Aircraft Certification, Elsevier Science, 2010.

[30] **ICAO**, "The History of ICAO and the Chicago Convention," ICAO, [Online]. Available: https://www.icao.int/about-icao/History/Pages/default.aspx. [Accessed 5 May 2024].

[31] **A. De Marco, E. Duke and J. Berndt**, "A General Solution to the Aircraft Trim Problem," in *AIAA Modeling and Simulation Technologies Conference and Exhibit*, Hilton Head, South Carolina, 2007.

[32] **ICAO**, "The Convention on International Civil Aviation: Annexes 1 to 18," ICAO.

[33] **EASA**, "The EASA story so far," [Online]. Available: https://www.easa.europa.eu/en/20th-anniversary. [Accessed 1 May 2024].

[34] **FAA**, "Software Load Control," [Online]. Available: https://www.faa.gov/.

[35] **INCOSE**, Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities, United States: Wiley, 2023.

[36] **SAE International**, "ARP-4754A - Guidelines for Development of Civil Aircraft and Systems," SAE International Press, Washington DC, USA, 2010.

[37] **L. Rierson**, Developing Safety-Critical Software: A Practical Guide for Aviation Software and DO-178C Compliance, CRC Press, 2017.

[38] **RTCA**, "DO-254 - Design Assurance Guidance for Airborne Electronic Hardware," RTCA, 2011.

[39] **B. Potter**, "DO178_case_study: Case Study for DO-178 using MathWorks tools," 2020. [Online]. Available: https://www.mathworks.com/matlabcentral/fileexchange/56056-do178_case_study.

[40] **RTCA**, "DO-332 - Object-Oriented Technology and Related Techniques Supplement to DO-178C and DO-278A," RTCA, 2011.

[41] **RTCA**, "DO-333 - Formal Methods Supplement to DO-178C and DO-278A," RTCA, 2011.

[42] **Y. Ertugrul**, "DO-178C/DO-331 Uyumlu Model Tabanlı Tasarım Yaklaşımı Kullanılarak Bulanık Mantık Tabanlı Uçuş Kontrol Sistemi Geliştirilmesi," MSc Thesis, 2023.

[43] **W. A. Babich**, Software Configuration Management, Chichester, U.K.: Wiley & Sons, 1991.

[44] **MathWorks**, "DO-178C Procedure with Matlab," Model-Based Systems Engineering. [Online].

[45] **R. Aarenstrup**, Managing Model-Based Design, Natick, MA, USA: The MathWorks Inc, 2015.

[46] **MathWorks**, "Model-Based Design with Automatic Code generation," [Online]. Available: https://www.mathworks.com/company/events/webinars/upcoming/model-based-design-with-automatic-code-generation-3986336.html. [Accessed 12 May 2024].

[47] **S. A. Jacklin**, "Certification of Safety-Critical Software Under DO-178C and DO-278A," in *Infotech@ Aerospace 2012*, 2012.

[48] **M. Millidere, U. Karaman, S. Uslu, C. Kasnakoglu and T. Cimen**, "Newton-Raphson Methods in Aircraft Trim: A Comparative Study," in *AIAA Aviation Forum*, Virtual Event, 2020.

[49] **M. Millidere and T. Yiğit**, "Local Search Based Hybrid Genetic Algorithm Method in Aircraft Trim," in *AIAA Aviation 2021 Forum*, 2021.

[50] **M. Millidere**, "Optimal Input Design and System Identification for an Agile Aircraft," PhD Thesis, METU, 2021.

[51] J.**-F. Magni and S. Bennani**, Robust Flight Control: A Design Challenge, Springer Science and Business Media LLC, 1997.

[52] **M. Millidere, M. Alam, S. Place and J. Whidborne**, "Development of High-Fidelity Saab 340B Aerodynamic Model," in *AIAA SCITECH Forum*, National Harbor, MD & Online, 2023.

[53] **T. Nguyen, P. Ogburn, S. Kibler, B. Brown and L. Deal**, "NASA Technical Paper 1538 - Simulator Study of Stall/Post-Stall Characteristics of a Fighter Airplane with Relaxed Longitudinal Static Stability," National Aeronautics and Space Administration, 1979.

[54] **O. Albostan**, "Flight Control System Design of F-16 Aircraft Using Robust Eigenstructure Assignment," PhD Thesis, 2018.

[55] **G. Ghazi, R. M. Botez and J. M. Achigui**, "Cessna Citation X Engine Model Identification from Flight Tests," *SAE International Journal of Aerospace*, vol. 8, no. 2, pp. 203-213, 2015.

[56] **Y. Huo**, "Model of F-16 Fighter Aircraft," University of Southern California, 2012.

[57] **H. Ballı and M. Millidere**, "Mission Profile Simulation of High Maneuverable Aircraft Using Trim Based Maneuver," in *Ankara International Aerospace Conference*, Ankara, Türkiye, 2023.

[58] **F. R. Garza and E. A. Morelli**, "Collection of Nonlinear Aircraft Simulations in MATLAB," NASA Technical Memorandum 212145, 2003.

[59] **B. L. Stevens, F. L. Lewis and E. N. Johnson**, Aircraft Control and Simulation: Dynamics, Controls Design, and Autonomous Systems, 3 ed., John Wiley & Sons, Inc, 2015.

[60] **J. R. Raol and J. Singh**, Flight Mechanics Modeling and Analysis, CRC Press, 2008.

[61] **R. V. Jategaonkar and F. Thielecke**, "An Integrated Software Tool for Nonlinear Parameter Estimation," *Aerospace Science and Technology*, vol. 6, no. 8, pp. 565-578, 2002.

[62] **United States Department of Defense**, "MIL-F-8785C Flying Qualities of piloted airplanes: Military specification".

[63] **O. Albostan and M. Gökaşan**, "High Angle of Attack Manoeuvring Control of F-16 Aircraft Based on Nonlinear Dynamic Inversion and Eigenstructure," in *EUCASS)*, 2017.

[64] **J. Valasek, D. Ito and D. Ward**, "Robust dynamic inversion controller design and analysis for the X-38," in *AIAA Guidance, Navigation, and Control Conference and Exhibit*, Montreal, Canada, 2001.

[65] **S. Snell, W. Garrard and D. Enns**, "Nonlinear Control of a Supermaneuverable Aircraft," in *AIAA Guidance, Navigation and Control Conference*, Boston, MA, USA, 1989.

[66] **G. Ghazi and R. M. Botez**, "Development of a High-Fidelity Simulation Model Identification for a Research Environment," in *SAE AeroTech Congress&Exhibition*, Seattle, WA, 2015.

[67] **J. D. Mattingly, W. H. Heiser and D. H. Daley**, Aircraft Engine Design, Reston, VA: AIAA, 1987.

[68] **L. F. Rodriguez and R. M. Botez**, "Generic New Modeling Technique for Turbofan Engine Thrust," *Journal of Propulsion and Power*, vol. 29, pp. 1492-1495, 2013.

[69] **B. E. Saleh**, "Modeling and Simulation of A Double Spool Turbofan Engine Using SIMULINK," Master's Thesis, Zazazig University, Zazazig, 2017.

[70] **RTCA**, "DO-330 - Software Tool Qualification Considerations," RTCA, 2011.

[71] **J. Antsaklis and A. Michel**, Linear Systems, University of Notr Dame, USA: Birkhauser, 2006.

[72] **J. Bugajski, D. Enns and M. Elgersma**, "A Dynamic Inversion Based Control Law with Application to the High Angle of Attack Research Vehicle," in *AIAA Guidance, Navigation and Control Conference*, Portland, OR, USA, 1990.

[73] **H. B. Chen and S. G. Zhang**, "Robust Dynamic Inversion Flight Control Law Design," in *2008 2nd International Symposium on Systems and Control in Aerospace and Astronautics*, Shenzhen, China, 2008.

[74] **D. Enns, D. Bugajski, R. Hendrick and G. Stein**, "Dynamic Inversion: An Evolving Methodology for Flight Control Design," *International Journal of Control* , pp. 71-91, 1992.

[75] **E. Lavretsky and K. Wise**, Robust and Adaptive Control with Aerospace Applications, Springer, 2013.

[76] **C. Miller**, "Nonlinear Dynamic Inversion Baseline Control Law: Architecture and Performance Predictions," in *AIAA Guidance, Navigation and Control Conference*, 2011.

[77] **E. Muir, J. Terlouw and S. Scala**, "Robust Flight Control Design Challenge Problem Formulation and Manual: The High Incidence Research Model," GARTEUR., 1996.

[78] **R. Stengel**, Flight Dynamics, Princeton: Princeton University Press, 2004.

[79] **M. Millidere, U. Karaman, H. Balli and T. Cimen**, "Further Investigations on Newton-Raphson Methods in Aircraft Trim," in *AIAA Scitech Forum*, Virtual Event, 2021.

[80] **T. Sarac**, "Model Tabanlı Geliştirme Teknolojisinin Hava Aracı Yazılımlarında Kullanımı ve Sertifikasyonu," MSc Thesis, 2019.

[81] **R. Bıyıklı**, "Nonlinear Dynamic Inversion Autopilot Design for An Air Defense System with Aerodynamic and Thrust Vector Control," MSc Thesis, 2022.

[82] **M. V. Cook,** Flight Dynamics Principles: A Linear Systems Approach to Aircraft Stability and Control, Butterworth-Heinemann, 2012.

[83] **R. Abeyratne**, Strategic Issues in Air Transport Legal, Economic and Technical Aspects, 2012.

[84] **I. Dodd and I. Habli**, "Safety Certification of Airborne Software: An Empirical Study," *ReliabilityEngineeringandSystemSafety*, vol. 98, pp. 7-23, 2012.

APPENDICES

APPENDIX A: System Requirements

APPENDIX A: System Requirements

Aircraft Level Performance Requirements

AVR- FLCS-01 : The flight control system must provide stable and controllable flight across all intended flight regimes (takeoff, cruise, landing, etc.).

Aircraft Level Functional Requirements

AVR-FLCS-01 : Air vehicle shall provide HOTAS interface with the pilot.

Aircraft Level Safety Requirements

AVR-FLCS-02 : The flight control system must have redundant components and pathways to ensure continued operation in case of a failure in one part of the system.

CURRICULUM VITAE

Name Surname **:** Hakan BALLI

EDUCATION

- **B.Sc.** **:** 2017, Yıldız Technical University,
  Faculty of Mechanical Engineering,
  Mechatronics Engineering *(Double Major)*

  2015, Yıldız Technical University,
  Faculty of Mechanical Engineering,
  Mechanical Engineering

PROFESSIONAL EXPERIENCE AND REWARDS

- 2017 – 2024 Turkish Aerospace
  KAAN, Turkish Fighter Aircraft Development Project
  Flight Dynamics, Simulation & Control Engineer
- 2016 – 2017 Turkish Aerospace
  HÜRKUŞ, Trainer Aircraft Project
  Flight Mechanics & Control Engineer

PUBLICATIONS, PRESENTATIONS AND PATENTS ON THE THESIS:

- **Ballı, H.**, Millidere, M., 2023, *Ankara International Aerospace Conference (AIAC)*, Online.