University of Cape Town

Guidance, Navigation & Control of a Small, Unmanned Blended Wing Body Aircraft

Author:
David van Wyk

Supervisor:
Prof. Hennie Mouton

Dissertation presented for the degree of Master of Science (MSc.)
in the Department of Mechanical Engineering

June 30, 2020

University of Cape Town

The copyright of this thesis vests in the author. No quotation from it or information derived from it is to be published without full acknowledgement of the source. The thesis is to be used for private study or non-commercial research purposes only.

Published by the University of Cape Town (UCT) in terms of the non-exclusive license granted to UCT by the author.

University of Cape Town

I know the meaning of plagiarism and declare that all the work in the document, save for that which is properly acknowledged, is my own. This dissertation has been submitted to the Turnitin module (or equivalent similarity and originality checking software) and I confirm that my supervisor has seen my report and any concerns revealed by such have been resolved with my supervisor.

Signed by candidate

David van Wyk

If I have seen further
it is by standing
on the shoulders of giants.

SIR ISAAC NEWTON

Abstract

The purpose of this research is to document the design and optimisation of a full suite of guidance, navigation and control (GNC) algorithms for a small unmanned aerial vehicle (UAV), the Skywalker X8. This was performed so as to fill a void in the available literature on the selected airframe, which currently only focuses on aspects such as aerodynamic modelling, advanced controller design, or uses of the airframe to perform higher level tasks. All of these research areas make use of off-the-shelf flight controllers, but these are not always the most appropriate foundations for more advanced work as they are inherently sluggish so as to be broadly applicable to a variety of airframes.

Subsequently, the Skywalker X8 airframe was modelled, using existing literature, and then characterised so as to establish what the goals might be for an optimal set of controllers. An autopilot was then designed which was optimised so as to be as close to the identified optimal performance characteristics as possible, with effort being put into ensuring that all non-linearities and disturbances were taken into account. This included advanced modelling of sensors, actuators, the environment, and the system itself.

The autopilot design was then extended with a set of guidance and navigation algorithms, also developed as part of this research. This consisted of both path planning and path following algorithms which allowed for the synthesis of general classes of paths useful to the application.

With both the autopilot and guidance laws developed, the system could be tested under several atmospheric flight conditions. These took the form of various wind directions and intensity levels being applied to the airframe whilst transitioning between a range of different waypoint configurations. The system was subsequently shown to be able to follow a set of waypoints very accurately, even with winds and turbulence with magnitudes of in excess of 60% of the aircraft's nominal airspeed. With a strong autopilot designed and illustrated in a high fidelity simulation environment, this work can now easily be extended into many fields.

All of the tools used for this research are available and well documented, and the processes followed repeatable with all justification available in the text. As such, should a project which aims to extend this work wish to adjust the autopilot design or guidance laws, based on different requirements, this is easily accomplished and recommendations of starting points are provided. The system model and autopilot are also made available and are usable exactly as they are should one wish to undertake additional research which does not aim to modify, but to extend this work.

Abbreviations & Symbols

Abbreviations & Acronyms

3D	Three Dimensional
6DoF	Six Degree of Freedom
AoA	Angle of Attack
CFD	Computational Fluid Dynamics
COESA	Committee on Extension to the Standard Atmosphere
EKF	Extended Kalman Filter
FPU	Floating Point Unit
GNC	Guidance, Navigation and Control
GPS	Global Positioning System
HiL	Hardware in the Loop
IMU	Inertial Measurement Unit
LTI	Linear Time Invariant
MEMS	Micro-Electro-Mechanical Systems
NED	North East Down (coordinate system)
PIDF	Proportional-Integral-Filtered-Derivative (controller)
PiL	Processor in the Loop
RMS	Root Mean Square
ROS	Robot Operating System
RTOS	Real Time Operating System
SiL	Software in the Loop
UAV	Unmanned Aerial Vehicle
VTOL	Vertical Take-Off and Landing

Symbols

α	Angle of attack (rad)
α_0	Stall angle (rad)
β	Sideslip angle (rad)
$\boldsymbol{\omega}^{\boldsymbol{b}}_{\boldsymbol{b/\mathcal{F}^i}}$	Angular velocity of the system with respect to the inertial frame, as expressed in the body frame (rad/s)
$\boldsymbol{\omega_{b/\mathcal{F}^i}}$	Angular velocity of the system with respect to the inertial frame (rad/s)
χ	Course angle (rad)
χ_c	Crab angle (rad)
χ_q	Course angle of the straight line path given by $\mathcal{P}_{line}(\mathbf{r}, \mathbf{q})$ (rad)
χ_{cmd}	Course angle command (rad)
δ_a	Aileron deflection angle (rad)
δ_e	Elevator deflection angle (rad)
$\delta_{e_{l_{cmd}}}$	Left elevon angular position command (rad)
δ_{e_l}	Left elevon angular position (rad)
$\delta_{e_{r_{cmd}}}$	Right elevon angular position command (rad)
δ_{e_r}	Right elevon angular position (rad)
$\delta_{t_{cmd}}$	Normalized propeller speed command (Unitless)
δ_t	Normalized propeller speed (Unitless)
$\dot{\phi}$	Temporal derivative of the roll angle, ϕ (rad/s)
$\dot{\psi}$	Temporal derivative of the yaw (or heading) angle, ψ (rad/s)
$\dot{\theta}$	Temporal derivative of the pitch angle, θ (rad/s)
$\dot{p_d}$	Temporal derivative of the inertial Down position of the aircraft (m/s)
$\dot{p_e}$	Temporal derivative of the inertial East position of the aircraft (m/s)
$\dot{p_n}$	Temporal derivative of the inertial North position of the aircraft (m/s)
η_{a_x}	Measurement noise on a_x (m/s^2)
η_{a_y}	Measurement noise on a_y (m/s^2)

η_{a_z}	Measurement noise on a_z (m/s^2)
$\frac{1}{2}\rho V_a^2$	Dynamic pressure (Pa)
γ	Flight path angle (rad)
γ_a	Air-mass-referenced flight-path angle (rad)
m	Mass of the system (kg)
$\mathbf{F^b}$	Sum of all external forces acting on the aircraft, as expressed in the body frame (N)
$\mathbf{F_a}$	Aerodynamic forces (N)
$\mathbf{F_g^b}$	Gravitational force in the body frame (N)
$\mathbf{F_g^v}$	Gravitational force in the vehicle frame (N)
$\mathbf{F_g}$	Gravitational forces (N)
$\mathbf{F_p}$	Propulsive forces (N)
$\mathbf{F}$	Sum of all external forces acting on the aircraft (N)
$\mathbf{h^b}$	Angular momentum in vector form, as expressed in the body frame (kg.m^2/s)
$\mathbf{h}$	Angular momentum in vector form (kg.m^2/s)
$\mathbf{J}$	Inertia matrix (or tensor) (kg.m^2
$\mathbf{M^b}$	Sum of all externally applied moments about the center of mass of the system, as expressed in the body frame (Nm)
$\mathbf{M_a}$	Aerodynamic moments (Nm)
$\mathbf{M_p}$	Propulsive moments (Nm)
$\mathbf{M}$	Sum of all externally applied moments about the center of mass of the system (Nm)
$\mathbf{V_a^b}$	Airspeed velocity in the body frame (m/s)
$\mathbf{V_a^i}$	Airspeed velocity in the inertial frame (m/s)
$\mathbf{V_a^w}$	Airspeed velocity in the wind frame (m/s)
$\mathbf{V_a}$	Airspeed vector (m/s)
$\mathbf{V_g^b}$	Velocity of the system with respect to the inertial frame - expressed in the body frame (m/s)

$\mathbf{V_g^i}$	Ground speed vector in the inertial frame (m/s)
$\mathbf{V_g}$	Ground speed vector (m/s)
$\mathbf{V_w^b}$	Wind velocity in the body frame (m/s)
$\mathbf{V_w}$	Wind velocity vector relative to the inertial frame (m/s)
$\mathbf{V_{w_g}}$	Stochastic process that represents wind gusts and other atmospheric disturbances (m/s)
$\mathbf{V_{w_s}}$	Constant ambient wind (m/s)
$\mathbf{W_i}$	ith waypoint, defined as $\mathbf{W_i} = (W_{n_i}, W_{e_i}, W_{d_i})^T \in \mathbb{R}^3$ (m)
$\mathbf{x}^*(t)$	Estimate of the state of the aircraft, $\mathbf{x}$, at time t
$\mathbf{x_m}(t)$	Measurements of the state of the aircraft, $\mathbf{x}$, at time t
$\mathbf{x}(t)$	State of the aircraft at time t
$\mathcal{F}^b$	Body coordinate frame
$\mathcal{F}^i$	Inertial coordinate frame
$\mathcal{F}^s$	Stability coordinate frame
$\mathcal{F}^v$	Vehicle coordinate frame
$\mathcal{F}^w$	Wind coordinate frame
$\mathcal{F}^{v_1}$	Vehicle-1 coordinate frame
$\mathcal{F}^{v_2}$	Vehicle-2 coordinate frame
$\mathcal{H}(\mathbf{r}, \mathbf{n})$	A half-plane in $\mathbb{R}^3$
$\mathcal{P}_{line}(\mathbf{r}, \mathbf{q})$	A straight-line path in $\mathbb{R}^3$
$\mathcal{P}_{orbit}(\mathbf{c}, \rho, \lambda)$	An orbit path described by a center, $\mathbf{c} \in \mathbb{R}^3$, a radius, $\rho \in \mathbb{R}$, and a direction, $\lambda \in \{-1, 1\}$
$\mathcal{R}_{v_2}^b(\phi)$	Rotation matrix from the vehicle-2 to the body coordinate frame
$\mathcal{R}_v^b(\phi, \theta, \psi)$	Rotation matrix from the vehicle to the body coordinate frame
$\mathcal{R}_w^b(\alpha, \beta)$	Rotation matrix from the wind to the body coordinate frame
$\mathcal{R}_b^s(\alpha)$	Rotation matrix from the body to the stability coordinate frame
$\mathcal{R}_v^{v_1}(\psi)$	Rotation matrix from the vehicle to the vehicle-1 coordinate frame

$\mathcal{R}_{v_1}^{v_2}(\theta)$	Rotation matrix from the vehicle-1 to the vehicle-2 coordinate frame
$\mathcal{R}_b^w(\alpha, \beta)$	Rotation matrix from the body to the wind coordinate frame
$\mathcal{R}_s^w(\beta)$	Rotation matrix from the stability to the wind coordinate frame
$\mathcal{R}_i^{\mathcal{P}}$	Rotation matrix from the inertial frame to the straight line path frame
$\mathcal{W}$	A set of waypoints defined as $\{\mathbf{W_1}, \mathbf{W_2}, ...\mathbf{W_n}\}$
ϕ	Roll angle (rad)
	Yaw (or heading) angle (rad)
ρ	Air density (kg/m^3)
θ	Pitch angle (rad)
ϱ	Fillet angle (rad)
ξ_ϕ	Process noise on ϕ (rad)
ξ_θ	Process noise on θ (rad)
a_x	Body frame acceleration along the $\mathbf{i^b}$ axis (m/s^2)
a_y	Body frame acceleration along the $\mathbf{j^b}$ axis (m/s^2)
a_z	Body frame acceleration along the $\mathbf{k^b}$ axis (m/s^2)
b	Aircraft wingspan (m)
c	Mean chord length of the aircraft wing (m)
C_χ	Course angle controller
C_ϕ	Roll angle controller
C_θ	Pitch angle controller
C_D	Non-dimensional drag force coefficient
C_L	Non-dimensional lift force coefficient
C_l	Non-dimensional roll moment coefficient
C_m	Non-dimensional pitch moment coefficient
C_n	Non-dimensional yaw moment coefficient
C_p	Roll rate damping controller

C_q	Pitch rate damping controller
C_X	Non-dimensional force coefficient in $\mathbf{i^b}$
C_Y	Non-dimensional force coefficient in $\mathbf{j^b}$
C_Z	Non-dimensional force coefficient in $\mathbf{k^b}$
C_{alt}	Altitude controller
C_{prop}	Dimensionless coefficient used to adjust the propeller thrust to account for unmodelled effects (eg. propeller pitch)
$C_{V_{a_{\delta_t}}}$	Airspeed controller via propeller speed (or thrust)
$C_{V_{a_\theta}}$	Airspeed controller via pitch angle
f_s	Sampling frequency (Hz)
f_x	Magnitude of the total force exerted on the aircraft along the $\mathbf{i^b}$ axis (N)
f_y	Magnitude of the total force exerted on the aircraft along the $\mathbf{j^b}$ axis (N)
f_z	Magnitude of the total force exerted on the aircraft along the $\mathbf{k^b}$ axis (N)
h	Altitude (m)
h_{cmd}	Altitude command (m)
k_Ω	Propeller speed scaling coefficient (rad/s)
k_{prop}	Scaling factor for propeller speed command, δ_t, to the exit velocity of air leaving the propeller (m/s)
k_{T_p}	Propeller reaction torque scaling coefficient (Nm.s/rad)
l	Sum of all externally applied moments about the $\mathbf{i^b}$ axis (Nm)
M	Flat plate stall transition rate
m	Sum of all externally applied moments about the $\mathbf{j^b}$ axis (Nm)
n	Sum of all externally applied moments about the $\mathbf{k^b}$ axis (Nm)
p	Roll rate measured around $\mathbf{i^b}$ in $\mathcal{F}^b$ (rad/s)
p_d	Inertial Down position of the system in the $\mathbf{k^i}$ direction in $\mathcal{F}^i$ (m)
p_e	Inertial East position of the system in the $\mathbf{j^i}$ direction in $\mathcal{F}^i$ (m)
p_n	Inertial North position of the system in the $\mathbf{i^i}$ direction in $\mathcal{F}^i$ (m)

q	Pitch rate measured around $\mathbf{j^b}$ in $\mathcal{F}^b$ (rad/s)
r	Yaw rate measured around $\mathbf{j^b}$ in $\mathcal{F}^b$ (rad/s)
S	Planform area of the wing (m^2)
S_{prop}	Area swept out by the aircraft propeller (m^2)
T_s	Sampling time (s)
u	Body frame velocity along $\mathbf{i^b}$ in $\mathcal{F}^b$ (m/s)
u_{w_g}	Speed of the stochastic wind in the body frame $\mathbf{i^b}$ direction (m/s)
v	Body frame velocity along $\mathbf{j^b}$ in $\mathcal{F}^b$ (m/s)
V_a	Airspeed magnitude (m/s)
V_g	Ground speed vector magnitude (m/s)
$V_{a_{cmd}}(t)$	Airspeed command at time t (m/s)
v_{w_g}	Speed of the stochastic wind in the body frame $\mathbf{j^b}$ direction (m/s)
w	Body frame velocity along $\mathbf{k^b}$ in $\mathcal{F}^b$ (m/s)
w_{d_s}	Speed of the steady wind in the Down direction (m/s)
w_{e_s}	Speed of the steady wind in the East direction (m/s)
w_{n_s}	Speed of the steady wind in the North direction (m/s)
w_{w_g}	Speed of the stochastic wind in the body frame $\mathbf{k^b}$ direction (m/s)

CONTENTS

1 Introduction **1**
1.1 Skywalker X8 . . . 1
1.2 Previous Research . . . 2
1.3 Dissertation Aims . . . 3
1.4 Simulation Environment . . . 6
1.5 Dissertation Structure . . . 6

2 Plant Modelling **10**
2.1 Coordinate Frames . . . 10
2.1.1 The Inertial Frame ($\mathcal{F}^i$) . . . 11
2.1.2 The Vehicle Frame ($\mathcal{F}^v$) . . . 12
2.1.3 The Vehicle-1 Frame ($\mathcal{F}^{v_1}$) . . . 12
2.1.4 The Vehicle-2 Frame ($\mathcal{F}^{v_2}$) . . . 13
2.1.5 The Body Frame ($\mathcal{F}^b$) . . . 14
2.1.6 The Stability Frame ($\mathcal{F}^s$) . . . 15
2.1.7 The Wind Frame ($\mathcal{F}^w$) . . . 16
2.1.8 Airspeed, Wind Speed, and Ground Speed . . . 17
2.1.9 The Wind Triangle . . . 19
2.2 Kinematics and Dynamics . . . 22
2.2.1 State Variables . . . 23
2.2.2 Kinematics . . . 24
2.2.3 Rigid Body Dynamics . . . 26
2.2.4 Implementation of 6DoF Model . . . 30
2.3 Forces and Moments . . . 32
2.3.1 Gravitational Forces . . . 32
2.3.2 Aerodynamic Forces and Moments . . . 33
2.3.3 Flat Plate Stall . . . 41
2.3.4 Propulsion Forces and Moments . . . 43
2.3.5 Implementation of Forces and Moments . . . 45
2.4 Implementation of Aircraft Model . . . 46
2.5 Wind Modelling . . . 47
2.5.1 Implementation of Wind Modelling . . . 48
2.6 Aircraft Performance . . . 49
2.6.1 Flight Envelope . . . 50
2.6.2 Performance Limits . . . 54
2.6.3 Skywalker X8 Performance Characterisation . . . 57
2.7 Linearisation . . . 63
2.7.1 Effects of Cross-Coupling . . . 64

2.7.2 Longitudinal Linearisation . . . 66
2.7.3 Lateral Linearisation . . . 74

3 Sensors and Actuators 79
3.1 Sensors . . . 79
3.1.1 Noise and Error Modelling . . . 80
3.1.2 GPS . . . 83
3.1.3 9-Axis IMU . . . 85
3.1.4 Airspeed Sensor . . . 88
3.1.5 Barometric Altitude Sensor . . . 89
3.1.6 Angle of Attack Sensor . . . 90
3.1.7 Benchmark Manoeuvre . . . 90
3.2 Actuators . . . 95
3.2.1 Elevon Servomotors . . . 95
3.2.2 Propeller Motor . . . 96

4 Autopilot Design 97
4.1 Gain Scheduling . . . 98
4.1.1 Scheduling Parameter Selection . . . 98
4.1.2 Longitudinal Scheduling Parameters . . . 99
4.1.3 Lateral Scheduling Parameters . . . 99
4.2 Longitudinal Controllers . . . 100
4.2.1 Pitch Angle Control . . . 103
4.2.2 Altitude Control . . . 108
4.2.3 Airspeed Control . . . 112
4.2.4 Longitudinal Controller State Machine . . . 117
4.2.5 Longitudinal Controller Verification . . . 118
4.3 Lateral Controllers . . . 124
4.3.1 Roll Angle Control . . . 127
4.3.2 Course Angle Control . . . 131
4.3.3 Lateral Controller Verification . . . 133
4.4 Digital Implementation . . . 137
4.4.1 Discretisation of Controllers . . . 137
4.4.2 Digital Longitudinal Controller Verification . . . 139
4.4.3 Digital Lateral Controller Verification . . . 142
4.5 Autopilot Verification . . . 144

5 State Estimation 148
5.1 Benchmark Manoeuvre . . . 149
5.2 Signal Conditioning . . . 149
5.3 Sensor Model Inversion . . . 150
5.4 Extended Kalman Filtering . . . 151
5.4.1 Attitude Estimation . . . 152
5.4.2 GPS Smoothing . . . 154

5.5 State Estimation Verification . . . 157

6 Guidance and Navigation 162
6.1 Straight Line and Orbit Following . . . 163
6.1.1 Straight Line Following . . . 164
6.1.2 Orbit Following . . . 170
6.2 Path Management . . . 174
6.2.1 Waypoint Following with Fillets . . . 174
6.3 Guidance and Navigation Verification . . . 181

7 System Integration and Verification 184
7.1 Benchmark Flight Paths . . . 185
7.1.1 Waypoint Path . . . 186
7.1.2 Alternate Waypoint Path . . . 187
7.2 Wind Inclusion . . . 189
7.2.1 Southwesterly Fresh Breeze . . . 190
7.2.2 Easterly Fresh Breeze . . . 195
7.3 Waypoint Path Switching . . . 200
7.3.1 Switching between $\mathbf{W_4}$ and $\mathbf{W_5}$ . . . 200
7.3.2 Switching between $\mathbf{W_6}$ and $\mathbf{W_7}$ . . . 202

8 Future Work and Project Extensions 204
8.1 Limitations and Improvements . . . 204
8.1.1 Plant Modelling . . . 205
8.1.2 Sensors and Actuators . . . 205
8.1.3 Autopilot Design . . . 206
8.1.4 State Estimation . . . 206
8.1.5 Guidance and Navigation . . . 207
8.1.6 System Integration and Verification . . . 208
8.2 Extension Points . . . 209

9 Conclusion 213

References 215

A Simulink Models 221
A.1 Plant . . . 222
A.1.1 Six Degree of Freedom Model . . . 222
A.1.2 Atmospheric Model . . . 223
A.1.3 Flight Envelope Generation Model . . . 224
A.2 Sensors . . . 225
A.2.1 GPS . . . 225
A.2.2 9-Axis IMU . . . 227
A.2.3 Airspeed Sensor . . . 232
A.2.4 Barometric Altitude Sensor . . . 233

A.2.5 Angle of Attack Sensor . . . 234
A.2.6 Sensor Suite . . . 235
A.2.7 Sensor Suite Verification . . . 236
A.3 Actuators . . . 236
A.3.1 Elevon Servomotors . . . 236
A.3.2 Propeller Motor . . . 237
A.4 Gain Scheduled Controllers . . . 238
A.4.1 Longitudinal Controllers . . . 238
A.4.2 Lateral Controllers . . . 239
A.5 Longitudinal Controller Tuning Model . . . 240
A.6 Lateral Controller Tuning Model . . . 241
A.7 Longitudinal State Machine . . . 242
A.8 Longitudinal Control Verification . . . 243
A.8.1 Continuous Controllers . . . 243
A.8.2 Discrete Controllers . . . 244
A.9 Lateral Control Verification . . . 245
A.9.1 Continuous Controllers . . . 245
A.9.2 Discrete Controllers . . . 246
A.10 Autopilot . . . 247
A.11 State Estimation . . . 249
A.12 Guidance and Navigation . . . 251
A.12.1 Path Following . . . 251
A.12.2 Path Management . . . 253
A.12.3 Guidance and Navigation Verification . . . 254
A.13 Guidance Navigation and Control Software in the Loop . . . 255

B MATLAB Scripts 256
B.1 Longitudinal Controller Tuning Goals . . . 257
B.1.1 Pitch Angle Controller . . . 257
B.1.2 Altitude Controller . . . 265
B.1.3 Airspeed Controller - Propeller Thrust . . . 273
B.1.4 Airspeed Controller - Pitch Angle . . . 279
B.2 Lateral Controller Tuning Goals . . . 286
B.2.1 Roll Angle Controller . . . 286
B.2.2 Course Angle Controller . . . 292
B.3 Guidance and Navigation Implementation . . . 298
B.3.1 Straight-Line Following Algorithm . . . 298
B.3.2 Orbit Following Algorithm . . . 300
B.3.3 Path Management Algorithm . . . 301

List of Figures

1 Introduction **1**

1.1 Skywalker X8 blended wing body aircraft 1
1.2 Guidance, Navigation and Control system architecture 3
1.3 Waypoint tracking reference objective 4
1.4 Alternate waypoint tracking reference objective 5

2 Plant Modelling **10**

2.1 The inertial frame, $\mathcal{F}^i$, and vehicle frame, $\mathcal{F}^v$ 12
2.2 The vehicle-1 frame, $\mathcal{F}^{v_1}$ 13
2.3 The vehicle-2 frame, $\mathcal{F}^{v_2}$ 13
2.4 The body frame, $\mathcal{F}^b$ 14
2.5 The stability frame, $\mathcal{F}^s$ 16
2.6 The wind frame, $\mathcal{F}^w$ 16
2.7 The flight path angle, γ, and course angle, χ 20
2.8 Aircraft following a ground track viewed from the $\mathbf{i^i}$-$\mathbf{j^i}$ plane to define: heading, ψ, sideslip angle, β, crab angle, χ_c, and course angle, χ 21
2.9 The wind triangle projected onto the $\mathbf{i}^b$-$\mathbf{k}^b$ plane to define: air-mass-referenced flight path angle, γ_a, angle of attack α, pitch angle, θ, and flight path angle, γ 21
2.10 Definition of axes for the equations of motion 24
2.11 Simulink implementation of 6DoF body kinematics Euler angles 31
2.12 Standard aircraft configuration with control surfaces [21] 34
2.13 Flying wing aircraft configuration with control surfaces [21] 34
2.14 Aerodynamic coefficient blending with flat plate stall model for $C_L(\alpha)$, $C_D(\alpha)$, and $C_m(\alpha)$ 43
2.15 Simulink implementation of aerodynamic forces and moments 46
2.16 Masked subsystem of the full Skywalker X8 aircraft model 47
2.17 Full Skywalker X8 aircraft model 49
2.18 Illustration of α for a pitching aircraft 54
2.19 Forces on an aircraft during a coordinated turn. Forces shown in $\mathbf{j^b}$-$\mathbf{k^b}$ plane 56
2.20 Performance limits of the Skywalker X8 across the flight envelope 61
2.21 Skywalker X8 longitudinal plant model 67
2.22 Bode plot of $\frac{q}{\delta_e}(s)$, with $V_a = \text{linspace}(9.89, 26.64, 10)$ m/s, $\alpha = 0$ rad . . 68
2.23 Bode plot of $\frac{\theta}{\delta_e}(s)$, with $V_a = \text{linspace}(9.89, 26.64, 10)$ m/s, $\alpha = 0$ rad . . 69
2.24 Bode plot of $\frac{h}{\delta_e}(s)$, with $V_a = \text{linspace}(9.89, 26.64, 10)$ m/s, $\alpha = 0$ rad . . 69
2.25 Bode plot of $\frac{\dot{q}}{\delta_e}(s)$, with varying (V_a, α) 72

2.26 Bode plot of $\frac{\theta}{\delta_e}(s)$, with varying (V_a, α) . . . 72
2.27 Bode plot of $\frac{h}{\delta_e}(s)$, with varying (V_a, α) . . . 73
2.28 Bode plot of $\frac{V_a}{\delta_t}(s)$, with varying V_a . . . 74
2.29 Skywalker X8 lateral plant model . . . 75
2.30 Bode plot of $\frac{p}{\delta_a}(s)$, with varying V_a . . . 77
2.31 Bode plot of $\frac{\phi}{\delta_a}(s)$, with varying V_a . . . 78
2.32 Bode plot of $\frac{\chi}{\delta_a}(s)$, with varying V_a . . . 78

3 Sensors and Actuators **79**
3.1 Gyroscope noise and error modelling [44] . . . 81
3.2 Generic sensor model indicating error and noise modelling utilized . . . 82
3.3 Sensor Suite model implementation . . . 91
3.4 Benchmark manoeuvre, h_c (m) . . . 92
3.5 Benchmark manoeuvre, χ_c (rad/s) . . . 92
3.6 Benchmark manoeuvre, V_{a_c} (m/s) . . . 93
3.7 Benchmark manoeuvre measurement, y_{p_e} (m) . . . 93
3.8 Benchmark manoeuvre measurement, y_χ (rad) . . . 94
3.9 Benchmark manoeuvre measurement, y_ψ (rad) . . . 94

4 Autopilot Design **97**
4.1 Successive loop feedback structure for the longitudinal controllers . . . 100
4.2 Gain-scheduled successive loop feedback structure for the longitudinal controllers . . . 100
4.3 Reduced-order linearisation of $\frac{q}{\delta_e}(s)$ in comparison to the full order model 101
4.4 Reduced-order linearisation of $\frac{\theta}{\delta_e}(s)$ in comparison to the full order model 102
4.5 Reduced-order linearisation of $\frac{h}{\delta_e}(s)$ in comparison to the full order model 102
4.6 Pitch angle controller architecture . . . 104
4.7 Pitch angle controller bode plot for $\frac{\theta}{\theta_c}(s)$ . . . 104
4.8 Pitch angle controller step plot for $\frac{\theta}{\theta_c}(s)$ . . . 105
4.9 Pitch angle controller step plot for $\frac{\theta}{\theta_c}(s)$, comparing a mid-point and gain scheduled controller . . . 106
4.10 Pitch angle controller open-loop bode plot for $\frac{\theta}{\theta_c}(s)$, comparing a mid-point and gain scheduled controller . . . 107
4.11 Pitch angle controller bode plot for $\frac{\theta}{\delta_e}(s)$ . . . 107
4.12 Altitude controller bode plot for $\frac{h}{h_c}(s)$ . . . 109
4.13 Altitude controller step plot for $\frac{h}{h_c}(s)$ . . . 109
4.14 Altitude controller bode plot for $\frac{h}{h_c}(s)$ comparing a mid-point and gain scheduled controller . . . 110
4.15 Altitude controller step plot for $\frac{h}{h_c}(s)$ comparing a mid-point and gain scheduled controller . . . 111
4.16 Altitude controller open loop bode plot for $\frac{h}{h_c}(s)$ comparing a mid-point and gain scheduled controller . . . 112

4.17 Airspeed controller, via propeller thrust, bode plot for $\frac{V_a}{V_{a_c}}(s)$ comparing a mid-point and gain scheduled controller 113
4.18 Airspeed controller, via propeller thrust, step plot for $\frac{V_a}{V_{a_c}}(s)$ comparing a mid-point and gain scheduled controller 114
4.19 Airspeed controller, via propeller thrust, open loop bode plot for $\frac{V_a}{V_{a_c}}(s)$ comparing a mid-point and gain scheduled controller 114
4.20 Airspeed controller, via pitch angle, bode plot for $\frac{V_a}{V_{a_c}}(s)$ comparing a mid-point and gain scheduled controller 115
4.21 Airspeed controller, via pitch angle, step plot for $\frac{V_a}{V_{a_c}}(s)$ comparing a mid-point and gain scheduled controller 116
4.22 Airspeed controller, via pitch angle, open loop bode plot for $\frac{V_a}{V_{a_c}}(s)$ comparing a mid-point and gain scheduled controller 117
4.23 Longitudinal autopilot state machine implementation [21] 118
4.24 Integrator freeze implementation 119
4.25 Longitudinal controller verification, α (rad) 120
4.26 Longitudinal controller verification, θ (rad) 121
4.27 Longitudinal controller verification, h (m) 121
4.28 Longitudinal controller verification, V_a (m/s) 122
4.29 Longitudinal controller verification, airspeed via pitch angle control, θ (rad) 123
4.30 Longitudinal controller verification, airspeed via pitch angle control, h (m) 123
4.31 Longitudinal controller verification, airspeed via pitch angle control, V_a (m/s) 124
4.32 Successive loop feedback structure for the lateral controllers 125
4.33 Gain scheduled successive loop feedback structure for the lateral controllers 125
4.34 Reduced order linearisation of $\frac{p}{\delta_a}(s)$ in comparison to the full order model 126
4.35 Reduced order linearisation of $\frac{\phi}{\delta_a}(s)$ in comparison to the full order model 126
4.36 Reduced order linearisation of $\frac{\chi}{\delta_a}(s)$ in comparison to the full order model 127
4.37 Roll angle controller architecture 128
4.38 Roll angle controller bode plot for $\frac{\phi}{\phi_c}(s)$, comparing a mid-point and gain scheduled controller 129
4.39 Roll angle controller step plot for $\frac{\phi}{\phi_c}(s)$, with mid-point controller omitted due to instabilities 130
4.40 Roll angle controller open-loop bode plot for $\frac{\phi}{\phi_c}(s)$, comparing a mid-point and gain scheduled controller 130
4.41 Course angle controller bode plot for $\frac{\chi}{\chi_c}(s)$, comparing a mid-point and gain scheduled controller 132
4.42 Course angle controller step plot for $\frac{\chi}{\chi_c}(s)$, comparing a mid-point and gain scheduled controller 132
4.43 Course angle controller open-loop bode plot for $\frac{\chi}{\chi_c}(s)$, comparing a mid-point and gain scheduled controller 133
4.44 Lateral controller verification, χ (rad) 135
4.45 Lateral controller verification, ϕ (rad) 135
4.46 Lateral controller verification, h (m) 136

4.47 Lateral controller verification, V_a (m/s) . . . 136
4.48 Digital longitudinal controller verification, α (rad) . . . 140
4.49 Digital longitudinal controller verification, θ (rad) . . . 140
4.50 Digital longitudinal controller verification, h (m) . . . 141
4.51 Digital longitudinal controller verification, V_a (m/s) . . . 141
4.52 Digital lateral controller verification, χ (rad) . . . 142
4.53 Digital lateral controller verification, ϕ (rad) . . . 143
4.54 Digital lateral controller verification, h (m) . . . 143
4.55 Digital lateral controller verification, V_a (m/s) . . . 144
4.56 Autopilot implementation masked subsystem . . . 145
4.57 Digital autopilot verification, h (m) . . . 145
4.58 Digital autopilot verification, χ (rad) . . . 146
4.59 Digital autopilot verification, V_a (m/s) . . . 146
4.60 Digital autopilot verification, α (rad) . . . 147

5 State Estimation **148**
5.1 Digital autopilot with observable feedback verification, h (m) . . . 158
5.2 Digital autopilot with observable feedback verification, χ (rad) . . . 158
5.3 Digital autopilot with observable feedback verification, V_a (m/s) . . . 159
5.4 State estimation verification, ϕ (rad) . . . 159
5.5 State estimation verification, θ (rad) . . . 160
5.6 State estimation verification, χ (rad) . . . 160
5.7 State estimation verification, h (m) . . . 161

6 Guidance and Navigation **162**
6.1 Orthographic view of an aircraft travelling relative to the straight-line segment $\mathcal{P}_{line}(\mathbf{r}, \mathbf{q})$ . . . 164
6.2 Top view of an aircraft travelling relative to the straight-line segment $\mathcal{P}_{line}(\mathbf{r}, \mathbf{q})$ . . . 165
6.3 Project of the straight-line path following problem onto the $\mathbf{q}$-$\mathbf{k}^\mathbf{i}$ plane . . . 167
6.4 Course angle command adjustment to account for current course angle . . . 169
6.5 Orbit following geometry . . . 171
6.6 Orbit following inclined plane altitude following . . . 172
6.7 b-ball switching between waypoint segments . . . 175
6.8 Fillet insertion between line segments $\overline{\mathbf{W_{i-1}W_i}}$ and $\overline{\mathbf{W_iW_{i+1}}}$ . . . 176
6.9 Geometry of fillet insertion between line segments $\overline{\mathbf{W_{i-1}W_i}}$ and $\overline{\mathbf{W_iW_{i+1}}}$ 177
6.10 Geometry of fillet insertion between line segments $\overline{\mathbf{W_{i-1}W_i}}$ and $\overline{\mathbf{W_iW_{i+1}}}$ indicating half planes needed for the path following algorithm . . . 178
6.11 Masked guidance and navigation system . . . 180
6.12 Guidance and navigation system verification, orthographic view . . . 182
6.13 Guidance and navigation system verification, $\mathbf{i}^\mathbf{i}$-$\mathbf{j}^\mathbf{i}$ plane view . . . 182
6.14 Guidance and navigation system verification, $\mathbf{j}^\mathbf{i}$-$\mathbf{k}^\mathbf{i}$ plane view . . . 183

7 System Integration and Verification **184**
7.1 SiL implementation of the full Guidance, Navigation and Control (GNC) system . . . 185
7.2 GNC system verification, orthographic view . . . 186
7.3 GNC system verification, $\mathbf{i^i}$-$\mathbf{j^i}$ plane view . . . 187
7.4 GNC system verification, $\mathbf{j^i}$-$\mathbf{k^i}$ plane view . . . 187
7.5 GNC system verification, alternate waypoints, orthographic view . . . 188
7.6 GNC system verification, alternate waypoints, $\mathbf{i^i}$-$\mathbf{j^i}$ plane view . . . 188
7.7 GNC system verification, alternate waypoints, $\mathbf{i^i}$-$\mathbf{k^i}$ plane view . . . 189
7.8 GNC system verification, southwesterly fresh breeze wind magnitude . . . 190
7.9 GNC system verification, southwesterly fresh breeze, orthographic view . . . 191
7.10 GNC system verification, southwesterly fresh breeze, $\mathbf{i^i}$-$\mathbf{j^i}$ plane view . . . 192
7.11 GNC system verification, southwesterly fresh breeze, $\mathbf{j^i}$-$\mathbf{k^i}$ plane view . . . 192
7.12 GNC system verification, alternate waypoints, southwesterly fresh breeze, orthographic view . . . 193
7.13 GNC system verification, alternate waypoints, southwesterly fresh breeze, $\mathbf{i^i}$-$\mathbf{j^i}$ plane view . . . 194
7.14 GNC system verification, alternate waypoints, southwesterly fresh breeze, $\mathbf{i^i}$-$\mathbf{k^i}$ plane view . . . 194
7.15 GNC system verification, easterly fresh breeze wind magnitude . . . 195
7.16 GNC system verification, easterly fresh breeze, orthographic view . . . 196
7.17 GNC system verification, easterly fresh breeze, $\mathbf{i^i}$-$\mathbf{j^i}$ plane view . . . 197
7.18 GNC system verification, easterly fresh breeze, $\mathbf{j^i}$-$\mathbf{k^i}$ plane view . . . 197
7.19 GNC system verification, alternate waypoints, easterly fresh breeze, orthographic view . . . 198
7.20 GNC system verification, alternate waypoints, easterly fresh breeze, $\mathbf{i^i}$-$\mathbf{j^i}$ plane view . . . 199
7.21 GNC system verification, alternate waypoints, easterly fresh breeze, $\mathbf{i^i}$-$\mathbf{k^i}$ plane view . . . 199
7.22 GNC system verification, waypoint switching between $\mathbf{W_4}$ and $\mathbf{W_5}$, orthographic view . . . 201
7.23 GNC system verification, waypoint switching between $\mathbf{W_4}$ and $\mathbf{W_5}$, $\mathbf{i^i}$-$\mathbf{j^i}$ plane view . . . 201
7.24 GNC system verification, waypoint switching between $\mathbf{W_6}$ and $\mathbf{W_7}$, orthographic view . . . 202
7.25 GNC system verification, waypoint switching between $\mathbf{W_6}$ and $\mathbf{W_7}$, $\mathbf{i^i}$-$\mathbf{j^i}$ plane view . . . 203

8 Future Work and Project Extensions **204**
8.1 Skywalker X8 as modelled in the Gazebo simulation environment . . . 209
8.2 Static obstacle avoidance path planning [76] . . . 210
8.3 Area mapping [77] . . . 211

9 Conclusion **213**

A Simulink Models **221**
A.1 Simulink implementation of the full Skywalker X8 aircraft model 222
A.2 Skywalker X8 aircraft model including wind modelling 223
A.3 Simulink implementation of the performance characterization model . . . 224
A.4 GPS Sensor implementation 225
A.5 GPS Sensor masked subsystem 226
A.6 3-Axis Accelerometer sensor model implementation 227
A.7 3-Axis Accelerometer masked subsystem 228
A.8 3-Axis Gyroscope sensor model implementation 229
A.9 3-Axis Gyroscope masked subsystem 230
A.10 Magnetometer Compass sensor model implementation 230
A.11 Magnetometer Compass masked subsystem 231
A.12 9-Axis MEMS IMU masked subsystem 231
A.13 Airspeed Sensor model 232
A.14 Airspeed Sensor model masked subsystem 232
A.15 Barometric Altitude Pressure Sensor model 233
A.16 Barometric Altitude Pressure Sensor model masked subsystem 233
A.17 Angle of Attack Sensor model 234
A.18 Angle of Attack Sensor model masked subsystem 234
A.19 Full sensor suite model 235
A.20 Sensor Suite verification model 236
A.21 Servo-Motor actuator model 236
A.22 Servo-Motor actuator model masked subsystem 237
A.23 Propeller Motor actuator model masked subsystem 237
A.24 Standard longitudinal gain scheduled controller as a function of (V_a, α), using the altitude controller as an example 238
A.25 Standard lateral gain scheduled controller as a function of V_a, using the airspeed controller as an example 239
A.26 Longitudinal control system tuning model 240
A.27 Lateral control system tuning model 241
A.28 Longitudinal State Machine implementation in Stateflow 242
A.29 Longitudinal controller verification implementation 243
A.30 Digital longitudinal controller verification implementation 244
A.31 Lateral controller verification implementation 245
A.32 Digital lateral controller verification implementation 246
A.33 Autopilot implementation 247
A.34 Autopilot implementation verification 248
A.35 State Estimation implementation 249
A.36 State Estimation verification 250
A.37 Path following algorithm implementation 251
A.38 Masked path following algorithm implementation 252
A.39 Path management shown in the context of the full guidance and navigation system 253
A.40 Guidance and navigation verification model implementation 254

A.41 Guidance, navigation and control implementation verification model . . . 255

B MATLAB Scripts **256**

List of Tables

1 Introduction **1**

2 Plant Modelling **10**
2.1 System State Variables 23
2.2 Physical System Parameters for the Skywalker X8 Aircraft [3] 32
2.3 C_D Aerodynamic Coefficients for the Skywalker X8 Aircraft 38
2.4 C_Y Aerodynamic Coefficients for the Skywalker X8 Aircraft 38
2.5 C_L Aerodynamic Coefficients for the Skywalker X8 Aircraft 39
2.6 C_l Aerodynamic Coefficients for the Skywalker X8 Aircraft 39
2.7 C_m Aerodynamic Coefficients for the Skywalker X8 Aircraft 39
2.8 C_n Aerodynamic Coefficients for the Skywalker X8 Aircraft 39
2.9 Stall Coefficients for the Skywalker X8 Aircraft 42
2.10 Propeller Coefficients for the Skywalker X8 Aircraft 45
2.11 Required Parameters for the Aircraft Performance Characterisation . . . 58

3 Sensors and Actuators **79**
3.1 GPS Gauss-Markov Error Model Parameters 83
3.2 9-Axis IMU Model Noise Parameters 87
3.3 9-Axis IMU Model Sensor Dynamics Parameters 87
3.4 9-Axis IMU Model Sensor Non-Linearity Parameters 88
3.5 Airspeed Sensor Model Parameters 89
3.6 Barometric Altitude Pressure Sensor Model Parameters 90
3.7 Angle of Attack Sensor Model Parameters 90
3.8 Elevon Actuator Model Parameters 95

4 Autopilot Design **97**
4.1 Longitudinal State Machine Parameters 118
4.2 Longitudinal Controller Saturation Parameters 120
4.3 Lateral Controller Saturation Parameters 133

5 State Estimation **148**
5.1 State Estimation Low Pass Filter Parameters 150

6 Guidance and Navigation **162**
6.1 Guidance and Navigation Design Parameters 181

7 System Integration and Verification **184**

8 Future Work and Project Extensions **204**

9 Conclusion **213**

A Simulink Models **221**

B MATLAB Scripts **256**

Listings

1 Introduction **1**

2 Plant Modelling **10**

3 Sensors and Actuators **79**

4 Autopilot Design **97**

5 State Estimation **148**

6 Guidance and Navigation **162**

7 System Integration and Verification **184**

8 Future Work and Project Extensions **204**

9 Conclusion **213**

A Simulink Models **221**

B MATLAB Scripts **256**

B.1 Pitch angle controller tuning goals, SkywalkerX8_Longitudinal_ControlDesign_VaAlpha.m 257

B.2 Altitude controller tuning goals, SkywalkerX8_Longitudinal_ControlDesign_VaAlpha.m 265

B.3 Airspeed controller, using propeller speed, tuning goals, SkywalkerX8_Longitudinal_ControlDesign_Va.m 273

B.4 Airspeed controller, using pitch angle, tuning goals, SkywalkerX8_Longitudinal_ControlDesign_VaAlpha.m 279

B.5 Roll angle controller tuning goals, SkywalkerX8_Lateral_ControlDesign_Va.m 286

B.6 Course angle controller tuning goals, SkywalkerX8_Lateral_ControlDesign_Va.m 292

B.7 Straight line following algorithm, straightLineFollowing.m 298

B.8 Orbit following algorithm, orbitFollowing.m 300

B.9 Path management fillets algorithm, pathManager_followWaypointsFillet.m 301

Introduction

The purpose of this dissertation is to document the design and optimisation of a full suite of guidance, navigation and control (GNC) algorithms for a small unmanned aerial vehicle (UAV), the Skywalker X8.

The subsequent sections of this chapter will discuss why the particular aircraft was chosen, what the specific aims of this dissertation are, and how this document has been structured in order to accomplish these aims.

1.1 Skywalker X8

Figure 1.1: Skywalker X8 blended wing body aircraft

The Skywalker X8 is a small, blended wing body type UAV that is most commonly found in the hobbyist category of airframes. It makes use of a single propeller for thrust, and two elevons (one per wing) in order to control its pitch and roll dynamics - with the term elevon being a combination of the words elevator and aileron, which perform these

two respective roles in a conventional airframe.

This airframe was selected for this project for a number of reasons:

1. It is commonly available and would be easily procurable should future work be done on the system whereby flight testing might need to be performed.

2. Given that it does not make use of a rudder to control its motion in yaw, it presents a more challenging control system design problem than a conventional airframe.

3. It is fairly well-established in the literature, and as such removes the need for experimental testing to be done to characterise the system as this has been done in other works already.

As such, the flight system that this dissertation will use as its focus will be the Skywalker X8. It should be noted, however, that a large majority of the work and methods presented in this text are broadly applicable to any rudderless system (indeed, even to conventional airframes should a sideslip controller be added), and as such, with a simple replacement of the plant, following the same process should enable similar results.

1.2 Previous Research

The majority of the previous work on the Skywalker X8 airframe has to do with: aerodynamic modelling and system identification [1, 2]; advanced controller design [3–6]; or uses of the airframe to perform tasks such as the observation of glacier dynamics [7, 8], alien plant species detection and monitoring [9], turbulence measurement in the atmospheric boundary layer [10], and crop analysis [11], amongst others.

However, there is very little literature that exists on the development of the autopilot and guidance laws upon which all of this research is based. The base commonly used for most of the existing research is some form of generic autopilot system from off the shelf autopilots such as the Pixhawk 4 [12]. While this is indeed a very well-developed and maintained system, the algorithms employed are applied to very many airframes and not optimised for any of them in particular, with only a minor calibration being done as part of pre-flight checks for the system. For very slow operation this will not prove to be much of a disadvantage, but for anything more advanced, or which requires faster dynamic response, this will not be adequate.

As such, the results of any research which makes use of a generic autopilot such as this, are based on a system that is very likely tuned to be quite slow so as to be broadly applicable to many airframes and only requiring minor calibration. This means that if more advanced techniques are built on top of it, it could severely impact the results of any analyses subsequently performed and conclusions drawn. This is because if the foundation of a system is not optimal, anything built on top of that system intuitively

cannot be fully optimal.

This work aims to fill this gap in the literature by providing details on how to develop a well-designed and well-tuned guidance, navigation and control system for the airframe. This can then be extended without any concern that the underlying system is not properly optimised. This is beneficial because one can then state more conclusively that a particular technique built around the system is either effective or not, without any concern that it is the underlying autopilot inhibiting the performance.

Another key contribution to the literature is that the methods by which this design has been accomplished, as well as how it could be repeated, and even adjusted, are also provided in the text. These are invaluable, as should there be a requirement to retune the design presented for a more specific case - or perhaps eve, a slight variation on the plant used - the same methodology can be followed, and even the same software used. This is because everything developed has purposefully been built to be extremely modular, well-documented and reusable, thus making extensions to this research trivial.

1.3 Dissertation Aims

The primary aim of this dissertation is to present the methodology used to develop a GNC suite for a Skywalker X8 aircraft. A summary of the overall system architecture utilized throughout the text is denoted in Figure 1.2.

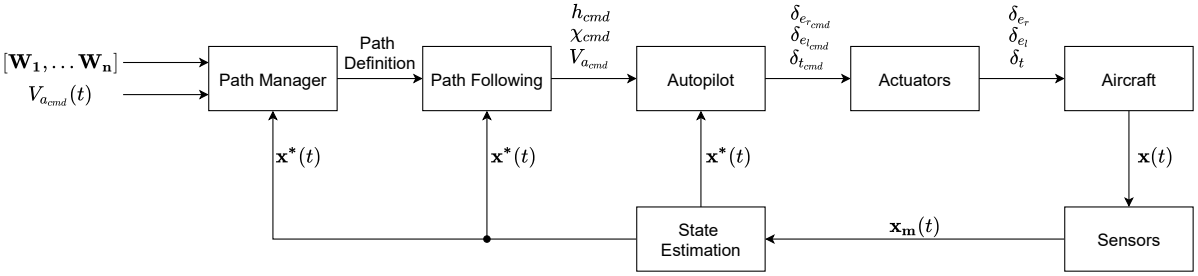


Figure 1.2: Guidance, Navigation and Control system architecture

In Figure 1.2 we see that our two main inputs are: 1. A set of waypoints, $[\mathbf{W_1}, ...\mathbf{W_n}]$, and 2. An airspeed command (as a function of time), $V_{a_{cmd}}(t)$. It should be noted that each of our waypoints is defined as a position in $\mathbb{R}^3$ such that $\mathbf{W_i} = (W_n, W_e, W_d)^T$. These values are then used by the path manager block, along with the estimate of the current state of the aircraft, $\mathbf{x}^*(t)$, to define the path which needs to be followed. This is done so as to accurately transition between each of the provided waypoints. The path definition is then provided to the path following block which takes this information, as well as the estimate of the current state of the aircraft, and determines the input commands to provide to the autopilot so as to accomplish the commanded manoeuvre. The autopilot takes the input commands from the path following algorithm - namely the altitude command, h_c, the course angle command, χ_c, and the airspeed command, $V_{a_{cmd}}$

- as well as the estimate of the system's current state, $\mathbf{x}^*(t)$, from the state estimator, and uses this information to determine the required actuator commands, $\delta_{e_{r_{cmd}}}$, $\delta_{e_{l_{cmd}}}$, and $\delta_{t_{cmd}}$, to safely transition to the required altitude, course angle and airspeed. These actuator commands are then sent to the actuators which respond in some small, but non-negligible, amount of time to the commanded values. Once the actuators transition from their initial state, they apply an input to the aircraft in the form of forces and moments acting on the airframe. These forces and moments induce motion in the system, which subsequently results in a change of the system's state, $\mathbf{x}(t)$. These state changes are measured by sensors located on the airframe. The sensed values, $\mathbf{x_m}(t)$, are then sent to the state estimator to determine an estimate of the current system state which is then used in the next time step by all of our previous systems as described.

Once each of the elements depicted in Figure 1.2 has been designed and optimised, the resultant GNC algorithms will then be confirmed to perform as expected. This will be accomplished by providing the system with a set of waypoints and an airspeed command and ensuring that we can accurately transition between each of the waypoints and along the resulting line segments connecting them.

The primary goal for this research is thus succinctly summed up by providing the set of reference waypoints seen in Figure 1.3 and stating that our system should be designed such that this set of waypoints can be travelled between, even in the presence of wind.

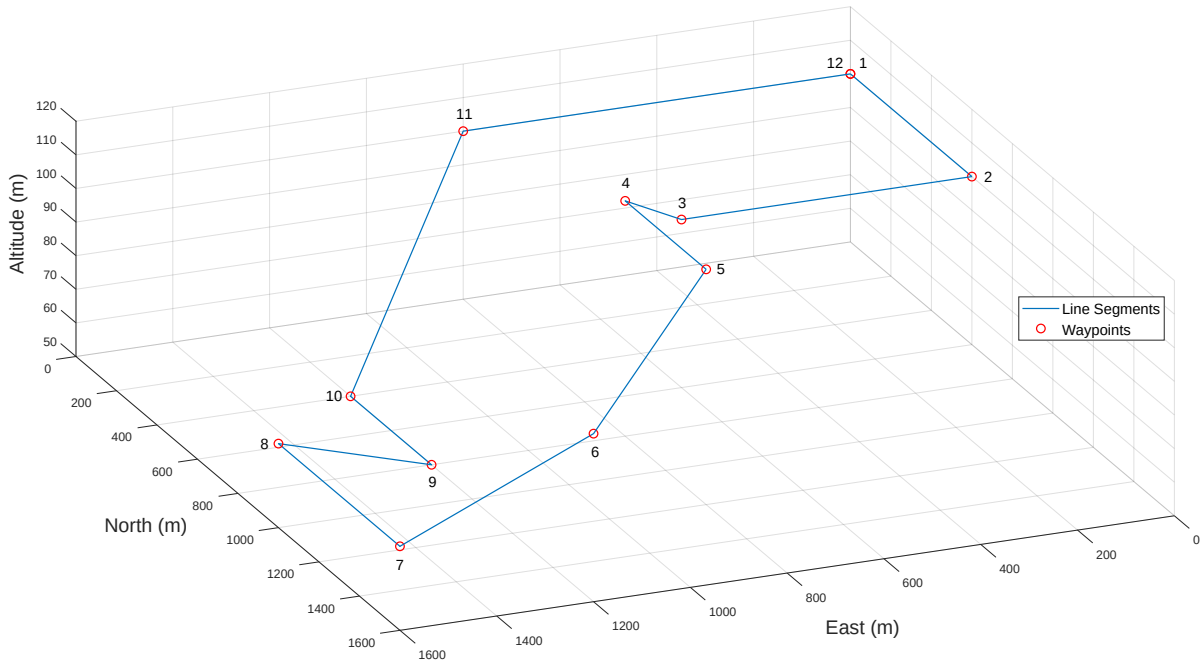


Figure 1.3: Waypoint tracking reference objective

In order to ensure that the system can work in a realistic environment, we extend this requirement to include the ability to switch between sets of waypoints. Once a new set

of waypoints is passed to the system, the aircraft should treat the closest waypoint of the new set as the last waypoint reached and should start travelling to the next. The alternate set of waypoints used as a reference are denoted in Figure 1.4. This is similar to the original set, but is such that waypoints 5 and 8, as well as waypoints 6 and 7, have been swapped around in comparison to the original set depicted in Figure 1.3.

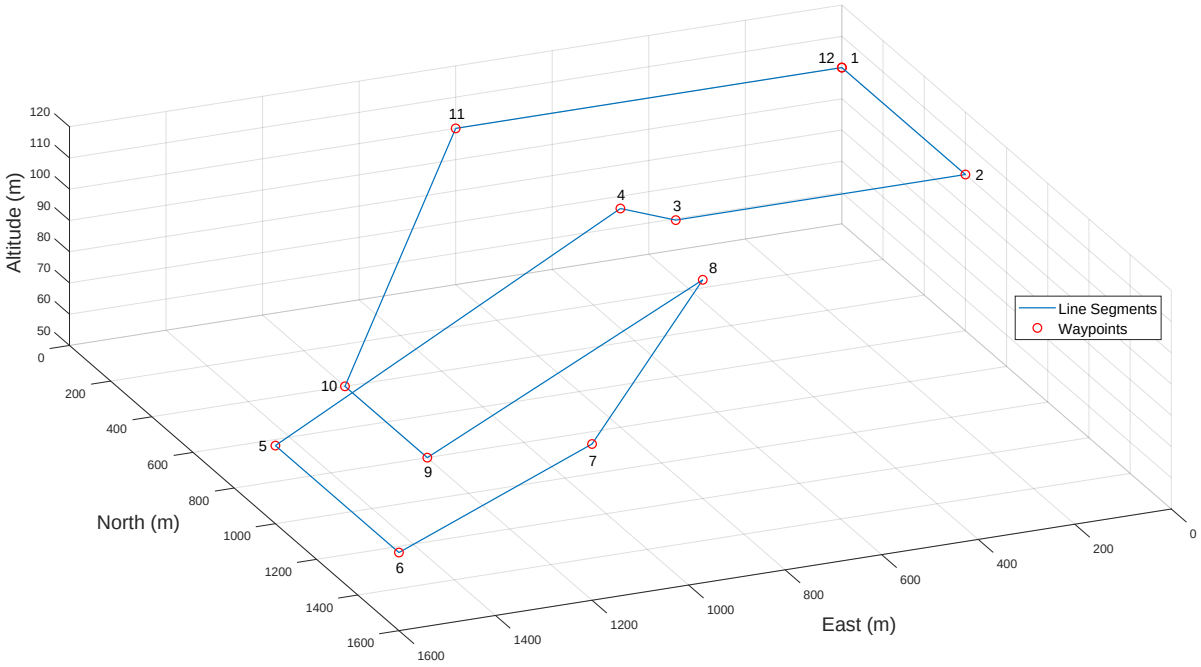


Figure 1.4: Alternate waypoint tracking reference objective

We require that these waypoints be traversed in the presence of wind and, given our aircraft size and expected velocities, we will add as a requirement that this needs to occur in fairly severe wind speeds. In order to quantify this, we state that we need to be able to operate in a fresh breeze, with a Beaufort scale number of 5. This will have a magnitude of between 8.0-10.8 m/s [13].

A secondary objective of this dissertation is to serve as the basis for future projects. Having a well-defined and modelled system and a full autopilot available, extending this work into other areas of research becomes a strong possibility.

As an example, computer vision research is a prime candidate as instead of a user providing the path manager with a set of waypoints, these waypoints can be determined on-board, or with a ground station, by using image processing techniques.

Extensions such as these are fairly complicated in and of themselves and have all of their own challenges. As such, having a solid foundation that already has been shown to work well in simulation, under even the most severe of conditions, means that the focus can be put on the actual objectives of the research being done, and not first having to develop a

working base system. Having access to all of the tools and processes which were used to generate the underlying autopilot and guidance laws is also invaluable. This is because should any adjustments need to be made, all of the tools used, as well as the insight and justification for all decisions, are available in both this text, as well as the accompanying MATLAB code itself.

Some suggestions on how to extend this work will be presented at the end of the document.

1.4 Simulation Environment

All work in this project is performed in the MATLAB/Simulink environment. The specifics of this environment are as follows:

1. MATLAB 9.4
2. Simulink 9.1
3. Aerospace Blockset 3.21
4. Aerospace Toolbox 2.2
5. Control System Toolbox 10.4
6. DSP System Toolbox 9.6
7. Robotics System Toolbox 2.0
8. Simulink Coder 8.14
9. Simulink Control Design 5.1
10. Stateflow 9.1
11. System Identification Toolbox 9.8

The reader is also directed to [14], where the full Simulink project is available with all required scripts, models, tools and utilities provided.

1.5 Dissertation Structure

This dissertation makes use of Figure 1.2 as the high level system architecture for the entire project. As such, each of the chapters to follow covers a specific element (or elements) of the high level design items presented in the figure, starting from the right hand side with the plant, and moving toward the left, with the guidance and navigation algorithms.

We will start by first establishing the full aircraft model from first principles in Chapter 2. This will firstly cover the required coordinate frames made use of, which will also serve as an introduction to the standards by which system parameters are referred to in this text. The kinematics and dynamics of a generic six degree of freedom system will then be developed, followed by a discussion on developing a model for all of the required forces and moments generated by, and applied to, the Skywalker X8 airframe. With this established, the implementation of the aircraft model can be discussed, as well as how this might be extended to include atmospheric effects such as wind. The aircraft performance limitations will then be derived and discussed, so as to create realistic goals for autopilot tuning moving forward. Finally, the system will be linearised, and the impact on the linearised dynamics of the system examined as a function of its state.

Having fully discussed the aircraft itself, the next item to consider is how one can interact with this system. Chapter 3 considers this by firstly looking at how one can measure the outputs from the system, or the system state. This is done by establishing how one can model each of the sensors which would be mounted on the airframe. These sensors are used to measure the current system state, but with several limitations which will be discussed. The outputs of the sensor models are then illustrated by making use of a benchmark manoeuvre, which is widely used throughout the text to illustrate operation of various elements. Following this, we consider how one might control the aircraft by deflecting its control surfaces or adjusting the thrust generated by the propeller. In order to do this, there are dynamics introduced by the actuators controlling these elements themselves, and these are discussed and characterised so that they can be taken into account when designing the autopilot and running simulations further on in the design process.

With the aircraft, as well as all of its interfaces, defined, the next item to consider in Figure 1.2 is the autopilot itself. This is covered in Chapter 4. In this chapter we first consider the impact on the aircraft dynamics by its state, as was done in Chapter 2, and use this to justify the concept of gain scheduling, as well as to determine the parameters that are most suitably used as scheduling parameters. The longitudinal controllers are then designed by using classical control techniques with a set of linearised plant models, but with the final designs being verified against the full non-linear plant model developed in Chapter 2. This same process is then followed for the lateral controllers. With the controllers designed in the continuous domain, we consider the effect of discretising these so that they might be implemented on a digital flight computer. We then discuss the selection of sampling times for each of the controllers and the impact this might have on their performance. The discretised controllers are all then verified in the same manner as their continuous counterparts. This is done to ensure that there is no significant performance degradation observed due to the discretisation process. Finally, the full autopilot is verified against the benchmark manoeuvre established in Chapter 2 in order to illustrate exemplary performance even with aggressive input commands likely to not be encountered in practice.

It is not always possible to directly measure each state of the aircraft required for the GNC suite, and certain parameters need to be estimated. For this reason, the next design element considered is that of state estimation, which is covered in Chapter 5. This first considers signal conditioning and how we go about filtering noisy sensor measurements. Following this, we consider the idea of sensor model inversion, where we use a sensor measurement that does not directly provide an output we are interested in, but can be converted to one with some simple algebraic manipulation. We then consider the extended Kalman filter and how this can be used to allow us to estimate the attitude of the aircraft, as well as to smooth GPS measurements and increase their update rate. Finally, our state estimation techniques are verified against the benchmark manoeuvre established in Chapter 2 in order to compare what our estimated state would be in comparison to the known state and ensure that this provides adequate performance to continue with the design.

It should be clear from Figure 1.2 that at this stage in the design process, a fully working autopilot has been developed which can take in altitude, course angle and airspeed commands and control the aircraft to achieve the commanded values. As such, the next step which needs to be covered in order to achieve our goal of tracking the waypoint paths defined in Figures 1.3 and 1.4 is to develop the guidance and navigation laws. These algorithms will create the command signals for the autopilot needed for the aircraft to accomplish transitioning between the defined waypoints. This is covered in Chapter 6, where we first consider our path following algorithms, and follow this with the path management algorithm, as depicted in Figure 1.2. The path following algorithms begin with the observation that any path can be made up of straight lines and circular orbits. Due to this, we establish a guidance algorithm for each of these cases and leave how these straight lines and orbits are to be combined to the path manager, which is subsequently discussed. At the end of this chapter, we consider the full GNC system in calm conditions and illustrate how it can traverse the waypoint path defined in Figure 1.3 with the expected level of accuracy.

This marks the end of the design phase in the project, with the full system as depicted in Figure 1.2 having now been designed, optimised and verified. All that remains to do is to verify that this works under all of the conditions specified as design goals in Section 1.3. These are namely that we can accurately traverse the waypoint paths specified in the presence of significant wind conditions, as well as allow for switching between sets of waypoints whilst still ensuring adequate flight behaviour if this is requested by the user. These requirements are verified in their entirety in Chapter 7. This chapter illustrates how each of the design requirements set out in Section 1.3 are met by making use of several high fidelity simulations which consider various scenarios constructed to test the system to its limits in order to do so.

Finally, in Chapter 8 we consider how this project might be improved upon, or used as a base for future research. This includes some examples of what this extension point research might focus on. In order to facilitate this, guidance is provided on the first

steps which could be undertaken to accomplish these tasks, as well as references which could be made use of as a starting point.

CHAPTER 2

Plant Modelling

The first, and perhaps the most critical, step in the development of a control system is to fully understand and characterise the plant one wishes to control - in this case the Skywalker X8 aircraft. This is depicted as the aircraft block in Figure 1.2. In order to do so we will first establish the coordinate frames within which we are working, so that all future discussions on angles and rates can be defined relative to the correct frame. We then will consider the kinematics and dynamics of the aircraft to establish our six degree of freedom (6DoF) model, followed next by establishing how to calculate the forces and moments acting on the 6DoF system which result in linear and angular accelerations, respectively. After having built the full 6DoF model, some atmospheric effects will be included in the form of wind modelling.

With the full aircraft model defined, some analysis can be performed so as to characterise the system and glean insight into what is and isn't possible. This analysis is key to future design work as it presents our first insights into the system which we want to control and what our design goals might look like for our controllers. We start this process off by firstly looking at the flight envelope within which the Skywalker X8 is able to operate, and define how this is restricted for this project. We then consider how turning might be accomplished by reviewing the coordinated turn methodology. Finally, we start linearising the model and gaining insight into its trim conditions and the small signal analysis around these points.

2.1 Coordinate Frames

In order to allow for discussion regarding the dynamic behaviour of the Skywalker X8, several coordinate systems are of interest. The reason why it is necessary to use several different coordinate systems is as follows:

1. Newtonian physics is defined relative to a fixed, inertial reference frame. However, motion is most easily described in a body-fixed frame.

2. Aerodynamic forces and moments act on the aircraft body and are most easily described in a body-fixed reference frame.

3. On-board sensors like accelerometers and rate gyroscopes measure information with respect to the body frame. Alternatively, GPS measures position, ground speed and course angle with respect to the inertial frame.

4. Most mission requirements, like waypoints, are specified in the inertial frame.

Any coordinate system can be transformed into another by some combination of rotation and translation. In this section, the following coordinate frames will be defined and described [15–18]:

1. The inertial frame ($\mathcal{F}^i$)
2. The vehicle frame ($\mathcal{F}^v$)
3. The vehicle-1 frame ($\mathcal{F}^{v_1}$)
4. The vehicle-2 frame ($\mathcal{F}^{v_2}$)
5. The body frame ($\mathcal{F}^b$)
6. The stability frame ($\mathcal{F}^s$)
7. The wind frame ($\mathcal{F}^w$)

The inertial and vehicle frames are related by a simple translation, whereas the remaining frames are related by rotations. All rotations in this text will be defined making use of Euler angles in order to allow for easier analysis and discussion, at the expense of a slightly more tricky implementation that a system like quaternions would easily resolve.

The angles that define the relative orientations of the vehicle, vehicle-1, vehicle-2 and body frame are the roll (ϕ), pitch (θ) and yaw (ψ) angles that describe the attitude of the aircraft [19, 20].

The angles that define the relative orientations of the body, stability and wind frames are the angle of attack (α) and sideslip (β) angles.

All coordinate frames discussed in this text assume a flat, non-rotating earth. Given the size and range of our aircraft, this assumption is perfectly valid.

2.1.1 The Inertial Frame ($\mathcal{F}^i$)

The inertial coordinate system is an earth-fixed coordinate system with its origin at the defined launch location (or wherever we choose our origin to be at launch). As depicted in Figure 2.1, the unit vector $\mathbf{i}^i$ is directed North, $\mathbf{j}^i$ is directed East, and $\mathbf{k}^i$ is directed toward the center of the earth (or Down). This is known as the North-East-Down (NED) coordinate system.

2.1.2 The Vehicle Frame ($\mathcal{F}^v$)

The origin of the vehicle frame is at the center of mass of the aircraft. This is the only difference to the inertial frame, with this being depicted in Figure 2.1, where see that the axes of $\mathcal{F}^v$ are aligned with $\mathcal{F}^i$. In other words, the unit vector $\mathbf{i}^v$ is directed North, $\mathbf{j}^v$ is directed East, and $\mathbf{k}^v$ is directed toward the center of the earth (or Down). As such, the transformation from $\mathcal{F}^i$ to $\mathcal{F}^v$ only requires a translation.

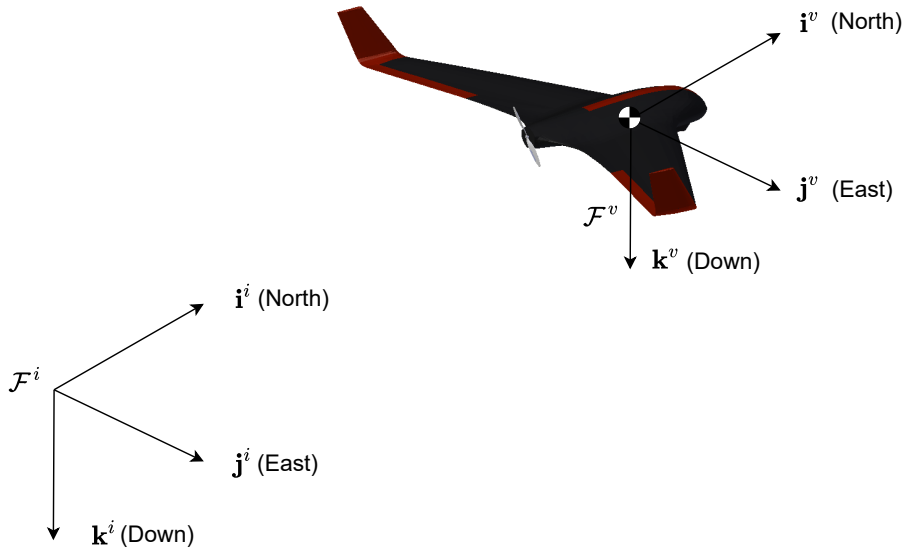


Figure 2.1: The inertial frame, $\mathcal{F}^i$, and vehicle frame, $\mathcal{F}^v$

2.1.3 The Vehicle-1 Frame ($\mathcal{F}^{v_1}$)

The origin of the Vehicle-1 frame is the same as that of the vehicle frame, at the center of mass of the aircraft. However, $\mathcal{F}^{v_1}$, is rotated in the positive right-hand direction about $\mathbf{k}^v$ by the yaw angle, ψ. As such, $\mathbf{i}^{v_1}$ points out the nose of the airframe, $\mathbf{j}^{v_1}$ out the right wing and $\mathbf{k}^{v_1}$ is aligned with $\mathbf{k}^v$ and points down. This is illustrated in Figure 2.2.

The transform from $\mathcal{F}^v$ to $\mathcal{F}^{v_1}$ is given by:

$$\mathbf{p}^{v_1} = \mathcal{R}_v^{v_1}(\psi)\mathbf{p}^v$$

where:

$$\mathcal{R}_v^{v_1}(\psi) = \begin{pmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

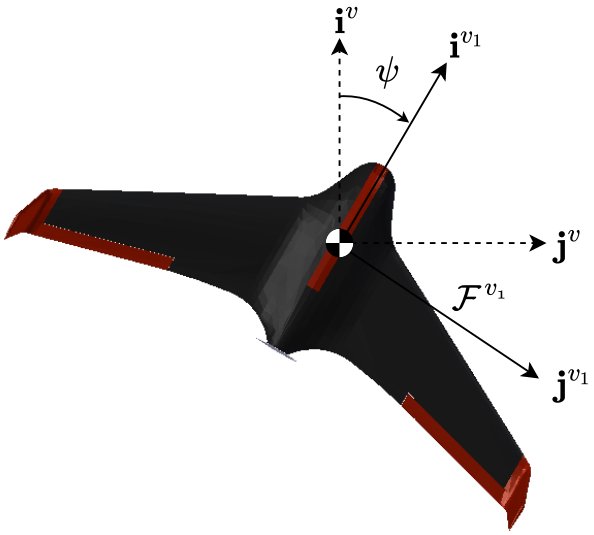


Figure 2.2: The vehicle-1 frame, $\mathcal{F}^{v_1}$

2.1.4 The Vehicle-2 Frame ($\mathcal{F}^{v_2}$)

The origin of the vehicle-2 frame is also that of the vehicle frame - the center of mass of the aircraft. It is obtained by rotating the vehicle-1 frame in the positive right-hand direction about the $\mathbf{j}^{v_1}$ axis by the pitch angle, θ. As such, $\mathbf{i}^{v_2}$ points out the nose of the airframe, $\mathbf{j}^{v_2}$ out the right wing and $\mathbf{k}^{v_2}$ points out the undercarriage. This is illustrated in Figure 2.3.

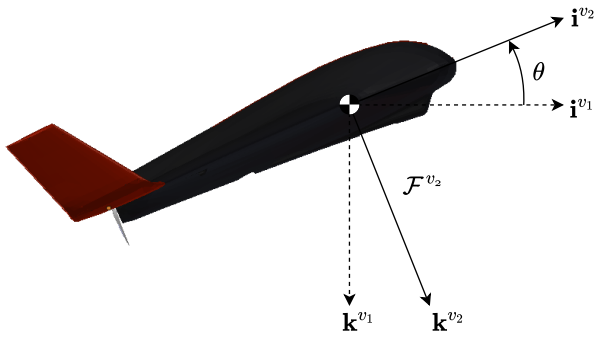


Figure 2.3: The vehicle-2 frame, $\mathcal{F}^{v_2}$

The transform from $\mathcal{F}^{v_1}$ to $\mathcal{F}^{v_2}$ is given by:

$$\mathbf{p}^{v_2} = \mathcal{R}^{v_2}_{v_1}(\theta)\mathbf{p}^{v_1}$$

where:

$$\mathcal{R}_{v_1}^{v_2}(\theta) = \begin{pmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{pmatrix}$$

2.1.5 The Body Frame ($\mathcal{F}^b$)

The origin of the body frame is also that of the vehicle frame - the center of mass of the aircraft. It is obtained by rotating the vehicle-2 frame in the positive right-hand direction about $\mathbf{i}^{v_2}$ by the roll angle, ϕ. As such, $\mathbf{i}^b$ points out the nose of the airframe, $\mathbf{j}^b$ out the right wing and $\mathbf{k}^b$ points out the undercarriage. This is illustrated in Figure 2.4.

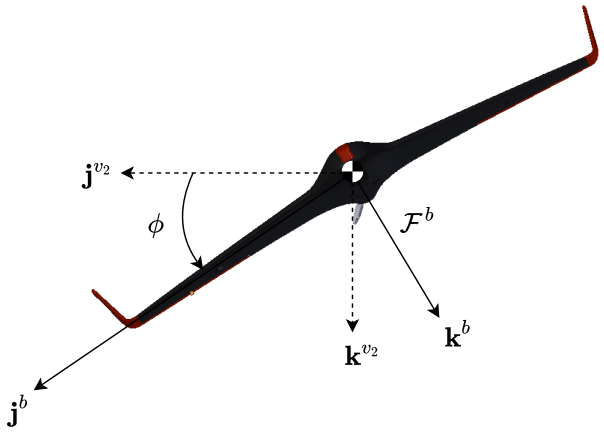


Figure 2.4: The body frame, $\mathcal{F}^b$

The transform from $\mathcal{F}^{v_2}$ to $\mathcal{F}^b$ is given by:

$$\mathbf{p}^b = \mathcal{R}_{v_2}^b(\phi)\mathbf{p}^{v_2}$$

where:

$$\mathcal{R}_{v_2}^b(\phi) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{pmatrix}$$

Having defined all of the required rotations, we can now determine the transformation

from the vehicle frame to the body frame as:

$$\begin{aligned}\mathcal{R}_v^b(\phi,\theta,\psi) &= \mathcal{R}_{v_2}^b(\phi)\mathcal{R}_{v_1}^{v_2}(\theta)\mathcal{R}_v^{v_1}(\psi) \\ &= \begin{pmatrix} c_\theta c_\psi & c_\theta s_\psi & -s_\theta \\ s_\phi s_\theta c_\phi - c_\phi s_\psi & s_\phi s_\theta s_\psi + c_\phi c_\psi & s_\phi c_\theta \\ c_\phi s_\theta c_\phi + s_\phi s_\psi & c_\phi s_\theta s_\psi - s_\phi c_\psi & c_\phi c_\theta \end{pmatrix}\end{aligned} \tag{2.1}$$

where we make use of the shorthand notation $c_x \triangleq \cos x$ and $s_y \triangleq \sin y$.

The angles ϕ, θ, and ψ are commonly referred to as Euler angles. These are commonly used because they provide an intuitive means by which the orientation of a body in three dimensions can be represented. The rotation sequence made use of in Equation (2.1) is of the ψ-θ-ϕ type, and is just one of many Euler angle systems in use [19]. For this rotation sequence, there is a singularity when the pitch angle, θ is $\pm 90°$, in which case the yaw angle is not defined. An alternate method which frees one from singularities such as these is the quaternion system, which are also more computationally efficient. However, these lack the intuitive appeal of Euler angles and so are not made use of for this dissertation.

2.1.6 The Stability Frame ($\mathcal{F}^s$)

Aerodynamic forces are generated as the aircraft moves through the air surrounding it. We refer to the velocity of the aircraft relative to surrounding air as the airspeed vector, denoted as $\mathbf{V_a}$. The magnitude of the airspeed vector is simply denoted as V_a. To generate lift, the wings of the airframe must fly at a positive angle with respect to the airspeed vector. This angle is called the angle of attack (AoA), which is denoted by α. As shown in Figure 2.5, the AoA is defined as a left handed rotation about $\mathbf{j}^b$ and is such that $\mathbf{i}^s$ aligns with the projection of $\mathbf{V_a}$ onto the plane spanned by $\mathbf{i}^b$ and $\mathbf{k}^b$. The left handed rotation is used because, by definition, angle of attack is positive when using a right-handed rotation from the stability frame $\mathbf{i}^s$ axis, to the body frame $\mathbf{i}^b$ axis.

The transform from $\mathcal{F}^b$ to $\mathcal{F}^s$ is given by:

$$\mathbf{p}^s = \mathcal{R}_b^s(\alpha)\mathbf{p}^b$$

where:

$$\mathcal{R}_b^s(\alpha) = \begin{pmatrix} \cos\alpha & 0 & \sin\alpha \\ 0 & 1 & 0 \\ -\sin\alpha & 0 & \cos\alpha \end{pmatrix}$$

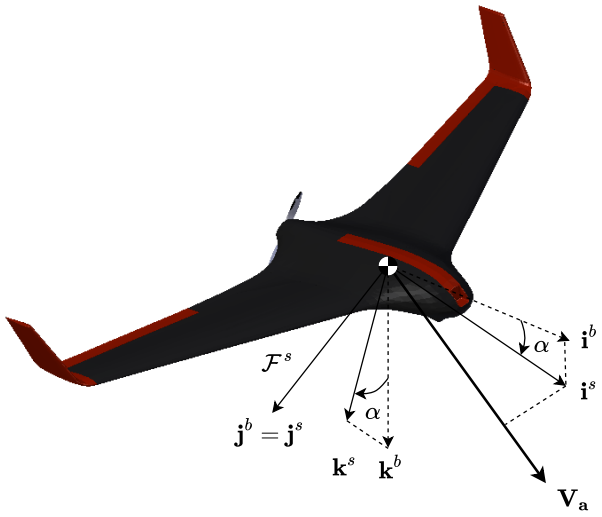


Figure 2.5: The stability frame, $\mathcal{F}^s$

2.1.7 The Wind Frame ($\mathcal{F}^w$)

The angle between the airspeed vector and the $\mathbf{i}^b$-$\mathbf{k}^b$ plane is called the sideslip angle and is denoted by β. As depicted in Figure 2.6, the wind frame is obtained by rotating the stability frame using a right-handed rotation of β about $\mathbf{k}^s$. The unit vector $\mathbf{i}^w$ is aligned with the airspeed vector, $\mathbf{V_a}$.

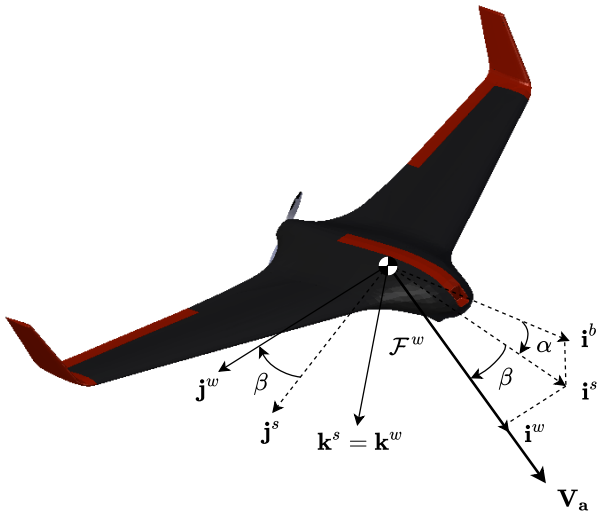


Figure 2.6: The wind frame, $\mathcal{F}^w$

The transform from $\mathcal{F}^s$ to $\mathcal{F}^w$ is given by:

$$\mathbf{p}^w = \mathcal{R}_s^w(\beta)\mathbf{p}^s$$

where:

$$\mathcal{R}_s^w(\beta) = \begin{pmatrix} \cos\beta & \sin\beta & 0 \\ -\sin\beta & \cos\beta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Having defined all of the required rotations, we can now derive the transformation from the body frame to the wind frame as:

$$\begin{aligned} \mathcal{R}_b^w(\alpha,\beta) &= \mathcal{R}_s^w(\beta)\mathcal{R}_b^s(\alpha) \\ &= \begin{pmatrix} c_\beta c_\alpha & s_\beta & c_\beta s_\alpha \\ -s_\beta c_\alpha & c_\beta & -s_\beta s_\alpha \\ -s_\alpha & 0 & c_\alpha \end{pmatrix} \end{aligned} \tag{2.2}$$

where we have used the same shorthand notation as was defined earlier, where $c_x \triangleq \cos x$ and $s_y \triangleq \sin y$.

However, a more useful transform is that from the wind frame to the body frame, which is calculated as:

$$\mathcal{R}_w^b(\alpha,\beta) = \mathcal{R}_b^w(\alpha,\beta)^T = \begin{pmatrix} c_\beta c_\alpha & -s_\beta c_\alpha & -s_\alpha \\ s_\beta & c_\beta & 0 \\ c_\beta s_\alpha & -s_\beta s_\alpha & c_\alpha \end{pmatrix} \tag{2.3}$$

2.1.8 Airspeed, Wind Speed, and Ground Speed

In order to develop the dynamic equations of motion for our system, it should be remembered that the inertial forces experienced by it are dependent on velocities and accelerations relative to the inertial frame. The aerodynamic forces and moments, however, depend on the velocity of the airframe relative to the surrounding air. If the system was in an environment with no wind, these velocities would be the same. However, there is almost always wind present in a realistic system and the relative magnitude for a small aircraft can be quite substantial - wind magnitude can be 20-50% of our nominal airspeed [21] - so we cannot treat it as negligible, as one might do for a larger airframe. As such, we must carefully distinguish between the velocity relative to the surrounding air, $\mathbf{V_a}$, and the ground speed, represented by the velocity with respect to the inertial frame, $\mathbf{V_g}$. The velocities are related by the expression:

$$\mathbf{V_a} = \mathbf{V_g} - \mathbf{V_w} \tag{2.4}$$

where $\mathbf{V_w}$ is the wind velocity relative to the inertial frame.

The system velocity, $\mathbf{V_g}$, can be expressed in the body frame in terms of its components along the $\mathbf{i}^b$, $\mathbf{j}^b$ and $\mathbf{k}^b$ axes as:

$$\mathbf{V_g^b} = \begin{pmatrix} u \\ v \\ w \end{pmatrix}$$

where $\mathbf{V}_g^b$ is the velocity of the system with respect to the inertial frame, but expressed in the body frame.

Since we usually know wind velocities in the North, w_n, East, w_e, and Down, w_d frame, we can express this in the body frame (where it is required to determine aerodynamic forces and moments) as:

$$\mathbf{V_w^b} = \begin{pmatrix} u_w \\ v_w \\ w_w \end{pmatrix} = \mathcal{R}_v^b(\phi, \theta, \psi) \begin{pmatrix} w_n \\ w_e \\ w_d \end{pmatrix} \tag{2.5}$$

By definition, the airspeed vector, $\mathbf{V_a}$, is the velocity of the system with respect to the wind, it can thus be expressed in the wind frame as:

$$\mathbf{V_a^w} = \begin{pmatrix} V_a \\ 0 \\ 0 \end{pmatrix}$$

If we define the components of our airspeed vector in the body frame as $(u_r, v_r, w_r)^T$, we can make use of Equation (2.4) to express it in the body frame as:

$$\mathbf{V_a^b} = \begin{pmatrix} u_r \\ v_r \\ w_r \end{pmatrix} = \begin{pmatrix} u - u_w \\ v - v_w \\ w - w_w \end{pmatrix}$$

This is an extremely important expression as this is the form of our airspeed which is used to calculate aerodynamic forces and moments. It is also very convenient as it is easily determined in simulation because our body-frame velocity components, u, v and w are states of the system and are available from the solution to the equations of motion. The wind velocity components, u_w, v_w and w_w, are usually derived from a wind model in the inertial frame (ie. w_n, w_e, w_d) which can then be rotated as in Equation (2.5).

Combining these expressions, we can express our airspeed vector body-frame components

in terms of the airspeed magnitude, angle of attack, and sideslip angle as:

$$\begin{aligned}\mathbf{V_a^b} &= \begin{pmatrix} u_r \\ v_r \\ w_r \end{pmatrix} = \mathcal{R}_w^b(\alpha, \beta) \begin{pmatrix} V_a \\ 0 \\ 0 \end{pmatrix} \\ &= \begin{pmatrix} c_\beta c_\alpha & -s_\beta c_\alpha & -s_\alpha \\ s_\beta & c_\beta & 0 \\ c_\beta s_\alpha & -s_\beta s_\alpha & c_\alpha \end{pmatrix} \begin{pmatrix} V_a \\ 0 \\ 0 \end{pmatrix}\end{aligned}$$

From this we can conveniently express our airspeed vector in the body frame as:

$$\mathbf{V_a^b} = \begin{pmatrix} u_r \\ v_r \\ w_r \end{pmatrix} = V_a \begin{pmatrix} \cos\alpha\cos\beta \\ \sin\beta \\ \cos\beta\sin\alpha \end{pmatrix} \tag{2.6}$$

Finally, we can invert this to derive equations for V_a, α and β in terms of known values which we will have in simulation. These are calculated as:

$$\begin{aligned} V_a &= \sqrt{u_r^2 + v_r^2 + w_r^2} \\ \alpha &= \tan^{-1}\left(\frac{w_r}{u_r}\right) \\ \beta &= \sin^{-1}\left(\frac{v_r}{\sqrt{u_r^2 + v_r^2 + w_r^2}}\right) \end{aligned} \tag{2.7}$$

Given that aerodynamic forces and moments are commonly stated in terms of these parameters, these expressions are essential in formulating the equations of motion for our system.

2.1.9 The Wind Triangle

For small UAVs, the wind speed is often in the range of 20-50% of the airspeed, as was mentioned in the previous section. As such, it contributes significantly to the system forces and moments and is thus more important to understand than it might be for much larger conventional aircraft, where the airspeed is much larger than the wind speed. Having previously discussed coordinate frames, airframe velocity, wind velocity and the airspeed vector, we can now discuss some important definitions relating to UAV navigation.

The direction of the ground speed vector relative to an inertial frame is specified using two angles. These angles are the course angle, χ, and the (inertially referenced) flight path angle, γ. Figure 2.7 shows how these angles are defined. The flight path angle, γ is defined as the angle between the horizontal plane and the ground velocity vector,

$\mathbf{V_g}$; while the course angle, χ, is the angle between the projection of the ground velocity vector to the horizontal plane and true north, or the $\mathbf{i^i}$ vector, as defined by the inertial reference frame.

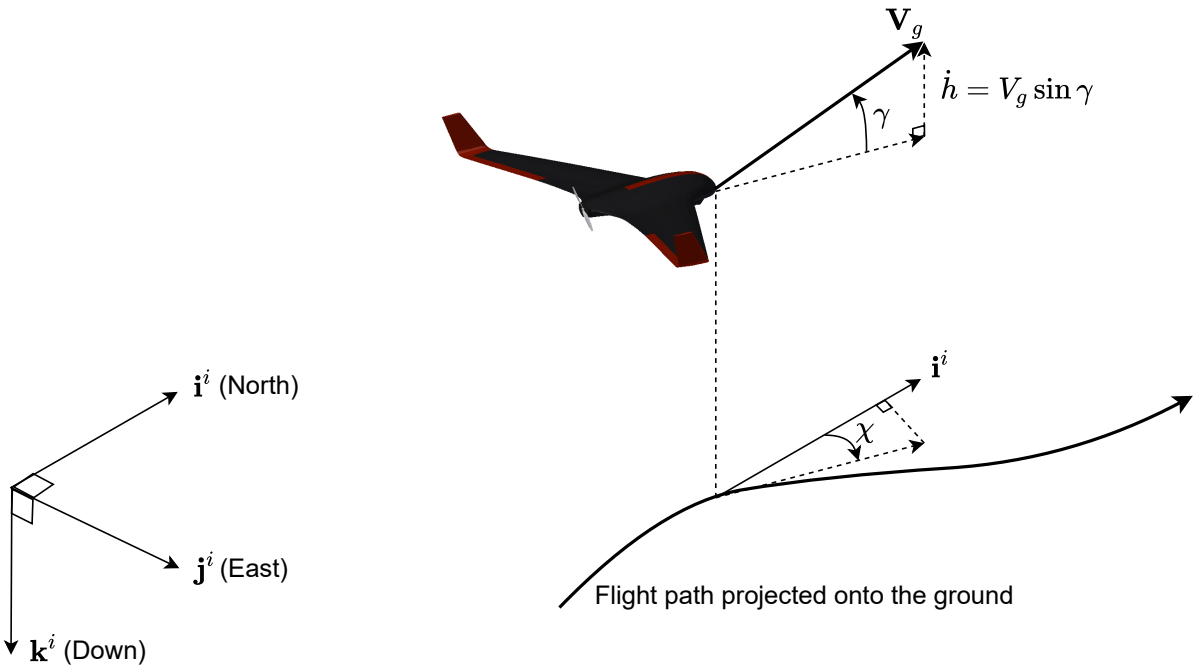


Figure 2.7: The flight path angle, γ, and course angle, χ

This relationship, between the ground speed vector, the airspeed vector, and the wind vector, which is given by Equation (2.4), is called the wind triangle. This is depicted in more detail in the inertial horizontal (or $\mathbf{i^i}$-$\mathbf{j^i}$) plane in Figure 2.8, and in the body vertical (or $\mathbf{i^b}$-$\mathbf{k^b}$) plane in Figure 2.9.

Figure 2.8 shows an aircraft following a ground track represented by the dashed line. The North direction is indicated with the $\mathbf{i^i}$ vector, and the direction that the vehicle is pointed is shown, by definition, by the $\mathbf{i^b}$ vector in the body frame. For level flight (ie. $\theta \approx 0$), the heading (or yaw) angle, ψ, is the angle between $\mathbf{i^i}$ and $\mathbf{i^b}$, and describes the direction that the vehicle is pointed. The direction the vehicle is travelling with respect to the surrounding air mass is given, by definition, by the airspeed vector, $\mathbf{V_a}$. In steady, level flight, $\mathbf{V_a}$ is commonly aligned with $\mathbf{i^b}$, meaning the sideslip angle, β, is usually zero. The direction the vehicle is travelling with respect to the ground is shown, by definition, by the velocity vector $\mathbf{V_g}$. The angle between the inertial north, defined as $\mathbf{i^i}$, and the inertial velocity vector projected on the local North-East (or $\mathbf{i^i}$-$\mathbf{j^i}$) plane is called the course angle, χ. If there is a constant ambient wind, the aircraft will need to "crab" into the wind in order to follow a ground track that is not aligned with the wind. The "crab angle", χ_c, is defined as the difference between the course and heading

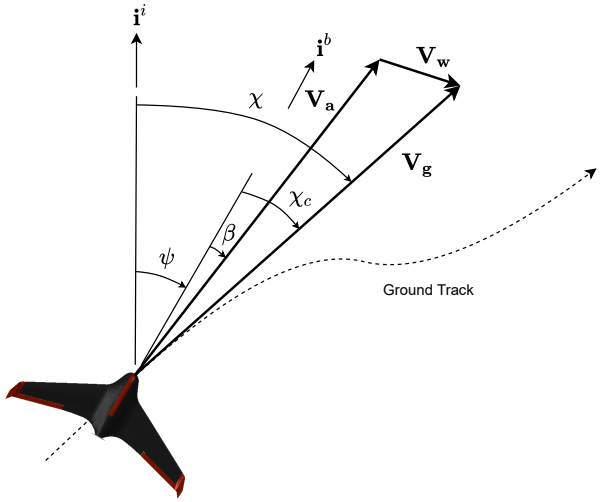


Figure 2.8: Aircraft following a ground track viewed from the $\mathbf{i^i}$-$\mathbf{j^i}$ plane to define: heading, ψ, sideslip angle, β, crab angle, χ_c, and course angle, χ

angles as follows:

$$\chi_c \triangleq \chi -$$

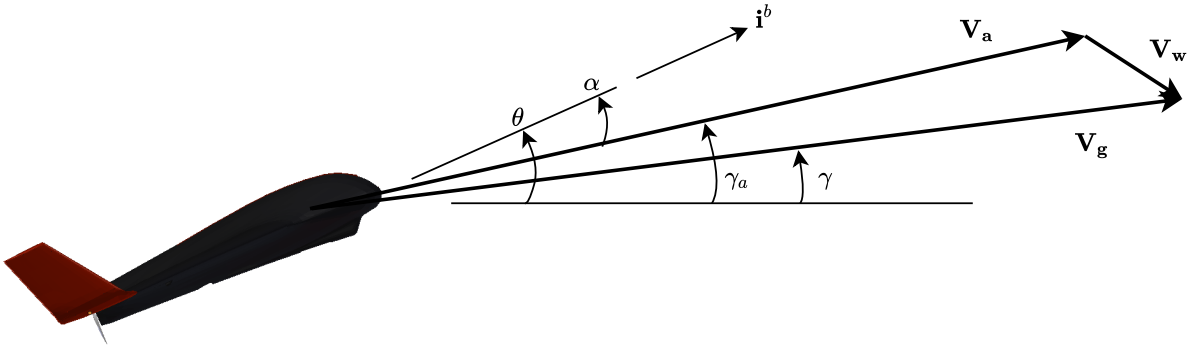


Figure 2.9: The wind triangle projected onto the $\mathbf{i}^b$-$\mathbf{k}^b$ plane to define: air-mass-referenced flight path angle, γ_a, angle of attack α, pitch angle, θ, and flight path angle, γ

Figure 2.9 depicts the vertical components of the wind triangle. When there is a down component of wind, we define the angle from the inertial North-East (or $\mathbf{i}^i$-$\mathbf{k}^i$) plane to $\mathbf{V_a}$ as the air-mass-referenced flight-path angle and denote it as γ_a. The relationship between the air-mass-referenced flight-path angle, γ_a, the angle of attack, α, and the pitch angle, θ, is given by:

$$\gamma_a = \theta - \alpha$$

In the absence of wind, the flight-path angle and air-mass-referenced flight-path angle are equivalent such that, $\gamma_a = \gamma$.

The ground speed vector in the inertial frame can be expressed as:

$$\mathbf{V_g^i} = \begin{pmatrix} \cos\chi & -\sin\chi & 0 \\ \sin\chi & \cos\chi & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos\gamma & 0 & \sin\gamma \\ 0 & 1 & 0 \\ -\sin\gamma & 0 & \cos\gamma \end{pmatrix} \begin{pmatrix} V_g \\ 0 \\ 0 \end{pmatrix} = V_g \begin{pmatrix} \cos\chi\cos\gamma \\ \sin\chi\cos\gamma \\ -\sin\gamma \end{pmatrix}$$

where $V_g = \|\mathbf{V_g}\|$.

Similarly, the airspeed vector in the inertial frame can be expressed as:

$$\mathbf{V_a^i} = V_a \begin{pmatrix} \cos\psi\cos\gamma_a \\ \sin\psi\cos\gamma_a \\ -\sin\gamma_a \end{pmatrix}$$

where $V_a = \|\mathbf{V_a}\|$.

With these in mind, and making use of Equation (2.4), the wind triangle can be expressed in inertial coordinates as:

$$V_g \begin{pmatrix} \cos\chi\cos\gamma \\ \sin\chi\cos\gamma \\ -\sin\gamma \end{pmatrix} - \begin{pmatrix} w_n \\ w_e \\ w_d \end{pmatrix} = V_a \begin{pmatrix} \cos\psi\cos\gamma_a \\ \sin\psi\cos\gamma_a \\ -\sin\gamma_a \end{pmatrix} \tag{2.8}$$

This expression becomes very useful when performing state estimation using GPS measurements.

By observation, we can also state our course angle as:

$$\chi = \tan^{-1}\left(\frac{V_e}{V_n}\right) \tag{2.9}$$

2.2 Kinematics and Dynamics

Having previously established our various co-ordinate frames and definitions of the Euler angles we will be making use of for the remainder of the project, we are now in a position where we can start to develop models for the dynamics of the system. The purpose of this section is to derive the non-linear equations of motion for the aircraft, which we can then use as a platform for analysis and eventually to design guidance, navigation and control algorithms. Deriving non-linear equations of motion is something that is done in various fields and similar discussions can be found in textbooks on mechanics

[22–24], space dynamics [25, 26], flight dynamics [16–18, 27–29], and robotics [30, 31]. This section primarily adapts the work presented in [21] as this is the most relevant to our application.

In Section 2.2.1 we will establish various state variables that will be used to describe the current state of our system at any particular moment in time. After this we will focus on defining the relationships between positions and velocities (the kinematics) in Section 2.2.2, and then on the relationships between forces, moments and the momentum of the system (the dynamics) in Section 2.2.3. In Section 2.2.4 we will then make use of all of these elements to create a full 6DoF dynamic model and discuss how this is implemented in the simulation environment. With this complete we can then start work on establishing the forces and moments required to complete the analysis, which will be done in Section 2.3.

2.2.1 State Variables

Twelve state variables will be introduced in order to develop the equations of motion for the aircraft. There are three position states and three velocity states associated with the translational motion of the aircraft. There are also three angular positions and three angular velocity states associated with the rotational motion. Each of these variables are listed in Table 2.1.

Table 2.1: System State Variables

Variable	Description
p_n	Inertial North position of the system in the $\mathbf{i^i}$ direction in $\mathcal{F}^i$
p_e	Inertial East position of the system in the $\mathbf{j^i}$ direction in $\mathcal{F}^i$
p_d	Inertial Down position of the system in the $\mathbf{k^i}$ direction in $\mathcal{F}^i$
u	Body frame velocity along $\mathbf{i^b}$ in $\mathcal{F}^b$
v	Body frame velocity along $\mathbf{j^b}$ in $\mathcal{F}^b$
w	Body frame velocity along $\mathbf{k^b}$ in $\mathcal{F}^b$
ϕ	Roll angle defined with respect to $\mathcal{F}^{v_2}$
θ	Pitch angle defined with respect to $\mathcal{F}^{v_1}$
	Yaw (heading) angle defined with respect to $\mathcal{F}^v$
p	Roll rate measured around $\mathbf{i^b}$ in $\mathcal{F}^b$
q	Pitch rate measured around $\mathbf{j^b}$ in $\mathcal{F}^b$
r	Yaw rate measured around $\mathbf{j^b}$ in $\mathcal{F}^b$

The state variables are shown diagrammatically in Figure 2.10, where the NED positions of the aircraft, (p_n, p_e, p_d), are defined relative to the inertial frame, $\mathcal{F}^i$. It should be

noted that sometimes we might use $h = -p_d$ when discussing the altitude of the aircraft. The linear velocities, (u, v, w), and the angular velocities, (p, q, r) are defined relative to the body frame, $\mathcal{F}^b$. The Euler angles: roll, ϕ, pitch, θ, and yaw, ψ; are defined with respect to the vehicle-2 frame, $\mathcal{F}^{v_2}$, the vehicle-1 frame, $\mathcal{F}^{v_1}$ and the vehicle frame, $\mathcal{F}^v$, respectively.

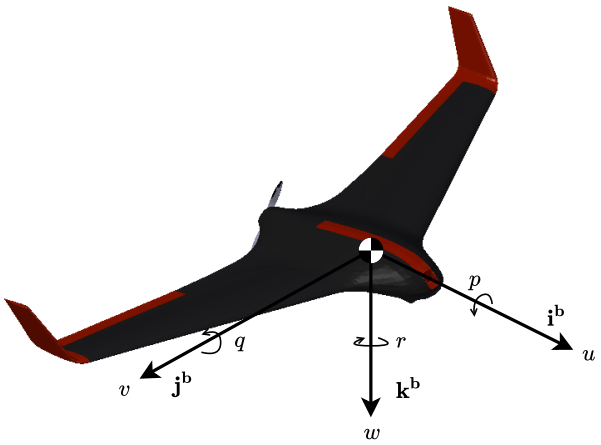


Figure 2.10: Definition of axes for the equations of motion

Because Euler angles are defined relative to intermediate frames of reference (ie. none of them are defined in the body frame), we cannot say that the angular rates, (p, q, r), are simply the temporal derivatives of the attitude angles (ϕ, θ, ψ). It is only the case that $p = \dot{\phi}$, $q = \dot{\theta}$ and $r = \dot{\psi}$ when $\phi = \theta = 0$. This will be shown in Section 2.2.2. In general, the angular rates, (p, q, r), are functions of the Euler angle temporal derivatives as well as the pitch and roll angles. In other words:

$$(p, q, r) = f(\dot{\phi}, \dot{\theta}, \dot{\psi}, \phi, \theta)$$

2.2.2 Kinematics

The translational velocity of the system is commonly expressed in terms of the velocity components along each of the axes in the body frame, $\mathcal{F}^b$. The components u, v, and w correspond to the inertial velocity of the vehicle projected onto the $\mathbf{i^b}$, $\mathbf{j^b}$, and $\mathbf{k^b}$ axes, respectively. On the other hand, the translational position of the system is usually measured and expressed in the inertial frame, $\mathcal{F}^i$. Relating the translational velocity and position can be done as follows:

$$\frac{d}{dt}\begin{pmatrix} p_n \\ p_e \\ p_d \end{pmatrix} = \mathcal{R}_b^v \begin{pmatrix} u \\ v \\ w \end{pmatrix} = (\mathcal{R}_v^b)^T \begin{pmatrix} u \\ v \\ w \end{pmatrix}$$

Which, using Equation (2.1), gives:

$$\begin{pmatrix} \dot{p_n} \\ \dot{p_e} \\ \dot{p_d} \end{pmatrix} = \begin{pmatrix} c_\theta c_\psi & s_\phi s_\theta c_\psi - c_\phi s_\psi & c_\phi s_\theta c_\psi + s_\phi s_\psi \\ c_\theta s_\psi & s_\phi s_\theta s_\psi + c_\phi c_\psi & c_\phi s_\theta s_\psi - s_\phi c_\psi \\ -s_\theta & s_\phi c_\theta & c_\phi c_\theta \end{pmatrix} \begin{pmatrix} u \\ v \\ w \end{pmatrix} \tag{2.10}$$

Which uses the previously defined shorthand notation of $c_x \triangleq \cos x$ and $s_x \triangleq \sin x$. This is a kinematic relationship in that it relates the derivative of position to velocity, while forces and accelerations are not considered.

As discussed in the previous section, the relationship between our angular positions, ϕ, θ, and ψ, and the angular rates p, q, and r is complicated by the fact that these quantities are defined in different coordinate frames. The angular rates are defined in the body frame, $\mathcal{F}^b$. The angular positions (Euler angles) are defined in 3 different coordinate frames: the roll angle ϕ is a rotation from $\mathcal{F}^{v_2}$ to $\mathcal{F}^b$ about the $\mathbf{i^{v_2}} = \mathbf{i^b}$ axis; the pitch angle, θ, is a rotation from $\mathcal{F}^{v_1}$ to $\mathcal{F}^{v_2}$ about the $\mathbf{j^{v_1}} = \mathbf{j^{v_2}}$ axis; and the yaw angle, ψ, is a rotation from $\mathcal{F}^v$ to $\mathcal{F}^{v_1}$ about the $\mathbf{k^v} = \mathbf{k^{v_1}}$ axis.

The body-frame angular rates can be expressed in terms of the derivatives of the Euler angles, provided that one applies the correct rotational transformations as follows:

$$\begin{aligned} \begin{pmatrix} p \\ q \\ r \end{pmatrix} &= \begin{pmatrix} \dot{\phi} \\ 0 \\ 0 \end{pmatrix} + \mathcal{R}^b_{v_2}(\phi) \begin{pmatrix} 0 \\ \dot{\theta} \\ 0 \end{pmatrix} + \mathcal{R}^b_{v_2}(\phi)\mathcal{R}^{v_2}_{v_1}(\theta) \begin{pmatrix} 0 \\ 0 \\ \dot{\psi} \end{pmatrix} \\ &= \begin{pmatrix} \dot{\phi} \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{pmatrix} \begin{pmatrix} 0 \\ \dot{\theta} \\ 0 \end{pmatrix} \\ &\quad + \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{pmatrix} \begin{pmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ \dot{\psi} \end{pmatrix} \\ &= \begin{pmatrix} 1 & 0 & -\sin\theta \\ 0 & \cos\phi & \sin\phi\cos\theta \\ 0 & -\sin\phi & \cos\phi\cos\theta \end{pmatrix} \begin{pmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{pmatrix} \end{aligned} \tag{2.11}$$

Inverting this relationship results in:

$$\begin{pmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{pmatrix} = \begin{pmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi\sec\theta & \cos\phi\sec\theta \end{pmatrix} \begin{pmatrix} p \\ q \\ r \end{pmatrix} \tag{2.12}$$

Which describes the temporal derivatives of our Euler angles in terms of the roll angle, ϕ, the pitch angle, θ, and the body rates p, q, and r, as stated.

2.2.3 Rigid Body Dynamics

To derive the dynamic equations of motion for the system, Newtonian physics will be applied. This will first be done to the translational degrees of freedom and then to the rotational degrees of freedom. However, Newton's laws only hold in inertial reference frames - meaning that the motion of the body of interest must be referenced to a fixed (ie. inertial) frame of reference, which in our case is the ground. As mentioned previously, we assume a flat earth model, which is appropriate for the size of the system in question.

Even though motion is referenced to a fixed frame, it can be expressed using vector components associated with other frames - such as the body frame. We do this with the (inertially referenced) ground speed velocity vector, $\mathbf{V_g}$, which for convenience is most commonly expressed in the body frame as $\mathbf{V_g^b} = (u, v, w)^T$. $\mathbf{V_g^b}$ is the inertially referenced ground speed velocity vector as expressed in the body frame.

Translational Motion

Newton's second law applied to a body undergoing translational motion can be stated as:

$$\mathsf{m}\frac{d^{\mathcal{F}^i}}{dt}\mathbf{V_g} = \mathbf{F} \tag{2.13}$$

where m is the mass of the system[1], $\frac{d^{\mathcal{F}^i}}{dt}$ is the temporal derivative in the inertial frame, and $\mathbf{F}$ is the sum of all external forces acting on the aircraft. These external forces include gravitational, aerodynamic and propulsive forces.

The derivative of our velocity taken in the inertial frame can be written in terms of the derivative in the body frame and the angular velocity as:

$$\frac{d^{\mathcal{F}^i}}{dt}\mathbf{V_g} = \frac{d^{\mathcal{F}^b}}{dt}\mathbf{V_g} + \boldsymbol{\omega_{b/\mathcal{F}^i}} \times \mathbf{V_g} \tag{2.14}$$

where $\boldsymbol{\omega_{b/\mathcal{F}^i}}$ is the angular velocity of the system with respect to the inertial frame.

Combining Equation (2.13) and Equation (2.14) results in an alternative representation of Newton's second law with differentiation carried out in the body frame:

$$\mathsf{m}\left(\frac{d^{\mathcal{F}^b}}{dt}\mathbf{V_g} + \boldsymbol{\omega_{b/\mathcal{F}^i}} \times \mathbf{V_g}\right) = \mathbf{F}$$

[1]Mass is denoted as m, as opposed to m, in order to differentiate it to the pitching moment which will be established as m in a subsequent chapter

As mentioned previously, one can express vectors using components associated with other frames. For an aircraft, we can most easily apply Newton's second law by expressing forces and velocities in the body frame. As such:

$$\mathsf{m}\left(\frac{d^{\mathcal{F}^b}}{dt}\mathbf{V}_{\mathbf{g}}^{\mathbf{b}} + \boldsymbol{\omega}_{\boldsymbol{b/\mathcal{F}^i}}^{\boldsymbol{b}} \times \mathbf{V}_{\mathbf{g}}^{\mathbf{b}}\right) = \mathbf{F}^{\mathbf{b}} \tag{2.15}$$

where $\mathbf{V}_{\mathbf{g}}^{\mathbf{b}} = (u, v, w)^T$ and $\boldsymbol{\omega}_{\boldsymbol{b/\mathcal{F}^i}}^{\boldsymbol{b}} = (p, q, r)^T$. The vector $\mathbf{F}^{\mathbf{b}}$ represents the sum of the externally applied forces and is defined in terms of its body-frame components as $\mathbf{F}^{\mathbf{b}} \triangleq (f_x, f_y, f_z)^T$.

The expression $\frac{d^{\mathcal{F}^b}}{dt}\mathbf{V}_{\mathbf{g}}^{\mathbf{b}}$ is the rate of change of the velocity expressed in the body frame, as viewed by an observer on the moving body. Since u, v, and w are instantaneous projects of $\mathbf{V}_{\mathbf{g}}^{\mathbf{b}}$ onto the $\mathbf{i}^{\mathbf{b}}$, $\mathbf{j}^{\mathbf{b}}$, and $\mathbf{k}^{\mathbf{b}}$ axes, it follows that:

$$\frac{d^{\mathcal{F}^b}}{dt}\mathbf{V}_{\mathbf{g}}^{\mathbf{b}} = \begin{pmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{pmatrix}$$

Finally, expanding the cross product in Equation (2.15) and doing some algebraic rearrangement, we get:

$$\begin{pmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{pmatrix} = \begin{pmatrix} rv - qw \\ pw - ru \\ qu - pv \end{pmatrix} + \frac{1}{\mathsf{m}} \begin{pmatrix} f_x \\ f_y \\ f_z \end{pmatrix} \tag{2.16}$$

Rotational Motion

For rotational motion, Newton's second law can be stated as:

$$\frac{d^{\mathcal{F}^i}}{dt}\mathbf{h} = \mathbf{M}$$

Where $\mathbf{h}$ is the angular momentum in vector form and $\mathbf{M}$ is the sum of all externally applied moments. This expression is true provided that moments are summed about the center of mass of the system.

The derivative of angular momentum taken in the inertial frame can be expanded as:

$$\frac{d^{\mathcal{F}^i}}{dt}\mathbf{h} = \frac{d^{\mathcal{F}^b}}{dt}\mathbf{h} + \boldsymbol{\omega}_{\boldsymbol{b/\mathcal{F}^i}} \times \mathbf{h}$$

As with translational motion, it is most convenient to express this equation in the body

frame, giving:

$$\frac{d^{\mathcal{F}^b}}{dt}\mathbf{h^b} + \boldsymbol{\omega}^{\boldsymbol{b}}_{\boldsymbol{b/\mathcal{F}^i}} \times \mathbf{h^b} = \mathbf{M^b} \tag{2.17}$$

For a rigid body, angular momentum is defined as the product of the inertia matrix (or tensor), $\mathbf{J}$, and the angular velocity vector. In other words:

$$\mathbf{h^b} \triangleq \mathbf{J}\boldsymbol{\omega}^{\boldsymbol{b}}_{\boldsymbol{b/\mathcal{F}^i}}$$

Where:

$$\begin{aligned}\mathbf{J} &= \begin{pmatrix} \int (y^2+z^2)d\mathsf{m} & -\int xyd\mathsf{m} & -\int xzd\mathsf{m} \\ -\int xyd\mathsf{m} & \int (x^2+z^2)d\mathsf{m} & -\int yzd\mathsf{m} \\ -\int xzd\mathsf{m} & -\int yzd\mathsf{m} & \int (x^2+y^2)d\mathsf{m} \end{pmatrix} \\ &\triangleq \begin{pmatrix} J_x & -J_{xy} & -J_{xz} \\ -J_{xy} & J_y & -J_{yz} \\ -J_{xz} & -J_{yz} & J_z \end{pmatrix}\end{aligned} \tag{2.18}$$

The diagonal terms of $\mathbf{J}$ are called the moments of inertia, whereas the off-diagonal terms are called the products of inertia. The moments of inertia are measures of the aircraft's tendency to oppose acceleration about a specific axis of rotation. For example, the larger J_x is in value, the more the aircraft opposes angular acceleration about the x axis. This applies to the moments of inertia J_y and J_z, too.

Because the integrals in Equation (2.18) are calculated with respect to the $\mathbf{i^b}$, $\mathbf{j^b}$, and $\mathbf{k^b}$ axes fixed on the rigid body, $\mathbf{J}$ is constant when viewed from the body frame - hence, $\frac{d^{\mathcal{F}^b}}{dt}\mathbf{J} = 0$. Using this fact and substituting into Equation (2.17), we get:

$$\mathbf{J}\frac{d^{\mathcal{F}^b}}{dt}\boldsymbol{\omega}^{\boldsymbol{b}}_{\boldsymbol{b/\mathcal{F}^i}} + \boldsymbol{\omega}^{\boldsymbol{b}}_{\boldsymbol{b/\mathcal{F}^i}} \times \mathbf{J}\boldsymbol{\omega}^{\boldsymbol{b}}_{\boldsymbol{b/\mathcal{F}^i}} = \mathbf{M^b} \tag{2.19}$$

The expression $\frac{d^{\mathcal{F}^b}}{dt}\boldsymbol{\omega}^{\boldsymbol{b}}_{\boldsymbol{b/\mathcal{F}^i}}$ is the rate of change of the angular velocity expressed in the body frame, as viewed by an observer on the moving body. Since p, q, and r are the instantaneous projections of $\boldsymbol{\omega}^{\boldsymbol{b}}_{\boldsymbol{b/\mathcal{F}^i}}$ onto the $\mathbf{i^b}$, $\mathbf{j^b}$, and $\mathbf{k^b}$ axes, it follows that:

$$\dot{\boldsymbol{\omega}}^{\boldsymbol{b}}_{\boldsymbol{b/\mathcal{F}^i}} = \frac{d^{\mathcal{F}^b}}{dt}\boldsymbol{\omega}^{\boldsymbol{b}}_{\boldsymbol{b/\mathcal{F}^i}} = \begin{pmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{pmatrix}$$

Rearranging Equation (2.19), we get:

$$\dot{\boldsymbol{\omega}}^{\boldsymbol{b}}_{\boldsymbol{b/\mathcal{F}^i}} = \mathbf{J}^{-1}\left[-\boldsymbol{\omega}^{\boldsymbol{b}}_{\boldsymbol{b/\mathcal{F}^i}} \times \left(\mathbf{J}\boldsymbol{\omega}^{\boldsymbol{b}}_{\boldsymbol{b/\mathcal{F}^i}}\right) + \mathbf{M^b}\right] \tag{2.20}$$

Given that aircraft are often symmetric about the plane spanned by $\mathbf{i^b}$, and $\mathbf{k^b}$, we can assume that $J_{xy} = J_{yz} = 0$, which implies that:

$$\mathbf{J} = \begin{pmatrix} J_x & 0 & -J_{xz} \\ 0 & J_y & 0 \\ -J_{xz} & 0 & J_z \end{pmatrix}$$

Under this symmetry assumption, the inverse of $\mathbf{J}$ is given by:

$$\begin{aligned} \mathbf{J^{-1}} &= \frac{adj(\mathbf{J})}{det(\mathbf{J})} \\ &= \frac{\begin{pmatrix} J_yJ_z & 0 & J_yJ_{xz} \\ 0 & J_xJ_z - J_{xz}^2 & 0 \\ J_{xz}J_y & 0 & J_xJ_y \end{pmatrix}}{J_xJ_yJ_z - J_{xz}^2J_y} \\ &= \begin{pmatrix} \frac{J_z}{\Gamma} & 0 & \frac{J_{xz}}{\Gamma} \\ 0 & \frac{1}{J_y} & 0 \\ \frac{J_{xz}}{\Gamma} & 0 & \frac{J_x}{\Gamma} \end{pmatrix} \end{aligned}$$

where $\Gamma \triangleq J_xJ_z - J_{xz}^2$.

Defining the components of the externally applied moments about the $\mathbf{i^b}$, $\mathbf{j^b}$, and $\mathbf{k^b}$ axes as $\mathbf{M^b} \triangleq (l, m, n)^T$, we can write Equation (2.20) in component form as:

$$\begin{aligned} \begin{pmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{pmatrix} &= \begin{pmatrix} \frac{J_z}{\Gamma} & 0 & \frac{J_{xz}}{\Gamma} \\ 0 & \frac{1}{J_y} & 0 \\ \frac{J_{xz}}{\Gamma} & 0 & \frac{J_x}{\Gamma} \end{pmatrix} \left[\begin{pmatrix} 0 & r & -q \\ -r & 0 & p \\ q & -p & 0 \end{pmatrix} \begin{pmatrix} J_x & 0 & -J_{xz} \\ 0 & J_y & 0 \\ -J_{xz} & 0 & J_z \end{pmatrix} \begin{pmatrix} p \\ q \\ r \end{pmatrix} + \begin{pmatrix} l \\ m \\ n \end{pmatrix} \right] \\ &= \begin{pmatrix} \Gamma_1 pq - \Gamma_2 qr + \Gamma_3 l + \Gamma_4 n \\ \Gamma_5 pr - \Gamma_6(p^2 - r^2) + \frac{1}{J_y}m \\ \Gamma_7 pq - \Gamma_1 qr + \Gamma_4 l + \Gamma_8 n \end{pmatrix} \end{aligned} \tag{2.21}$$

where:

$$
\begin{aligned}
\Gamma_1 &= \frac{J_{xz}(J_x - J_y + J_z)}{\Gamma} \\
\Gamma_2 &= \frac{J)z(J_z - J_y) + J_{xz}^2}{\Gamma} \\
\Gamma_3 &= \frac{J_z}{\Gamma} \\
\Gamma_4 &= \frac{J_{xz}}{\Gamma} \\
\Gamma_5 &= \frac{J_z - J_x}{J_y} \\
\Gamma_6 &= \frac{J_{xz}}{J_y} \\
\Gamma_7 &= \frac{(J_x - J_y)J_x + J_{xz}^2}{\Gamma} \\
\Gamma_8 &= \frac{J_x}{\Gamma}
\end{aligned}
\tag{2.22}
$$

2.2.4 Implementation of 6DoF Model

Making use of Equations (2.10), (2.12), (2.16), and (2.21), we can define the twelve-state model for the system equations of motion as:

$$
\begin{pmatrix} \dot{p_n} \\ \dot{p_e} \\ \dot{p_d} \end{pmatrix} = \begin{pmatrix} c_\theta c_\psi & s_\phi s_\theta c_\psi - c_\phi s_\psi & c_\phi s_\theta c_\psi + s_\phi s_\psi \\ c_\theta s_\psi & s_\phi s_\theta s_\psi + c_\phi c_\psi & c_\phi s_\theta s_\psi - s_\phi c_\psi \\ -s_\theta & s_\phi c_\theta & c_\phi c_\theta \end{pmatrix} \begin{pmatrix} u \\ v \\ w \end{pmatrix} \tag{2.23}
$$

$$
\begin{pmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{pmatrix} = \begin{pmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi\sec\theta & \cos\phi\sec\theta \end{pmatrix} \begin{pmatrix} p \\ q \\ r \end{pmatrix} \tag{2.24}
$$

$$
\begin{pmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{pmatrix} = \begin{pmatrix} rv - qw \\ pw - ru \\ qu - pv \end{pmatrix} + \frac{1}{\mathsf{m}} \begin{pmatrix} f_x \\ f_y \\ f_z \end{pmatrix} \tag{2.25}
$$

$$
\begin{pmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{pmatrix} = \begin{pmatrix} \Gamma_1 pq - \Gamma_2 qr + \Gamma_3 l + \Gamma_4 n \\ \Gamma_5 pr - \Gamma_6(p^2 - r^2) + \frac{1}{J_y} m \\ \Gamma_7 pq - \Gamma_1 qr + \Gamma_4 l + \Gamma_8 n \end{pmatrix} \tag{2.26}
$$

These equations are not complete in that the externally applied forces and moments are not yet defined. These will be established in Section 2.3.

With this set of differential equations established, we can make use of a Level-2 Matlab S-Function in the Simulink environment to model the system. Discussing the detailed operation of this is beyond the scope of this text, and the reader is directed to [32] for

more insight into the operation of S-Functions. In brief, however, this gives us the ability to model differential equations in the Simulink environment and we make use of this to generate a custom Simulink block which takes in our forces and moments as an input, as well as initial conditions and system parameters, and then maps this to a set of outputs which are required for other aspects of the simulation. This is depicted in Figure 2.11, which graphically illustrates all of the required inputs and generated outputs from this custom block. The implementation code for this can be found as ms_sixdof.m as part of the code repository provided in Section 1.4.

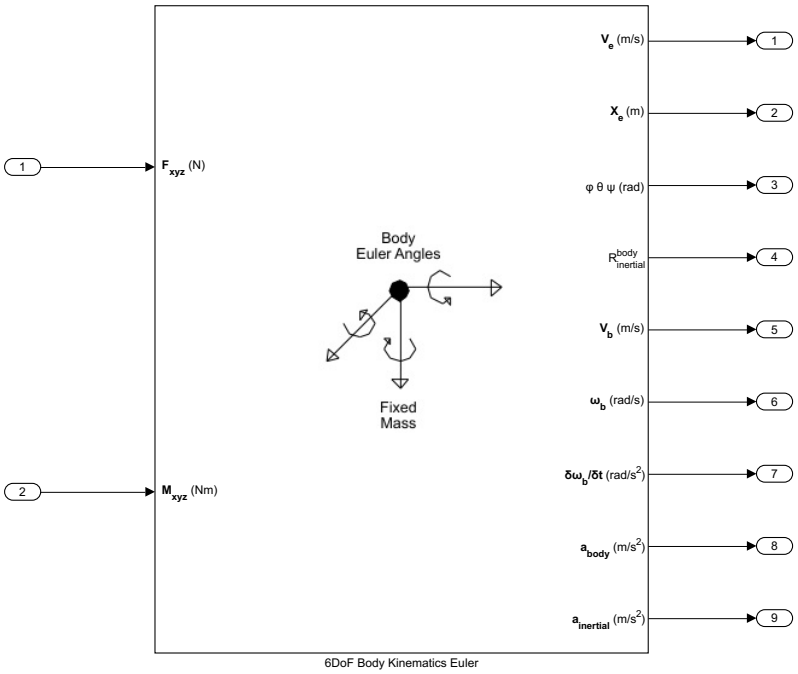


Figure 2.11: Simulink implementation of 6DoF body kinematics Euler angles

System testing and characterisation is beyond the scope of this dissertation, with previous work done on this aircraft serving as a basis for determining the required system parameters. In order to calculate our moments of inertia and inertia products, a simple pendulum method can be used; with mass being known by simply weighing the system on a scale. These methods were made use of in [3] with the results depicted in Table 2.2.

These results are made use of in the Simulink block depicted in Figure 2.11 and already allow us to start gaining some insight into the aircraft as they immediately tell us that the system will require a larger moment to accelerate at the same rate in roll than it would in yaw, which itself requires a larger moment than in pitch. This is a useful observation and one which can be made use of when performing controller design as we immediately expect that our pitch dynamics will be substantially faster than those in yaw or roll.

Table 2.2: Physical System Parameters for the Skywalker X8 Aircraft [3]

System Parameter	Value	Unit
m	3.364	kg
J_x	1.2290	kg.m^2
J_y	0.1702	kg.m^2
J_z	0.8808	kg.m^2
J_{xz}	0.9343	kg.m^2

2.3 Forces and Moments

The purpose of this section is to describe the forces and moments that act on our system. Following [28], we will assume that the forces and moments are primarily due to three sources: gravity, aerodynamics, and propulsion. Letting $\mathbf{F_g}$ be the gravitational force, ($\mathbf{F_a}$, $\mathbf{M_a}$) be the aerodynamic forces and moments, and ($\mathbf{F_p}$, $\mathbf{M_p}$) be the propulsive forces and moments, we have:

$$\begin{aligned}\mathbf{F} &= \mathbf{F_g} + \mathbf{F_a} + \mathbf{F_p} \\ \mathbf{M} &= \mathbf{M_a} + \mathbf{M_p}\end{aligned}$$

where $\mathbf{F}$ is the total force acting on the aircraft and $\mathbf{M}$ is the total moment acting on the aircraft.

In this section we will derive expressions for each of the forces and moments described. It should be noted that the majority of the work presented can be found in most textbooks on flight dynamics, including [16–18, 27–29, 33]. The discussions on lift, drag, and moment coefficients primarily follow [29], as does the decomposing of the wind vector into a constant and a random term.

2.3.1 Gravitational Forces

The effect of the Earth's gravitational field on the system is simply the weight of the aircraft, which we know acts in the $\mathbf{k^i}$ direction, has a magnitude proportional to the system's mass, and acts through the center of mass. In the vehicle frame, $\mathcal{F}^v$, the gravity force acting through the center of mass is given by:

$$\mathbf{F_g^v} = \begin{pmatrix} 0 \\ 0 \\ \mathsf{m}g \end{pmatrix}$$

However, in order to make use of this force in the equations established in Equation (2.25)

we need to align it with the axes in the body frame. As such, we establish:

$$\begin{aligned}\mathbf{F_g^b} &= \mathcal{R}_v^b \begin{pmatrix} 0 \\ 0 \\ \mathsf{m}g \end{pmatrix} \\ &= \begin{pmatrix} -\mathsf{m}g\sin\theta \\ \mathsf{m}g\cos\theta\sin\phi \\ \mathsf{m}g\cos\theta\cos\phi \end{pmatrix} \end{aligned} \tag{2.27}$$

Since gravity acts through the center of mass, there are no moments generated.

2.3.2 Aerodynamic Forces and Moments

As an aircraft moves through the air, a pressure distribution is generated around the body. The strength and distribution of the pressure acting on the system is a function of the airspeed, the air density, and the shape and attitude of the system. Accordingly, the dynamic pressure is given by $\frac{1}{2}\rho V_a^2$, where ρ is the air density and V_a is the speed of the system through the surrounding air mass.

Instead of attempting to characterize the pressure distribution around the wing, a more common approach is to capture the effect of the pressure with a combination of forces and moments. For airfoils, generally the lift, drag, and pitching moment coefficients are significantly influenced by the airfoil shape, Reynolds Number, Mach Number, and the angle of attack. For the airspeeds under consideration, we can approximate the Reynolds number and Mach Number effects as approximately being constant. So we will only consider the effects of the angles, α and β; the angular rates, p, q, and r; and the deflection of the control surfaces on the aerodynamic coefficients.

It is common to decompose the aerodynamic forces and moments into two groups: longitudinal and lateral. The longitudinal forces and moments act in the $\mathbf{i^b}$-$\mathbf{k^b}$ plane, also called the pitch plane. They include the forces in the $\mathbf{i^b}$ and $\mathbf{k^b}$ directions (caused by lift and drag) and the moment about the $\mathbf{j^b}$ axis. The lateral forces and moments include the force in the $\mathbf{j^b}$ direction and the moments about the $\mathbf{i^b}$ and $\mathbf{k^b}$ axes.

Control Surfaces

In order to fully define the variables used to define the aerodynamic coefficients, we need to define the control surface parameters. For a standard aircraft configuration, the control surfaces include the elevator, the aileron and the rudder.

Figure 2.12 shows the control surfaces for a standard configuration aircraft, where the aileron deflection is denoted by δ_a, the elevator deflection is denoted by δ_e, and the rudder deflection is denoted by δ_r. The positive direction of a control surface can be determined by applying the right-hand-rule to the hinge axis of the control surface. The

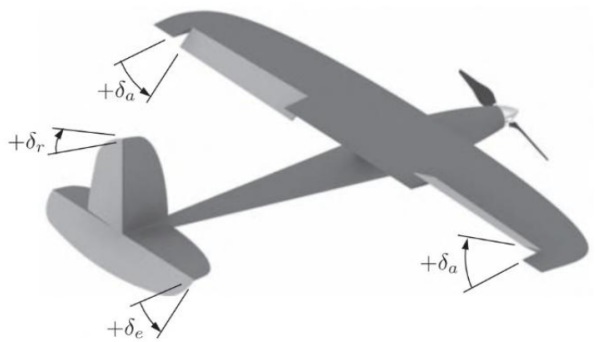


Figure 2.12: Standard aircraft configuration with control surfaces [21]

impact to most aerodynamic coefficients is given as a function of these three standard parameters, even if the system configuration is not standard.

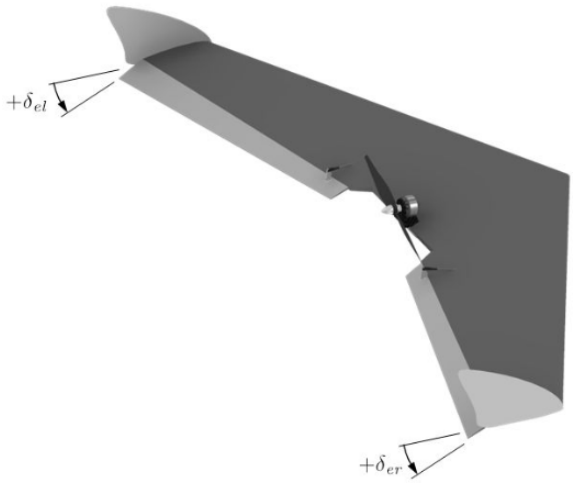


Figure 2.13: Flying wing aircraft configuration with control surfaces [21]

The Skywalker X8 is what is known as a flying wing, or a blended wing body aircraft, denoted in Figure 2.13 - so it should be clear that δ_r is not available and will have no impact on the aerodynamic coefficients. Indeed, there are no specific ailerons or elevators either - but their functionality can be accomplished by the control surfaces for the flying wing, known as elevons, by applying a mixing matrix.

The angular deflection of the left and right elevons are denoted δ_{el} and δ_{er}, respectively. Driving the elevons differentially has the same effect as ailerons producing a moment about $\mathbf{i^b}$, while driving the elevons together has the same effect as an elevator, producing a moment about $\mathbf{j^b}$. As such, we can convert between elevon and aileron-elevator signals

with:

$$\begin{pmatrix} \delta_e \\ \delta_a \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ -1 & 1 \end{pmatrix} \begin{pmatrix} \delta_{er} \\ \delta_{el} \end{pmatrix} \tag{2.28}$$

Longitudinal Aerodynamics

The longitudinal aerodynamic forces and moments cause motion in the body frame, $\mathcal{F}^b$, in the $\mathbf{i^b}$-$\mathbf{k^b}$ plane, also known as the pitch plane. They are comprised of the lift, drag and pitching moment. By definition, the lift and drag forces are aligned with the axes of the stability frame (ie. $\mathbf{k^s}$ and $\mathbf{i^s}$, in $\mathcal{F}^s$, respectively). When represented as a vector, the pitching moment also aligns with the $\mathbf{j^s}$ axis of the stability frame.

The lift and drag forces, as well as the pitching moment, are heavily influenced by the angle of attack, α. The pitch rate, q, and the elevator deflection, δ_e also influence the longitudinal forces and moment. Based on this we can write the equations for lift, drag and pitching moment as:

$$\begin{aligned} F_{lift} &= \frac{1}{2}\rho V_a^2 S C_L(\alpha, q, \delta_e) \\ F_{drag} &= \frac{1}{2}\rho V_a^2 S C_D(\alpha, q, \delta_e) \\ m &= \frac{1}{2}\rho V_a^2 S c C_m(\alpha, q, \delta_e) \end{aligned}$$

Where C_L, C_D, and C_m are non-dimensional aerodynamic coefficients, while S is the planform area, and c is the mean chord length of the aircraft wing.

In general, the coefficients are non-linear. However, for small angles of attack, the flow over the wing will remain laminar and attached. Under these conditions the lift, drag, and pitching moment can be modelled with acceptable accuracy using linear approximations. Using the lift equation as an example, a first-order Taylor series approximation of the lift coefficient can be written as:

$$C_L = C_{L_0} + \frac{\delta C_L}{\delta \alpha}\alpha + \frac{\delta C_L}{\delta q}\frac{c}{2V_a}q + \frac{\delta C_L}{\delta \delta_e}\delta_e \tag{2.29}$$

Where the coefficient C_{L_0} is the value of C_L when $\alpha = q = \delta_e = 0$. It is common to non-dimensionalise the partial derivatives of this linear approximation. Since C_L and the angles α and δ_e (in radians) are already dimensionless, the only partial derivative requiring nondimensionalisation is $\delta C_L/\delta q$. Since the units of q are rad/s, a standard

factor to use is $c/(2V_a)$. We can then rewrite this equation as:

$$C_L = C_{L_0} + C_{L_\alpha}\alpha + C_{L_q}\frac{c}{2V_a}q + C_{L_{\delta_e}}\delta_e \tag{2.30}$$

where the coefficients C_{L_0}, $C_{L_\alpha} \triangleq \frac{\delta C_L}{\delta \alpha}$, $C_{L_q} \triangleq \frac{\delta C_L}{\delta q}$, and $C_{L_{\delta_e}} \triangleq \frac{\delta C_L}{\delta \delta_e}$ are dimensionless quantities. C_{L_α} and C_{L_q} are commonly referred to as stability derivatives, while $C_{L_{\delta_e}}$ is an example of a control derivative. Using a similar approach, we express linear approximations of the drag coefficient and pitching moment coefficient as:

$$C_D = C_{D_0} + C_{D_\alpha}\alpha + C_{D_q}\frac{c}{2V_a}q + C_{D_{\delta_e}}\delta_e \tag{2.31}$$

$$C_m = C_{m_0} + C_{m_\alpha}\alpha + C_{m_q}\frac{c}{2V_a}q + C_{m_{\delta_e}}\delta_e \tag{2.32}$$

As mentioned previously, these linearised versions of our aerodynamic coefficients are only true under typical, low-angle-of-attack flight conditions. However, when our angle of attack deviates too far from zero, we get a strong deviation from these assumptions as we encounter separation and turbulent flow as the boundary layer separates from the surface. As such, Equation (2.30) and Equation (2.31) will be extended with a stall model in a subsequent section.

From Equation (2.25) we know we need to align our forces, currently in the stability frame, into the body frame. As such, we define aerodynamic coefficients in X and Z, or along $\mathbf{i^b}$ and $\mathbf{k^b}$, respectively, as:

$$\begin{aligned}\begin{pmatrix} C_X \\ C_Z \end{pmatrix} &= \mathcal{R}_s^b \begin{pmatrix} -C_D(\alpha, q, \delta_e) \\ -C_L(\alpha, q, \delta_e) \end{pmatrix} \\ &= \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix} \begin{pmatrix} -C_D(\alpha, q, \delta_e) \\ -C_L(\alpha, q, \delta_e) \end{pmatrix}\end{aligned} \tag{2.33}$$

From this, we can state our longitudinal aerodynamic forces and moments in a more convenient form as:

$$\begin{aligned} f_x &= \frac{1}{2}\rho V_a^2 S C_X(\alpha, q, \delta_e) \\ f_z &= \frac{1}{2}\rho V_a^2 S C_Z(\alpha, q, \delta_e) \\ m &= \frac{1}{2}\rho V_a^2 S c C_m(\alpha, q, \delta_e) \end{aligned} \tag{2.34}$$

Lateral Aerodynamics

The lateral aerodynamic force and moments cause translational motion in the lateral direction along the $\mathbf{j^b}$ axis as well as rotational motion in roll and yaw, about the $\mathbf{i^b}$ and $\mathbf{k^b}$ axes, respectively, that will result in directional changes in the flight path of the aircraft. The lateral aerodynamics are most significantly influenced by the sideslip angle, β. They are also influenced by the roll rate, p, the yaw rate, r, the deflection of the ailerons, δ_a, and the deflection of the rudder, δ_r. However, given that we are working with a flying wing, the influence from the rudder will be omitted.

Denoting the lateral force as f_y, and the roll and yaw moments as l and n, respectively, we have:

$$\begin{aligned} f_y &= \frac{1}{2}\rho V_a^2 S C_Y(\beta, p, r, \delta_a) \\ l &= \frac{1}{2}\rho V_a^2 S b C_l(\beta, p, r, \delta_a) \\ n &= \frac{1}{2}\rho V_a^2 S b C_n(\beta, p, r, \delta_a) \end{aligned} \tag{2.35}$$

where C_Y, C_l, and C_n are non-dimensional aerodynamic coefficients, and b is the wingspan of the aircraft.

As with the longitudinal aerodynamic forces and moments, the coefficients C_Y, C_l, and C_n are non-linear in their constitutive parameters. We make use of a similar process as defined in the longitudinal aerodynamics to define a set of linearised aerodynamic coefficients as follows:

$$C_Y = C_{Y_0} + C_{Y_\beta}\beta + C_{Y_p}\frac{b}{2V_a}p + C_{Y_r}\frac{b}{2V_a}r + C_{Y_{\delta_a}}\delta_a \tag{2.36}$$

$$C_l = C_{l_0} + C_{l_\beta}\beta + C_{l_p}\frac{b}{2V_a}p + C_{l_r}\frac{b}{2V_a}r + C_{l_{\delta_a}}\delta_a \tag{2.37}$$

$$C_n = C_{n_0} + C_{n_\beta}\beta + C_{n_p}\frac{b}{2V_a}p + C_{n_r}\frac{b}{2V_a}r + C_{n_{\delta_a}}\delta_a \tag{2.38}$$

The coefficient C_{Y_0} is the value of the lateral force coefficient, C_Y, when $\beta = p = r = \delta_a = 0$. For aircraft that are symmetrical about the $\mathbf{i^b}$-$\mathbf{k^b}$ plane, C_{Y_0} is typically zero. This is also true for C_{l_0} and C_{n_0}.

Given that these forces and moments are already aligned with the body axes of the aircraft, they do not require a rotational transformation to be implemented in the equations of motion.

With Equations (2.30), (2.31), (2.36), (2.37), (2.32) and (2.38) defined, we note that our coefficients are most easily determined by making use of wind tunnel testing or

computational fluid dynamic (CFD) software to analyse our airframe. From [1] we restate our coefficients as follows, where it should be noted that C_D has been extended to include higher order effects so as to more accurately fit the test data generated:

$$\begin{aligned}
\begin{pmatrix} C_D \\ C_Y \\ C_L \end{pmatrix} &= \begin{pmatrix} C_{D_0} + C_{D_{\alpha_1}}\alpha + C_{D_{\alpha_2}}\alpha^2 + C_{D_{\delta_{e2}}}\delta_e^2 + C_{D_q}\frac{c}{2V_a}q + C_{D_{\beta_2}}\beta^2 + C_{D_\beta}\beta \\ C_{Y_0} + C_{Y_\beta}\beta + C_{Y_p}\frac{b}{2V_a}p + C_{Y_r}\frac{b}{2V_a}r + C_{Y_{\delta_a}}\delta_a \\ C_{L_0} + C_{L_\alpha}\alpha + C_{L_q}\frac{c}{2V_a}q + C_{L_{\delta_e}}\delta_e \end{pmatrix} \\
\begin{pmatrix} C_l \\ C_m \\ C_n \end{pmatrix} &= \begin{pmatrix} C_{l_0} + C_{l_\beta}\beta + C_{l_p}\frac{b}{2V_a}p + C_{l_r}\frac{b}{2V_a}r + C_{l_{\delta_a}}\delta_a \\ C_{m_0} + C_{m_\alpha}\alpha + C_{m_q}\frac{c}{2V_a}q + C_{m_{\delta_e}}\delta_e \\ C_{n_0} + C_{n_\beta}\beta + C_{n_p}\frac{b}{2V_a}p + C_{n_r}\frac{b}{2V_a}r + C_{n_{\delta_a}}\delta_a \end{pmatrix}
\end{aligned} \tag{2.39}$$

The aerodynamic coefficients made use of in this text are taken from the literature, particularly from [1, 3]. The only values not used from [1] were those where the authors raised concerns about the validity of their results, or the results were not available from wind tunnel tests, in which case the values specified in [3] were made use of. In order to ensure that the lateral coefficients aligned with test data, [2] was also used. These coefficients are presented in Tables 2.3, 2.4, 2.5, 2.6, 2.7 and 2.8.

Table 2.3: C_D Aerodynamic Coefficients for the Skywalker X8 Aircraft

Aerodynamic Coefficient	**Value**
C_{D_0}	0.0197
$C_{D_{\alpha_1}}$	0.0791
$C_{D_{\alpha_2}}$	1.06
C_{D_q}	0
$C_{D_{\beta_2}}$	0.148
C_{D_β}	-0.00584
$C_{D_{\delta_{e2}}}$	0.0633

Table 2.4: C_Y Aerodynamic Coefficients for the Skywalker X8 Aircraft

Aerodynamic Coefficient	**Value**
C_{Y_0}	0
C_{Y_β}	-0.224
C_{Y_p}	-0.1172
C_{Y_r}	0.0959
$C_{Y_{\delta_a}}$	0.0433

The aerodynamic coefficients C_{m_α}, C_{l_β}, C_{n_β}, C_{m_q}, C_{l_p}, and C_{n_p} are known as our stability derivatives because their values determine the static and dynamic stability of the aircraft.

Table 2.5: C_L Aerodynamic Coefficients for the Skywalker X8 Aircraft

Aerodynamic Coefficient	**Value**
C_{L_0}	0.0867
C_{L_α}	4.02
C_{L_q}	3.8954
$C_{L_{\delta_e}}$	0.278

Table 2.6: C_l Aerodynamic Coefficients for the Skywalker X8 Aircraft

Aerodynamic Coefficient	**Value**
C_{l_0}	0
C_{l_β}	-0.0849
C_{l_p}	-0.4018
C_{l_r}	0.025
$C_{l_{\delta_a}}$	0.12

Table 2.7: C_m Aerodynamic Coefficients for the Skywalker X8 Aircraft

Aerodynamic Coefficient	**Value**
C_{m_0}	0.0302
C_{m_α}	-0.126
C_{m_q}	-1.3047
$C_{m_{\delta_e}}$	-0.206

Table 2.8: C_n Aerodynamic Coefficients for the Skywalker X8 Aircraft

Aerodynamic Coefficient	**Value**
C_{n_0}	0
C_{n_β}	0.0283
C_{n_p}	-0.0247
C_{n_r}	-0.1252
$C_{n_{\delta_a}}$	-0.00339

Static stability deals with the direction of the aerodynamic moments as the aircraft is perturbed away from its nominal flight condition. If the moments tend to restore the aircraft to the nominal condition - the aircraft is said to be statically stable. The coefficients C_{m_α}, C_{l_β}, and C_{n_β} determine the static stability of the aircraft as they represent

the change in the moment coefficients with respect to changes in the direction of the relative airspeed, as represented by α and β.

For the aircraft to be statically stable longitudinally, or in pitch, C_{m_α} must be negative. From Table 2.7 this is indeed the case so we can already state that our system is statically stable in the longitudinal plane and for an increase in α, the aircraft will tend to nose down in order to maintain the nominal angle of attack, thus rejecting the disturbance in α.

For the aircraft to be statically stable in roll, C_{l_β} must be negative. From Table 2.6, this is indeed the case so we know the aircraft is statically stable in roll and the system will generate rolling moments that roll the aircraft away from the direction of sideslip, thus driving the sideslip angle, β, to zero, and rejecting the disturbance.

For the aircraft to be statically stable in yaw, C_{n_β} must be positive. C_{n_β} is sometimes called the weathercock stability derivative because if an aircraft is statically stable in yaw, it will naturally point into the wind like a weathervane (or weathercock). From Table 2.8, this is indeed the case so we know the aircraft is statically stable in yaw and the system will generate a positive yawing moment which will yaw the aircraft into the direction of the relative airspeed, thus driving the sideslip angle, β, to zero, and rejecting the disturbance.

Dynamic stability deals with the dynamic behaviour of the airframe in response to disturbances. If a disturbance is applied to the aircraft, the system is said to be dynamically stable if the response of the airframe damps out over time. If we use a second-order mass-spring-damper analogy to analyse the system, the stability derivatives, C_{m_α}, C_{l_β}, and C_{n_β} behave as torsional springs, while the derivatives C_{m_q}, C_{l_p} and C_{n_p} act as torsional dampers. The moments of inertia of the aircraft provide the mass to the analogy.

C_{m_q}, C_{l_p} and C_{n_p} are referred to as damping derivatives and each of these should be negative so that a moment is produced that opposes the direction of motion - thus causing damping. From Tables 2.6, 2.7 and 2.8 this is indeed the case so our aircraft will tend to damp out disturbances and will thus be dynamically stable.

The aerodynamic coefficients of $C_{m_{\delta_e}}$ and $C_{l_{\delta_a}}$ are referred to as the primary control derivatives. This is because the moments produced are the intended result of the specific control surface deflection. For example, the intended result of an elevator deflection, δ_e, is to cause a pitching moment, m, so $C_{m_{\delta_e}}$ results in its primary function. $C_{n_{\delta_a}}$ is a cross-control derivative because it defines an off-axis moment that occurs when the control surface is deflected.

Given the sign convention presented in Figures 2.10 and 2.12, a positive elevator deflection results in a nose-down pitching moment (negative about $\mathbf{j^b}$) and a positive aileron deflection causes a right-wing-down rolling moment (positive about $\mathbf{i^b}$). As such, we

define the signs of the control derivatives such that positive deflections cause positive moments. For this to be the case, $C_{m_{\delta_e}}$ must be negative and $C_{l_{\delta_a}}$ positive. Tables 2.6 and 2.7 indicate that this is the case so our sign convention holds in the selection of our aerodynamic coefficients.

It should also be noted from Tables 2.4, 2.6 and 2.8 that our C_{Y_0}, C_{l_0}, and C_{n_0} are all zero. As discussed previously, this means that our aircraft is, as expected, symmetric in the $\mathbf{i^b}$-$\mathbf{k^b}$ plane.

Similarly to the simple analysis performed on the moments of inertia of our aircraft, the above discussion on our aerodynamic coefficients already provides us with significant insight into how we expect our system to behave in flight - even though we have yet to build a model of the aircraft. We expect the system to be very well-behaved and completely stable both statically and dynamically. However, how long disturbances take to damp out and the magnitude of the deviation from the nominal state before our natural aerodynamics reject the disturbance is difficult to determine from these simplified analogies and will have to be seen in more detailed characterisation. Having said that, we expect that our system should be naturally stable and so the controller design for the system should not pose any significant difficulties unless we want to ensure rapid response times, in which case we might have to more heavily rely on our control derivatives and not the natural disturbance rejection properties of the airframe.

2.3.3 Flat Plate Stall

In order to include some of the impact of non-steady, turbulent flow and essentially increase the range of α values for which our aerodynamic model is acceptably accurate, we need to have a model for stall. Stall occurs when the angle of attack increases to the point that flow separates from the wing, known as the stall angle, α_0, resulting in a drastic loss of lift. Under stall conditions, our standard lift coefficient is exceptionally optimistic in the forces that will be generated. The main weakness with the model is that it cannot predict the abrupt drop in lift force with increasing angle of attack. In order to incorporate wing stall into our longitudinal aerodynamic model, we modify our lift coefficient equation so that the lift and drag forces become non-linear in angle of attack.

On the onset of stall, the wing acts roughly like a flat plate, whose lift and drag coefficients can be modelled as [3, 21, 29]:

$$C_{L_{\text{flat plate}}} = 2\text{sign}(\alpha)\sin^2\alpha\cos\alpha \tag{2.40}$$

$$C_{D_{\text{flat plate}}} = 2\sin^3\alpha \tag{2.41}$$

We can merge this with our original equations by making use of a blending function

known as the sigmoid function. This is done as follows:

$$C_L(\alpha) = (1 - \sigma(\alpha))(C_{L_0} + C_{L_\alpha}\alpha) + \sigma(\alpha)\left[2\text{sign}(\alpha)\sin^2\alpha\cos\alpha\right] \tag{2.42}$$

$$C_D(\alpha) = (1 - \sigma(\alpha))(C_{D_0} + C_{D_\alpha}\alpha + C_{D_{\alpha_2}}\alpha^2) + \sigma(\alpha)\left[2\sin^3\alpha\right] \tag{2.43}$$

Where

$$\sigma(\alpha) = \frac{1 + e^{-M(\alpha-\alpha_0)} + e^{M(\alpha+\alpha_0)}}{(1 + e^{-M(\alpha-\alpha_0)})(1 + e^{M(\alpha+\alpha_0)})} \tag{2.44}$$

and α_0 is the previously mentioned stall angle, while M is known as the transition rate and defines how quickly we transition between the two models.

We extend this further to our pitching moment coefficient by making use of [3], where we state that:

$$C_{m_{\text{flat plate}}} = C_{m_{fp}}\text{sign}(\alpha)\sin^2\alpha \tag{2.45}$$

where $C_{m_{fp}}$ is a constant defined by the wing. We then make use of the same sigmoid blending function such that:

$$C_m(\alpha) = (1 - \sigma(\alpha))(C_{m_0} + C_{m_\alpha}\alpha) + \sigma(\alpha)\left[C_{m_{fp}}\text{sign}(\alpha)\sin^2\alpha\right] \tag{2.46}$$

For the Skywalker X8, these parameters are as follows [3]:

Table 2.9: Stall Coefficients for the Skywalker X8 Aircraft

Stall Coefficient	**Value**
M	50
α_0	0.2670
$C_{m_{fp}}$	-0.2168

In order to view the effects of our stall model, Figure 2.14 illustrates the variation in $C_L(\alpha)$, $C_D(\alpha)$, and $C_m(\alpha)$ as a function of alpha with the above parameters. These results illustrates that we generate the expected effect and see a sharp drop off in the magnitude of our parameters as we encounter our stall angle. This is most pronounced in lift, because lift has the largest magnitude of the coefficients.

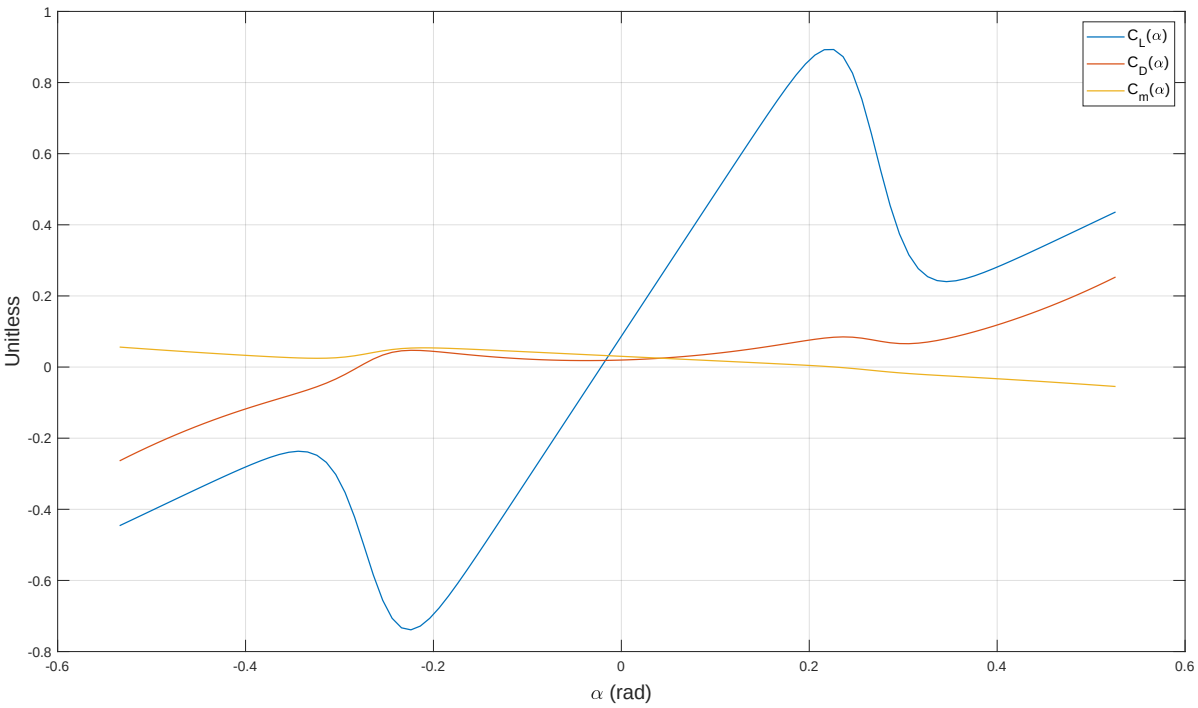


Figure 2.14: Aerodynamic coefficient blending with flat plate stall model for $C_L(\alpha)$, $C_D(\alpha)$, and $C_m(\alpha)$

2.3.4 Propulsion Forces and Moments

The Skywalker X8 makes use of a single propeller in a push configuration (in other words air is pushed away from the aircraft) to generate thrust in the $\mathbf{i^b}$ direction by accelerating air in the $-\mathbf{i^b}$ direction. Due to the reaction torque from the motor, which requires some non-zero torque to hold the propeller at a particular speed, we will also have a torque applied to the aircraft about the $\mathbf{i^b}$ axis in the same direction as the torque being applied to the other (or the opposite direction to that generated by the motor).

Propeller Thrust

By applying Bernoulli's principle to calculate pressure both ahead of and behind the propeller, and then applying the pressure difference to the propeller area, we can generate an approximate model for propeller thrust. As such, the total pressure up and downstream of the propeller can be determined by:

$$P_{\text{upstream}} = P_0 + \frac{1}{2}\rho V_a^2$$
$$P_{\text{downstream}} = P_0 + \frac{1}{2}\rho V_{exit}^2$$

where P_0 is the static pressure and ρ is the air density. V_{exit} is simply the speed of the air as it leaves the propeller. Ignoring motor transients we can assume for an input

command to the motor, $\delta_{t_{cmd}} \approx \delta_t$, we will get some angular velocity of the propeller. For any particular angular velocity of the propeller, we assume that there is a particular V_{exit}. As such, we can model our exit velocity as:

$$V_{exit} = k_{motor}\delta_t$$

where k_{motor} is a scaling factor for normalized propeller speed, δ_t, to the exit velocity of air leaving the propeller.

If S_{prop} is the area swept out by the propeller, then the thrust produced by the motor is given by:

$$\begin{aligned} F_{x_p} &= S_{prop}C_{prop}(P_{\text{downstream}} - P_{\text{upstream}}) \\ &= \frac{1}{2}\rho S_{prop}C_{prop}\left[(k_{motor}\delta_t)^2 - V_a^2\right] \end{aligned}$$

Where C_{prop} is a dimensionless coefficient used to adjust the propeller thrust to account for unmodelled effects (eg. propeller pitch).

As such, in the body coordinates we can model our propulsion thrust as:

$$\mathbf{F_p} = \frac{1}{2}\rho S_{prop}C_{prop}\begin{pmatrix}(k_{motor}\delta_t)^2 - V_a^2 \\ 0 \\ 0\end{pmatrix} \tag{2.47}$$

Propeller Torque

The reaction torque from the motor to the aircraft is opposite to the direction of the propeller rotation and can be approximated to be proportional to the square of the propeller angular velocity. In other words:

$$T_p = -k_{T_p}(k_\Omega\delta_t)^2$$

where $\Omega = k_\Omega\delta_t$ is the propeller speed and k_{T_p} is a constant determined by experiment. As such, the moments due to the propulsion system are subsequently stated as:

$$\mathbf{M_p} = \begin{pmatrix}-k_{T_p}(k_\Omega\delta_t)^2 \\ 0 \\ 0\end{pmatrix} \tag{2.48}$$

The effects of this torque are usually minor and can often be neglected. This is because if we leave this unaccounted for, we will simply cause a slow rolling motion in the direc-

tion opposite to the propeller rotation which is easily corrected for by applying a small aileron deflection which our roll controller would do regardless.

For the Skywalker X8, the propeller coefficients are summarised in Table 2.10.

Table 2.10: Propeller Coefficients for the Skywalker X8 Aircraft

Propeller Coefficient	**Value**
S_{prop}	0.1018
C_{prop}	0.5
k_{motor}	40
k_{T_p}	0
k_Ω	0

2.3.5 Implementation of Forces and Moments

The total forces and moments acting on the aircraft can be stated as follows, which makes use of Equations (2.27), (2.34), (2.35), (2.47) and (2.48):

$$\begin{pmatrix} f_x \\ f_y \\ f_z \end{pmatrix} = \begin{pmatrix} -\mathsf{m}g\sin\theta \\ \mathsf{m}g\cos\theta\sin\phi \\ \mathsf{m}g\cos\theta\cos\phi \end{pmatrix} + \frac{1}{2}\rho V_a^2 S \begin{pmatrix} C_X(\alpha, q, \delta_e) \\ C_Y(\beta, p, r, \delta_a) \\ C_Z(\alpha, q, \delta_e) \end{pmatrix} + \frac{1}{2}\rho S_{prop} C_{prop} \begin{pmatrix} (k_{motor}\delta_t)^2 - V_a^2 \\ 0 \\ 0 \end{pmatrix} \tag{2.49}$$

$$\begin{pmatrix} l \\ m \\ n \end{pmatrix} = \frac{1}{2}\rho V_a^2 S \begin{pmatrix} bC_l(\beta, p, r, \delta_a) \\ cC_m(\alpha, q, \delta_e) \\ bC_n(\beta, p, r, \delta_a) \end{pmatrix} + \begin{pmatrix} -k_{T_p}(k_\Omega\delta_t)^2 \\ 0 \\ 0 \end{pmatrix} \tag{2.50}$$

Where the expansion of the aerodynamic coefficients, C_X, C_Y, C_Z, C_l, C_m, and C_n are discussed in the preceding sections.

The calculation of our forces and moments is implemented in Simulink as a Matlab function, which is shown as a masked subsystem in Figure 2.15, and the code used to implement this functionality can be found as f_ForcesAndMoments.m in the code repository provided in Section 1.4.

This subsystem takes in all of the inputs required in our forces and moments equation and outputs the forces described in Equation (2.49) and the moments described in Equation (2.50). Which are in the exact form required for the 6DoF dynamic model depicted in Figure 2.11.

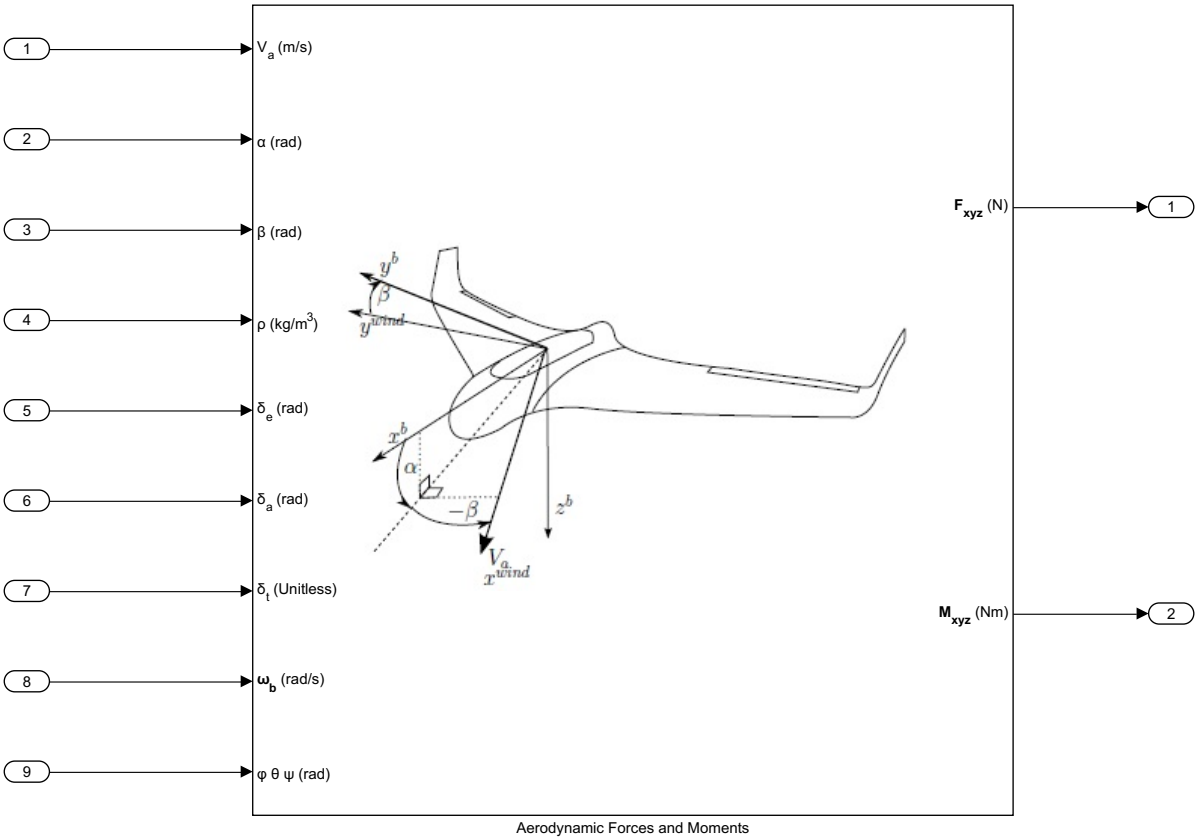


Figure 2.15: Simulink implementation of aerodynamic forces and moments

2.4 Implementation of Aircraft Model

The subsystem shown in Figure 2.15 can be combined with the 6DoF model which is depicted in Figure 2.11 in order to generate our full aircraft model.

As such, our entire Simulink model is shown in its implemented form in Figure A.1. Where one can see that our 2 main blocks have been augmented with: 1. A body frame airspeed calculation block which is used to perform the equations described by Equation (2.7) to determine V_a, α, and β; 2. A mixing matrix which makes use of Equation (2.28), denoted as Elevon Conversion in the diagram; and finally, 3. A calculation for course and crab angle as per Equation (2.9) in the Course Angle Calculation block.

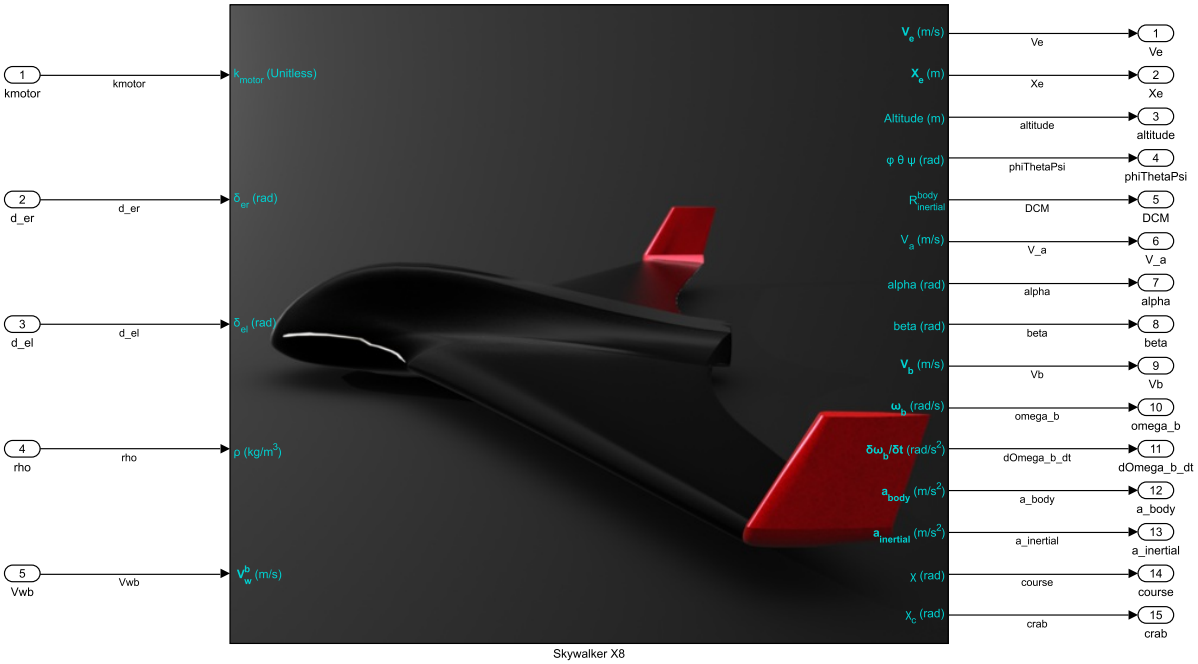


Figure 2.16: Masked subsystem of the full Skywalker X8 aircraft model

2.5 Wind Modelling

In order to expand our model to include the effects of atmospheric disturbances, such as wind, we need to review how these disturbances enter into the dynamics of the system.

We have defined the relationship between ground velocity, air velocity, and wind velocity in Equation (2.4), which is rearranged as:

$$\mathbf{V_g} = \mathbf{V_a} + \mathbf{V_w}$$

We assume that our total wind vector can be represented as

$$\mathbf{V_w} = \mathbf{V_{w_s}} + \mathbf{V_{w_g}}$$

where $\mathbf{V_{w_s}}$ is a constant vector that represents a steady ambient wind, and $\mathbf{V_{w_g}}$ is a stochastic process that represents wind gusts and other atmospheric disturbances. The

ambient wind is usually expressed in the inertial frame, $\mathcal{F}^i$ as:

$$\mathbf{V}^{\mathbf{i}}_{\mathbf{w_s}} = \begin{pmatrix} w_{n_s} \\ w_{e_s} \\ w_{d_s} \end{pmatrix}$$

where w_{n_s}, w_{e_s}, and w_{d_s} are the speeds of the steady wind in the north, east and down directions, respectively.

The stochastic component of the wind is typically expressed in the aircraft body frame because atmospheric effects experienced by the aircraft in the direction of its forward motion occur at a higher frequency than do those in the lateral and down directions. The gust portion of the wind can be written in terms of its body-frame components as:

$$\mathbf{V}^{\mathbf{b}}_{\mathbf{wg}} = \begin{pmatrix} u_{w_g} \\ v_{w_g} \\ w_{w_g} \end{pmatrix}$$

In order to generate the values for the gust components, we make use of the Dryden Wind Turbulence Model. This takes in a white noise signal and passes it through a linear time-invariant filter. The parameters for the Dryden Wind Turbulence Model are defined in MIL-F-8785C and a block is available from the Simulink Aerospace Toolbox [34] which will do the calculation for the user. This is what was made use of in this project.

2.5.1 Implementation of Wind Modelling

The combination of the steady and gust terms can be expressed mathematically as:

$$\mathbf{V}^{\mathbf{b}}_{\mathbf{w}} = \begin{pmatrix} u_w \\ v_w \\ w_w \end{pmatrix} = \mathcal{R}^b_v(\phi, \theta, \psi) \begin{pmatrix} w_{n_s} \\ w_{e_s} \\ w_{d_s} \end{pmatrix} + \begin{pmatrix} u_{w_g} \\ v_{w_g} \\ w_{w_g} \end{pmatrix} \tag{2.51}$$

where $\mathcal{R}^b_v$ is the rotation from the vehicle to the body frame.

From Equation (2.7), stated again as:

$$\begin{aligned} V_a &= \sqrt{u_r^2 + v_r^2 + w_r^2} \\ \alpha &= \tan^{-1}\left(\frac{w_r}{u_r}\right) \\ \beta &= \sin^{-1}\left(\frac{v_r}{\sqrt{u_r^2 + v_r^2 + w_r^2}}\right) \end{aligned}$$

we know that our airspeed vector depends on $\mathbf{V_w^b}$. Given that our forces and moments subsequently depend on the magnitude of our airspeed as well as α and β, which define its direction, the critical observation here is that the impact of wind on our system is coupled in via our airspeed vector. In our Simulink model, this is implemented as is depicted in Figure A.2, where we note that we have implemented the wind model described mathematically in Equation (2.51) in the Simulink model. It should also be noted that we take the full wind vector and rotate this back into the inertial frame before passing it to an output, as our output is required to be in the inertial frame and not the body frame and needs to include the full wind effect.

This is then masked into a subsystem which includes the aerodynamic effects and makes up the full airframe model (ie. does not include actuators and sensors). This system is depicted in Figure 2.17 where the background has been adjusted from Figure 2.16 in order to illustrate the inclusion of atmospheric effects. It should be noted that this block includes the phrase "Observable Feedback" in its title, this is because it only has outputs which are measurable by sensors. This will be discussed in a subsequent section.

Figure 2.17: Full Skywalker X8 aircraft model

With this, the plant model has now been fully defined and as such we can perform some analysis as well as linearise the system at various trim conditions, which will allow for additional analysis. This will allow us to draw many insights into the system dynamics and how we might design the autopilot controllers to follow.

2.6 Aircraft Performance

Having established the equations for our forces and moments, as well as knowing the aerodynamic coefficients which define the Skywalker X8 airframe, we can start reviewing

various performance limitations on our system by making use of simple force and moment balances. This allows us to gain unique insight into the limitations of our system so that we don't create unrealistic controller goals during autopilot design.

2.6.1 Flight Envelope

The flight envelope is a good starting point in understanding the limitations of our airframe. The flight envelope of the system which will be presented here will be a simple model which indicates the velocity bounds within which the aircraft can operate in trim - defined in this case as a simple balance in lift such that altitude is neither gained nor lost. Intuitively we expect that the lowest airspeed possible will be when we are generating our maximum amount of lift via means other than the dynamic pressure, while the highest possible airspeed will be when our drag is minimized.

Minimum Airspeed

For our lowest airspeed, we expect to be generating our maximum amount of lift in C_L, so that we maximise the lift force as defined in Equation (2.30) such that the dynamic pressure contribution can be kept as small as possible in order to still balance the weight of our aircraft. If we assume zero pitch rate, this condition will occur when $\alpha = \alpha_{max}$ and $\delta_e = \delta_{e_{max}}$. Using a zero angle approximation for our pitch angle, or $\theta \approx 0$, means that we can make use of Equation (2.49) to balance our lift force with weight by simply balancing f_z as

$$0 = \mathsf{m}g + \frac{1}{2}\rho V_a^2 S C_Z(\alpha, q, \delta_e)$$

$$\sqrt{\rho} V_{a_{min}} = \sqrt{-\frac{2\mathsf{m}g}{S C_Z(\alpha_{max}, 0, \delta_{e_{max}})}}$$

Where we have indicated that our pitch rate, q, is zero as well as $\alpha = \alpha_{max}$ and $\delta_e = \delta_{e_{max}}$. We have kept $\sqrt{\rho}$ together with the airspeed parameter because this value will vary as a function of altitude, which is something we will manually adjust when determining the flight envelope, while the portion on the RHS of the equation will always be constant regardless of the altitude. Expanding C_Z as per Equation (2.33) results in

$$\sqrt{\rho} V_{a_{min}} = \left(\frac{2\mathsf{m}g}{S\left[\sin\alpha_{max} C_D(\alpha_{max}, 0, \delta_{e_{max}}) + \cos\alpha_{max} C_L(\alpha_{max}, 0, \delta_{e_{max}})\right]} \right)^{\frac{1}{2}} \tag{2.52}$$

We know from our previous discussion on the lift coefficient that we have maximum lift just prior to stall. As such, we can approximate $\alpha_{max} \approx \alpha_0$. In other words, our maximum angle of attack will be approximately the stall angle. In order to determine our maximum elevator deflection angle, we can make use of a simple requirement that

we have a moment balance in pitch.

From Equation (2.50) we know that in order for us to have trim in the pitch axis moments, ie. $m = 0$, we need $C_m(\alpha, q, \delta_e) = 0$. Using Equation (2.32) we can thus define our elevator deflection as:

$$\delta_e = -\frac{C_{m_0} + C_{m_\alpha}\alpha + C_{m_q}\frac{c}{2V_a}q}{C_{m_{\delta_e}}}$$

Where we can further assume that at the instant of consideration, we have zero pitch rate, such that

$$\delta_e = -\frac{C_{m_0} + C_{m_\alpha}\alpha}{C_{m_{\delta_e}}}$$

Knowing that $\alpha = \alpha_{max} = \alpha_0$ for maximum lift, we know that

$$\delta_{e_{max}} = -\frac{C_{m_0} + C_{m_\alpha}\alpha_0}{C_{m_{\delta_e}}} \tag{2.53}$$

Which allows us to fully define our minimum airspeed.

Maximum Airspeed

Intuitively we expect that our maximum airspeed will occur when we run our propeller at its maximum speed and we minimise drag. From Equation (2.31) we know that drag is minimized when $\alpha = 0$.

Using the same pitch moment balance as in the preceding section, we know that:

$$\delta_{e_{min}} = -\frac{C_{m_0}}{C_{m_{\delta_e}}} \tag{2.54}$$

Using the same small pitch angle approximation, $\theta \approx 0$, as in the preceding section, we can use the same lift balance such that:

$$\sqrt{\rho}V_{a_{max}} = \sqrt{\frac{2\mathsf{m}g}{SC_L(0, 0, \delta_{e_{min}}))}} \tag{2.55}$$

Absolute Maximum Airspeed

The airspeed in the preceding section only deals with the maximum airspeed possible in order to maintain lift. Airspeeds slower than this (but not slower than the minimum airspeed) will also have a trim condition that will allow us to maintain altitude. However,

although our lift balance indicates that a certain maximum speed is allowable, there exist two additional constraints. These are drag and power.

In order for our aircraft to fly with a particular forward velocity, we need to ensure we have enough propeller thrust to balance our drag force. In other words, we need a force balance in f_x. This might happen prior to the maximum air velocity previously calculated, as we are constrained by how much thrust the propeller can generate for a particular airspeed because $\delta_t \leq 1$.

In order to review our thrust limit airspeed, we assume several things.

1. Small pitch angle approximation, $\theta \approx 0$
2. Pitch moment balance, $m \approx 0$
3. Minimum drag angle of attack, $\alpha \approx 0$
4. Zero pitch rate, $q \approx 0$

We then make use of f_x as described in Equation (2.49) and note that

$$f_x = 0 = \frac{1}{2}\rho V_a^2 S C_X(\alpha, q, \delta_e) + \frac{1}{2}\rho S_{prop} C_{prop} \left[(k_{motor}\delta_t)^2 - V_a^2\right]$$

Which, after applying our assumptions and rearranging our equation gives us

$$V_{a_{max_{\text{thrust limit}}}} = \sqrt{\frac{S_{prop} C_{prop} (k_{motor}\delta_t)^2}{S_{prop} C_{prop} + S C_D(0, 0, \delta_{e_{min}})}}$$

Which is maximum when $\delta_t = 1$. As such,

$$V_{a_{max}} \leq \sqrt{\frac{S_{prop} C_{prop} (k_{motor})^2}{S_{prop} C_{prop} + S C_D(0, 0, \delta_{e_{min}})}} \tag{2.56}$$

Finally, the only additional constraint on maximum airspeed is that the motor which is driving the propeller can only provide a constrained amount of power. Since we can approximate the power requirement of the aircraft as $P = f_x V_a$, and we know at our maximum speed we have a force balance in f_x, we know that

$$f_x = \frac{1}{2}\rho V_a^2 S C_X(\alpha, q, \delta_e) = \frac{1}{2}\rho V_a^2 S C_D(0, 0, \delta_{e_{min}})$$

Such that,

$$P = \frac{1}{2}\rho V_a^3 S C_D(0, 0, \delta_{e_{min}})$$

Finally, we can reduce this to:

$$V_{a_{max}} \leq \left(\frac{P_{max}}{\frac{1}{2}\rho S C_D(0, 0, \delta_{e_{min}})} \right)^{\frac{1}{3}} \tag{2.57}$$

While these approximations provide a good starting point, they do not take into account some critical relationships between parameters and as such, some of the assumptions are not entirely valid.

For example, if we consider an aircraft flying under conditions where an approximately-zero angle of attack if required to maintain trim, it would subsequently be impossible to have no pitch rate while simultaneously having a non-negligible angle of attack in a steady state condition - steady state in this case being defined as no nett forces or moments acting on the aircraft. One can think of this by considering an aircraft that is travelling forward while retaining its altitude. It is now commanded to start ascending so needs to pitch upward to do so. During the pitch manoeuvre, the elevator (or equivalent in our case) is deflected, which causes the aircraft to pitch upward. This increases the angle of attack of the aircraft which increases the lift. As the lift increases the aircraft will start to accelerate upward, which means the flight path angle will increase. The pitch rate remains constant, but now the flight path angle also increases and at some stage will be increasing at the same rate as the pitch angle. Since, by definition, $\alpha = \theta - \gamma$, we know that if $\dot{\theta} = \dot{\gamma}$, then $\dot{\alpha} = 0$ - and as such, the moment $\dot{\theta} = \dot{\gamma}$, we will hold our current angle of attack as we continue pitching upward. At some point the desired pitch angle will be reached and the elevators will be set back to their neutral position as the pitch rate is reduced to zero to hold the required pitch angle - at this stage γ will catch up to θ, and α will be reduced to the required trim value. This effect is captured in Figure 2.18, where we assume that the trim condition is such that $\alpha = 0$.

From this discussion it should be clear there is an inextricable link between q and α. This observation allows us to intuitively understand that our zero pitch rate assumption is only an approximation to keep the maths simpler, because it cannot be realistic. As such, the non-linear model of the aircraft defined previously will be used to capture this, as well as any other dynamics which we have reduced for our simple approximations.

This insight into the link between pitch rate, q, and angle of attack, α, will be extremely beneficial when designing our longitudinal autopilot. This is because we know that in order to avoid stalling the aircraft, we need to ensure that our pitch control system does not react too aggressively. Alternatively, if we want a fast-acting controller for disturbance rejection purposes, we need to ensure a pitch angle command that does not vary too quickly.

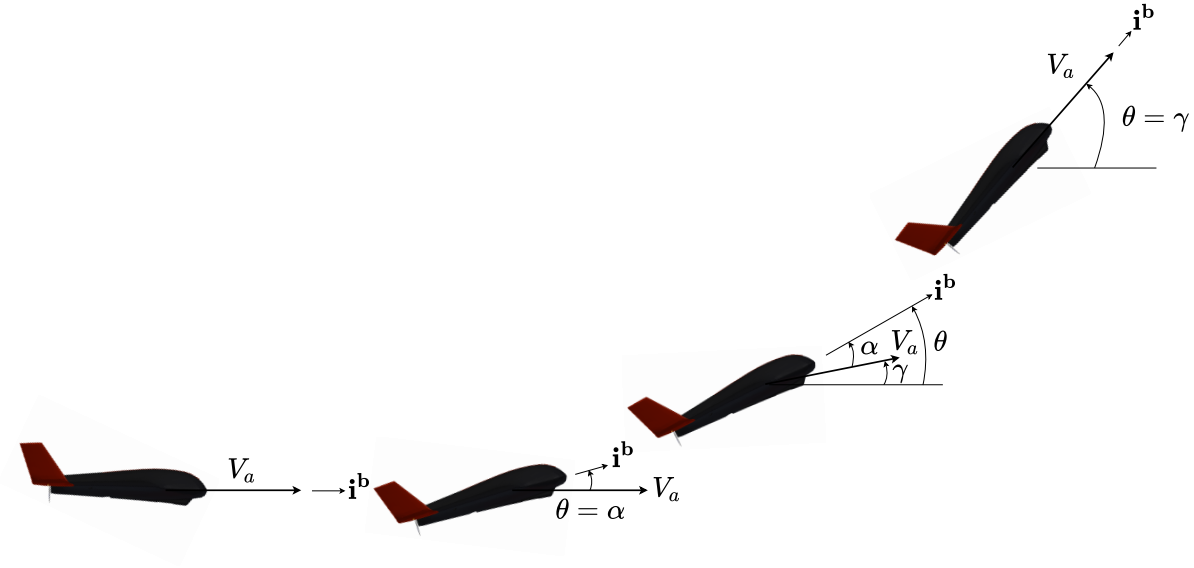


Figure 2.18: Illustration of α for a pitching aircraft

2.6.2 Performance Limits

Similarly to our discussion in the preceding section, there are many other performance parameters which we can estimate with some simple analytical methods. These metrics are by no means exhaustive or extremely accurate, but they provide some good insight into the overall system performance which will prove to be invaluable when designing controllers. This is because they show us what is physically possible with this system so that controllers are not tuned with goals in mind which are unrealistic. These metrics will be calculated with a particular airspeed in mind, with the value for this airspeed coming from our minimum and maximum airspeeds as discussed in the preceding section.

Maximum Roll/Pitch Rates

In order to determine the maximum rate at which we can roll (ie. roll rate, p), or pitch (ie. pitch rate, q), for a particular airspeed, we make use of the assumption that these rates need to be constant. As such, we know that $l = 0$ and $m = 0$, and we can utilize the expressions as per Equation (2.50) to state

$$l = 0 = \frac{1}{2}\rho V_a^2 SbC_l(\beta, p, r, \delta_a) - k_{T_p}(k_\Omega \delta_t)^2$$
$$m = 0 = \frac{1}{2}\rho V_a^2 ScC_m(\alpha, q, \delta_e)$$

We also then approximate $\beta \approx 0$ and $r \approx 0$, and we know from our propeller coefficients

in Table 2.10 that $k_{T_p} = 0$. As such:

$$C_l(0, p, 0, \delta_a) = 0$$
$$C_m(\alpha, q, \delta_e) = 0$$

Intuitively we know that for our maximum roll rate, $p = p_{max}$, we will need to apply our maximum aileron deflection, $\delta_a = \delta_{a_{max}}$. Likewise, for our maximum pitch rate, $q = q_{max}$, we will need to apply our maximum elevator deflection, $\delta_e = \delta_{e_{max}}$. We also know from the preceding discussion, and indeed just from reviewing Figure 2.14, that for $C_m = C_{m_{max}}$, we need $\alpha \approx \alpha_0$, which will allow us to determine the maximum attainable steady state pitch rate. As such,

$$C_l(0, p_{max}, 0, \delta_{a_{max}}) = 0$$
$$C_m(\alpha_0, q_{max}, \delta_{e_{max}}) = 0$$

We can now make use of Equations (2.37) and (2.38) to calculate our maximum roll and pitch rates as:

$$p_{max} = \frac{-2V_a(C_{l_0} + C_{l_{\delta_a}}\delta_{a_{max}})}{bC_{l_p}} \tag{2.58}$$

$$q_{max} = \frac{-2V_a(C_{m_0} + C_{m_\alpha}\alpha_0 + C_{m_{\delta_e}}\delta_{e_{max}})}{cC_{m_q}} \tag{2.59}$$

Coordinated Turn

In order to determine performance limits for maximum roll angle, ϕ_{max}, maximum yaw rate, $\dot{\psi}_{max}$, and minimum turn radius, R_{min}, for a given airspeed, we need to define what is known as a coordinated turn [35].
During a coordinated turn, there is no lateral acceleration in the body frame of the aircraft. The aircraft banks to turn, as opposed to skidding laterally. Making use of these dynamics allows us to develop simple expressions that relate course, $\dot{\chi}$, or heading $\dot{\psi}$ rate, to the bank (or roll) angle, ϕ [33]. As shown in Figure 2.19, the centripetal force acting on the aircraft is equal to the horizontal component of the lift force acting in the radial direction. Summing forces in the horizontal direction gives:

$$\begin{aligned} F_{\text{lift}} \sin\phi \cos(\chi - \psi) &= \mathsf{m}\frac{v^2}{R} \\ &= \mathsf{m}v\omega \\ &= \mathsf{m}(V_g \cos\gamma)\dot{\chi} \end{aligned} \tag{2.60}$$

where F_{lift} is the lift force, γ is the flight path angle, V_g is the ground speed, and χ is the course angle.

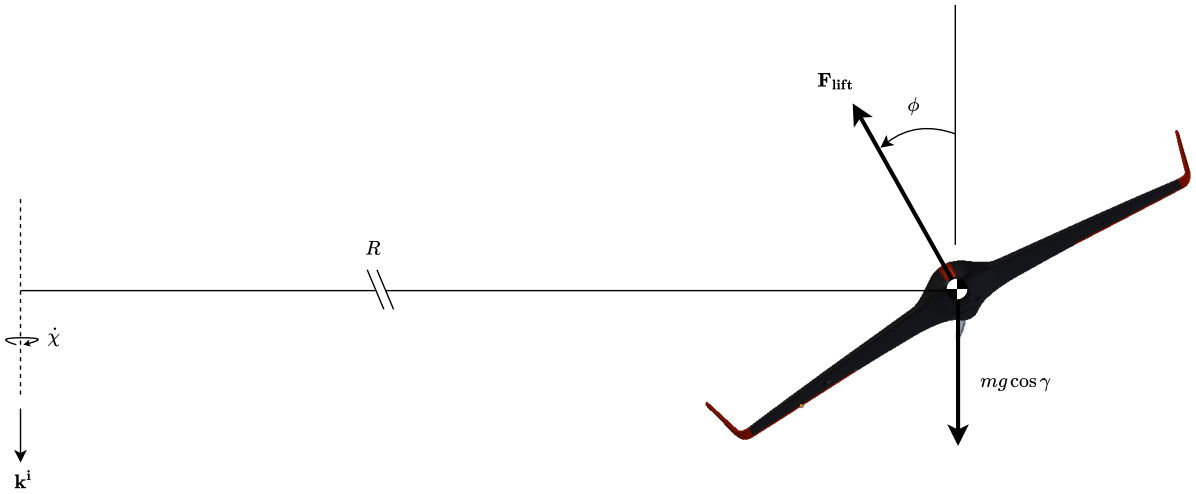


Figure 2.19: Forces on an aircraft during a coordinated turn. Forces shown in $\mathbf{j^b}$-$\mathbf{k^b}$ plane

Balancing forces in the vertical direction we get:

$$F_{\text{lift}} \cos\phi = \mathsf{m} g \cos\gamma \tag{2.61}$$

Combining Equations (2.60) and (2.61) allows us to solve for $\dot{\chi}$ as

$$\dot{\chi} = \frac{g}{V_g} \tan\phi \cos(\chi - \psi) \tag{2.62}$$

Which is the equation for a coordinated turn.

Given that the turn radius is given by $R = V_g \cos\gamma / \dot{\chi}$, we get:

$$R = \frac{V_g^2 \cos\gamma}{g \tan\phi \cos(\chi - \psi)} \tag{2.63}$$

Where, in the absence of wind or sideslip, we have that, $V_a = V_g$, and $\quad = \chi$, which leads to:

$$\dot{\chi} = \frac{g}{V_g} \tan\phi = \dot{\psi} = \frac{g}{V_a} \tan\phi \tag{2.64}$$

From [21] we know that Equation (2.64) also holds true in the presence of wind.

We can use these results to calculate our maximum roll angle, ϕ_{max}, maximum yaw rate,

$\dot{\psi}_{max}$, and minimum turn radius, R_{min} for a given airspeed as follows:

$$\phi_{max} = \cos^{-1}\left(\frac{\mathsf{m}g}{\frac{1}{2}\rho V_a^2 S C_Z(\alpha, q, \delta_e, \beta)}\right) \tag{2.65}$$

$$\dot{\psi}_{max} = \frac{g}{V_a}\tan\phi_{max} \tag{2.66}$$

$$R_{min} = \frac{V_a}{\dot{\psi}_{max}} \tag{2.67}$$

which assume that we have no wind or sideslip such that $V_a = V_g$, and $\psi = \chi$. We also assume a constant altitude manoeuvre such that $\gamma = 0$.

In order to calculate ϕ_{max} with Equation (2.65) we make the assumption that $\beta = 0$ and that $C_{m_q}\frac{c}{2V_a}q \ll C_{m_0} + C_{m_\alpha}\alpha + C_{m_{\delta_e}}\delta_e$ and then assume that in order to maintain our maximum bank angle, we will need to actuate our elevators to their maximum values. As such,

$$\alpha_{\phi_{max}} = -\frac{C_{m_0} + C_{m_{\delta_e}}\delta_{e_{max}}}{C_{m_\alpha}}$$

However, if this results in $\alpha \geq \alpha_0$, we instead say that $\alpha = \alpha_0$ and rearrange to solve for $\delta_{e_{\phi_{max}}}$. Assuming, however, that this isn't the case, we rewrite Equation (2.65) as:

$$\phi_{max} = \cos^{-1}\left(\frac{\mathsf{m}g}{\frac{1}{2}\rho V_a^2 S C_Z(\alpha_{\phi_{max}}, 0, \delta_{e_{max}}, 0)}\right)$$

If it is, we would rewrite Equation (2.65) as:

$$\phi_{max} = \cos^{-1}\left(\frac{\mathsf{m}g}{\frac{1}{2}\rho V_a^2 S C_Z(\alpha_0, 0, \delta_{e_{\phi_{max}}}, 0)}\right)$$

2.6.3 Skywalker X8 Performance Characterisation

In order to determine the performance of the Skywalker X8 aircraft, we will first generate our flight envelope and use this as a base for establishing all of our other performance parameters previously defined in this section.

In order to perform this analysis, we need to define certain parameters which will describe the physical constraints on the aircraft. Some of these parameters have already been

provided earlier in this text, but are restated here for ease of reading. These are stated in Table 2.11.

Table 2.11: Required Parameters for the Aircraft Performance Characterisation

Parameter	**Value**	**Unit**
$\delta_{e_{max}}$	30	$^\circ$
$\delta_{a_{max}}$	30	$^\circ$
$\delta_{t_{max}}$	1	Unitless
P_{max}	900	W
α_0	0.2670	rad

It should be noted from Table 2.11 that $\delta_{e_{max}}$ and $\delta_{a_{max}}$ are assumed physical limitations of the servomotors used to actuate the control surfaces, with the physical limit on the servomotor itself being half of the quoted value, given that we control our system with elevons which allow us to make use of Equation (2.28) and as such we are able to get twice the effective angle by using both control surfaces. It should be noted that [3] states that a more realistic value for these would be closer to $\delta_{e_{max}} = \delta_{a_{max}} = 60°$, but we take a conservative approach here. This does not in any way impact the analysis of the system, it just means that it might be the case that one would be able to generate larger forces and moments should one make use of the servomotors with a larger travel angle than that assumed for this text.

The only other parameter not previously stated in the text thus far is P_{max}. This is also provided in [3], where the limit is determined by looking at the burst power capability of the Hacker A40-12S V2 14-Pole 3-Phase Brushless DC Motor which was used to drive the propeller for the aircraft.

In order to determine our flight envelope we make use of the following algorithm:

Algorithm 1 Flight Envelope Calculation

$altitude$ = linspace(0, 30000, 300)
$\delta_{e_{max}}$ = deg2rad(30)
$\delta_{a_{max}}$ = deg2rad(30)
$P_{max} = 900$
$\alpha_0 = 0.267$
$\delta_{e_{\text{Max Lift}}} \leftarrow$ Equation (2.53)
$\delta_{e_{MinLift}} \leftarrow$ Equation (2.54)
$\sqrt{\rho}V_{a_{max}} \leftarrow$ Equation (2.55)
$\sqrt{\rho}V_{a_{min}} \leftarrow$ Equation (2.52)
$V_{a_{max}} \leftarrow$ Equation (2.56)
$opspec$ = operspec(sys)

for i `= 1:length(`$altitude$`)` **do**

$alt = altitude(i)$
$\rho =$ atmoscoesa(alt)
$V_{a_{max}} = \sqrt{\rho} V_{a_{max}} \times \frac{1}{\sqrt{\rho}}$
$V_{a_{min}} = \sqrt{\rho} V_{a_{min}} \times \frac{1}{\sqrt{\rho}}$

if $V_{a_{max}} \geq V_{a_{max}_{\text{thrust limit}}}$ from Equation (2.56) **then**
$V_{a_{max}} \leftarrow$ Equation (2.56)
end if
if $V_{a_{max}} \geq V_a \leftarrow$ Equation (2.57) **then**
$V_{a_{max}} \leftarrow$ Equation (2.57)
end if

$V_{a_{max}} = 1.25 \times V_{a_{max}}$
$V_{a_{min}} = 0.75 \times V_{a_{min}}$

while true **do**

$V_{a_{max}} = V_{a_{max}} - 0.1$

if $V_{a_{max}} \leq V_{a_{min}}$ **then**
break
end if

$op =$ findop$(sys, opspec)$

if op was found **then**
break
end if

end while

while true **do**

$V_{a_{min}} = V_{a_{min}} + 0.1$

if $V_{a_{min}} \geq V_{a_{max}}$ **then**
break
end if

$op =$ findop$(sys, opspec)$

if *op* was found **then**
break
end if

end while

$V_{a_{min_{trim}}} = V_{a_{min}}$
$V_{a_{max_{trim}}} = V_{a_{max}}$

end for

This algorithm is implemented in Matlab as SkywalkerX8_ FlightEnvelopePerformance-Modes.m, which is available in the code repository provided in Section 1.4, and essentially is just an implementation of our equations derived for the flight envelope.

We first define our constants, which are just some arbitrary altitude values which go much higher than we expect we will be able to fly, followed by our physical limits. We then determine $\delta_{e_{\text{Max Lift}}}$ and $\delta_{e_{\text{Min Lift}}}$ from the stated equations. Finally, we calculate our reduced-order linear approximations of $\sqrt{\rho}V_{a_{max}}$ and $\sqrt{\rho}V_{a_{min}}$, as well as our absolute maximum velocity based on maximum thrust and power conditions.

With all of these initial values calculated we can now start incrementally increasing our altitude and seeing what our maximum and minimum velocities end up being. This is done by firstly calculating our air density at the current altitude using the COESA (Committee on Extension to the Standard Atmosphere) atmospheric model in Matlab [36]. This is then used to calculate our linear approximations for $V_{a_{max}}$ and $V_{a_{min}}$ by scaling our previously calculated values by $\frac{1}{\sqrt{\rho}}$. Indeed, this is why we derived our equations with the $\sqrt{\rho}$ term grouped with the V_a term. We then ensure that our calculated maximum velocity does not exceed the maximum allowable velocity specified by either Equation (2.56) or Equation (2.56), setting the velocity to this value if that is indeed the case. Having now calculated our expected minimum and maximum velocity, we make use of our previously determined 6DoF model to find trim conditions at all altitudes. The model used to find the trim conditions is presented in Figure A.3.

We do this by making use of our calculated linear $V_{a_{max}}$ and $V_{a_{min}}$ values to calculate starting points for a search algorithm which is designed to iteratively look for the first solution we can find for a trim condition. For $V_{a_{max}}$, we start with a slightly higher velocity and incrementally decrease this until we find the first trim condition. For $V_{a_{min}}$ we start with a slightly lower velocity and incrementally increase this until we find the first trim condition. However, if either $V_{a_{max}} \leq V_{a_{min}}$, or $V_{a_{min}} \geq V_{a_{max}}$, we break out of the loop early.

Making use of this algorithm we can determine the flight envelope for our system. This is done by making use of a variant of our full 6DoF model that makes use of a the

COESA atmosphere block in Simulink [37] in order to create a dependence between air density, ρ, and altitude, h. This configuration is depicted in Figure A.3.

The resultant flight envelope is depicted in Figure 2.20. It should be noted from this figure that we get the expected behaviour of a standard flight envelope, except for the staircase-looking portion in the lower right. While the exact cause of this is unknown, it is expected that it is due to the tolerancing in the findop() function in the Matlab environment [38].

With a fully defined flight envelope, we can now make good use of our calculated V_a values to determine our expected performance characteristics as were derived in Equations (2.58), (2.59), (2.65), (2.66) and (2.67). This will give us some very good insight into how the performance of our aircraft varies around the flight envelope. These values are also depicted in Figure 2.20.

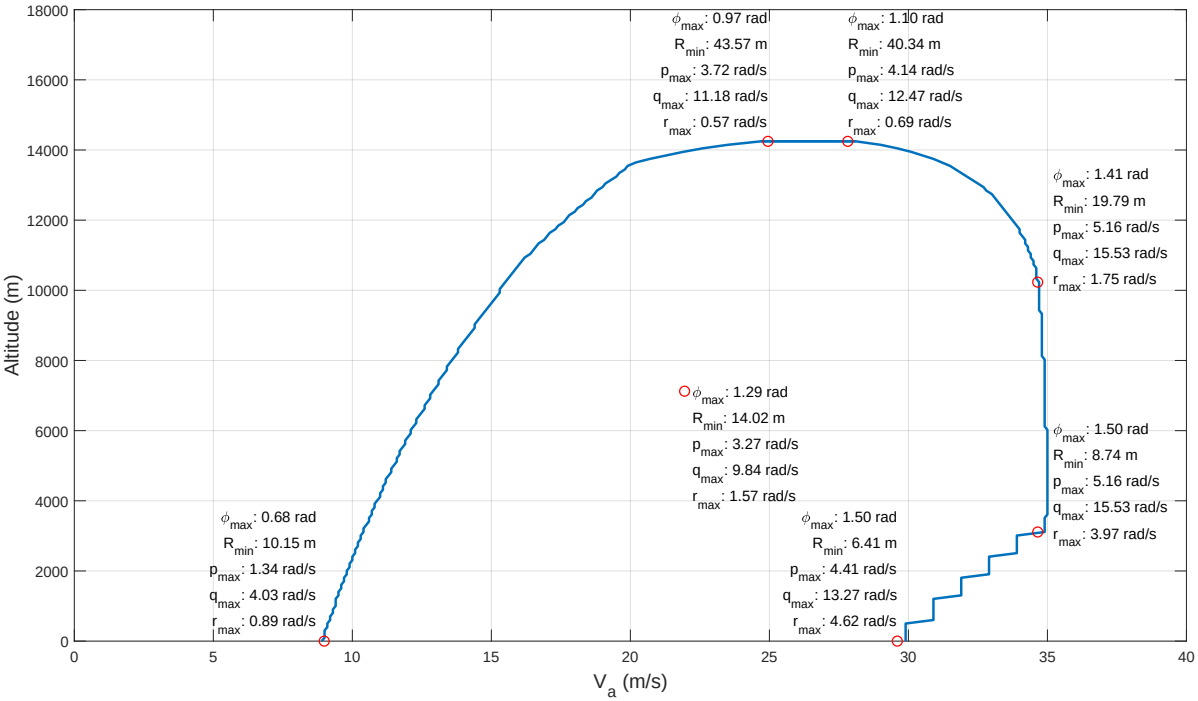


Figure 2.20: Performance limits of the Skywalker X8 across the flight envelope

In order to get a good idea of how these values might vary across the full flight envelope, the corner points were chosen as references. From Figure 2.20 our performance variation is quite evident and quite dramatic across the full flight envelope. It indicates that our

maximum performance limits are bounded by:

$$\begin{aligned} 0.68 \text{ rad} \leq \phi_{max} \leq 1.5 \text{ rad} \\ 6.41 \text{ m} \leq R_{min} \leq 43.57 \text{ m} \\ 1.34 \text{ rad/s} \leq p_{max} \leq 5.16 \text{ rad/s} \\ 4.03 \text{ rad/s} \leq q_{max} \leq 15.53 \text{ rad/s} \\ 0.57 \text{ rad/s} \leq r_{max} \leq 4.62 \text{ rad/s} \end{aligned}$$

The general trend from all of these results is what one might expect from the basic physics. As our airspeed increases, we are able to generate much larger forces and moments due to the dynamic pressure, $\frac{1}{2}\rho V_a^2$, increasing quadratically with V_a. As such, we note that as our airspeed increases, our performance limits will generally increase.

For example, our maximum roll angle increases dramatically as airspeed increases because at very high airspeeds we can generate substantially more lift for the same angle of attack, α, or elevator deflection δ_e. Indeed, if we consider our zero altitude condition we will find that at our slowest airspeed, $V_a = 9.0$ m/s, we would generate forces and moments which are 11.3 times smaller than those generated at our maximum airspeed of $V_a = 29.6$ m/s, assuming we have the same inputs and only our dynamic pressure varies.

We also note that as our altitude increases, the dynamic pressure, $\frac{1}{2}\rho V_a^2$, will tend to decrease because $\rho \propto \frac{1}{h}$, where h is the altitude. This is visible from the flight envelope where we see that for an airspeed comparable to our zero-altitude maximum airspeed of $V_a = 29.6$ m/s, $V_a = 27.8$ m/s, R_{min} increases and ϕ_{max}, p_{max}, q_{max}, and r_{max} all decrease.

What this indicates is that one would likely not be able to get an optimally performing classical control system to work across this vast variation in plant dynamics. As such, the idea of extending classical control theory into the realms of gain-scheduling starts to look promising as we expect our system dynamics will vary as a function of the system state, so it intuitively would make sense to also vary our controllers as a function of those same states. However, maximum limits are not fully indicative of the full picture when it comes to the dynamic behaviour of the system as the environmental conditions change, with these observations only being fully realizable when one delves deeper into the linearisation of the system and the analysis thereof.

It should be noted at this stage in the dissertation that while Figure 2.20 is very useful in illustrating the changing dynamics of the system as a function of both altitude and airspeed, the reality is that small UAVs are legally not allowed to fly to their maximum altitudes. As such, the decision was made to constrain this project to a more physically realisable system in which we abide by the laws and regulations surrounding small UAVs

[39, 40]. In general, these state that the maximum altitude achieved during a flight must be such that:

$$h \leq 122 \text{ m}$$

As such, for the remainder of the text we will assume low altitude flight. This means that we can also make a constant air density, ρ, assumption because variation between ground level and our maximum altitude will be minute. As such, we make use of the ISA (International Standard Atmosphere) [41] standard day values such that:

$$\rho = 1.225 \frac{\text{kg}}{\text{m}^3}$$

2.7 Linearisation

The previous section dealt with establishing our performance characteristics. While this gives us a simple, algebraic way of determining the bounds within which we expect to operate, it is only a guideline and more advanced techniques are needed to draw any additional insight. It is for this reason that in this section we will consider the linearisation of our plant at several airspeeds between the previously identified maximum and minimum values. This will give us additional insight into the dynamics of the system and how we might go about controlling it. The linearisation of the systems will be done by making use of the state space notation which is widely used in the literature. This is stated as

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

where $\mathbf{x}$ is the state vector, $\mathbf{u}$ is the input vector, and $\mathbf{y}$ is the output vector.

We will first consider the effect of cross-coupling so that we can establish if we will be able to isolate our longitudinal and lateral dynamics and design two separate sets of controllers for each set of dynamics, or if we require a more complicated set of controllers the controls for both longitudinal and lateral dynamics simultaneously.

Finally, we will consider the longitudinal and lateral linearisation of the system and how this can be accomplished in the selected simulation environment. We will also take this opportunity to consider the variation in plant dynamics with some critical parameters which we will identify by observation of the equations of motion for the aircraft.

It should be noted that this section will primarily make use of the linearize() function within MATLAB [42] which takes in a Simulink model as an input and linearises this for the user based on some specified operating condition. Given that this section is more

focused on discussing results instead of attempting to create algebraic linearised models, this approach removes most of the time in performing these analyses, but none of the complexity as the analysis of the results remains the same as if one were to perform the linearisation manually.

Another important note is that the effects of stall are turned off during linearisation - as these are only needed during the verification step.

2.7.1 Effects of Cross-Coupling

Adopting the approach from [21], we know that

$$\mathbf{x}_{lon} \triangleq (u, w, q, \theta)^T \tag{2.68}$$

$$\mathbf{x}_{lat} \triangleq (v, p, r, \phi)^T \tag{2.69}$$

where $\mathbf{x}_{lon}$ is our longitudinal state vector, which should capture all of our longitudinal dynamics, and $\mathbf{x}_{lat}$ is our lateral state vector, which should capture all of our lateral dynamics. In order to review the impact of cross-coupling, we concatenate these vectors, such that:

$$\mathbf{x} = (u, w, q, \theta, v, p, r, \phi)^T$$

If we do this, all we need to review in order to determine the impact of cross-coupling is to look at the terms in the state matrix, A, from our state-space model and see if there are any terms in $A(5:8, 1:4)$, which would illustrate that our longitudinal states have an impact on the lateral states, or in $A(1:4, 5:8)$, which would illustrate that our lateral states have an impact on the longitudinal states. This is because we know, by definition, that the state matrix can also be defined as:

$$A = \frac{\delta f(\mathbf{x}^*, \mathbf{u}^*)}{\delta \mathbf{x}}$$

which comes from a general non-linear system of equations stated as

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$$

and $f(\mathbf{x}^*, \mathbf{u}^*)$ is $f(\mathbf{x}, u)$ at some trim condition defined by $(\mathbf{x}^*, \mathbf{u}^*)$.

In other words, in our case,

$$A = \begin{pmatrix} \frac{\delta \dot{u}}{\delta u} & \frac{\delta \dot{u}}{\delta w} & \frac{\delta \dot{u}}{\delta q} & \frac{\delta \dot{u}}{\delta \theta} & \frac{\delta \dot{u}}{\delta v} & \frac{\delta \dot{u}}{\delta p} & \frac{\delta \dot{u}}{\delta r} & \frac{\delta \dot{u}}{\delta \phi} \\ & & & \vdots & \vdots & & & \\ & & & \vdots & \vdots & & & \\ \frac{\delta \dot{\phi}}{\delta u} & \frac{\delta \dot{\phi}}{\delta w} & \frac{\delta \dot{\phi}}{\delta q} & \frac{\delta \dot{\phi}}{\delta \theta} & \frac{\delta \dot{\phi}}{\delta v} & \frac{\delta \dot{\phi}}{\delta p} & \frac{\delta \dot{\phi}}{\delta r} & \frac{\delta \dot{\phi}}{\delta \phi} \end{pmatrix}$$

which means that if $A(5:8, 1:4)$ and $A(1:4, 5:8)$ are all zeros, there is no cross-coupling between the longitudinal and lateral dynamics for the specific trim state considered because there is no impact on the longitudinal states by the lateral states, or vice versa.

In order to illustrate this for our case we will look at two cases, $V_{a_{min}}$ and $V_{a_{max}}$, as defined in Figure 2.20. Where it should be noted that these trim states are for a wings-level, constant airspeed and constant altitude condition. This results in:

$$A_{V_{a_{min}}} = \begin{pmatrix} -0.1643 & 0 & -1.6305 & -9.6109 & 0 & 0 & 0.0000 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.2578 & 0 & -2.0183 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1.0000 & 0 & 0 & 0 & -0.0000 & -0.0000 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -0.0000 & 0 & -0.0000 & 0 & 0 & -16.3774 & -4.1251 & 0 \\ -0.0000 & 0 & -0.0000 & 0 & 0 & -17.6275 & -5.6699 & 0 \\ 0 & 0 & 0.0000 & 0.0000 & 0 & 1.0000 & 0.2046 & -0.0000 \end{pmatrix}$$

$$A_{V_{a_{max}}} = \begin{pmatrix} -0.7224 & 0 & 0.3147 & -9.8093 & 0 & 0 & -0.0000 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -0.0422 & 0 & -6.6462 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1.0000 & 0 & 0 & 0 & -0.0000 & 0.0000 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0.0000 & 0 & 0.0000 & 0 & 0 & -53.9311 & -13.5842 & 0 \\ 0.0000 & 0 & 0.0000 & 0 & 0 & -58.0478 & -18.6712 & 0 \\ 0 & 0 & -0.0000 & -0.0000 & 0 & 1.0000 & -0.0117 & -0.0000 \end{pmatrix}$$

It should be noted that the elements that are not precisely 0 simply indicate that there is some excitation that occurs at these points, but the value is negligible up to the fourth decimal place and so can be approximated as zero.

A very simple check on these matrices is to look at the elements $A(4,3)$ and $A(8,6)$. We know from the Jacobian approximation stated earlier that these elements are $A(4,3) = \frac{\delta \dot{\theta}}{\delta q}$

and $A(8,6) = \frac{\delta\dot{\phi}}{\delta p}$. Given that we are in a wings-level trim state with constant altitude and airspeed, we expect that $\dot{\theta} = q$ and $\dot{\phi} = p$. This is exactly what these elements indicate, as $\frac{\delta\dot{\theta}}{\delta q} = \frac{\delta\dot{\phi}}{\delta p} = 1$, which is a good sanity check that our linearisation results line up with an expected condition.

This result indicate that, at least for these constant altitude, constant airspeed trim conditions, we have no cross-coupling between dynamics and so one can immediately make the decision that the autopilot can be split into longitudinal and lateral controllers. These can then be designed and implemented separately. This is a critical decision in the design process, and also means that performing linearisation in the longitudinal and lateral planes will serve us well, as these linearised models can subsequently be used to design controllers.

2.7.2 Longitudinal Linearisation

From Figure 2.20 we know that our performance varies significantly across the flight envelope. Given that we've now restricted this to low altitude flight, the main observation we can draw is that our performance varies substantially with V_a. This is not surprising, as if we consider Equations (2.49) and (2.50), there is a V_a^2 term in both the forces and moments equations, which means that one would expect performance to vary significantly with V_a.

In order to review the longitudinal linearisation, let us make the assumption that we want to look at variations in plant dynamics within some predefined bounds. We know that V_a is a good candidate for this, so we will use this parameter as our trim condition and linearise around various values of V_a in order to do so. Forcing $\alpha = 0$, we calculate a trim condition, which results in a non-zero pitch rate, q. We then linearise around these points for $V_a = \text{linspace}(1.1V_{a_{min}}, 0.9V_{a_{max}}, 10)$ where linspace(first, last, elements) [43] is a MATLAB function that creates a vector with equal spacing starting at our first value, going to our second value and with the number of elements as specified by the third. It should be noted that we scale $V_{a_{min}}$ upward by 10% so that we aren't right on the limit of what is possible for trim, and do the same downward for $V_{a_{max}}$.

We then make use of a system model which isolates a reduced subset of our full plant dynamics. This is done in order to isolate the longitudinal states (we can do this based on our previous observation of no cross-coupling) of interest. This is depicted in Figure 2.21.

Figure 2.21: Skywalker X8 longitudinal plant model

As an example, our trim condition for $V_a = 1.1(V_{a_{min}})$, $\alpha = 0$ is as follows:

$$\mathbf{x}^* = \begin{pmatrix} p_n \\ p_e \\ p_d \\ u \\ v \\ w \\ \phi \\ \theta \\ \\ p \\ q \\ r \\ \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 9.89 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ -0.865 \\ 0 \\ 0 \\ 0 \end{pmatrix} \qquad \mathbf{u}^* = \begin{pmatrix} \delta_t \\ \delta_e \end{pmatrix} = \begin{pmatrix} 0.287 \\ 0.246 \end{pmatrix}$$

and for $V_a = 0.9(V_{a_{max}})$, $\alpha = 0$, it is:

$$\mathbf{x}^* = \begin{pmatrix} p_n \\ p_e \\ p_d \\ u \\ v \\ w \\ \phi \\ \theta \\ \\ p \\ q \\ r \\ \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 26.6 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0.101 \\ 0 \\ 0 \\ 0 \end{pmatrix} \qquad \mathbf{u}^* = \begin{pmatrix} \delta_t \\ \delta_e \end{pmatrix} = \begin{pmatrix} 0.762 \\ 0.142 \end{pmatrix}$$

With all intermediate trim values being calculated in the same manner. To consider the variation in plant dynamics we make use of the I/O relationship between δ_e, and q, θ, h. In other words we consider the transfer functions $\frac{q}{\delta_e}(s)$, $\frac{\theta}{\delta_e}(s)$, and $\frac{h}{\delta_e}(s)$. These are depicted in Figures 2.22, 2.23 and 2.24, respectively.

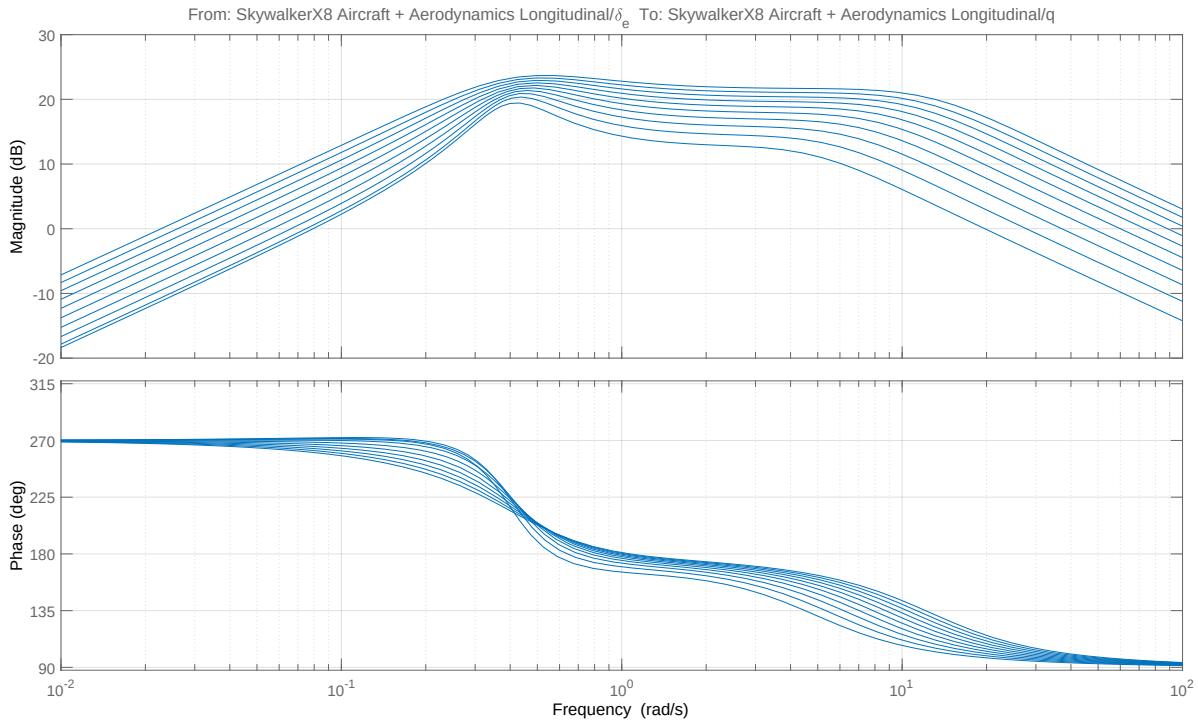


Figure 2.22: Bode plot of $\frac{q}{\delta_e}(s)$, with $V_a = \text{linspace}(9.89, 26.64, 10)$ m/s, $\alpha = 0$ rad

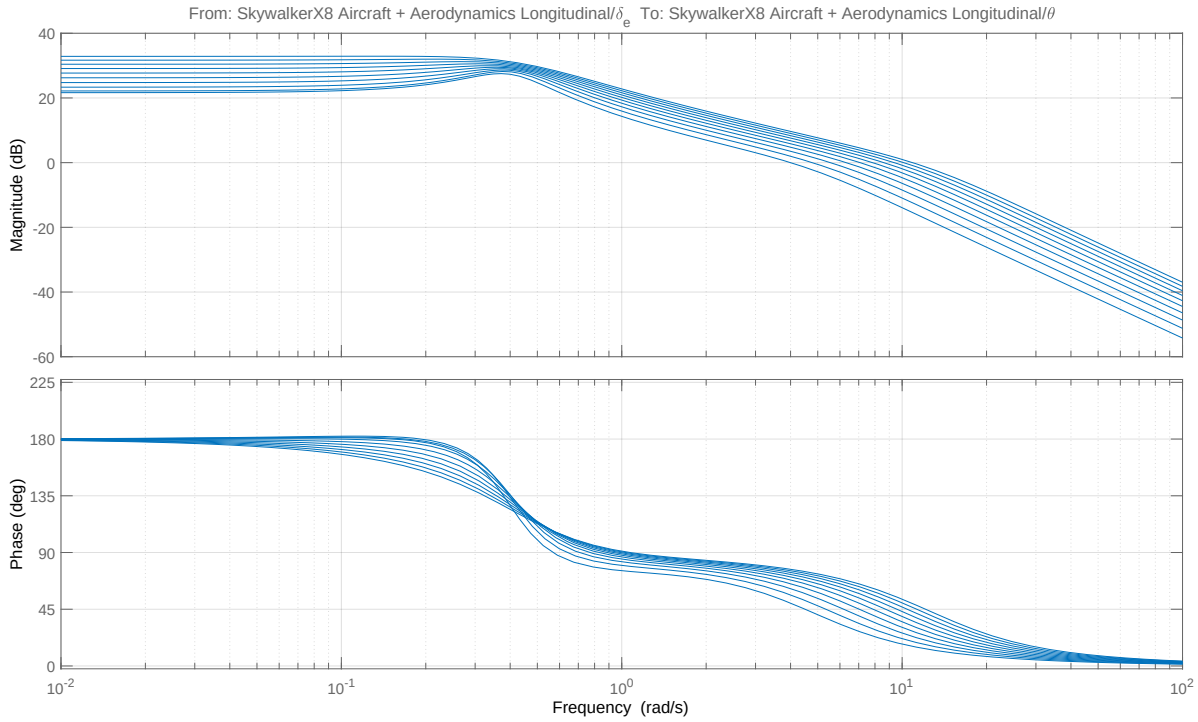


Figure 2.23: Bode plot of $\frac{\theta}{\delta_e}(s)$, with $V_a = \text{linspace}(9.89, 26.64, 10)$ m/s, $\alpha = 0$ rad

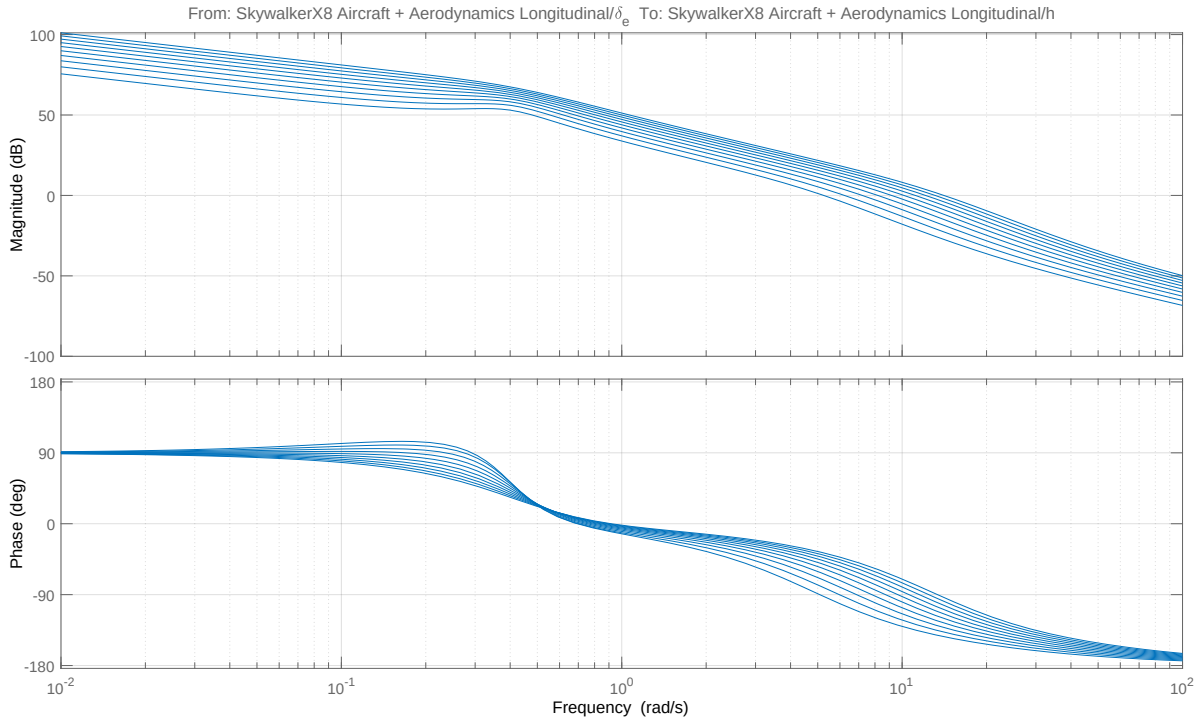


Figure 2.24: Bode plot of $\frac{h}{\delta_e}(s)$, with $V_a = \text{linspace}(9.89, 26.64, 10)$ m/s, $\alpha = 0$ rad

From these plots we can note some key behaviours. Firstly, everything is offset by 180° - this is expected because we know that $+\delta_e$ results in a negative pitching moment. So if we want to view this from the perspective of a pitch up manoeuvre we can simply subtract 180° from all of the plots - which would be the response for a negative actuator input. However, we can also note some key insights as follows.

We immediately can see that $\frac{q}{\delta_e}(s)$ has a dominant low frequency zero, and a dominant high frequency pole. This is because in the lower frequency domain we ascend with a gradient of approximately 20 dB/decade, which indicates a single zero. In the higher frequency domain we note that we start to descend with a constant gradient of approximately -20 dB/decade, which indicates a single zero. As such, our system is likely of the form:

$$\frac{q}{\delta_e}(s) = K_1 \frac{(s+a)Z(s)}{(s+b)(s+c)P(s)}$$

Which is a useful observation as if we want to reduce this at a later stage, a second order system will likely be adequate to get a reasonable fit.

If we move on to $\frac{\theta}{\delta_e}(s)$ we note that it seems as though it is simply the same transfer function as $\frac{q}{\delta_e}(s)$, with an added integrator (or low frequency pole) as the integrator will have the effect of adding 20 dB/decade in the lower frequencies and also providing a constant phase offset of $-90°$, which seems to be the exact difference between the two systems. This, however, should not be surprising as $q \approx \dot{\theta}$, which implies that $\frac{\theta}{q}(s) = \frac{1}{s}$, which is precisely what we observe. As such, we expect our system to be of the form:

$$\frac{\theta}{\delta_e}(s) = K_1 \frac{(s+a)Z(s)}{s(s+b)(s+c)P(s)}$$

Where we note that we do not expect there to be a change in the gain, and indeed we see that this is the case by observation on the bode plot.

Finally, we can consider $\frac{h}{\delta_e}(s)$. Since we have forced $\alpha = 0$, we know that $\theta = \gamma$. We are also in a windless environment with no roll, so we know that $V_a = V_g$ and that $V_g \sin\gamma = V_a \sin\theta = -\dot{p}_d = \dot{h}$. Making a small angle approximation for θ means that we can state

$$V_a \theta = \dot{h}$$

or, in the form of a transfer function,

$$\frac{h}{\theta}(s) = \frac{V_a}{s}$$

So we expect that the difference between $\frac{\theta}{\delta_e}(s)$ and $\frac{h}{\delta_e}(s)$ will be a scaled integrator with the scaling factor being V_a. If we consider a simple example of this we know that our minimum velocity is $V_a \approx 10$ m/s, as such, we know from first principles that an integrator scaled by this amount should provide a gain at 10^{-2} rad/s of 60 dB. If we consider our plot, this puts us approximately in the same region where we want to be so we know that our estimate holds. As such, we can state that

$$\frac{h}{\delta_e}(s) = K_1 \frac{V_a(s+a)Z(s)}{s^2(s+b)(s+c)P(s)}$$

This aligns with the theory presented in [15] and will prove to be very valuable moving forward.

However, we expect that there will be significant variations in the dynamics of the system based parameters other than just V_a. V_a will likely have the largest impact, given that it is the only squared term, but we expect that other trim states will have non-negligible impacts. For the longitudinal dynamics we expect that this will take the form of either α or θ - which is easily observable in Equations (2.49) and (2.50). However, we know that in practice θ is not directly measurable, and we can also see that it is only α that has an impact in both the forces and the moments generated, whereas θ only has an impact in the forces. α is also an important parameter for small aircraft as it is highly dependant on wind, and we know that the wind velocity for a small aircraft can be in excess of 20-50% of its total airspeed, so utilizing α also assists in taking these effects into account. As such, we consider α as our additional linearisation parameter.

With this in mind we perform a similar methodology to that used in the linearisation at particular values of V_a, as was presented previously, except now we extend this to include the effects of α; where $\alpha = \text{linspace}(0, \alpha_0, 6)$. It should be noted that from our definition of the α trim states, we are assuming that α is symmetric about zero in terms of the dynamics - this assumption should be fairly accurate based on the impact of α on the forces and moments.

As such, we calculate trim conditions for each combination of (V_a, α), resulting in 60 linearised plant models. In order to review whether α does indeed have a large impact on the plant dynamics, we consider the same figures as before, although now with an additional 50 plots. These are depicted in Figures 2.25, 2.26 and 2.27.

From these figures we see that our low and high frequency dynamics remain approximately consistent with our previous discussion, but resonance peaks now exist in our mid-range frequencies that are non-negligible, and we have higher order models generated from the linearisation algorithm to account for this because our simplified models discussed previously start to behave quite differently as α varies. As such, α, clearly has a large impact on overall longitudinal aircraft dynamics, which is not entirely un-

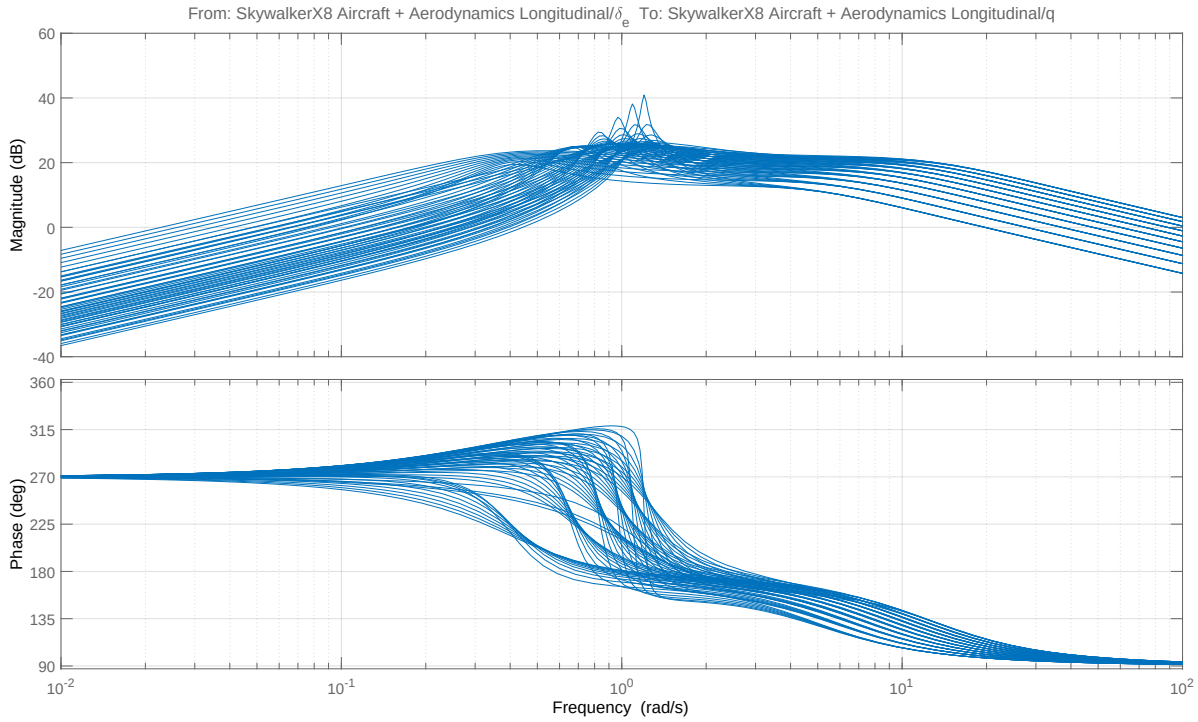


Figure 2.25: Bode plot of $\frac{q}{\delta_e}(s)$, with varying (V_a, α)

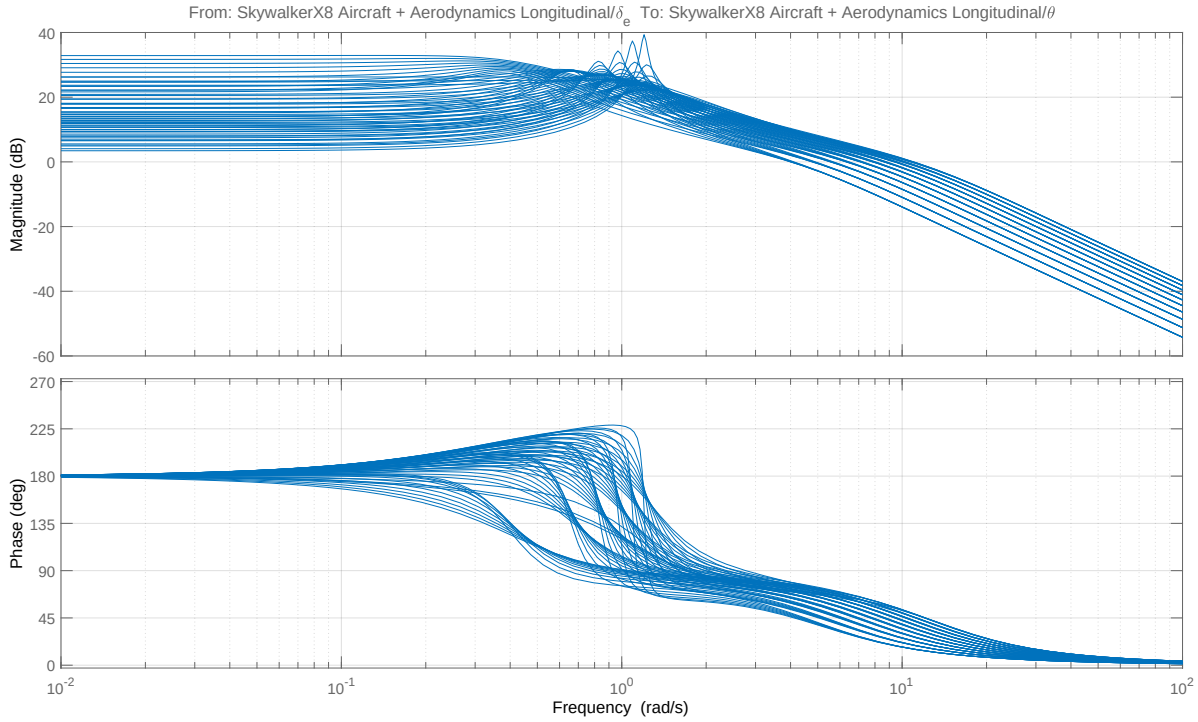


Figure 2.26: Bode plot of $\frac{\theta}{\delta_e}(s)$, with varying (V_a, α)

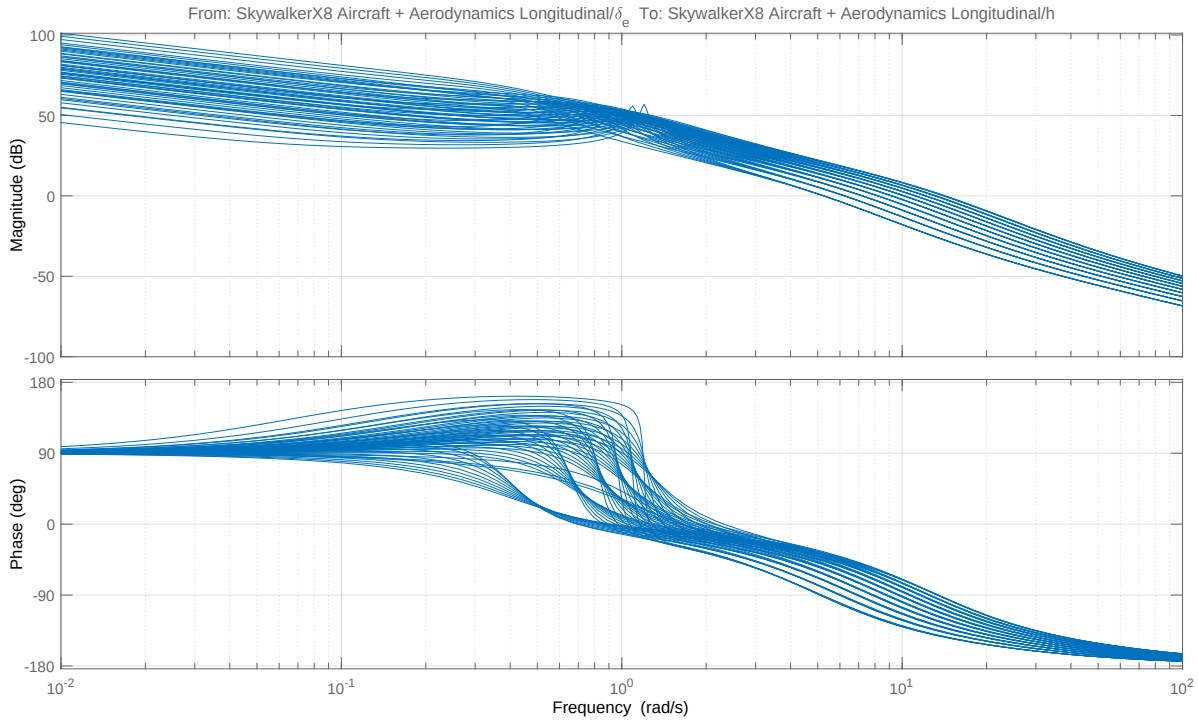


Figure 2.27: Bode plot of $\frac{h}{\delta_e}(s)$, with varying (V_a, α)

expected. This is a critical observation as it makes (V_a, α) an ideal candidate for gain scheduling, which will be discussed in the development of the controllers for the system.

The remaining I/O relationship that we need to consider the linearisation of is $\frac{V_a}{\delta_t}(s)$. This will allow us to design a controller to control our airspeed, V_a, with our propeller speed, δ_t. This is determined in much the same way as the previous controllers, but we adjust our 6DoF model to force θ to remain constant. This is provided for in the code repository given in Section 1.4 with the script ms_sixdof_thetaConst.m. This is required due to how the Simulink linearisation algorithm perturbs the model to get the transfer function, and a perturbation in δ_t results in a pitching moment upward and then a drop downward. Given that we will have a controller to regulate pitch angle, θ, in the final system, forcing this constant in the model is entirely valid. Given that we expect $\frac{V_a}{\delta_t}(s)$ to not be impacted by α, by simple observation of Equations (2.49) and (2.50), we linearise simply as a function of V_a. This is depicted in Figure 2.28.
From this figure we note that we get the expected linearisation of a simple lag. This is expected as δ_t causes a direct change in f_x, which then causes an acceleration in u which subsequently results in an increased V_a over time. As V_a increases, the thrust provided for a constant δ_t reduces and our drag increases such that we eventually get to a steady state value. Because of this, we expect a simple gain at low frequencies, and a lag at higher frequencies, which is precisely what is seen.

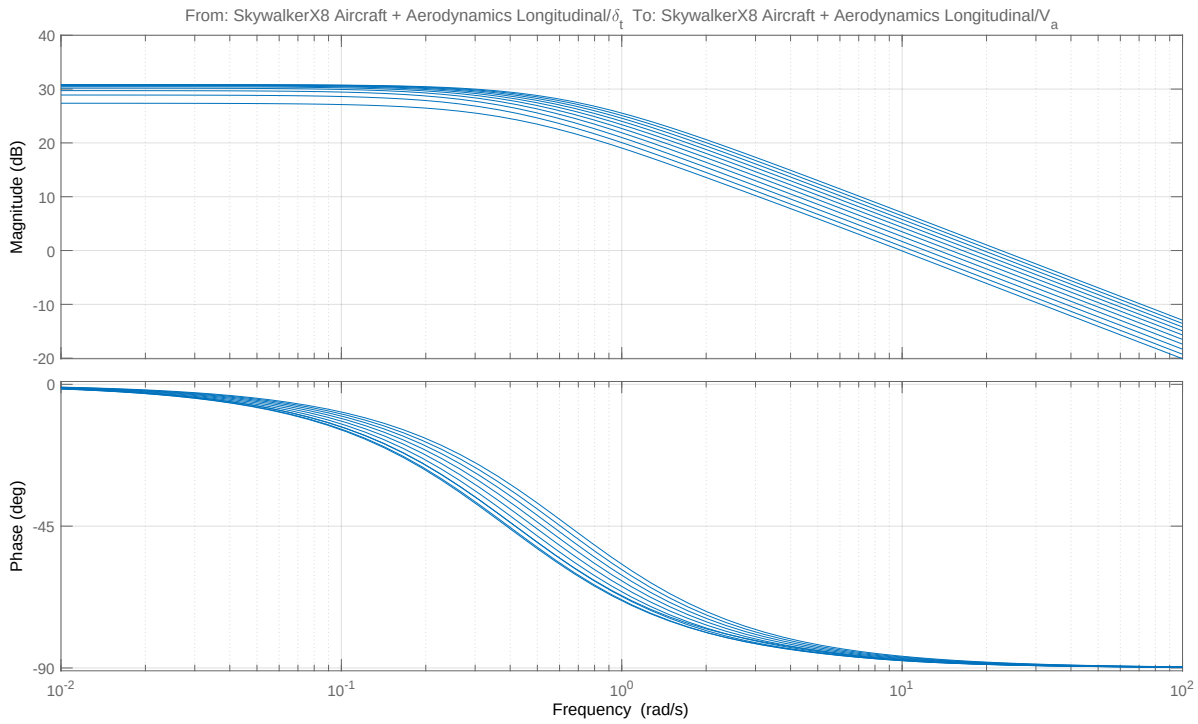


Figure 2.28: Bode plot of $\frac{V_a}{\delta_t}(s)$, with varying V_a

2.7.3 Lateral Linearisation

Similarly to the longitudinal linearisation, we expect that our lateral dynamics will vary significantly with V_a, as it not only has a quadratic relationship with the forces and moments in the longitudinal, but also the lateral plane. We also note that from Figure 2.20 we see a significant impact on ϕ_{max}, R_{min}, p, and r, which are all lateral performance parameters.

Considering our forces and moments equations (Equations (2.49) and (2.50)), we note that unlike the longitudinal forces and moments, α plays no role. The analogue in the lateral plane is β, but since this airframe does not have a rudder, we have no direct control over this parameter and as such we will not consider its impact on the system dynamics. Subsequently, the only parameter we will consider for variations in the lateral dynamics of the system will be V_a.

For the linearisation of the system in the lateral plane we make use of a system model which isolates a reduced subset of our full plant dynamics. This is done in order to isolate the lateral states (we can do this based on our previous observation of no cross-coupling) of interest. This lateral dynamic model is illustrated in Figure 2.29.

The lateral system is trimmed slightly differently to the longitudinal. Since we are only interested in V_a, we do not create a specification for α. We also specify that our angular

Figure 2.29: Skywalker X8 lateral plant model

rates should be held at zero. We then consider the same two trim conditions as in the longitudinal dynamics section and see that our trim condition for $V_a = 1.1(V_{a_{min}})$ is as follows:

$$\mathbf{x}^* = \begin{pmatrix} p_n \\ p_e \\ p_d \\ u \\ v \\ w \\ \phi \\ \theta \\ \\ p \\ q \\ r \\ \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 9.81 \\ 0 \\ 1.21 \\ 0 \\ 0.123 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0.123 \\ 0 \end{pmatrix} \qquad \mathbf{u}^* = \begin{pmatrix} \delta_t \\ \delta_e \end{pmatrix} = \begin{pmatrix} 0.344 \\ 0.524 \end{pmatrix}$$

and for $V_a = 0.9(V_{a_{max}})$, it is:

$$\mathbf{x}^* = \begin{pmatrix} p_n \\ p_e \\ p_d \\ u \\ v \\ w \\ \phi \\ \theta \\ \\ p \\ q \\ r \\ \alpha \\ \beta \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 26.6 \\ 0 \\ 0.0701 \\ 0 \\ 0.00263 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0.00263 \\ 0 \end{pmatrix} \qquad \mathbf{u}^* = \begin{pmatrix} \delta_t \\ \delta_e \end{pmatrix} = \begin{pmatrix} 0.758 \\ 0.014 \end{pmatrix}$$

Where we can see these differ slightly from those stated in the longitudinal dynamics trim conditions because we allow a non-zero α and constrain the system to not have any angular rates (ie. $p = q = r = 0$).

The characteristics we are interested for our lateral dynamics require us to consider the I/O relationship between δ_a and p, ϕ, and χ. In other words we are interested in the transfer functions $\frac{p}{\delta_a}(s)$, $\frac{\phi}{\delta_a}(s)$, and $\frac{\chi}{\delta_a}(s)$. These are depicted in Figures 2.30, 2.31 and 2.32.

From these figures we firstly note that there is significant deviation in system dynamics as a function of V_a, which illustrates that it is a good selection for a scheduling parameter and does indeed contribute significantly to the variation in the lateral dynamics. We also observe similar trends to those seen in the longitudinal dynamics, at least in the high and low frequency regions. Where it seems, as an approximation, that $\frac{\phi}{p}(s) = \frac{1}{s}$, and that $\frac{\chi}{\phi}(s)$ is some scaled integrator. The scaling factor can be determined from Equation (2.64) as $\frac{g}{V_g}$, giving us

$$\frac{\chi}{\phi}(s) = \frac{g/V_a}{s}$$

where we have made the $V_g \approx V_a$ approximation.

Looking at $V_{a_{min}} \approx 10$ m/s, we know that

$$\frac{\chi}{\phi}(s) \approx \frac{1}{s}$$

which means that we expect a gain of 40 dB at 10^{-2} rad/s. Given that our $\frac{\chi}{\delta_a}(s)$ plot indicates a gain of 43-52 dB at 10^{-2} rad/s, with $\frac{\phi}{\delta_a}(s)$ at 12-28 dB at the same frequency our previous calculation holds true as it puts us in approximately the same range.

It should be noted that these frequency responses have much higher order dynamics than the longitudinal plots, which is clear from the resonance peaks and fairly sharp transitions in phase. These resonance peaks occur at around 2 rad/s in $\frac{p}{\delta_a}(s)$ and $\frac{\phi}{\delta_a}(s)$, and around the 0.5 rad/s mark for $\frac{\chi}{\delta_a}(s)$. This is a strong indication that we will have to constrain our controllers to roll off before these dynamics are excited, or else we risk encountering instability or difficult to control dynamics. As such, we will likely require slower lateral controllers than our longitudinal controllers. This is not surprising, because we noted very early on in our observations around Table 2.2 that we expected slower lateral dynamics than longitudinal.

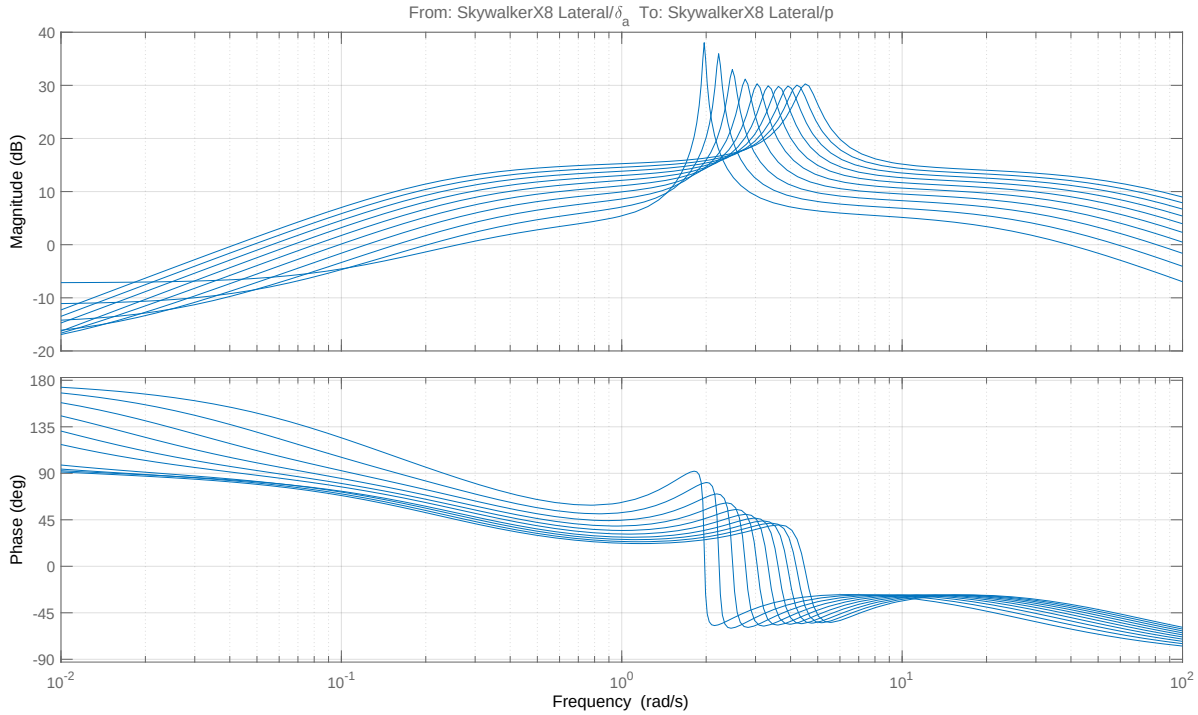


Figure 2.30: Bode plot of $\frac{p}{\delta_a}(s)$, with varying V_a

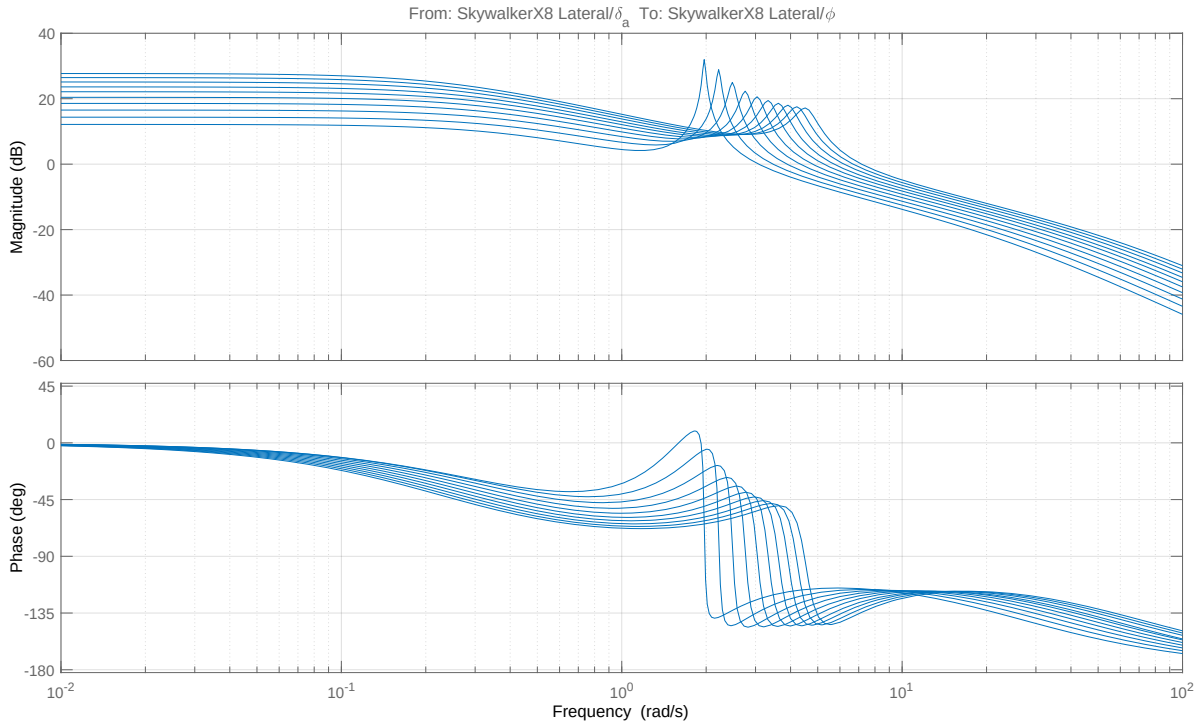


Figure 2.31: Bode plot of $\frac{\phi}{\delta_a}(s)$, with varying V_a

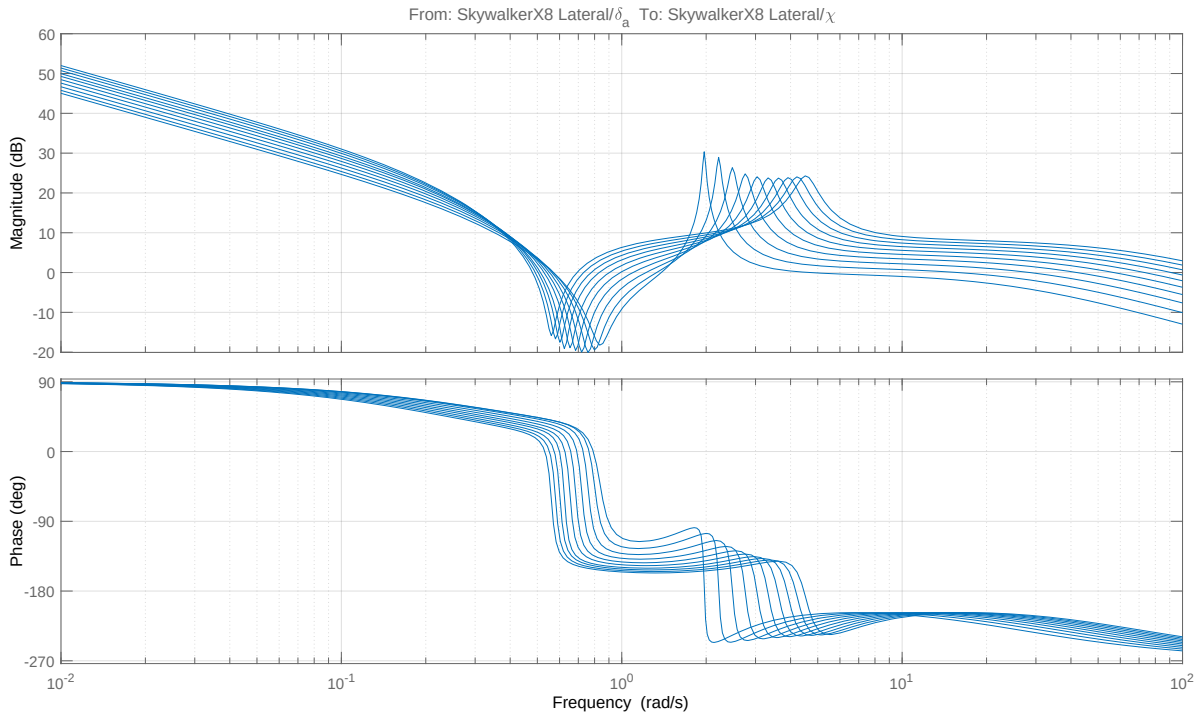


Figure 2.32: Bode plot of $\frac{\chi}{\delta_a}(s)$, with varying V_a

Sensors and Actuators

With the system dynamics and performance having now been quantified and discussed, we can move on to the development of how our digital control system can interact with the aircraft. In order to determine the state of our aircraft, we will need access to data from the sensors that are used to measure this state. In order to adjust this state, we will need actuators. These elements are both depicted as the primary interfaces with the aircraft in Figure 1.2.

This chapter will deal with how these sensors and actuators are modelled in our analyses, with the inclusion of non-linear effects like noise on the sensors and rate limiting on the actuators. This will allow us to develop a more realistic control system that can be shown to be able to handle a non-ideal environment, as opposed to just assuming we have full state feedback with perfect measurements and actuators that can respond instantaneously to our commands.

3.1 Sensors

In reality, one does not have perfect measurements for all of the states of the aircraft. As an example, there is no method to directly measure the current angular position of the aircraft, (ϕ, θ, ψ), with no sensor technology able to do this explicitly. As such, these states are not measured directly, but are estimated by making use of sensor measurements which one does have access to, namely rate gyroscopes. Rate gyroscopes do not measure angular position, but angular velocity, (p, q, r), and these values are then used to estimate the current angular position. A simple way to do this would be to simply integrate these signals - but due to noise and drift in our sensor measurements, this would lead to significant errors over time and our control system subsequently acting on incorrect assumptions about the current state of the aircraft. As such, more advanced techniques are needed to perform these estimates - these will be discussed in Chapter 5.

The sensor models that will be used in the simulations to follow in this dissertation need to match the expected performance of these sensors in a realistic environment. They also need to exist as sensors that are easily procurable for a physical system. As such, this section will be broken down into firstly establishing how noise and error modelling has been applied to our various sensor types. After which we will define the sensor models for each of the expected sensors to be used. These are listed below, along with the states which they will be used to sense.

1. Global Positioning System (GPS): $(V_g, \chi, p_n, p_e, p_d)$
2. Inertial Measurement Unit (IMU): $(p, q, r, a_x, a_y, a_z, \psi)$
3. Airspeed Sensing: V_a
4. Altitude Sensing: $h = -p_d$
5. Angle of Attack Sensing: α

The sensors discussed are readily available in the market, and an example for each sensor is provided which will be used to quantify the performance of each of the models. How this data will be used in our autopilot will be discussed in Chapter 5, which is depicted in the system architecture shown in Figure 1.2.

3.1.1 Noise and Error Modelling

In order to more accurately model our sensors, we need to ensure that we model the effects of sensor noise and other errors that might find their way into the measurement of the particular system state. An exhaustive treatment of measurement error and noise can be found in [44], where a gyroscope model is provided as an example in Figure 3.1. Advanced noise and error modelling falls outside of the scope of this text, with the main limitation on not generating more exhaustive noise models being that a lot of the required parameters are not available on a sensor datasheet and would need to be determined experimentally on a sensor-by-sensor basis. This project aims to provide a more general approach that does not have any extremely strong dependencies on very particular hardware, so the noise and error models utilised are reduced. As such, we make some assumptions - these are stated as:

1. Accelerometers and gyroscopes are perfectly aligned with their measurement axis (ie. there is no axis misalignment)
2. All bias is removed before the flight is started (ie. there is no constant offset)
3. Gyroscopes are not impacted by acceleration
4. Only GPS includes the effects of random walk drift
5. Temperature effects are negligible

With this in mind, aside from our GPS model, the remaining sensor models will include noise which is modelled heuristically with a zero-mean, independent, continuous-time Gaussian white noise process, $\eta(t)$, of strength σ, which is modelled in discrete time as:

$$\eta[k] = \sigma w[k]$$

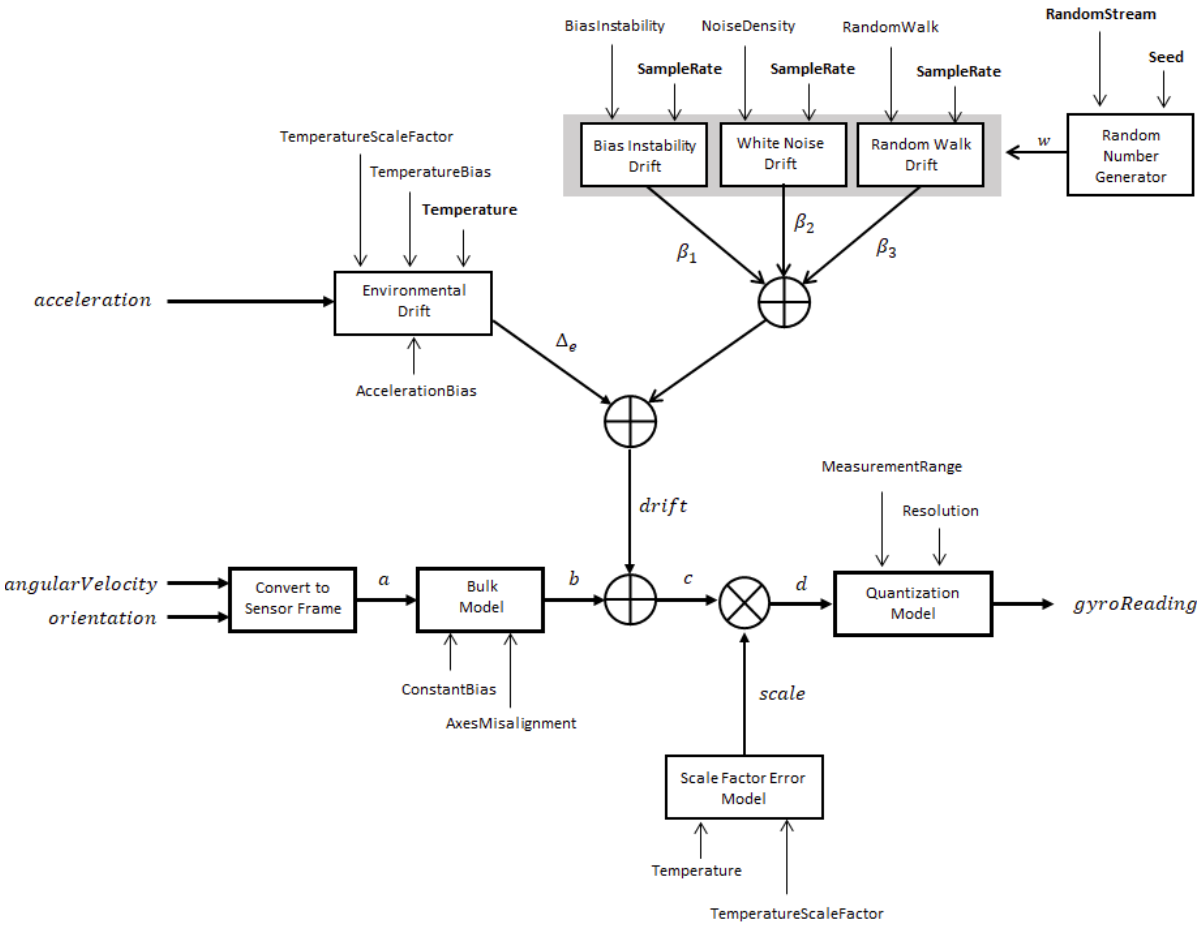


Figure 3.1: Gyroscope noise and error modelling [44]

where $w[k]$ is a zero-mean Gaussian white noise process with a variance of 1, and σ is generally provided on a sensor datasheet as either an RMS term, or can be calculated as as a percentage of the overall sensor accuracy.

However, for gyroscopes and accelerometers, σ is not stated directly and one is provided with a noise density term, σ_d, which is given with the units $x/\sqrt{Hz}$, where x are the units of the measurand. In other words, accelerometers and gyroscopes provide one with a way to quantify the variation in noise strength as a function of the sampling period used on the measurement. As such, we need to scale this value by the sensor sample rate before it can be used to scale the zero-mean, independent, continuous-time Gaussian white noise process. This is done as:

$$\sigma = \sigma_d \frac{1}{\sqrt{T_s}} \tag{3.1}$$

Where σ_d is the noise density term, and T_s is the sample time of the sensor.

Finally, we know that in a digital system sensor values are either communicated digitally or sampled by an analogue to digital converter (ADC). In either case, the measured value is constrained by the quantization process as we only have access to a specific number of digital bits to represent a continuous signal. This is an additional source of error that is quite easily included in the sensor models making use of the specifications provided on the sensor datasheet, and is implemented as follows:

$$y = R\text{floor}\left(\frac{u}{R}\right)$$

where y is the output of the quantization process, u the input, floor() is a function that simply rounds down, and R is the resolution calculated as

$$R = \frac{\text{span}}{(2^{\text{ADC Bits}} - 1)}$$

with span being the physical span which the sensor can measure, and ADC Bits being how many bits of resolution the system has available to represent the measurement.

It should also be noted that we assume in all cases that sensor noise is applied prior to the sensor dynamics, where this assumption can be amended on a sensor-by-sensor basis as it depends on how the sensor was characterised to determine the data presented on the datasheet, but should not impact our models too significantly. Having covered all of the error sources made use of, we present the standard noise and error model utilized in this text, which is illustrated by means of a generic sensor model in Figure 3.2.

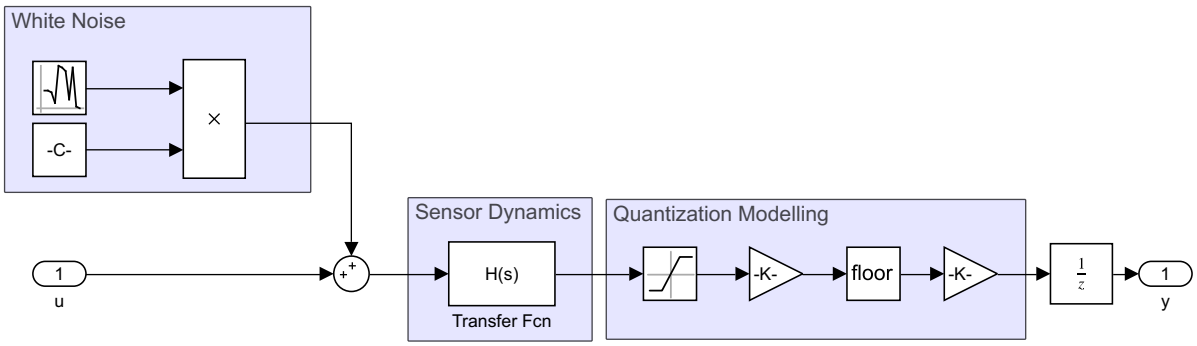


Figure 3.2: Generic sensor model indicating error and noise modelling utilized

It should be noted that a unit delay is added just prior to the output of the model. The reason for this is to account for any delays which might be introduced in the processing of the measurement by the sensor, subsequent signal conditioning and digitisation. This is assumed to occur at a worst case delay of the sampling time of the sensor.

3.1.2 GPS

Our GPS model is based on the work presented in [45] and is split between an inertial position measurement; as well as both velocity and course angle measurements.

Position Measurement

Our position measurement is modelled as

$$y_{GPS_N}[n] = p_n[n] + v_n[n] \tag{3.2}$$
$$y_{GPS_E}[n] = p_e[n] + v_e[n] \tag{3.3}$$
$$y_{GPS_h}[n] = -p_d[n] + v_h[n] \tag{3.4}$$

Where we model the error, v, as a Gauss-Markov process as follows,

$$v[n+1] = e^{-k_{GPS}T_s}v[n] + \eta_{GPS}[n]$$

where η_{GPS} is a zero-mean Gaussian white noise signal, of variance σ^2_{GPS}; $1/k_{GPS}$ is the time constant of the process; and T_s is the sample time. To quantify our model, we use the parameters presented in [21] as follows:

Table 3.1: GPS Gauss-Markov Error Model Parameters

Direction	σ_{GPS} **(m)**	$\mathbf{1}/k_{GPS}$ **(s)**	T_s **(s)**
North	0.21	1100	0.2
East	0.21	1100	0.2
Altitude	0.4	1100	0.2

where the sample time used is the maximum available on the VN-200 GPS-Aided INS Unit [46].

Velocity and Course Angle

Most modern GPS receivers provide velocity information as part of the received data packet. They provide this information in the form of horizontal ground speed and course angle over the ground. We can calculate these parameters from the north and east velocity components from the GPS as:

$$V_g = \sqrt{V_N^2 + V_E^2} \tag{3.5}$$
$$\chi = \tan^{-1}\left(\frac{V_N}{V_E}\right) \tag{3.6}$$

where $V_N = V_a \cos\psi + w_n$, and $V_E = V_a \sin\psi + w_e$.

Making use of the basic principles of uncertainty analysis presented in [47], we estimate the uncertainty in the ground speed and course angle measurements as

$$\sigma_{V_g} = \sqrt{\frac{V_N^2\sigma_{V_N}^2 + V_E^2\sigma_{V_E}^2}{V_N^2 + V_E^2}}$$
$$\sigma_\chi = \sqrt{\frac{V_N^2\sigma_{V_E}^2 + V_E^2\sigma_{V_N}^2}{\left(V_N^2 + V_E^2\right)^2}}$$

Which, if the uncertainty in the North and East directions have the same magnitude, or $\sigma_{V_N} = \sigma_{V_E} = \sigma_V$, then simplifies our expressions to

$$\sigma_{V_g} = \sigma_V$$
$$\sigma_\chi = \frac{\sigma_V}{V_g}$$

This is an important observation to note, as we see that our uncertainty in course angle, χ, scales with the inverse of the ground speed. What this means is that for high speeds our uncertainty is low, whereas for low speeds the error is large. This is not surprising as at zero speed, we know that the course angle is undefined so we expect this sort of relationship.

Finally, we can model our ground speed and course angle measurements from the GPS as follows:

$$y_{GPS_{V_g}} = \sqrt{\left(V_a \cos\psi + w_n\right)^2 + \left(V_a \sin\psi + w_e\right)^2} + \eta_V \tag{3.7}$$
$$y_{GPS_\chi} = \text{atan2}\left(V_a \sin\psi + w_e, V_a \cos\psi + w_n\right) + \eta_\chi \tag{3.8}$$

Where η_V and η_χ are zero-mean Gaussian processes which can be modelled as:

$$\eta[k] = \sigma w[k]$$

as discussed previously.

For this text we quantify the model by making use of

$$\sigma_V = 0.0125 \text{ m/s}$$

which uses the error band stated for the VectorNav VN200 of 0.05 m/s, which we treat as 4-σ [46]. This aligns with the range specified in [21].

Finally, this is all implemented in Simulink as depicted in Figure A.4, which is then reduced to a single masked subsystem as depicted in Figure A.5.

3.1.3 9-Axis IMU

A so-called 9-Axis IMU is a very common device used in most inertial navigation systems. The nine axes are:

1. 3-Axis Accelerometer
2. 3-Axis Gyroscope
3. 3-Axis Magnetometer

Each of these items will be discussed to follow.

Accelerometers

The accelerometers in the IMU are assumed to be aligned perfectly with the body axes of the aircraft and mounted at the center of mass. These accelerometers measure the difference between the acceleration of the aircraft and the gravitational acceleration. As such, we can model these systems as

$$y_{accel_x} = \frac{f_x}{\mathsf{m}} + g\sin\theta + \eta_{accel_x} \tag{3.9}$$

$$y_{accel_y} = \frac{f_y}{\mathsf{m}} - g\cos\theta\sin\phi + \eta_{accel_y} \tag{3.10}$$

$$y_{accel_z} = \frac{f_z}{\mathsf{m}} - g\cos\theta\cos\phi + \eta_{accel_z} \tag{3.11}$$

where η_{accel_x}, η_{accel_y}, and η_{accel_z} are zero-mean Gaussian processes with variances of $\sigma^2_{accel_x}$, $\sigma^2_{accel_y}$, and $\sigma^2_{accel_z}$, respectively. These values are converted to a discretised value by a digital computation device and are assumed to be sampled at a synchronous sample rate defined as T_s.

This is implemented in Simulink as is depicted in Figure A.6, which is then reduced to a masked subsystem as in Figure A.7.

Gyroscopes

The sensitive axes of the rate gyroscopes are aligned, by design, with the body axes of the aircraft, $\mathbf{i^b}$, $\mathbf{j^b}$, and $\mathbf{k^b}$. These then measure the angular rates of the aircraft about

these axes, p, q, and r, directly. As such, we can model the outputs of these sensors as

$$y_{gyro_x} = p + \eta_{gyro_x} \tag{3.12}$$
$$y_{gyro_y} = q + \eta_{gyro_y} \tag{3.13}$$
$$y_{gyro_z} = r + \eta_{gyro_z} \tag{3.14}$$

Where η_{gyro_x}, η_{gyro_y}, and η_{gyro_z} are zero-mean Gaussian processes with variances, $\sigma^2_{gyro_x}$, $\sigma^2_{gyro_x}$, and $\sigma^2_{gyro_x}$, respectively. These values are converted to a discretised value by a digital computation device and are assumed to be sampled at a synchronous sample rate defined as T_s.

This is implemented in Simulink as is depicted in Figure A.8, which is then reduced to a masked subsystem as in Figure A.9.

Magnetometers

The 3-axis magnetometer can be modelled as a simple digital compass which provides us with a measurement of our yaw (or heading) angle, ψ. This can be modelled as,

$$y_\psi = \psi + \eta_\psi \tag{3.15}$$

Where η_ψ is a zero-mean Gaussian processes with variance, σ^2_ψ. This value is converted to a discretised value by a digital computation device and is assumed to be sampled at a synchronous sample rate defined as T_s.

9-Axis IMU Implementation

The Accelerometer, Gyroscope and Magnetometer models are all grouped together to create a full 9-Axis IMU model which is masked as a subsystem and implemented in Simulink as per Figure A.12.

The required parameters are drawn from the Invensense MPU6000 [48] for the accelerometer and gyroscope models, whereas the magnetometer model parameters are drawn from the VN-200 [46].

The noise model parameters are provided as noise density terms, which are scaled in the models as per Equation (3.1) in order to get to the required std. dev. value needed to scale the zero-mean, unit variance Gaussian processes. These are stated in Table 3.2.

The dynamic models are provided in terms of the time constant of the sensors, τ. These are stated in Table 3.3.

Table 3.2: 9-Axis IMU Model Noise Parameters

Parameter	**Value**	**Unit**
$\sqrt{T_s}\sigma_{accel_x}$	0.0039	$\frac{\text{m/s}}{\sqrt{\text{Hz}}}$
$\sqrt{T_s}\sigma_{accel_y}$	0.0039	$\frac{\text{m/s}}{\sqrt{\text{Hz}}}$
$\sqrt{T_s}\sigma_{accel_z}$	0.0039	$\frac{\text{m/s}}{\sqrt{\text{Hz}}}$
$\sqrt{T_s}\sigma_{gyro_x}$	8.73E-05	$\frac{\text{rad/s}}{\sqrt{\text{Hz}}}$
$\sqrt{T_s}\sigma_{gyro_y}$	8.73E-05	$\frac{\text{rad/s}}{\sqrt{\text{Hz}}}$
$\sqrt{T_s}\sigma_{gyro_z}$	8.73E-05	$\frac{\text{rad/s}}{\sqrt{\text{Hz}}}$
$\sqrt{T_s}\sigma_{\psi}$	0.0012	$\frac{\text{rad}}{\sqrt{\text{Hz}}}$
T_s	0.01	s

Table 3.3: 9-Axis IMU Model Sensor Dynamics Parameters

Parameter	**Value**	**Unit**
τ_{accel}	0.05	s
τ_{gyro}	0.05	s
τ_{ψ}	0.05	s

The non-linearity modelling is accomplished by a combination of saturation limits and quantization modelling. The parameters needed to quantify these models are provided in terms of resolution terms, as well as upper and lower saturation limits. These are stated in Table 3.4.

Table 3.4: 9-Axis IMU Model Sensor Non-Linearity Parameters

Parameter	**Value**	**Unit**
Upper Sat_{accel}	156.96	$\frac{\text{m}}{\text{s}^2}$
Lower Sat_{accel}	-156.96	$\frac{\text{m}}{\text{s}^2}$
$\text{Resolution}_{accel}$	0.0048	$\frac{\text{m}}{\text{s}^2}$
Upper Sat_{gyro}	34.91	$\frac{\text{rad}}{\text{s}}$
Lower Sat_{gyro}	-34.91	$\frac{\text{rad}}{\text{s}}$
Resolution_{gyro}	0.0011	$\frac{\text{rad}}{\text{s}}$
Upper $\text{Sat}_{compass}$	3.1416	rad
Lower $\text{Sat}_{compass}$	-3.1416	rad
$\text{Resolution}_{compass}$	8.73E-04	rad

3.1.4 Airspeed Sensor

Airspeed is measured using a pitot-static probe in conjunction with a differential pressure transducer. This makes use of Bernouli's equation which states that the total pressure is the sum of the static and dynamic pressure. As such, we can write our airspeed in terms of pressure as:

$$P_t - P_s = \frac{\rho V_a^2}{2}$$

Where $P_t - P_s$ is the quantity that the differential pressure sensor measures. As such, we can model the output of the differential pressure sensor as

$$y_{\Delta P_{V_a}} = \frac{\rho V_a^2}{2} + \eta_{\Delta P_{V_a}} \tag{3.16}$$

where $\eta_{\Delta P_{V_a}}$ is a zero-mean Gaussian process with variance, $\sigma^2_{\Delta P_{V_a}}$.

This is implemented in Simulink as is depicted in Figure A.13, with the masked subsystem being illustrated in Figure A.14.

For this project we make use of a TE MS4525DO [49], which is an airspeed sensor made use of in the common Pixhawk flight controller series. The parameters which define this sensor model are provided in Table 3.5.

Table 3.5: Airspeed Sensor Model Parameters

Parameter	Value	Unit
$\sigma_{\Delta P_{Va}}$	0.1724	Pa
$\tau_{\Delta P_{Va}}$	0.02	s
Upper $\text{Sat}_{\Delta P_{Va}}$	6894.76	Pa
Lower $\text{Sat}_{\Delta P_{Va}}$	0	Pa
$\text{Resolution}_{\Delta P_{Va}}$	0.4204	Pa
T_s	0.01	s

3.1.5 Barometric Altitude Sensor

We are typically interested in the altitude, or height, of the system above a ground station. The corresponding change in pressure due to altitude is given by the basic equation of hydrostatics as

$$P - P_{ground} = -\rho g(h - h_{ground}) = -\rho g h_{AGL}$$

Where h is the absolute altitude of the aircraft, h_{ground} is the absolute altitude of the ground, and h_{AGL} is the altitude above ground level.

In order to get this information, we can make use of a barometric altitude sensor. This uses a fixed reference pressure which is compared to the static pressure measurement at the aircraft's current altitude. This is done in a similar way to the airspeed sensing, albeit by outputting an absolute pressure measurement because the reference pressure does not change. As such, we can model our altitude sensor pressure measurement as

$$y_{P_{\text{abs}}} = \rho g h_{AGL} + \eta_{P_{\text{abs}}} \tag{3.17}$$

Where h_{AGL} is the altitude above ground level, and $\eta_{P_{\text{abs}}}$ is zero-mean Gaussian noise with variance $\sigma^2_{P_{\text{abs}}}$.

This is implemented in Simulink as is depicted in Figure A.15, with the masked subsystem being illustrated in Figure A.16.

For this project we make use of a BMP280 [50], which is a barometric altitude sensor made use of in the common Pixhawk flight controller series. The parameters which define this sensor model are provided in Table 3.6.

Table 3.6: Barometric Altitude Pressure Sensor Model Parameters

Parameter	Value	Unit
$\sigma_{P_{\text{abs}}}$	1.3	Pa
$\tau_{P_{\text{abs}}}$	0.02	s
Upper $\text{Sat}_{P_{\text{abs}}}$	120000	Pa
Lower $\text{Sat}_{P_{\text{abs}}}$	0	Pa
$\text{Resolution}_{P_{\text{abs}}}$	0.16	Pa
T_s	0.01	s

3.1.6 Angle of Attack Sensor

Angle of attack sensors make use of a wind-vane probe to determine angle of attack and sideslip angle. We will only make use of the angle of attack for this project. The probe provides calibrated absolute angular data and this can be modelled as

$$y_\alpha = \alpha + \eta_\alpha \tag{3.18}$$

where η_α is a zero-mean Gaussian processes with variance, σ_α^2.

This is implemented in Simulink as is depicted in Figure A.17, with the masked subsystem being illustrated in Figure A.18.

For this project we make use of a Smart Miniature Vane SMV-1 [51], which is a fully integrated wind-vane probe. The parameters which define this sensor model are provided in Table 3.7.

Table 3.7: Angle of Attack Sensor Model Parameters

Parameter	Value	Unit
σ_α	0.0031	rad
τ_α	0.02	s
Upper Sat_α	3.1416	rad
Lower Sat_α	-3.1416	rad
Resolution_α	4.36E-04	rad
T_s	0.01	s

3.1.7 Benchmark Manoeuvre

In order to view the effects of our sensor modelling, we first create a masked subsystem of the preceding sensor models which takes in all of the required states from the aircraft

model, and converts them into sensor measurements as per the preceding models. This is depicted in Figure 3.3, where the unmasked implementation is depicted in Figure A.19.

Figure 3.3: Sensor Suite model implementation

We then feed in data from a benchmark manoeuvre, which makes use of the full state feedback plant model used in the autopilot implementation depicted in Figure A.34, in order to get realistic values for how our system states might vary during actual flight. The implementation of this verification run is depicted in Figure A.20, where it should be noted that the same manoeuvre will be used to verify the state estimation algorithms presented in Chapter 5.

The input commands for the reference manoeuvre are depicted in Figures 3.4, 3.5 and 3.6.

In order to depict the results of the sensor models, three examples are selected. These are, y_{p_e}, y_χ, and y_ψ. These values are selected because they are direct measurements of a particular state, so a direct comparison can be performed between the state and the measured parameter. The results of these comparisons are depicted in Figures 3.7, 3.8 and 3.9, respectively. From these figures it should be clear that our sensor models

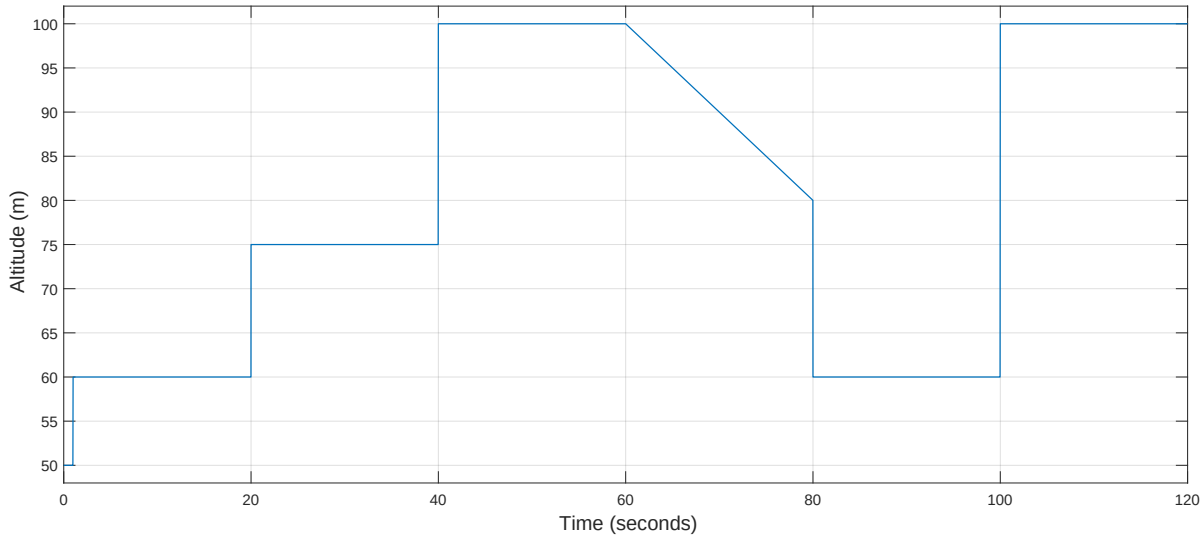


Figure 3.4: Benchmark manoeuvre, h_c (m)

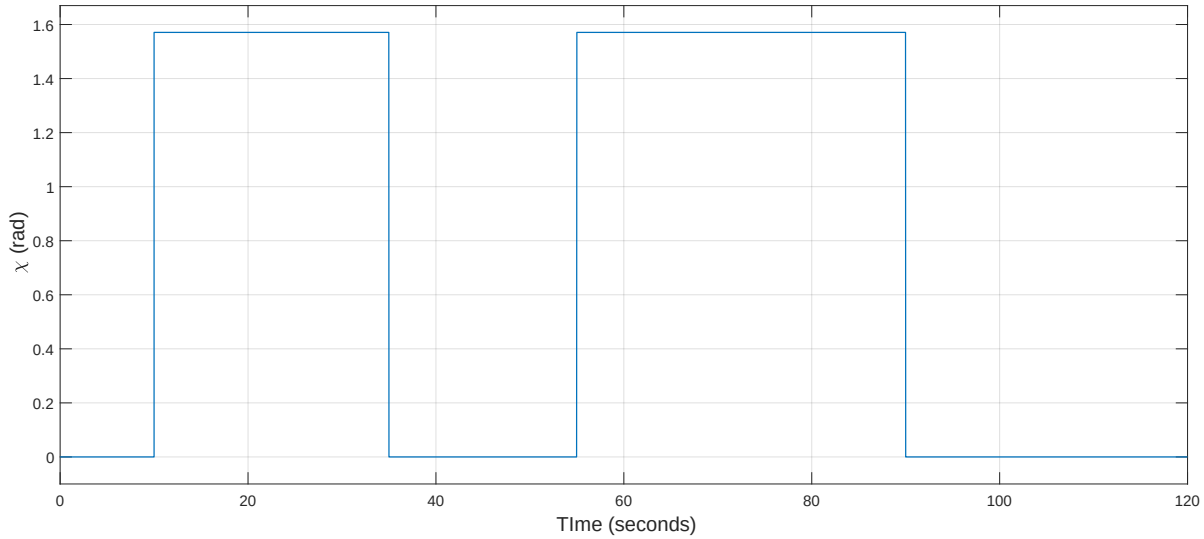


Figure 3.5: Benchmark manoeuvre, χ_c (rad/s)

perform as expected and simply act to add effects such as sampling time discretisation errors, most notable in Figure 3.8; and sensor noise, most notable in Figure 3.9. These will thus enable us to create much more realistic simulations of the system.

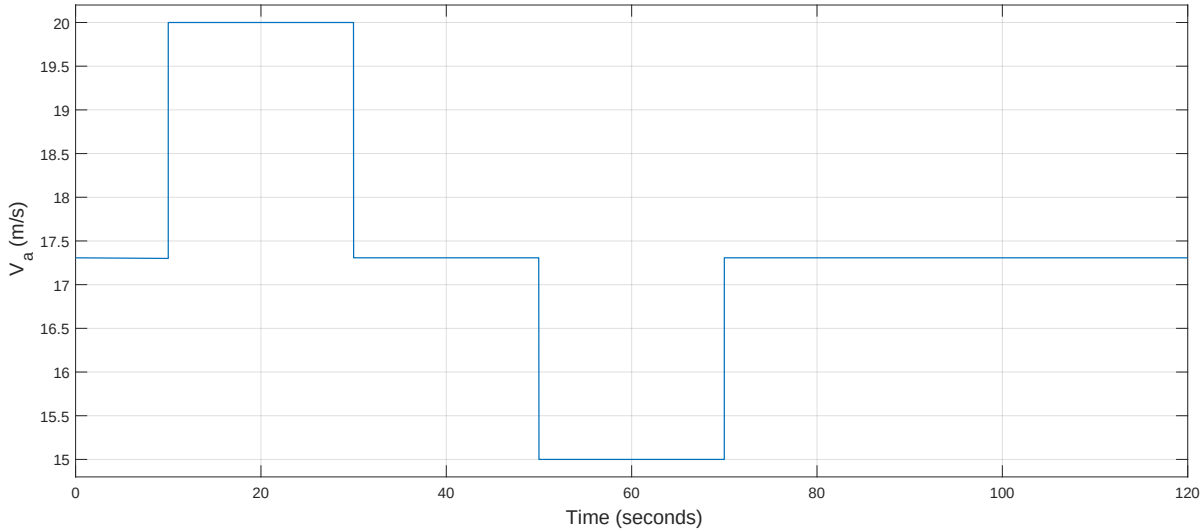


Figure 3.6: Benchmark manoeuvre, V_{a_c} (m/s)

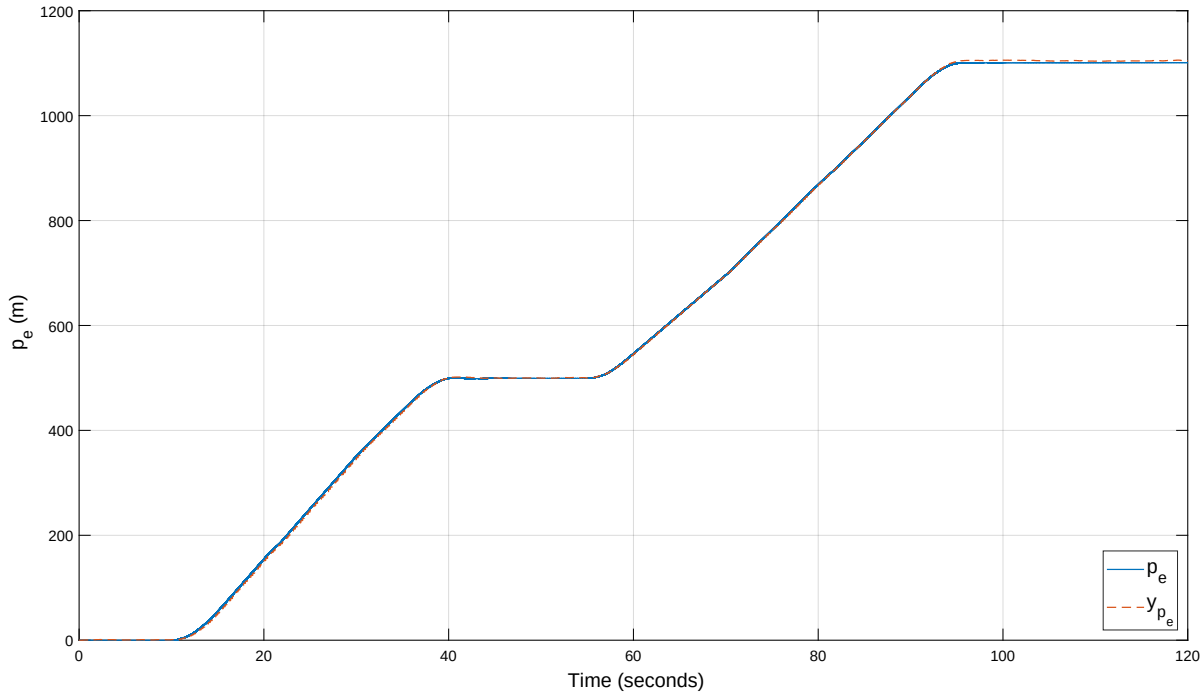


Figure 3.7: Benchmark manoeuvre measurement, y_{p_e} (m)

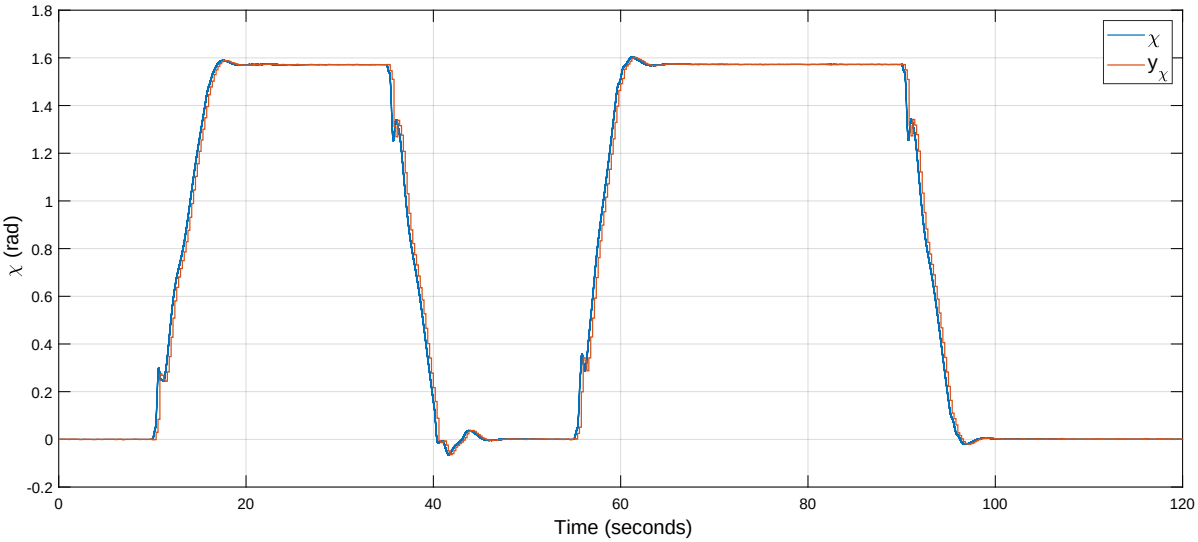


Figure 3.8: Benchmark manoeuvre measurement, y_χ (rad)

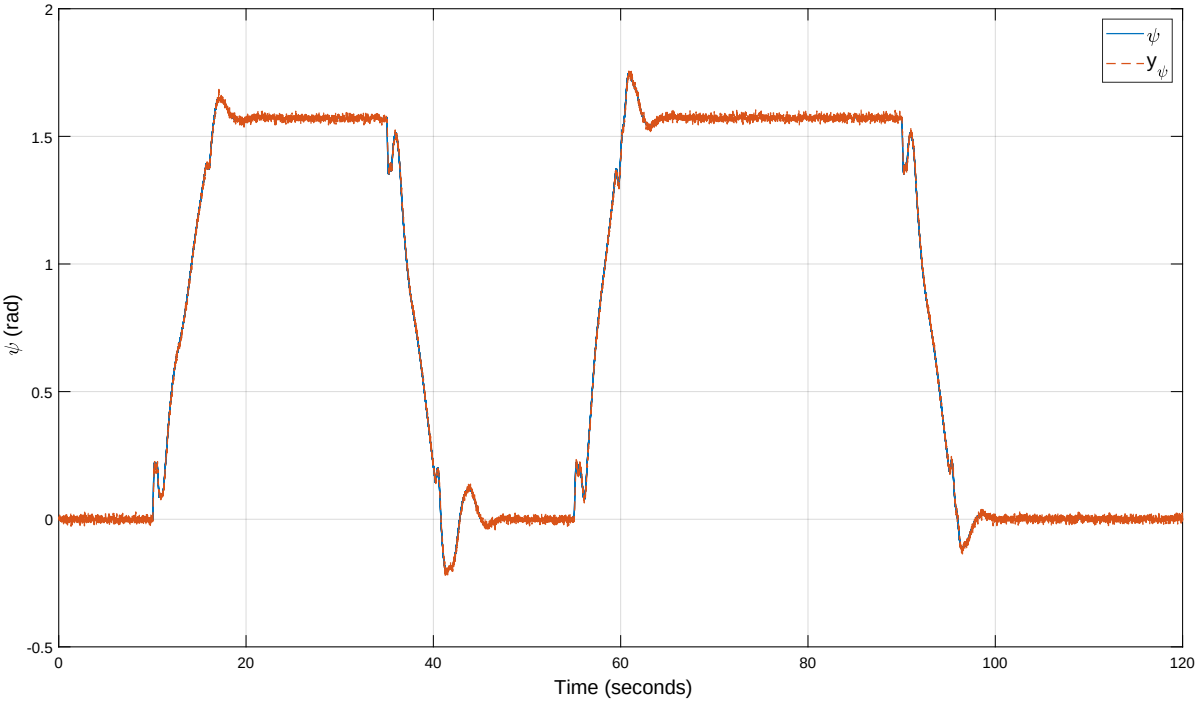


Figure 3.9: Benchmark manoeuvre measurement, y_ψ (rad)

3.2 Actuators

In a realistic system, the position or speed which we command our actuators to maintain is not instantly provided. This is because the servomotors that drive the elevons, or the motor that drives the propeller, cannot respond instantaneously. In order to ensure that we have the most representative model of the system possible, we need to include the lag introduced by the actuators themselves. This will be done by first considering the dynamics of the servomotors which actuate the elevons on the aircraft, and then the dynamics of the motor that drives the propeller.

3.2.1 Elevon Servomotors

In order to model the dynamic constraints of the servomotors which drive our elevons to their desired positions, δ_{e_r} and δ_{e_l}, we make use of a second-order transfer function in order to capture the small-signal dynamics. This is of the form:

$$\frac{\delta e_r}{\delta_{e_{r_{cmd}}}} = \frac{\omega^2}{s^2 + 2\zeta\omega s + \omega^2} \tag{3.19}$$

Where we use δe_r as an example, but with the effect applying to both elevons.

We also enforce some non-linearities on the servomotors to handle large-signal limitations, which are in the form of rate limitation and saturation.

This is implemented in Simulink as is depicted in Figure A.21, with the masked subsystem being illustrated in Figure A.22.

The parameters used for the small signal model are derived from [52], whereas the large signal parameters are from a commercially available servo-motor (ie. a Team Corally - CS-3007 HV High Speed Mini Servo [53]). These are captured in Table 3.8.

Table 3.8: Elevon Actuator Model Parameters

Parameter	**Value**	**Unit**
ω	100	rad/s
ζ	0.7071	Unitless
Upper Rate Limit	13.09	rad/s
Lower Rate Limit	-13.09	rad/s
Upper Sat	0.5236	rad
Lower Sat	-0.5236	rad

3.2.2 Propeller Motor

The propeller motor is modelled as a simple first order lag, with no non-linearities imposed (ie. the small-signal and large-signal models are treated equally), except for the previously stated saturation limits of $0 \leq \delta_t \leq 1$. This takes the form of

$$\frac{\delta_t}{\delta_{t_{cmd}}} = \frac{1}{\tau s + 1} \tag{3.20}$$

where τ is the time constant of the system.

This is implemented in Simulink as is depicted with the masked subsystem being illustrated in Figure A.23, which is just a simple transfer function model of the form provided previously.

The time constant used for this system is derived from [3], which makes use of the Hacker A40-12S V2 14-Pole 3-Phase Brushless DC Motor, as stated previously, and is given as

$$\tau = 0.2 \text{ s}$$

From this we know that it will not be realistic to attempt to drive our propeller speed in a closed loop system faster than approximately 31.4 rad/s. In other words, the actuator dynamics will constrain us to a closed loop response with cutoff frequency below this value.

CHAPTER 4

Autopilot Design

At this stage in the text we have defined and characterised the aircraft dynamics; considered how the state of the system can be measured and the limitations and non-linearities thereof; as well as quantified the dynamic response of the actuators which will be made use of in order to induce nett forces and moments on the aircraft, which will result in the ability to change its state. As such, we have completely quantified and identified the system, as well as how we can interface with both from an input and output perspective. The next element in the system architecture denoted in Figure 1.2, is the autopilot, which we can now consider the design and optimisation of.

The primary goal in autopilot design is to control the inertial position, (p_n, p_e, h), and attitude, (ϕ, θ, χ), of the aircraft. In the discussion to follow we will first consider the use of gain scheduling, whereby we can make use of a combination of controllers to control a system whose dynamics change as a function if its state. We will then consider our longitudinal controller design, where we will first consider our longitudinal attitude controller, which controls pitch angle, θ. We will subsequently consider the controllers responsible for altitude, h, as well as airspeed, V_a - which can be controlled both by pitch angle, θ, as well as by throttle command, δ_t. Finally, we will look at the longitudinal state machine which will handle the take-off, climb, altitude-hold and descend states of aircraft operation.

The lateral controller will then be considered, which will firstly deal with the lateral attitude controller in roll, ϕ, followed by course angle, χ. The course angle will be used to control the inertial position of the aircraft in p_n, and p_e.

We will then consider the impacts of discretisation, which is used to transition from the continuous controllers thus far designed into digital controllers, so that they are able to be implemented in a digital system, such as a microcontroller. It will be discussed how this was done for this project, as well as how the performance of the digital controllers compares to the previously-designed continuous controllers.

Finally, a final set of autopilot verification tests will be run to illustrate the performance of the autopilot under various conditions.

It should be noted that in order to constrain the development of our control system, we will make the assumption that all controllers are of the classical control PIDF (PID with a filtered derivative term) type. Variations of this architecture are also permitted,

but more exotic controllers will not be considered as classical controllers of this type are easily implemented and unless there is a strong requirement to deviate from them, are the best starting point for control system design. In order to ensure that our classical control system is appropriate, we will compare the results to the performance bounds provided in Figure 2.20 and provide justification if the system responds slower than what would be considered the optimal performance denoted in the figure.

4.1 Gain Scheduling

Gain scheduling is a practical and powerful method for the control of non-linear systems. A gain-scheduled controller is formed by interpolating between a set of linear controllers derived for a corresponding set of plant linearisations associated with several operating points. The interpolation is based on scheduling parameters that vary slowly (relative to the state(s) one wishes to control) and capture the plant non-linearities. The resulting controller is a linear system whose parameters (gains) are adjusted (scheduled) as a function of the scheduling variables.

Gain scheduling is attractive because it exploits linear control design methods that are theoretically mature, well understood, and involve computationally efficient synthesis procedures. Because the control design is based on linear approximations of the plant, one can guarantee that at each operating point, the feedback system has the desired stability and performance properties. In addition, because the controller gains are adjusted to reflect the plant operating conditions, the gain-scheduled control system can respond rapidly to changing operating conditions. The gain-scheduling method has been proven sound by successful application in various industries ranging from power generation to aerospace [54].

As was shown in Figures 2.25, 2.26 and 2.27 for the longitudinal dynamics, and Figures 2.30, 2.31 and 2.32 for the lateral dynamics, the Skywalker X8 has a substantial deviation in dynamics based on its current state. As such, we could potentially develop a single, linear controller which performs adequately across the variations in the system dynamics, but optimally for none of them. Alternatively, we can design a set of controllers that are optimised for each of the system linearisations, and then combine them so that we generate a non-linear controller that is optimised at all times, instead of just being passable.

4.1.1 Scheduling Parameter Selection

Scheduling parameters should be selected that adequately capture the plant non-linearities, and vary slowly [54] relative to the dynamics which one is attempting to control. In this case, attempting to use states like the angular rates would not work well because these can have large transients in a very short period of time. Attempting to use something like the aircraft attitude would also be unwise, because the attitude cannot be measured

directly, so the control system might select for a linear controller based on an estimate of the state which might be relative far from reality, which might result in instability. As such, we need to be very careful in our selection of the scheduling parameters.

4.1.2 Longitudinal Scheduling Parameters

As was discussed in Section 2.7.2, the longitudinal system dynamics are best captured by V_a, and α. From Section 3.1, we also know that both V_a and α are directly measurable quantities so we need not be concerned about introducing unfavourable controllers based on incorrect state estimation. Given that the longitudinal control system is responsible for controlling θ, and θ is closely tied to α, we can reasonably state that we do not expect α to vary too quickly relative to the dynamics which we are controlling. As such, both of these parameters make ideal scheduling parameters for the longitudinal controllers and these will be designed such that

$$C_{lon} = f(V_a, \alpha)$$

where C_{lon} is a generic longitudinal controller.

It should be noted, however, that in order to control airspeed via propeller thrust, we do not expect α to have a significant impact, as was alluded to in the discussion surrounding Figure 2.28, and as such $C_{V_{a_{\delta_t}}}$ will be scheduled as a function of V_a only.

As such, our controller implementation will start off with the form depicted in Figure A.24, with any adjustments made being discussed in the relevant section.

4.1.3 Lateral Scheduling Parameters

As was discussed in Section 2.7.3, the lateral system dynamics are best captured by V_a, with β not being considered as we have no control over it with a rudderless aircraft. As stated in the previous section, we also know that V_a is directly measurable and is not expected to vary quickly relative to the response of the controllers used to govern the lateral dynamics. As such, V_a is an ideal scheduling parameter for the lateral controllers and these will be designed such that

$$C_{lat} = f(V_a)$$

where C_{lat} is a generic lateral controller. These controllers will start off with the form depicted in Figure A.25, with any adjustments made being discussed in the relevant section.

4.2 Longitudinal Controllers

We know from the preamble to this section that the autopilot is responsible for controlling the attitude and inertial position of the aircraft. We also established in Section 2.7.1 that we could isolate the longitudinal and lateral dynamics from a control perspective. As such, we are able to design a control system which can control for θ, h, and V_a, without needing to consider the impact of ϕ or χ. We have also have established in Section 2.7.2, and in the preceding section on the selection of the longitudinal scheduling parameters, that it is wise to include the effects of gain scheduling, as a function of (V_a, α) into the longitudinal controllers. However, whether this is required in all cases will be verified, because reducing controller complexity is ideal if possible. As such, if it is determined that there is no appreciable benefit of the gain-scheduling methodology suggested, one can reduce this to a single controller so that implementation is made simpler.

With this in mind, the longitudinal control system architecture is selected to be of the successive loop closure type, of which the standard implementation is depicted in Figure 4.1.

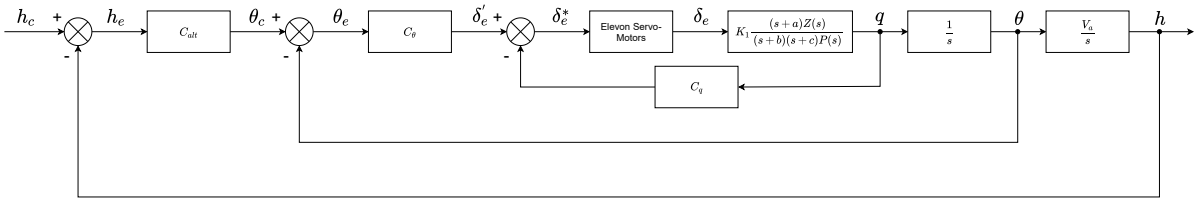


Figure 4.1: Successive loop feedback structure for the longitudinal controllers

It should be noted from Figure 4.1 that we have made use of the observed transfer functions which were discussed in Section 2.7.2. However, we know from our previous discussion that we will extend this standard configuration to be of the gain-scheduled variety. We will also include the effects of the pitch rate damping controller, C_q, into the main pitch angle controller, C_θ. This results in the configuration denoted in Figure 4.2.

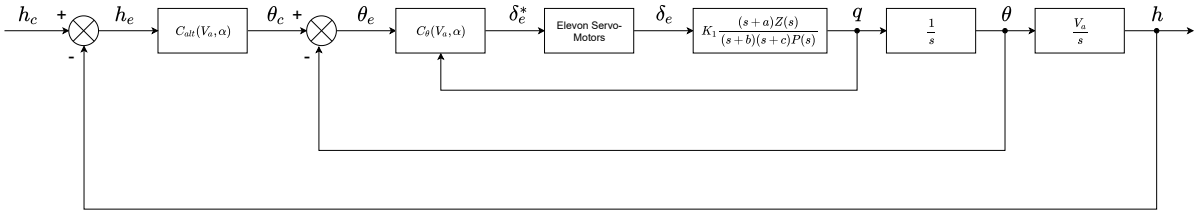


Figure 4.2: Gain-scheduled successive loop feedback structure for the longitudinal controllers

As noted, we have made use of the transfer function estimates which were developed in Section 2.7.2. In order to make it easier for the tuning algorithms in MATLAB (these are H_∞-based optimisation algorithms) to optimise our controller gains, we can leverage

this knowledge to generate a reduced-order model of the aircraft. This reduced-order model will likely deviate from the actual linearised model, because we are essentially assuming that $Z(s) = P(s)$, which is likely not true across the entire frequency spectrum. However, as long as we ensure that the controllers are designed such that they roll off the frequency response of the plant before the difference between the full linearised model and reduced order model becomes non-negligible, these approximations will provide an adequate means to design the controllers in a simplified environment. We will also verify that they produce desirable results in simulation with the full non-linear aircraft model.

In order to generate reduced order linearisations, we make use of the $rsys = \text{balred}(sys, \text{orders})$ function in MATLAB [55], which computes a reduced order approximation, $rsys$, of the linear time invariant (LTI) model, sys, with the desired number of states for $rsys$ specified by the orders parameter. Since we will assume that $Z(s) = P(s)$, we can assume two states for $\frac{q}{\delta_e}(s)$. We can then apply the transfer functions of $\frac{\theta}{q}(s)$ and $\frac{h}{\theta}(s)$, which were discussed in Section 2.7.2 in order to generate all of the reduced order models for the system. The comparison between the full order and reduced order models are shown in Figures 4.3, 4.4 and 4.5.

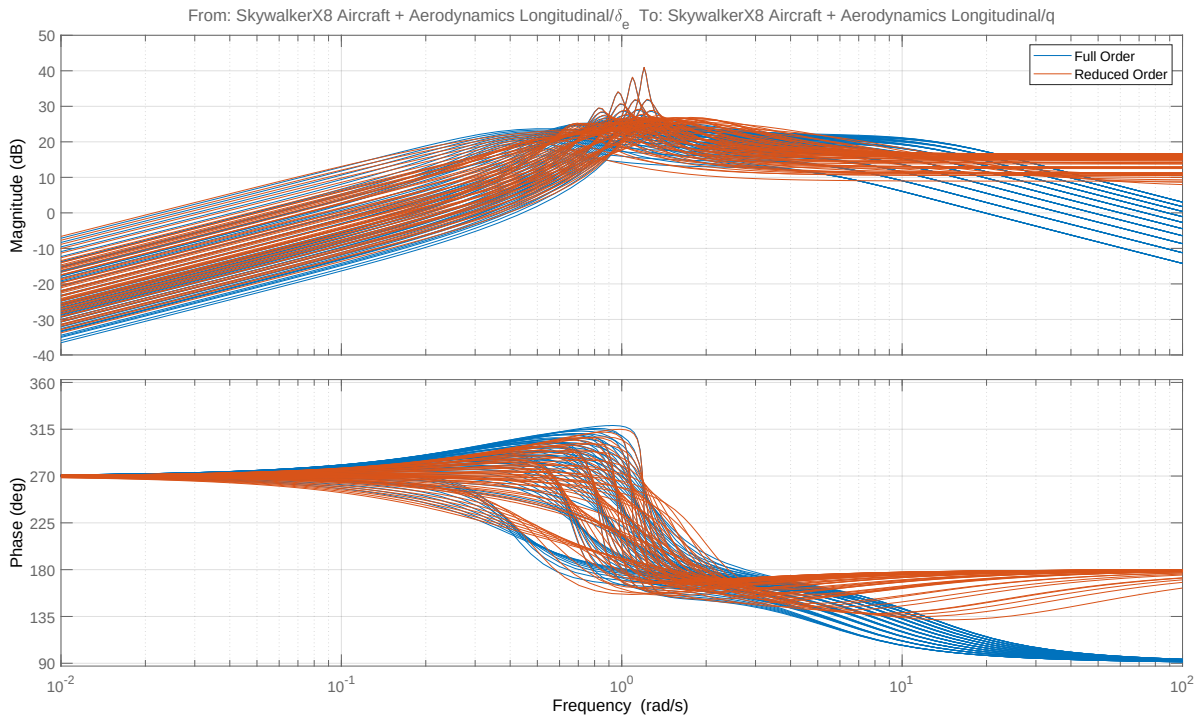


Figure 4.3: Reduced-order linearisation of $\frac{q}{\delta_e}(s)$ in comparison to the full order model

Reviewing our reduced order models it is clear that they provide an adequate approximation of the full order models. It should be noted that the model reduction exercise results in a reduced number of states by two for all cases, which is a significant reduction as the maximum number is five. With these reductions, we must ensure that any con-

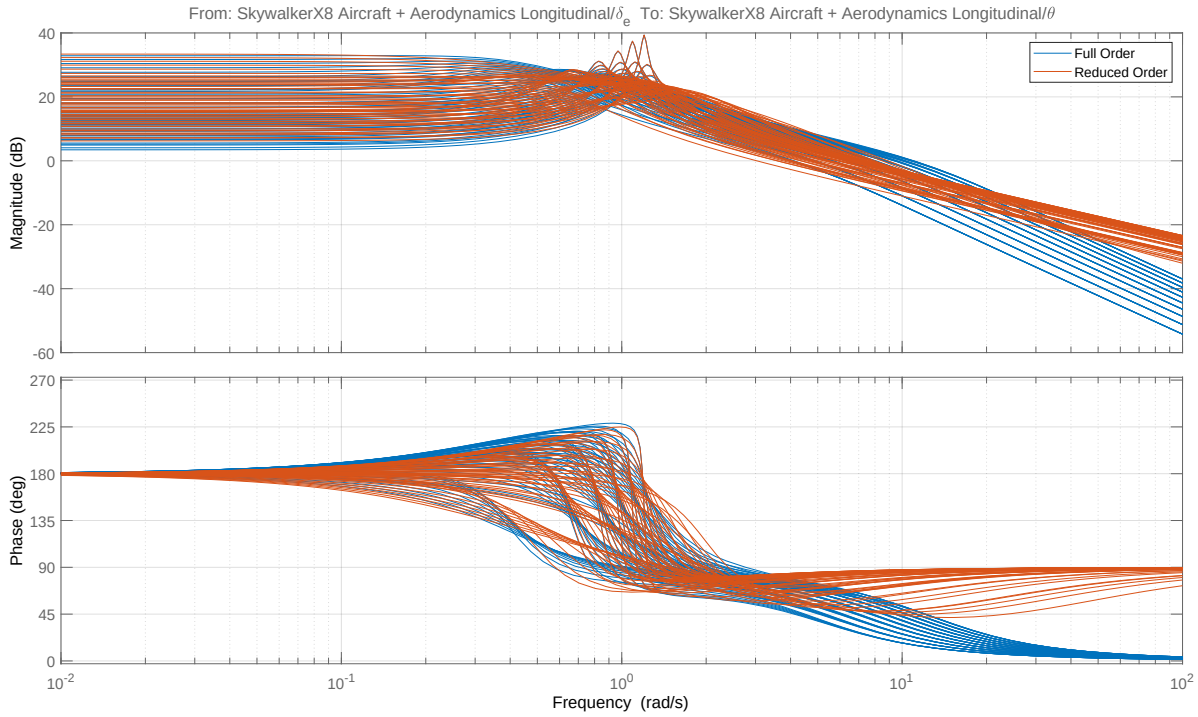


Figure 4.4: Reduced-order linearisation of $\frac{\theta}{\delta_e}(s)$ in comparison to the full order model

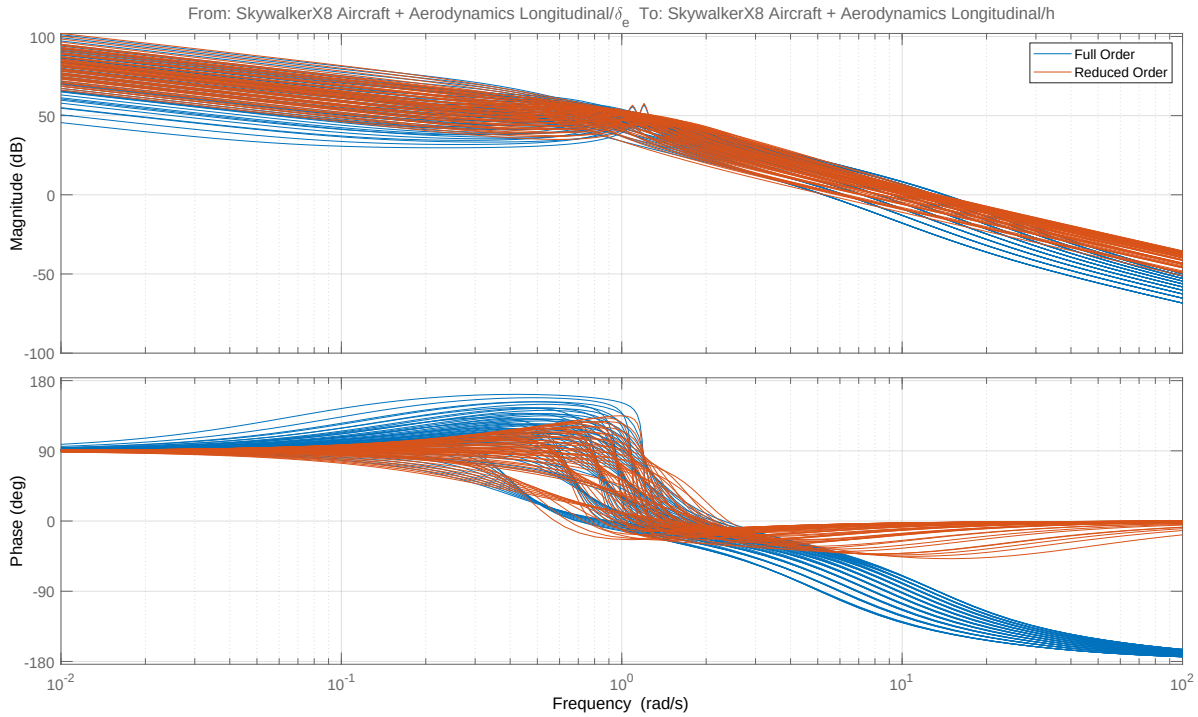


Figure 4.5: Reduced-order linearisation of $\frac{h}{\delta_e}(s)$ in comparison to the full order model

troller dynamics that impact q, or θ - or C_θ - must roll off prior to 5 rad/s, after which there is significant deviation in phase between the reduced and full order models. Any controller dynamics which impact h - or C_h - should roll off at approximately 3 rad/s, after which there is significant deviation in phase as well. This is something to keep in mind during the controller design, but is not a hard requirement as ultimately one can review if this has significant impact during the controller verification step, as this makes use of the full, non-linear, 6DoF model.

Having established the linearisation of the system that will be used, as well as having generated a set of reduced order models, we can now focus on the design of the controllers themselves. The Simulink model used to tune these longitudinal controllers can be found in Figure A.26.

4.2.1 Pitch Angle Control

As mentioned previously, the pitch angle controller, C_θ, will be a variant of the classical PIDF controller.

From our discussion on trim, we know that we will require a non-zero δ_e in order to maintain a wings-level trim condition. As such, we know that we will need an integrator term so that we can maintain a non-zero value with a zero error input.

From the standard configuration presented in Figure 4.1, we know that the standard successive loop closure architecture generally makes use of a pitch rate damping term, which is denoted in the figure as C_q. In the selected configuration, depicted in Figure 4.2, this is still made use of, but is simply included in the pitch angle controller. This is done because we know that $q \approx \dot{\theta}$, and as such we can make use of standard variant of the PIDF controller where one does not take the derivative of the error signal, but the feedback term itself. We extend that even further in this case, such that the derivative is not calculated, but simply used directly from our measured pitch rate gyroscope. In other words, the PIDF controller can then simply be of the form:

$$C_{1_\theta} = K_p + K_i\frac{1}{s} + K_d\frac{q}{\theta_e}\frac{N}{s+N}$$

where it should be noted that $\frac{q}{\theta_e}$ is used to illustrate that the error term is not made use of in the derivative portion of the controller, but which still allows us to use the more convenient controller architecture depicted in Figure 4.6.

In order to allow for the MATLAB optimisation algorithm to have additional parameters to tune for to get ideal dynamics, we add a second degree of freedom to our control system by adding a feed-forward controller of PDF type. It should be noted that an integrator can not be added here because we might have a fixed non-zero command, which would

cause δ_e^* to grow unbound. As such, this controller is of the form:

$$C_{2_\theta} = K_p + K_d \frac{sN}{s+N}$$

This then gives us the controller architecture as depicted in Figure 4.6.

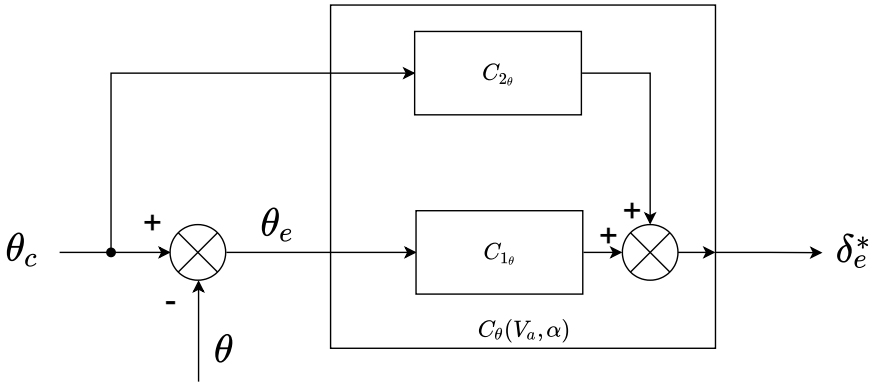


Figure 4.6: Pitch angle controller architecture

With this controller architecture we make use of the tuning goals defined in Listing B.1. This results in 60 controllers, each optimised at a particular combination of (V_a, α). The results of this design are depicted in Figures 4.7 and 4.8.

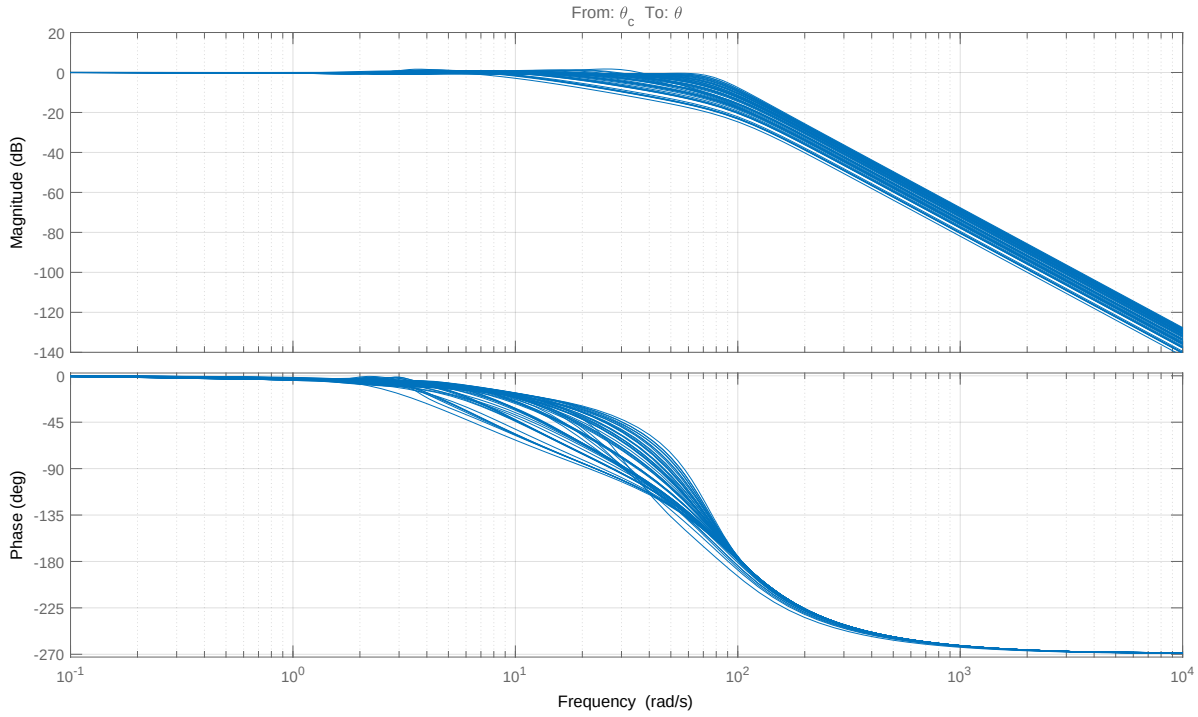


Figure 4.7: Pitch angle controller bode plot for $\frac{\theta}{\theta_c}(s)$

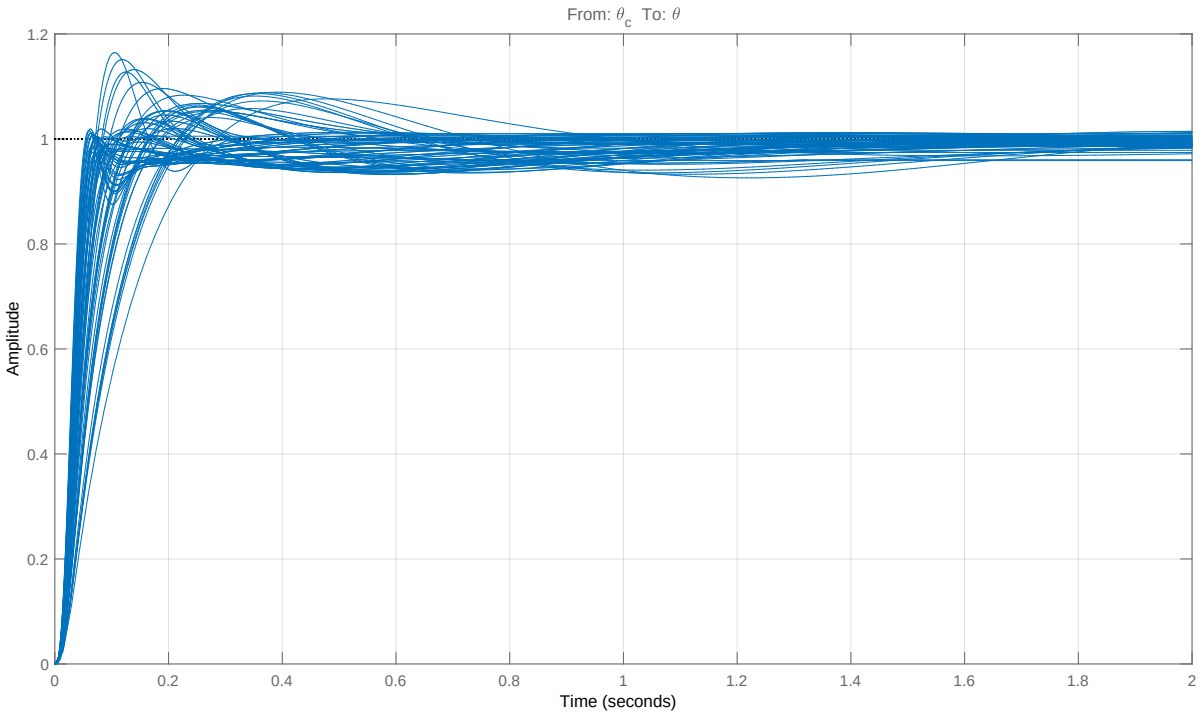


Figure 4.8: Pitch angle controller step plot for $\frac{\theta}{\theta_c}(s)$

Both Figures 4.7 and 4.8 show rapid performance without any excessive resonance peaks in the frequency domain, or overshoot in the time domain. We also note from the time domain response an ability to track input commands quite accurately. It should be noted that from Figure 2.20, we know that our maximum possible pitch rate, q, varies between 4.03 rad/s, and 13.27 rad/s, albeit only considering variations in V_a. This would correspond to changes of 1 rad in θ taking between 0.075 and 0.25 s. From the step response shown in Figure 4.8, we have rise times between 0.05 s and 0.2 s, which correspond quite well to what would be considered the optimal response times. These are thus fairly impressive controller designs and illustrate that near-optimal performance is achievable with a classical controller topology and a more exotic controller need not be considered.

In order to ensure that our original assumption of requiring gain-scheduled controllers holds, we can make use of a mid-point controller, $C_\theta = C_\theta(V_{a_{max}}/2, \alpha_0/2)$, and review what the control system dynamics would look like if this was used at all of the (V_a, α) combinations. This is depicted in Figure 4.9, where it is clear that a mid-point controller can stabilise the plant and provide adequate performance, but is much slower than the set of gain-scheduled controllers and provides a significant increase in overshoot, thus justifying the use of the extended controller set.

In order to review the effect on the stability margins, we can also examine the open loop bode plot depicted in Figure 4.10, which indicates that the gain margin remains consistent between the mid-point and gain scheduled controllers, with the gain scheduled

controllers providing significant additional phase margin. This too justifies the use of the gain scheduled controller over a single controller as we have shown that both in the time and frequency domains we are able to generate better, more robust performance using the former over the latter.

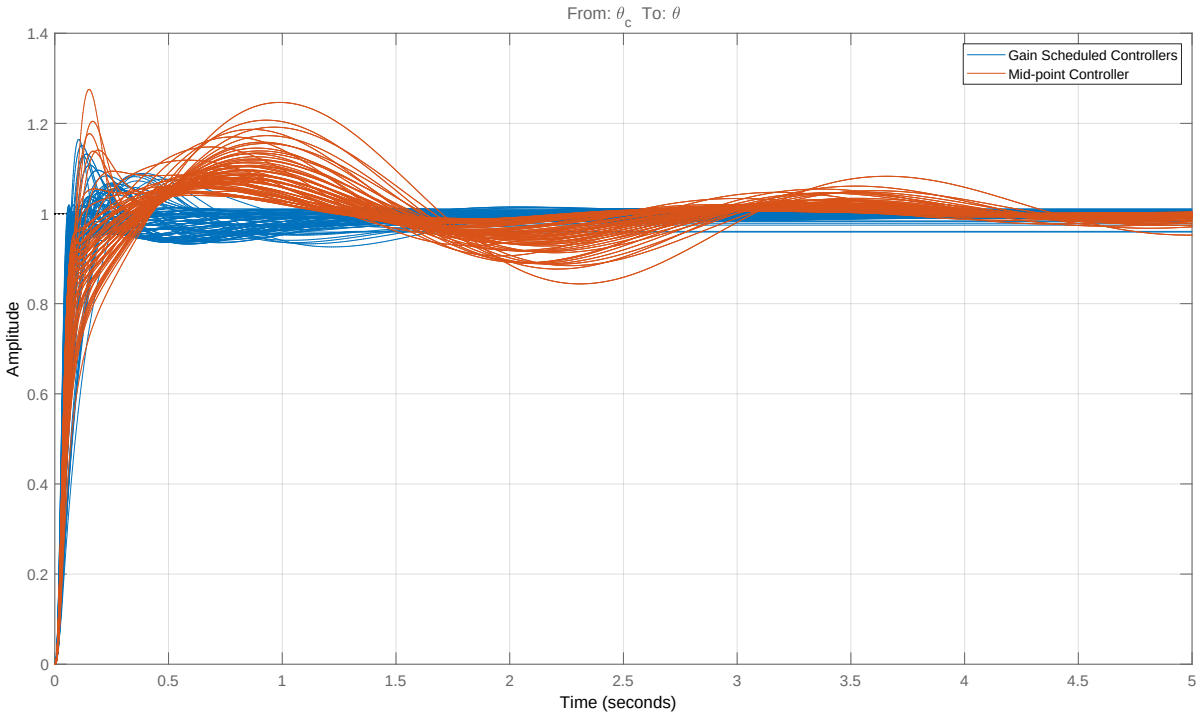


Figure 4.9: Pitch angle controller step plot for $\frac{\theta}{\theta_c}(s)$, comparing a mid-point and gain scheduled controller

One might note, however, that the frequency response is outside of the bounds which were specified when calculating the reduced order models, and indeed that is the case. However, from this perspective we are only concerned with exciting the plant dynamics from δ_e to θ. If we illustrate that this transfer function, $\frac{\theta}{\delta_e}(s)$, does not allow for the passing of higher frequency dynamics with the controller in the loop, we can use this as a first step to ensuring that we do not encounter any issues that might be introduced by making use of the reduced order models. This is depicted in Figure 4.11, where we see that we already start to roll off at the previously stated 5 rad/s, with a maximum gain at this point in the plot of -1 dB, and a minimum gain at -10 dB. While -1 dB is not a large reduction in gain, we start to roll off at -20 dB/decade after this point, which results in signals of fairly insignificant magnitude much beyond this point.

This is by no means an exhaustive demonstration that the reduced-order models will not include unfavourable dynamics further on in the design process, but it does seem as though these errors be diminished in terms of magnitude and are unlikely to present any problems. The only real way of illustrating this fully will be once the full longitudinal

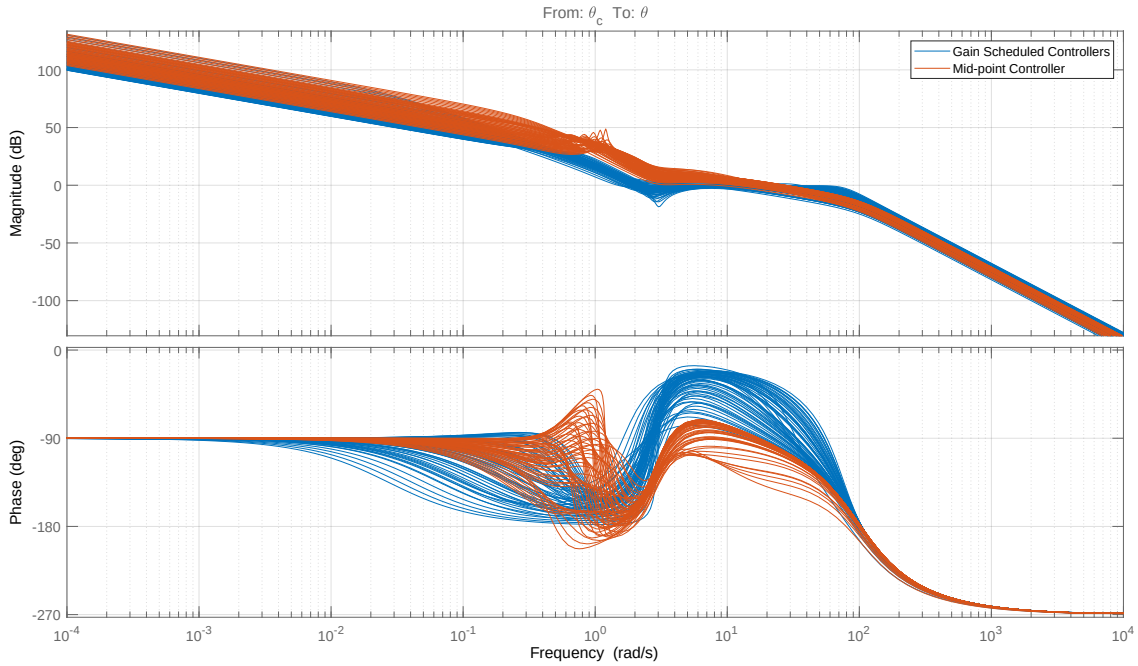


Figure 4.10: Pitch angle controller open-loop bode plot for $\frac{\theta}{\theta_c}(s)$, comparing a mid-point and gain scheduled controller

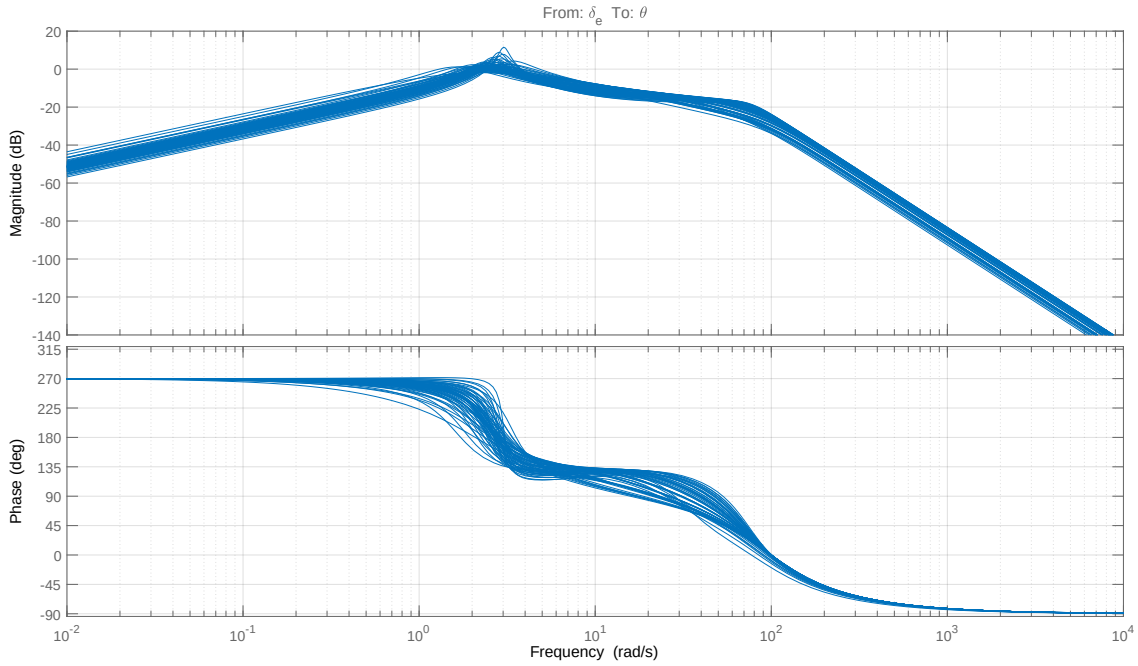


Figure 4.11: Pitch angle controller bode plot for $\frac{\theta}{\delta_e}(s)$

control system is verified by simulation with the full non-linear model.

4.2.2 Altitude Control

Figure 4.2 illustrates that the altitude controller is simply built around the pitch angle controller - which is the premise behind successive loop closure. We know that from Figure 4.7 we can estimate $\frac{\theta}{\theta_c}(s)$ as unity gain until approximately 10 rad/s. As such, we can approximate

$$\frac{h}{\theta_c}(s) \approx \frac{V_a}{s}$$

As long as we ensure that we start rolling off the system at a frequency 2-3x lower than the previously stated 10 rad/s.

We also know that for our altitude controller we might require a non-zero pitch angle, and as such, pitch command, θ_c, in order to hold altitude constant. This is because if our flight path angle is zero, $\gamma = 0$, we know we are not gaining altitude. In this condition, we have that $\theta = \alpha$, which might mean that we have to hold a particular pitch angle in order to generate the correct amount of lift to remain at a particular altitude. As such, we know that we require an integrator term so that we can maintain a non-zero pitch angle command with a zero error input signal. However, we also know that altitude is severely rate-limited from a large-signal analysis perspective as we cannot gain altitude faster than is allowed by our current airspeed and pitch angle. As such, we will make use of a standard PIDF controller topology such that,

$$C_{alt} = K_p + K_i\frac{1}{s} + K_d\frac{sN}{s+N}$$

This controller topology is then utilised with the tuning goals listed in Listing B.2 in order to design the required controllers. This results in 60 altitude controllers, each optimised at a particular combination of (V_a, α), as per the pitch angle controllers. The results of this design are depicted in Figures 4.12 and 4.13.

Similarly to the pitch angle controller, we need to ensure that gain scheduling is necessary here. Prior to doing this, we observe that this might not be the case as it could be that since we have a set of gain-scheduled controllers in pitch angle ensuring that these dynamics are handled well, the altitude controller might provide adequate performance as just a single controller. In other words, but having a gain-scheduled pitch controller, we might have adequately removed the majority of the undesirable dynamics from the system and as such might only require a simple outer-loop controller.

In order to establish if this is the case, we will make use of a comparison between a mid-point controller and the full gain scheduled controller, as in the discussion on C_θ. However, we will make use of the higher order linearised model in order to ensure that our reduced-order model is not providing any benefit to either of the controller types.

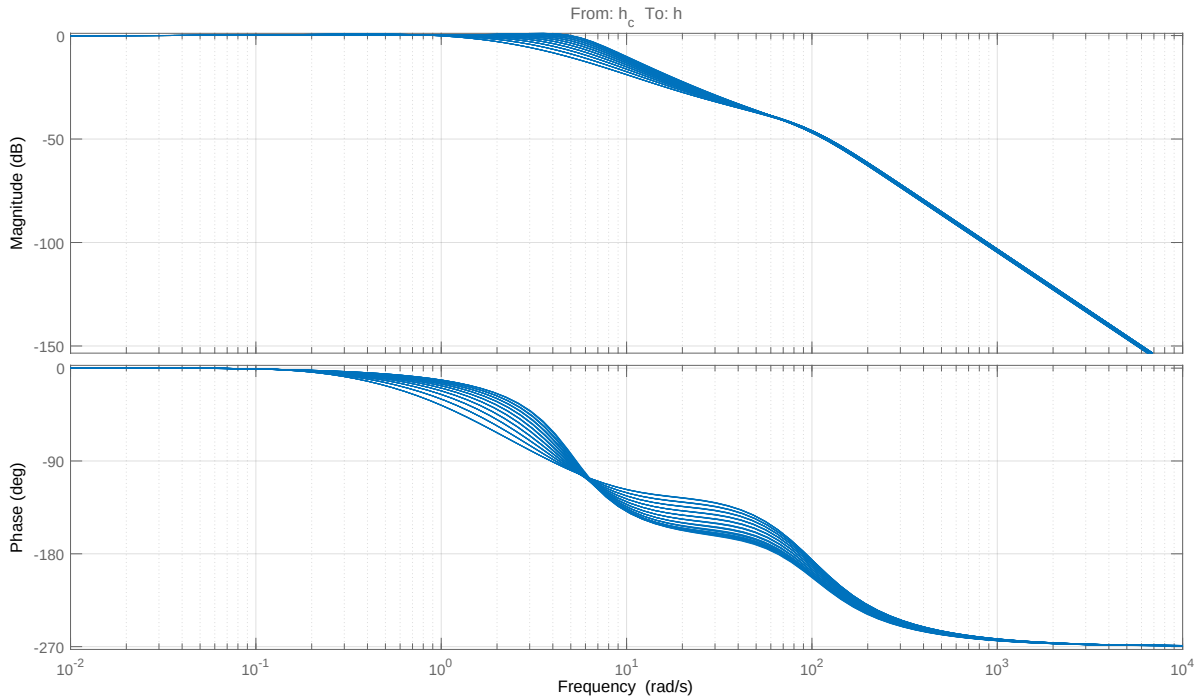


Figure 4.12: Altitude controller bode plot for $\frac{h}{h_c}(s)$

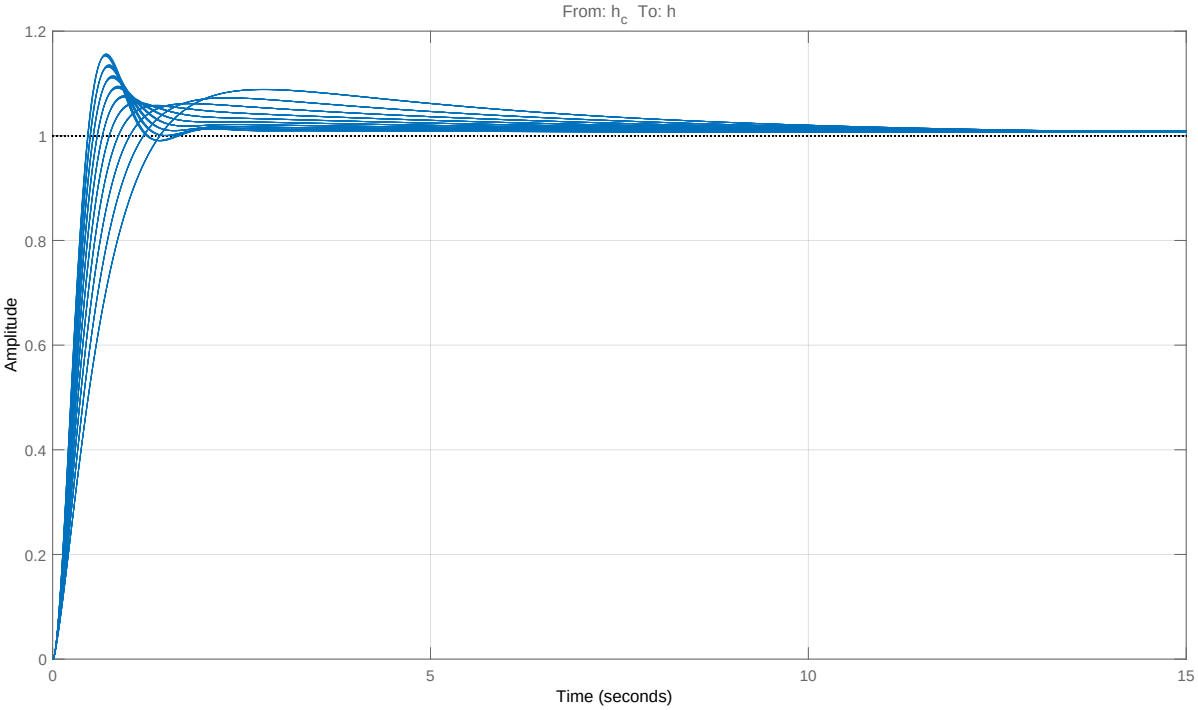


Figure 4.13: Altitude controller step plot for $\frac{h}{h_c}(s)$

The results of this comparison are depicted in Figures 4.14, 4.15 and 4.16.

From these results, it should be clear that there is no significant benefit in including a gain-scheduled controller. As such, the altitude controller will be constrained to be a simple PIDF controller, with gains based on the selected mid-point controller.

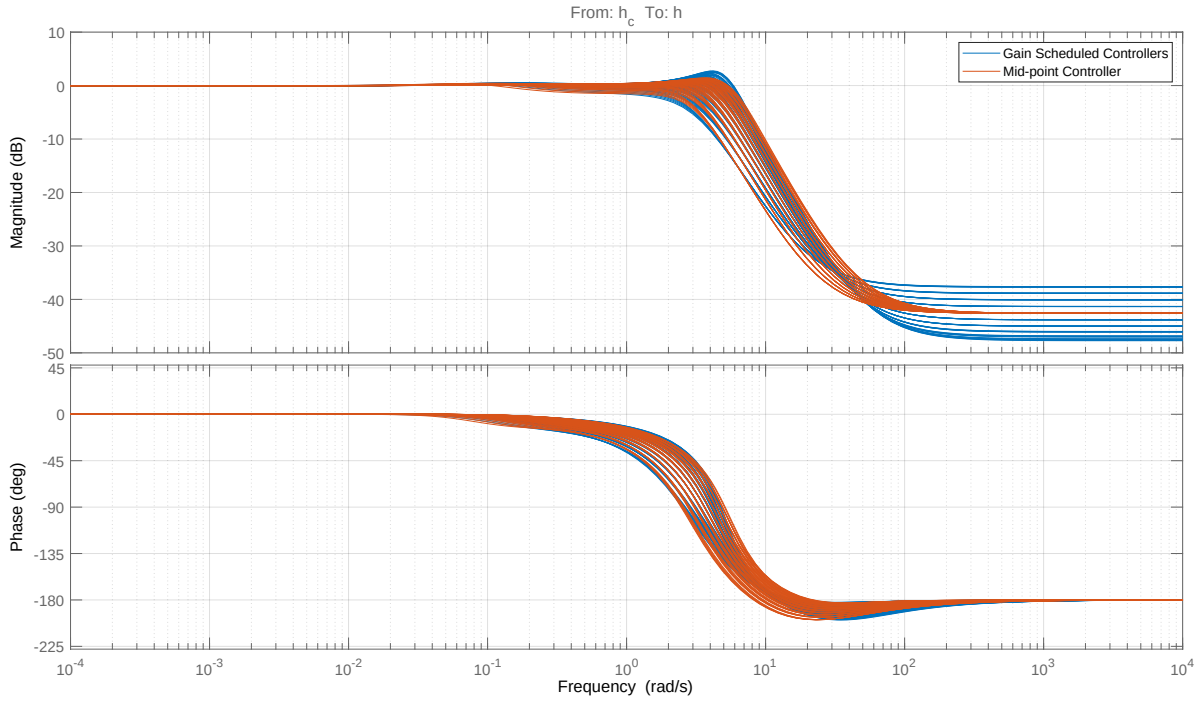


Figure 4.14: Altitude controller bode plot for $\frac{h}{h_c}(s)$ comparing a mid-point and gain scheduled controller

We can approximate our expected altitude performance as:

$$\dot{h} = V_a \sin\theta$$

Where we know that we have airspeeds ranging from 9.0 m/s to 29.6 m/s. For the pitch angle, we specify that $\theta_{max} = 60°$ (which is set as a saturation limit on the controller output), but which can only be applied for approximately half the time as we have to transition to this maximum value before transitioning back to approximately zero at our wings-level condition, or $\theta_{avg} = \frac{1}{2}\theta_{max}$. As such, we can approximate our expected performance as

$$\dot{h}_{max} \approx V_a \sin\theta_{avg} \tag{4.1}$$

Where we expect a maximum response time of between 0.07 s and 0.2 s. However, the tuned controllers provide us with reduced performance relative to this, with a rise time of between 0.5 s to 1.2 s, or about six times slower. This is intentional because, as demonstrated in the discussion surrounding Figure 2.18, if θ_c is allowed to vary too

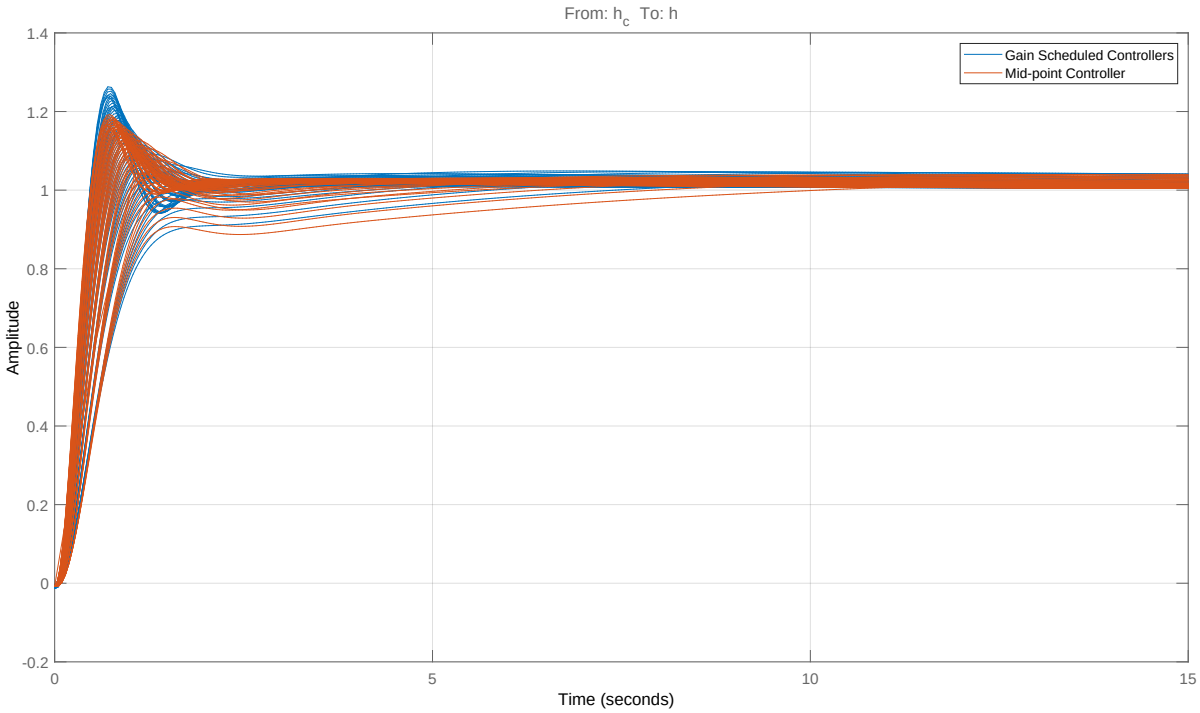


Figure 4.15: Altitude controller step plot for $\frac{h}{h_c}(s)$ comparing a mid-point and gain scheduled controller

quickly with optimal $\frac{\theta}{\theta_c}$ controllers - which are very fast so as to reject disturbances more easily - we will drive the aircraft into stall as we pitch aggressively. As such, we reduce the response time of the altitude controller to avoid these conditions by ensuring θ_c does not vary too quickly. However, this will be shown to not result in performance which is too drastically diminished once we consider the large signal effects because it only significantly impacts performance at the start and end of an altitude adjustment, with the majority of the manoeuvre being spent at a saturation limit as discussed.

It should also be noted that given the decision to make use of the full linearised model, as opposed to the reduced one, in the comparison between the mid-point and gain scheduled controllers, we have established that our model reduction technique produced a valid model in order to perform controller tuning as the resulting controllers perform well.

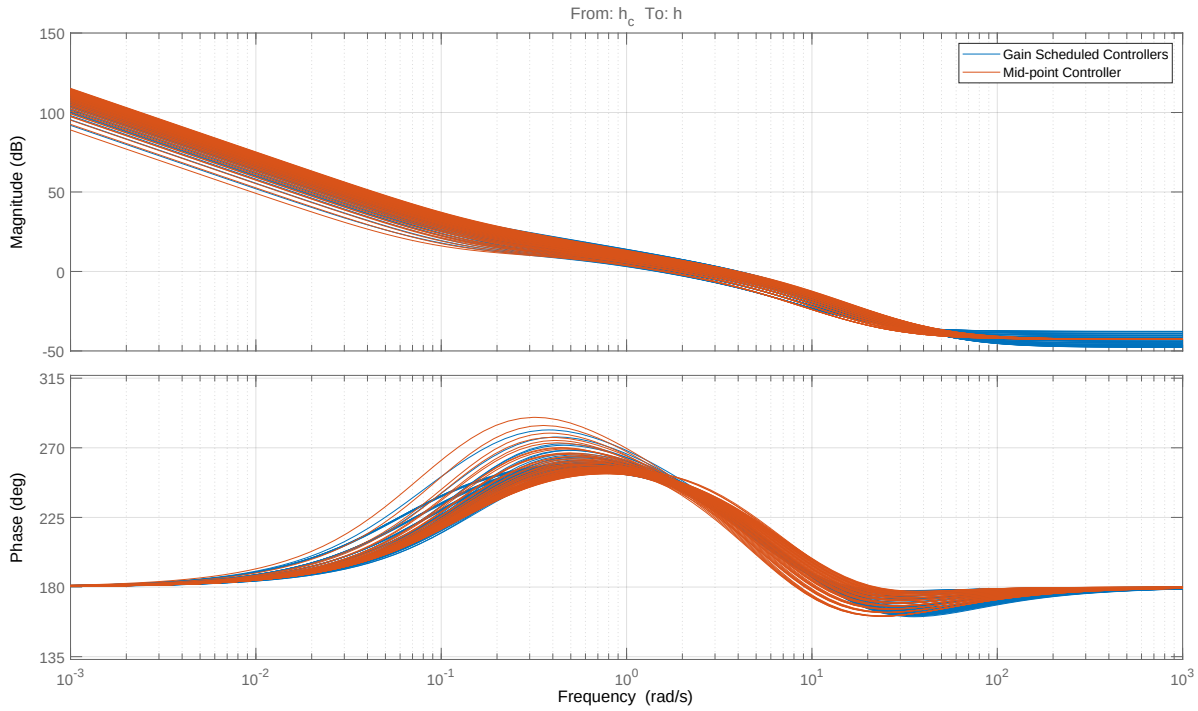


Figure 4.16: Altitude controller open loop bode plot for $\frac{h}{h_c}(s)$ comparing a mid-point and gain scheduled controller

4.2.3 Airspeed Control

From Figure A.26 we can see that we are able to control V_a in two ways - via propeller thrust, and via pitch angle. Propeller thrust has been discussed before, with the system dynamics being illustrated in Figure 2.28. However, control via pitch angle hasn't been discussed. For brevity, a detailed discussion will not be provided - except to say that this provides a useful control mechanism for if there is a drastic step change in altitude command for the aircraft to descend - where the autopilot can then simply switch off the propeller and switch into a glider-mode so as to descend gracefully into the altitude-hold zone, which is some bounded area around the altitude command where we would like to control altitude. This will be discussed in Section 4.2.4.

Airspeed Control via Thrust

For airspeed control we will make use of a standard PIDF controller topology, or

$$C_{V_{a_{\delta_t}}} = K_p + K_i\frac{1}{s} + K_d\frac{sN}{s+N}$$

The tuning goals for the airspeed control via thrust are denoted in Listing B.3. From these, our gain-scheduled controllers (note that these are only a function of V_a, as was discussed in the discussion surrounding Figure 2.28), are designed. We can then consider

the performance of these controllers, and compare this with the mid-point controller discussed in the preceding sections in order to determine if gain-scheduling is required for the system. These results are denoted in Figures 4.17, 4.18 and 4.19, where it should be clear that the gain-scheduled controller produces much more consistent, superior performance in all metrics.

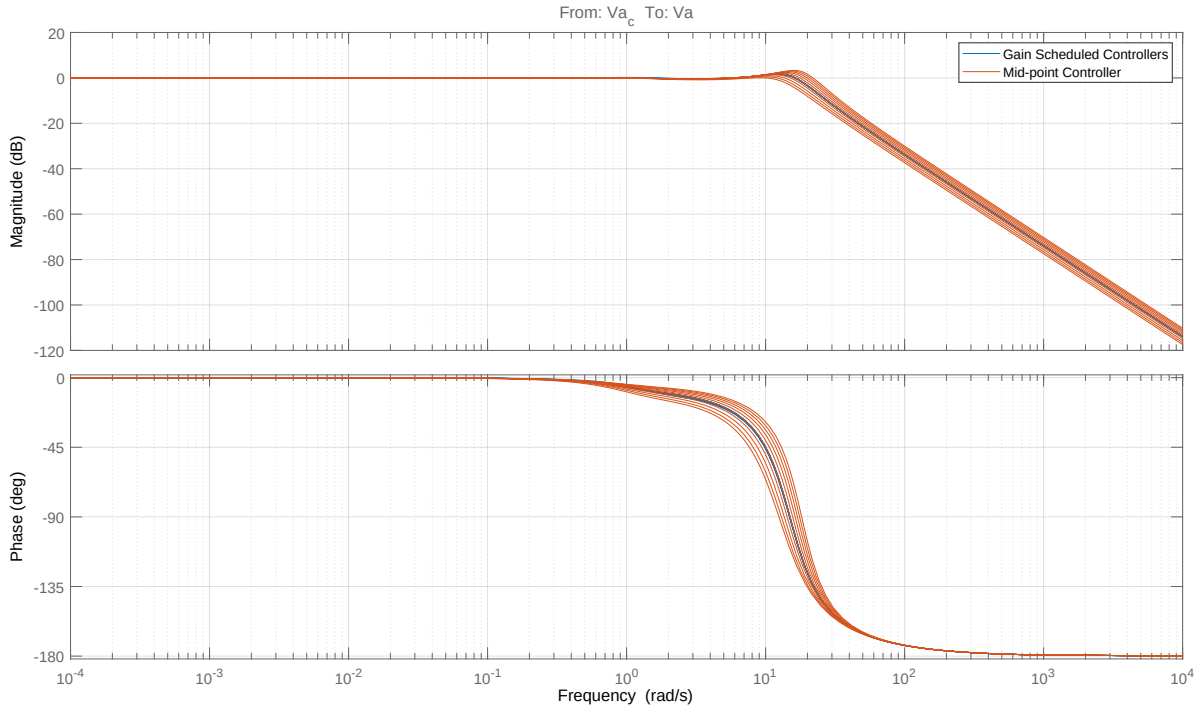


Figure 4.17: Airspeed controller, via propeller thrust, bode plot for $\frac{V_a}{V_{ac}}(s)$ comparing a mid-point and gain scheduled controller

From Figures 4.17, 4.18 and 4.19 we see that we expect a rise time of 0.1 s across all airspeeds, with a phase margin of 45° and an infinite gain margin. We expect virtually no resonant conditions in this control loop with the gain scheduled controller made use of.

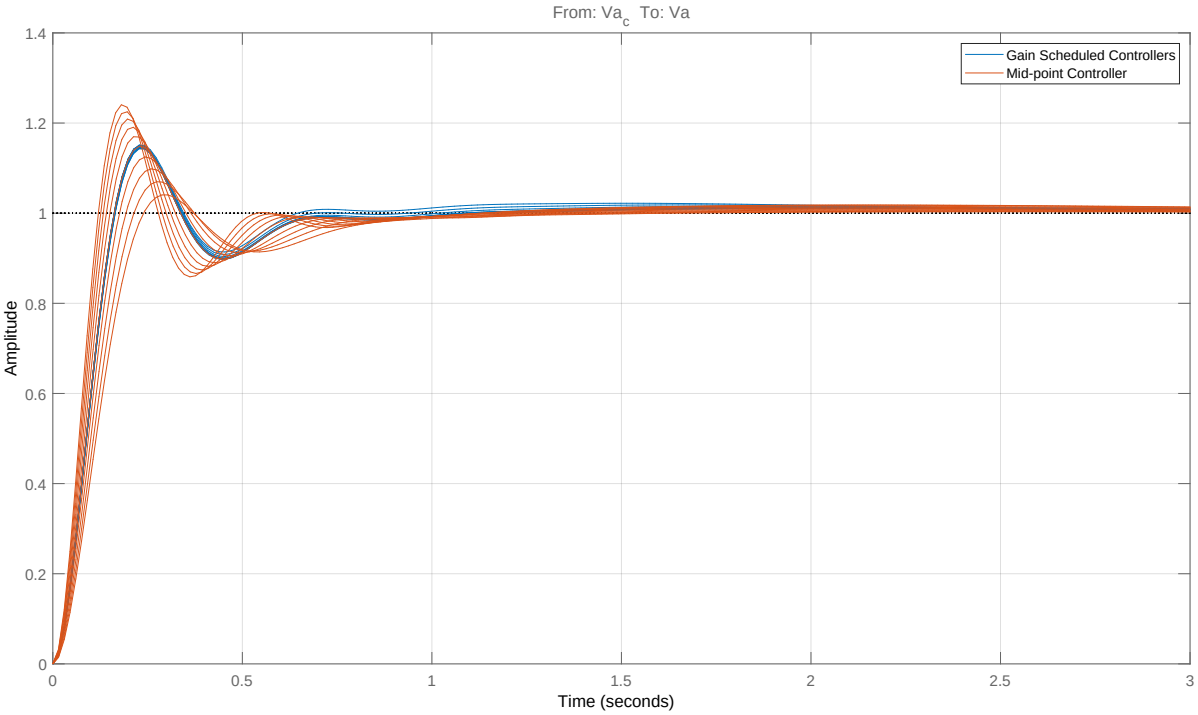


Figure 4.18: Airspeed controller, via propeller thrust, step plot for $\frac{V_a}{V_{a_c}}(s)$ comparing a mid-point and gain scheduled controller

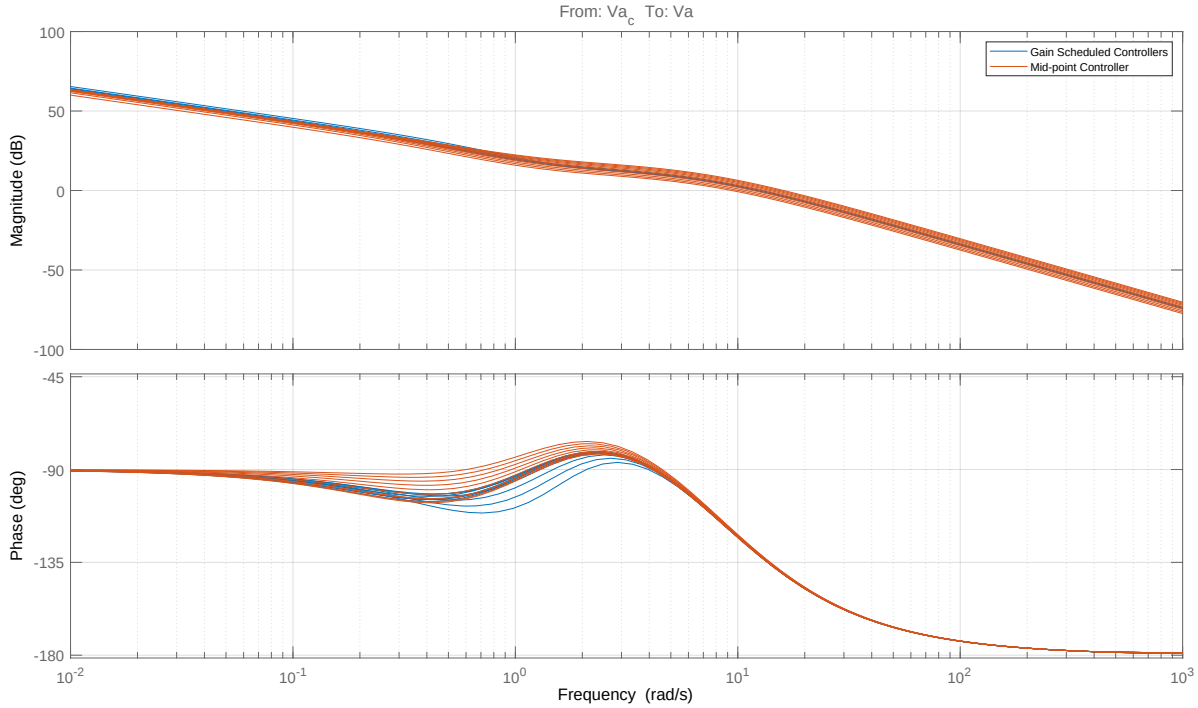


Figure 4.19: Airspeed controller, via propeller thrust, open loop bode plot for $\frac{V_a}{V_{a_c}}(s)$ comparing a mid-point and gain scheduled controller

Airspeed Control via Pitch Angle

For airspeed control we will make use of a standard PIDF controller topology, or

$$C_{V_{a_\theta}} = K_p + K_i \frac{1}{s} + K_d \frac{sN}{s+N}$$

The tuning goals for the airspeed control via pitch angle are denoted in Listing B.4. From these, our gain-scheduled controllers, are designed. We can then consider the performance of these controllers, and compare this with the mid-point controller discussed in the preceding sections in order to determine if gain-scheduling is required for the system. These results are denoted in Figures 4.20, 4.21 and 4.22, where it should be clear that the gain-scheduled controller produces much more consistent, superior performance by all metrics.

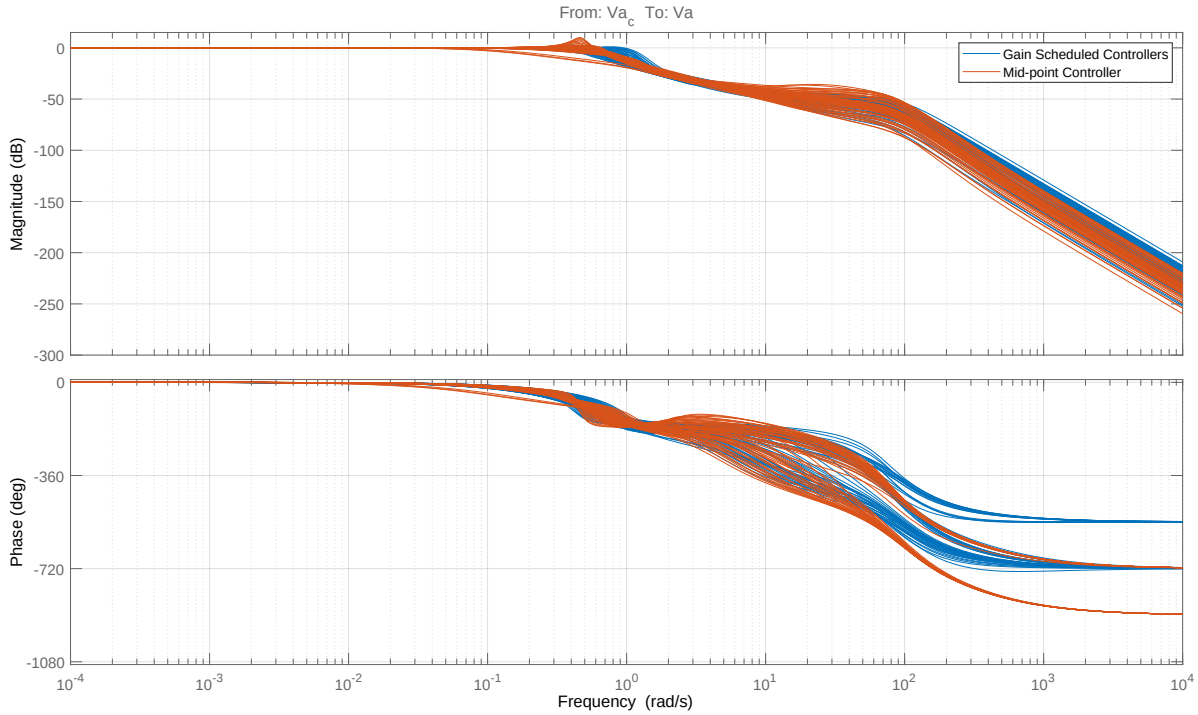


Figure 4.20: Airspeed controller, via pitch angle, bode plot for $\frac{V_a}{V_{a_c}}(s)$ comparing a mid-point and gain scheduled controller

From Figures 4.17, 4.18 and 4.19 we see that we expect a rise time of between 2 and 6 s across all (V_a, α) combinations. We have a phase margin of close to 90° for all cases and a gain margin of 10-20 dB. We expect virtually no resonant conditions in this control loop with the gain scheduled controller made use of.

It should be noted that these dynamics are substantially slower than those seen when controlling airspeed with propeller speed - this is intentional as controlling airspeed with pitch angle is not as straightforward a process as with a propeller, so forcing slower

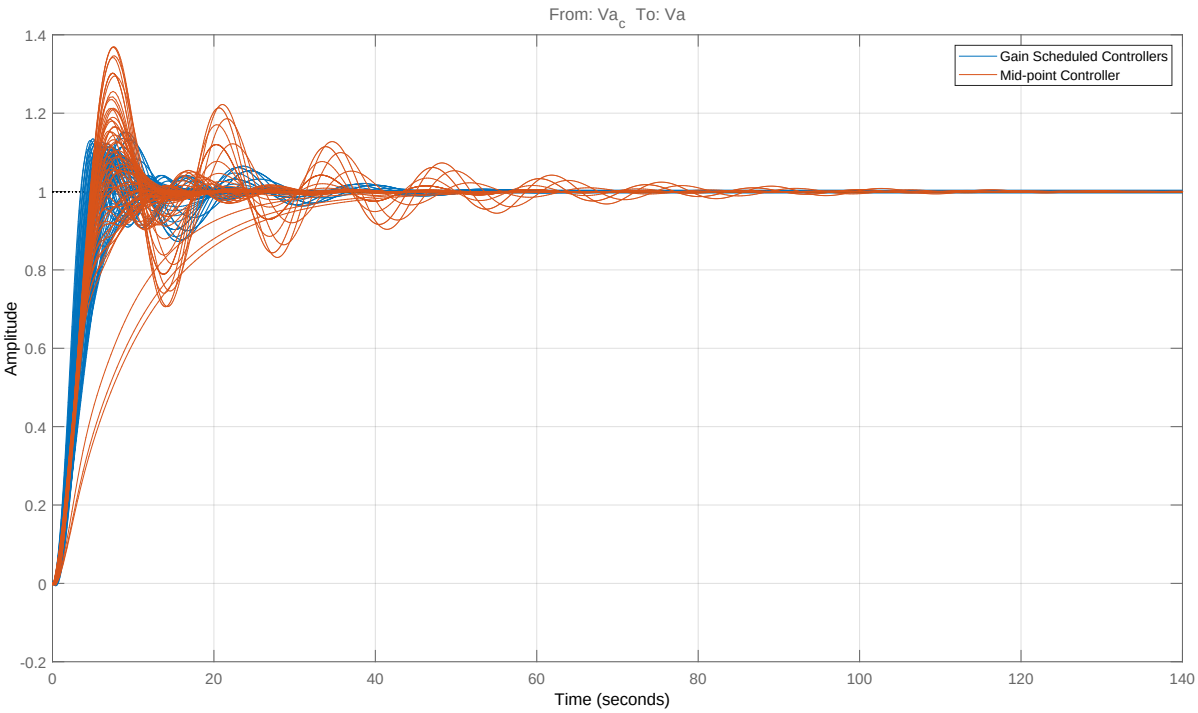


Figure 4.21: Airspeed controller, via pitch angle, step plot for $\frac{V_a}{V_{a_c}}(s)$ comparing a mid-point and gain scheduled controller

dynamics helps to alleviate some of the issues encountered. It is also a set of controllers that is only very rarely active, so has not been optimised fully and there is scope to improve on the performance if this functionality becomes more critical to the higher level application in any extensions of this work.

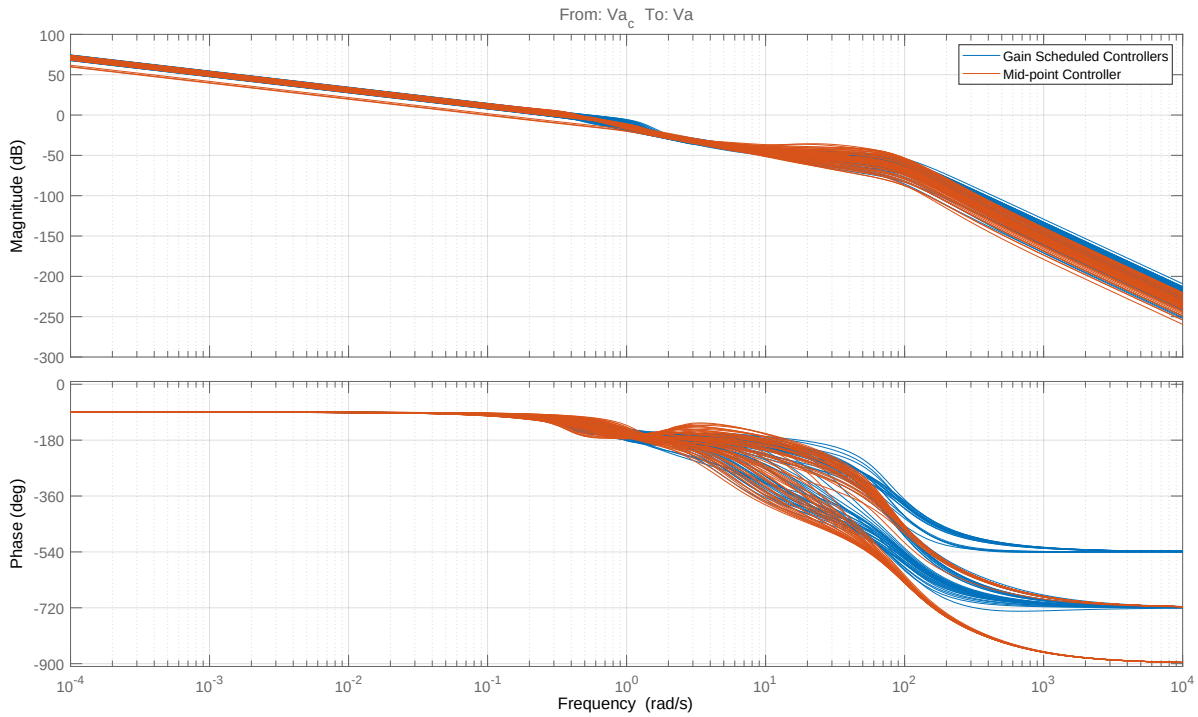


Figure 4.22: Airspeed controller, via pitch angle, open loop bode plot for $\frac{V_a}{V_{a_c}}(s)$ comparing a mid-point and gain scheduled controller

4.2.4 Longitudinal Controller State Machine

In order to handle events such as take-off, some mechanism is needed by which we can ensure adequate operation. As such, this work makes use of the state machine presented in [21], which has been extended and is depicted in Figure 4.23. This is implemented by making use of Simulink's Stateflow toolbox [56] and this implementation is denoted in Figure A.28.

From Figure A.28 it should be clear that the system presented in Figure 4.23 has been extended by adding hysteresis to the h_{hold} band which creates the altitude hold zone, this is to alleviate the state switching problem one would encounter if they were on the transition for a non-negligible amount of time. We have also added the ability to bypass the takeoff state and move straight into the altitude hold state so that testing of the system is made easier because we can start simulations already in-flight.

The parameters used in the state machine implementation are provided in Table 4.1.

This state machine works by ensuring that if a $h \leq (h_c + h_{\text{hold}})$ we shift the system into a glider mode whereby it will lose altitude and attempt to keep airspeed constant with pitch angle (hence the requirement for an airspeed controller as a function of pitch angle) with a zero throttle command. Conversely, should $h \geq (h_c - h_{\text{hold}})$, we give the aircraft maximum throttle and start to climb at the fastest rate possible - whereby we

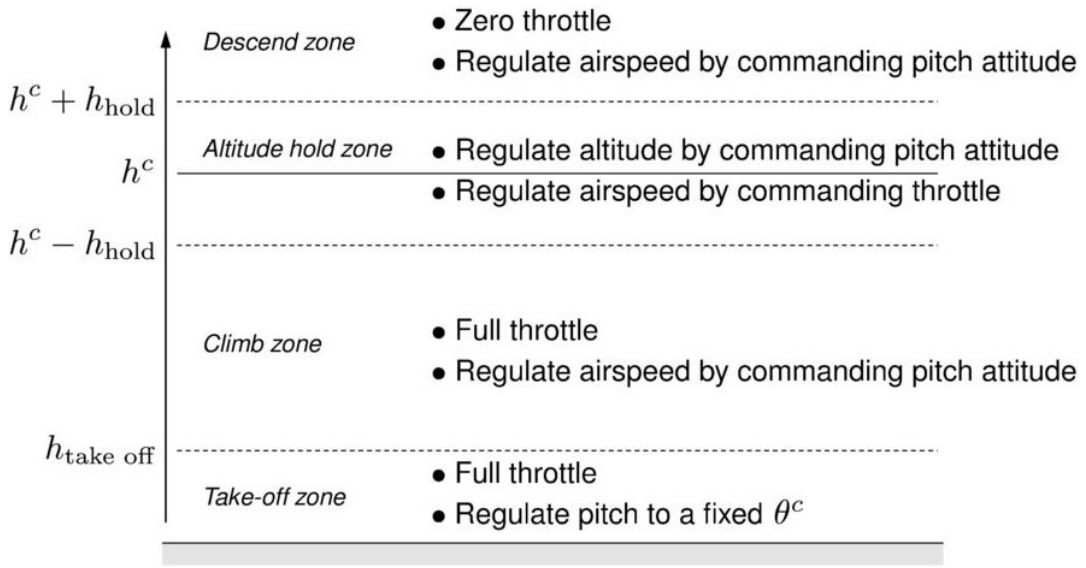


Figure 4.23: Longitudinal autopilot state machine implementation [21]

Table 4.1: Longitudinal State Machine Parameters

Parameter	Value	Unit
$h_{\text{take off}}$	10	m
h_{hold}	100	m
$\theta_{\text{take off}}$	30	°
Hysteresis Fraction	0.05	Unitless

also make use of the airspeed controller as a function of pitch angle to ensure that we do not climb at a dangerous rate and to attempt to avoid stall conditions. During take-off, we simply set our pitch angle command to a fixed value and apply full throttle so that we ascend as quickly as possible along a constant flight path angle. Most of our time will be spent in the altitude hold zone, where we regulate altitude by controlling pitch angle, and regulate airspeed by controlling throttle (or propeller thrust/speed).

All of the required controllers to implement this state machine have been developed in the preceding sections, with the state machine simply determining how the system should switch between them.

4.2.5 Longitudinal Controller Verification

Before the longitudinal controllers are verified - some non-linearities are included in the controller models. The first of these effects is integrator wind-up protection, where a simple integrator-freeze methodology is utilized, this is simply to stop the integrator present in the PIDF controllers from "winding up", which can occur if we encounter a

saturation limit on the output of the controller and the system's large signal response causes a longer delay than would be expected from the small signal analysis by which the integrator was designed. Since we will have a non-zero error for an extended period of time, the integrator "winds up" and will require an error of the opposite sign for an extended period to then wind down - which will general result in significant overshoot. As such, we simply constrain the integrator to only be active if we are not in a saturation condition - this is most easily illustrated using the Simulink block diagram implementation depicted in Figure 4.24, where we see that our integrator is simply fed a zero value if Pre-Saturation $\neq$ Post-Saturation, otherwise is fed the normal value.

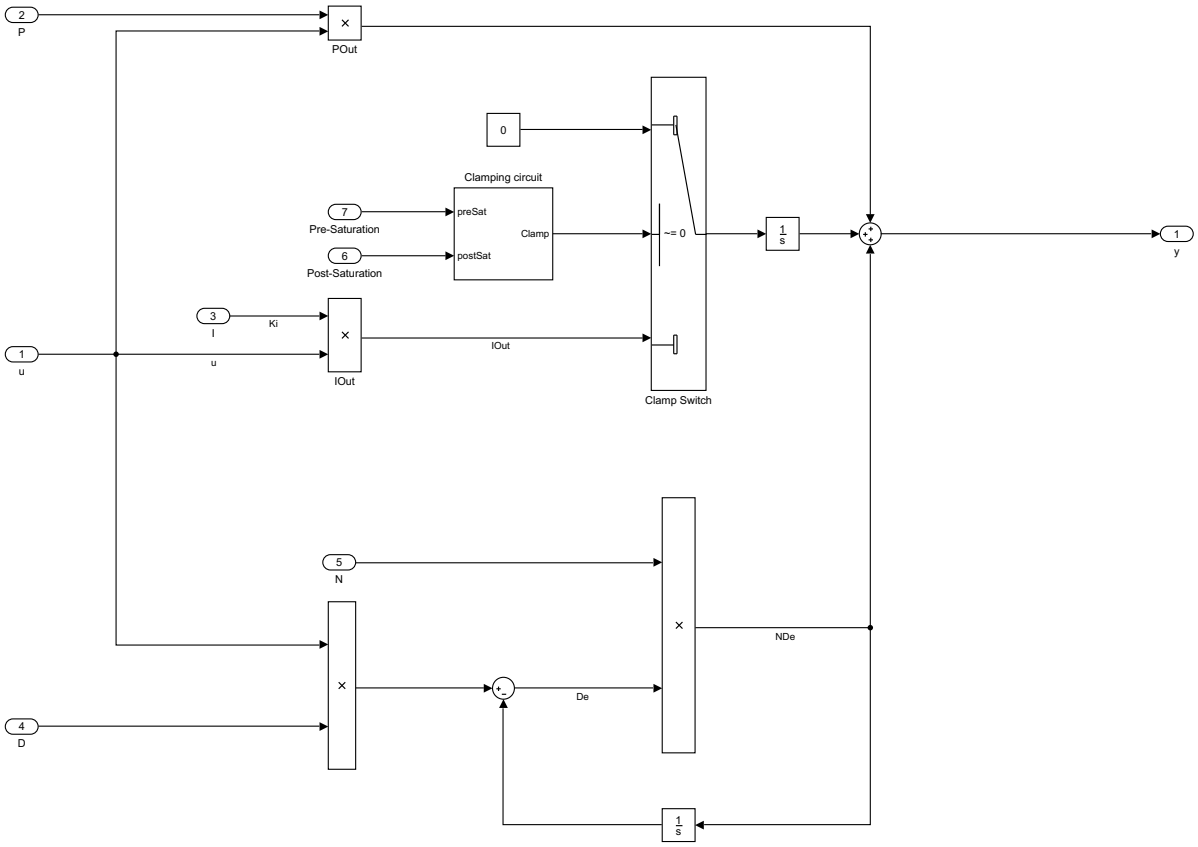


Figure 4.24: Integrator freeze implementation

The other non-linearity introduced is a rate limit on the altitude command so that we constrain this to:

$$\left|\frac{dh_c}{dt}\right| \leq V_a \sin\left(\frac{\theta_{max}}{2}\right) \tag{4.2}$$

We summarise our controller saturation parameters in Table 4.2.

With these defined, the longitudinal controllers are verified making use of the implementation depicted in Figure A.29, where it should be noted that (V_a, α) is not made

Table 4.2: Longitudinal Controller Saturation Parameters

Parameter	**Value**	**Unit**
$\delta_{e_{max}}$	0.523	rad
$\delta_{e_{min}}$	-0.523	rad
θ_{max}	1.047	rad
θ_{min}	-1.047	rad
$\delta_{t_{max}}$	1	Unitless
$\delta_{t_{min}}$	0	Unitless

use of for the altitude controller, as was determined in the design. The results of the verification run, which makes use of the full 6DoF model, including the stall modelling, are depicted in Figures 4.25, 4.26, 4.27 and 4.28.

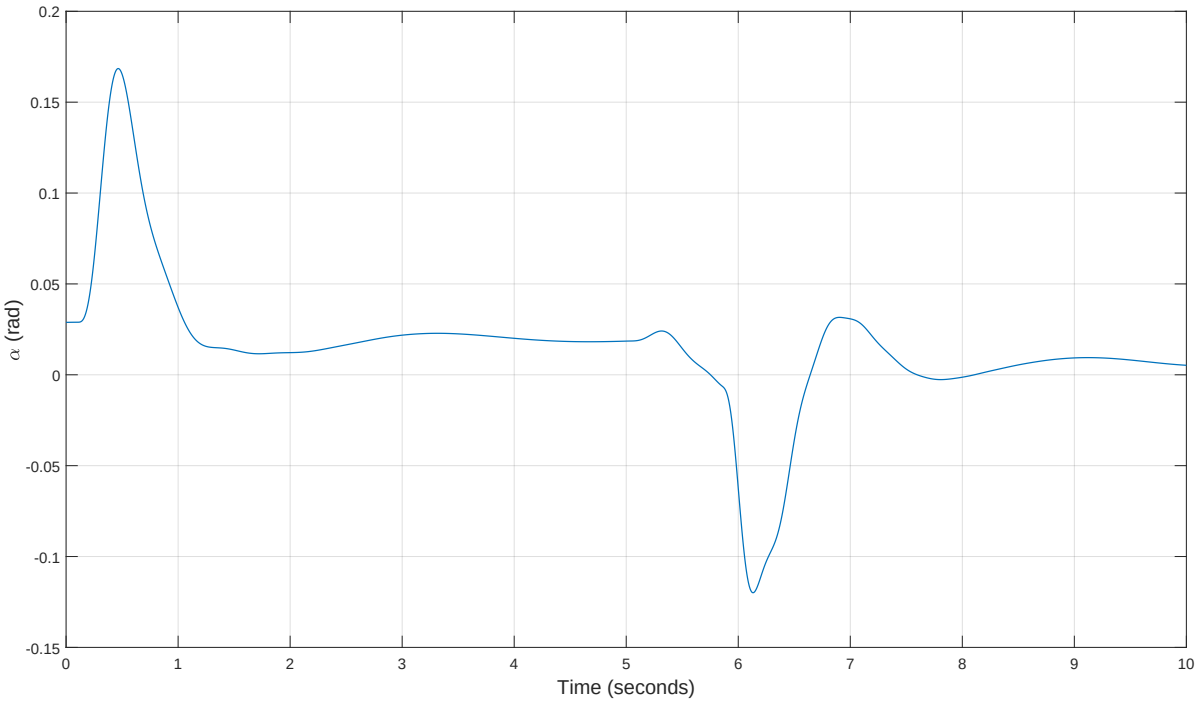


Figure 4.25: Longitudinal controller verification, α (rad)

These results illustrate the longitudinal controllers operate as expected, with particular attention being drawn to Figure 4.25, where we note that our angle of attack is constantly kept below the stall value - with a maximum value of $\alpha_{max} = 0.169$ rad for a large step input. Indeed, this justifies the reduced transient performance from the absolute maximum possible $\dot{h}$ as was calculated in Equation (4.1), where we see that our altitude

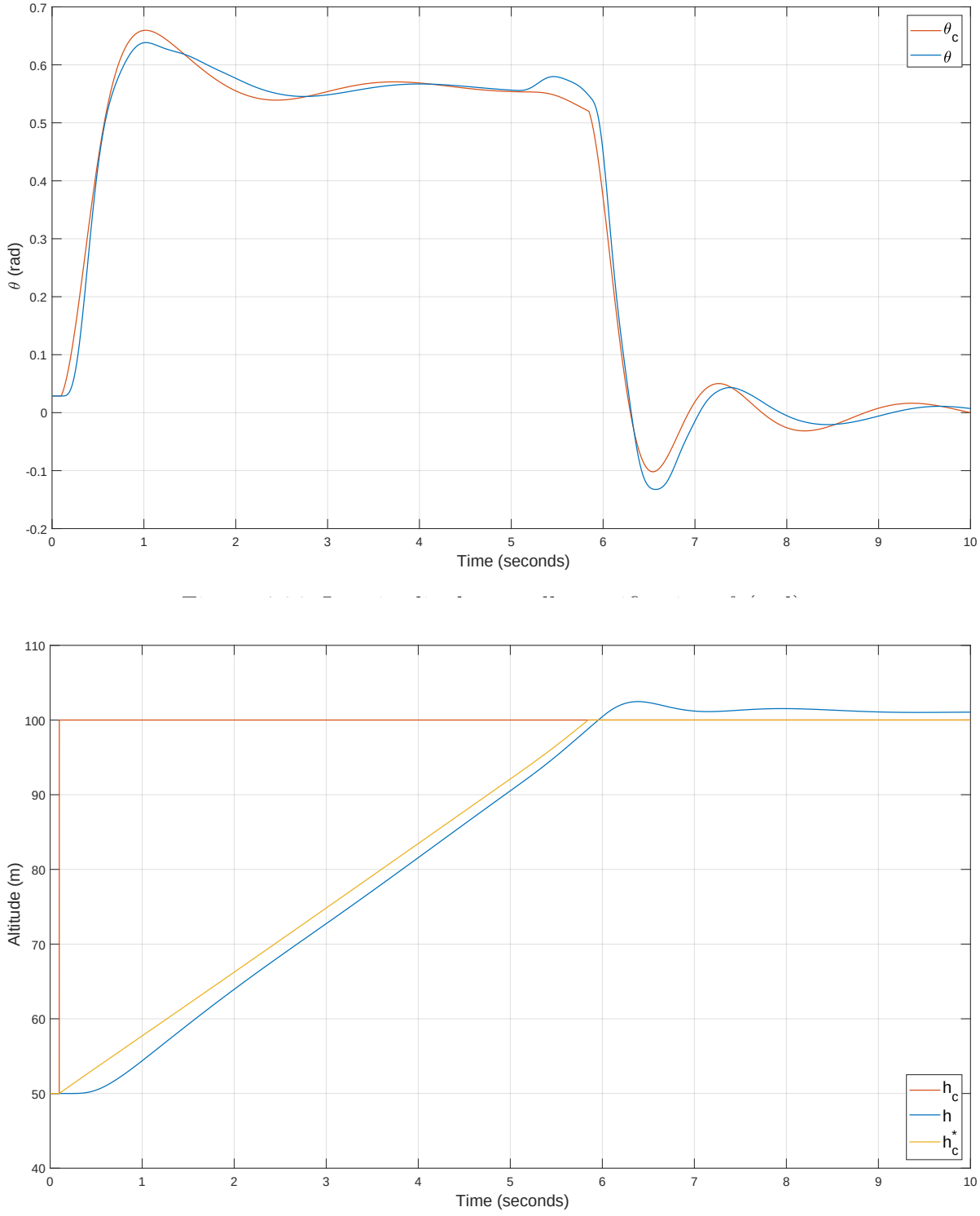


Figure 4.27: Longitudinal controller verification, h (m)

controller allows us to provide a safety factor on the stall angle of

$$SF = \frac{\alpha_0}{\alpha_{max}} = 1.58$$

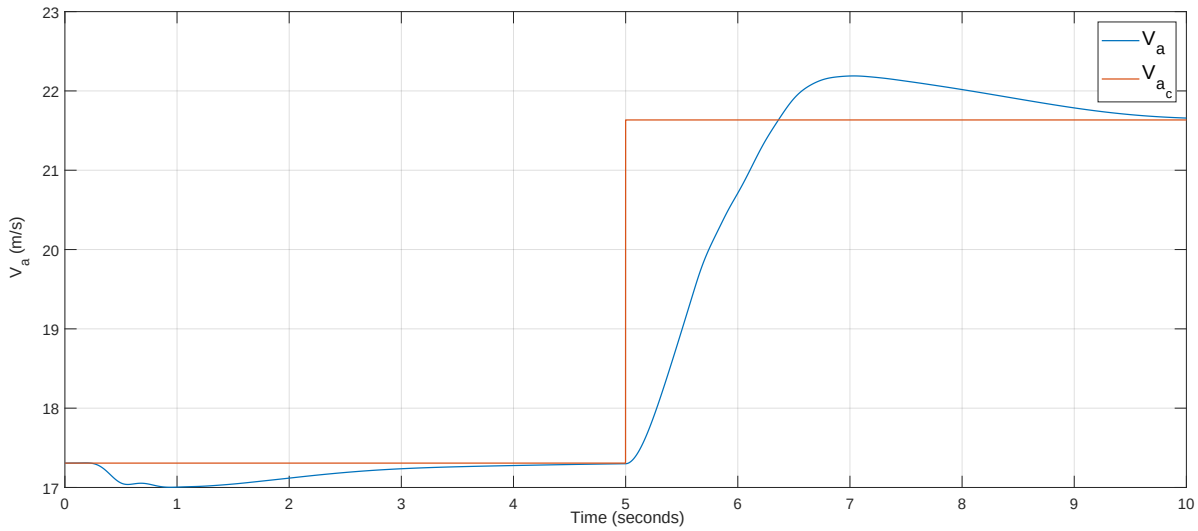


Figure 4.28: Longitudinal controller verification, V_a (m/s)

This safety factor provides adequate head room for allowing for turbulence and wind impacts which will cause additional variations in α, and also allows for unmodelled effects which might result in the early onset of stall.

We also see that in Figure 4.27, we illustrate the rate limited input command with h_c^*, which ensures that we do not cause too rapid a change in θ_c, as was previously indicated in Equation (4.2).

It should also be noted that in the altitude control loop, we have a slight deviation from constant steady state. However, this is within the kind of accuracy we can expect from a sensor measurement - so this isn't a problem in a realistic scenario, which will be shown later in the report.

We then consider the operation of our airspeed controller using pitch angle. The results of this are depicted in Figures 4.29, 4.30 and 4.31, where we see that we reach a point where we have constant airspeed at a constant descent angle. As expected, the response is quite slow because we have purposefully designed the controller to be slow given the more complicated dynamics involved in controlling airspeed via pitch angle, and because this controller is very rarely used in this particular application, as was discussed previously. It should also be noted that propeller speed was kept constant with a magnitude of the initial condition trim state throughout this verification run.

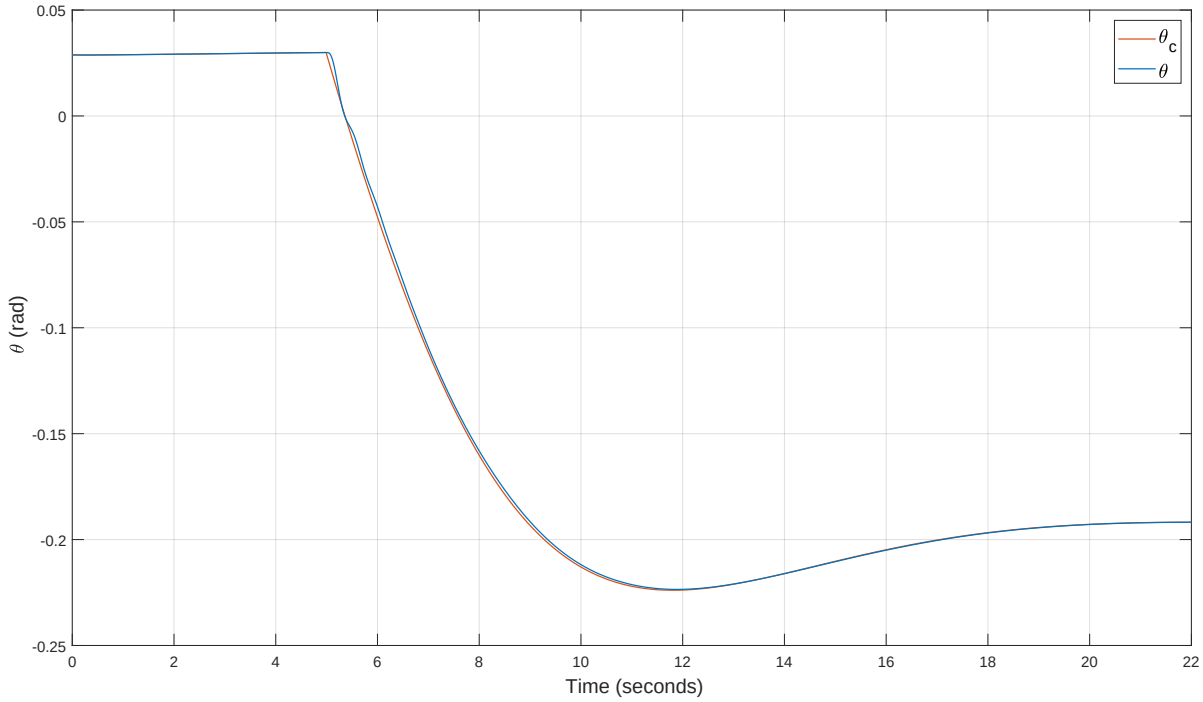


Figure 4.29: Longitudinal controller verification, airspeed via pitch angle control, θ (rad)

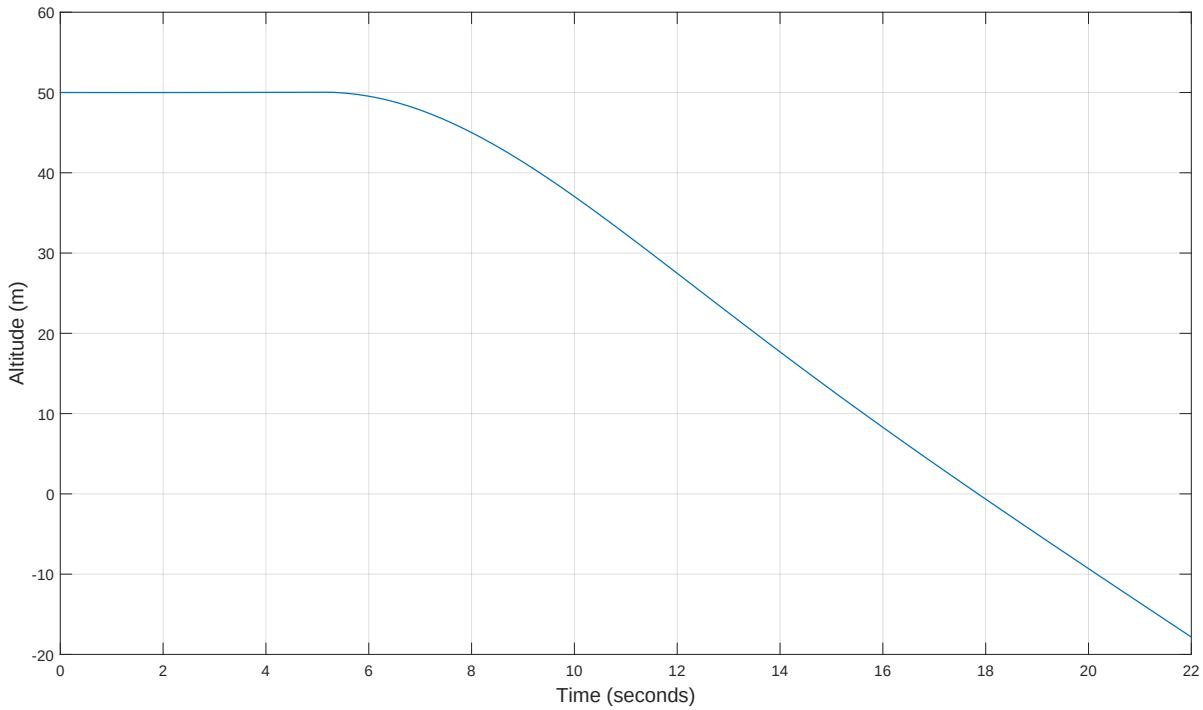


Figure 4.30: Longitudinal controller verification, airspeed via pitch angle control, h (m)

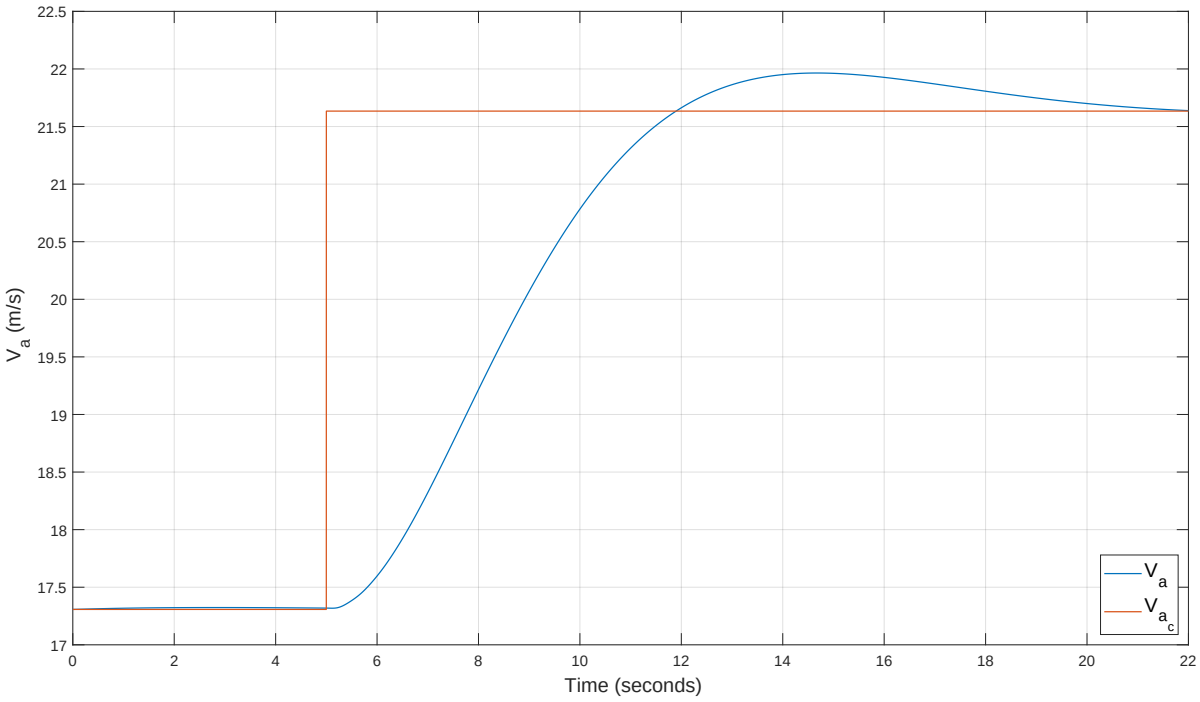


Figure 4.31: Longitudinal controller verification, airspeed via pitch angle control, V_a (m/s)

4.3 Lateral Controllers

As has been alluded to previously, we know that our lateral controllers are developed so as to control the course angle of the aircraft, χ, via the roll angle, ϕ. This is what is known as a coordinated turn, or a bank-to-turn, system, which was elaborated on in the discussion surrounding Figure 2.19. We also know from Section 2.7.1 that we can isolate our longitudinal and lateral dynamics, so we can ignore the effects of our longitudinal dynamics in the development of our lateral controllers - although it must be noted that we cannot verify these controllers without using the longitudinal controllers. This is because we need a mechanism which ensures that altitude is held constant, or else we will start descending as we roll because we will lose lift in the $\mathbf{k^i}$ direction. The rejection of this disturbance is the job of the altitude controller.

Similarly to our longitudinal controller, the lateral control system architecture selected is of the successive loop closure type, of which the standard implementation is depicted in Figure 4.32.

In Figure 4.32 we have made use of the observed transfer functions which were discussed in Section 2.7.3. However, we know from our previous discussion that we will extend this standard configuration, similarly again to the longitudinal controllers, to be of the gain-scheduled variety, albeit purely as a function of V_a, as opposed to (V_a, α). Also similarly to the longitudinal controllers, we include the effects of the roll rate damping

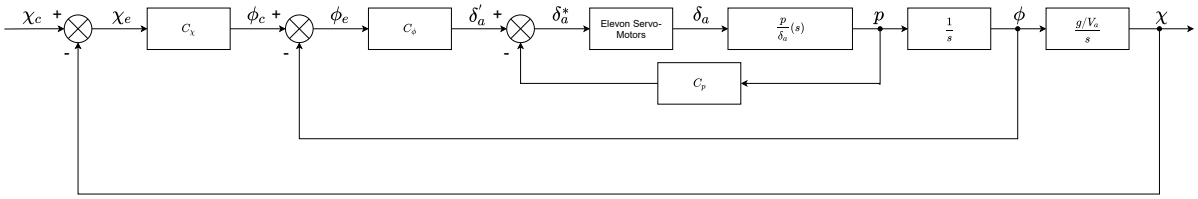


Figure 4.32: Successive loop feedback structure for the lateral controllers

controller, C_p, into the main roll controller, C_ϕ. This results in the configuration denoted in Figure 4.33.

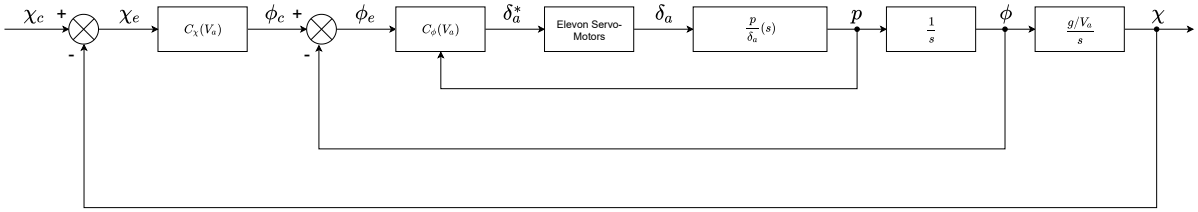


Figure 4.33: Gain scheduled successive loop feedback structure for the lateral controllers

As noted, we have made use of the transfer function estimates which were developed in Section 2.7.3, although we do not have a simple estimate for the transfer function $\frac{p}{\delta_a}(s)$. In the same manner as the longitudinal controller, we will make use of the $rsys = \text{balred}(sys, orders)$ function in MATLAB [55] to reduce our full order linearised models, which were presented in Section 2.7.3, to approximates which will be used in the design of the controllers.

In order to do this we have to work with four states for $\frac{p}{\delta_a}(s)$. This number was determined by experimentation, and is also an appropriate number of states to allow us to approximate $\frac{\phi}{p}(s)$ and $\frac{\chi}{\phi}(s)$. The comparison between the full order and reduced order models are shown in Figures 4.34, 4.35 and 4.36.
For the lateral models, we do not gain much by reducing the number of states, except that in the case of $\frac{\chi}{\delta_a}(s)$ we reduce the number of states by one. As such, we are essentially working with our full order model in almost all of the controller designs - this is not a bad thing if we can select tuning goals which allow the optimisation algorithms to find results with the higher order models, and if the higher order models do not contain dynamics which make the plant difficult to control.

Having established the linearisation of the system that will be used, as well as having generated a set of reduced order models, we can now focus on the design of the controllers themselves. The Simulink model used to tune the lateral controllers can be found in Figure A.27.

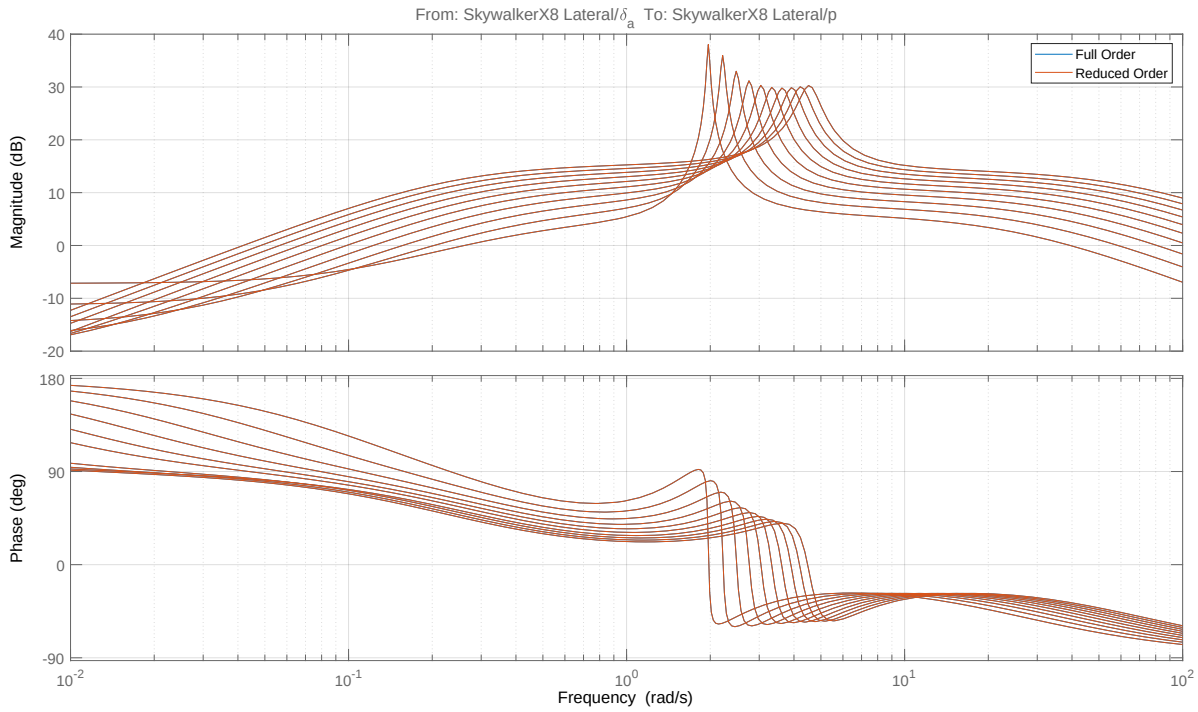


Figure 4.34: Reduced order linearisation of $\frac{p}{\delta_a}(s)$ in comparison to the full order model

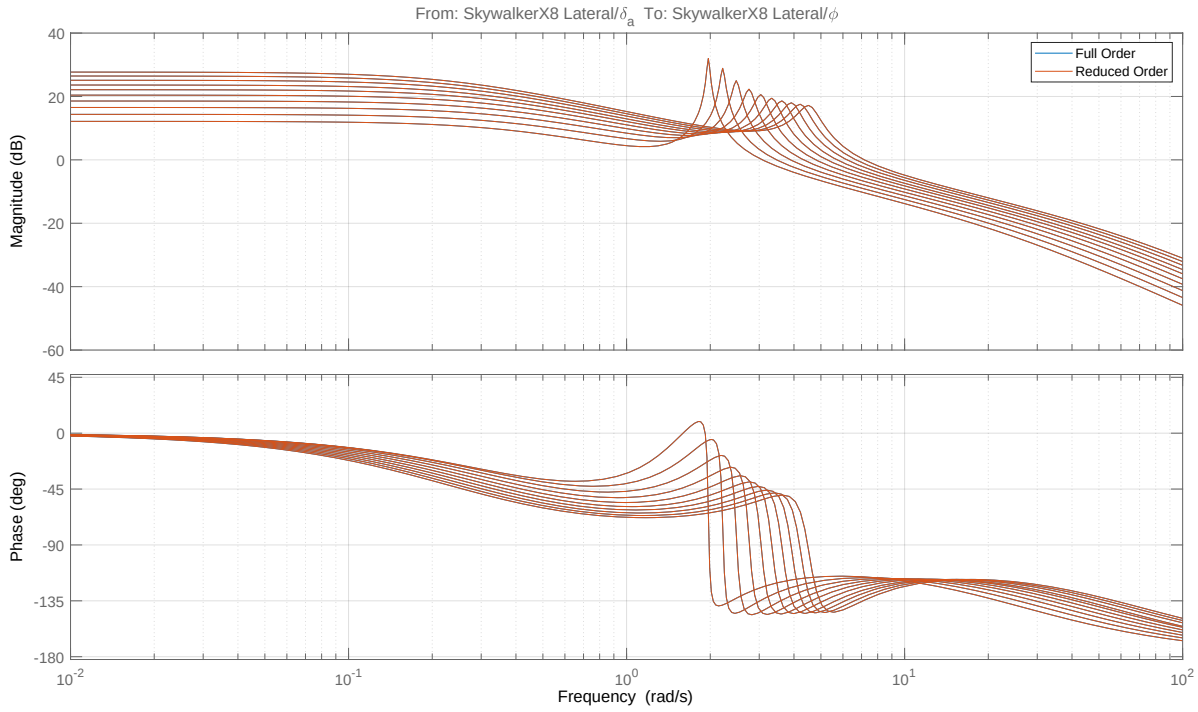


Figure 4.35: Reduced order linearisation of $\frac{\phi}{\delta_a}(s)$ in comparison to the full order model

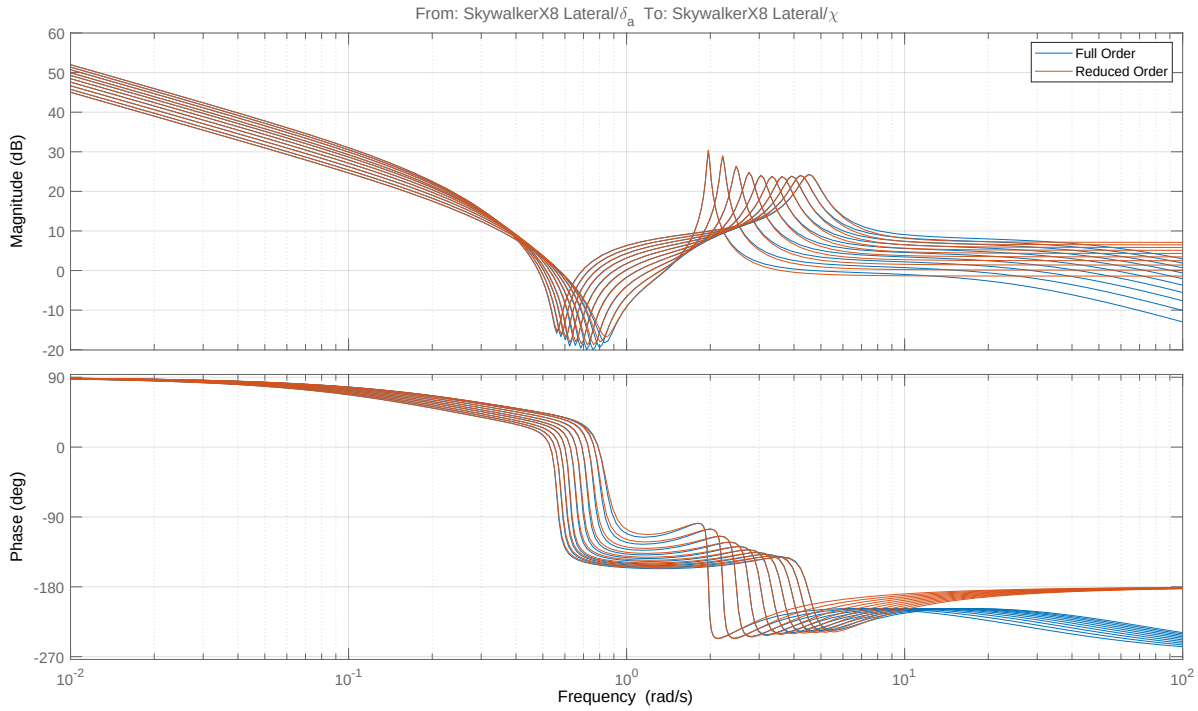


Figure 4.36: Reduced order linearisation of $\frac{\chi}{\delta_a}(s)$ in comparison to the full order model

4.3.1 Roll Angle Control

Similarly to the pitch angle controller, the roll angle controller, C_ϕ, will be a variant of the classical PIDF controller.

We know that we will usually not require a non-zero δ_a in steady flight. As such, we know that the integrator is not a hard requirement, but is still included in the initial controller topology in order to the remove steady state tracking error as we know that $\dot{\chi}$ depends on ensuring that we maintain a fixed roll angle to some commanded value, for which we will need an integrator. However, if the tuning algorithms result in very low gains for the K_i term, we know that we can get satisfactory performance without the need for the integrator.

Similarly to the pitch rate damping consideration, we make the approximation of $p \approx \dot{\phi}$, and as such we can augment the PIDF controller such that the derivative is not calculated, but simply used directly from our measured roll rate gyroscope. In other words, the PIDF controller will be of the form:

$$C_{1_\phi} = K_p + K_i\frac{1}{s} + K_d\frac{p}{\phi_e}\frac{N}{s+N}$$

We perform a similar extension to the pitch rate controller in that we add a second degree of freedom to the controller. However, in the case of the roll angle controller,

an adjustment was made whereby the second degree of freedom was added onto the feedback term, ϕ, instead of in the feed-forward path. This is done because the lateral dynamics are not as easily controlled as the longitudinal dynamics - which is indicated clearly in the frequency responses seen from our linearisation performed in Section 2.7.3.

As such, we need to make use of the roll angle measurement to ensure adequate stability, because our lateral dynamics are inherently slow due to the much larger rotational inertia of the Skywalker X8 airframe in the roll direction (ie. $J_x x$), as per Table 2.2, so attempting to get these to be very fast with a feed-forward term will not accomplish as much as will the increased stability by using the roll angle itself in our controller algorithm. Similarly to the pitch controller, this has to be of the PDF type, because an integrator would cause δ_a^* to grow unbound. As such, the controller is of the form:

$$C_{2_\phi} = K_p = K_d \frac{sN}{s+N}$$

This then gives us the controller architecture as depicted in Figure 4.37.

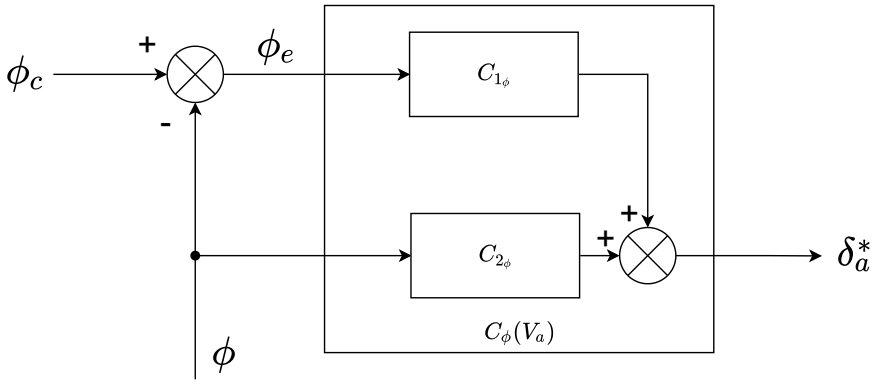


Figure 4.37: Roll angle controller architecture

Having defined the controller architecture, we can make use of the tuning goals defined in Listing B.5, which results in 10 controllers, each optimised at a value of V_a. The results of this design are depicted in Figures 4.38, 4.39 and 4.40, where a mid-point controller has been used to illustrate if gain scheduling is necessary.

From the results depicted it is clear that making use of a gain-scheduled controller is necessary with the tuning goals used, because if we do not we have an instability. This unstable mode is illustrated effectively in Figure 4.40, where we see we have 180° phase with a 0 dB gain at 2.7 rad/s.

Another consideration is whether or not we need an integrator in our controller topology. If this was not the case, we would expect our tuning algorithm to assign a near-zero value to K_i. However, in this case the values for K_i range from 16 to 31 - which, using

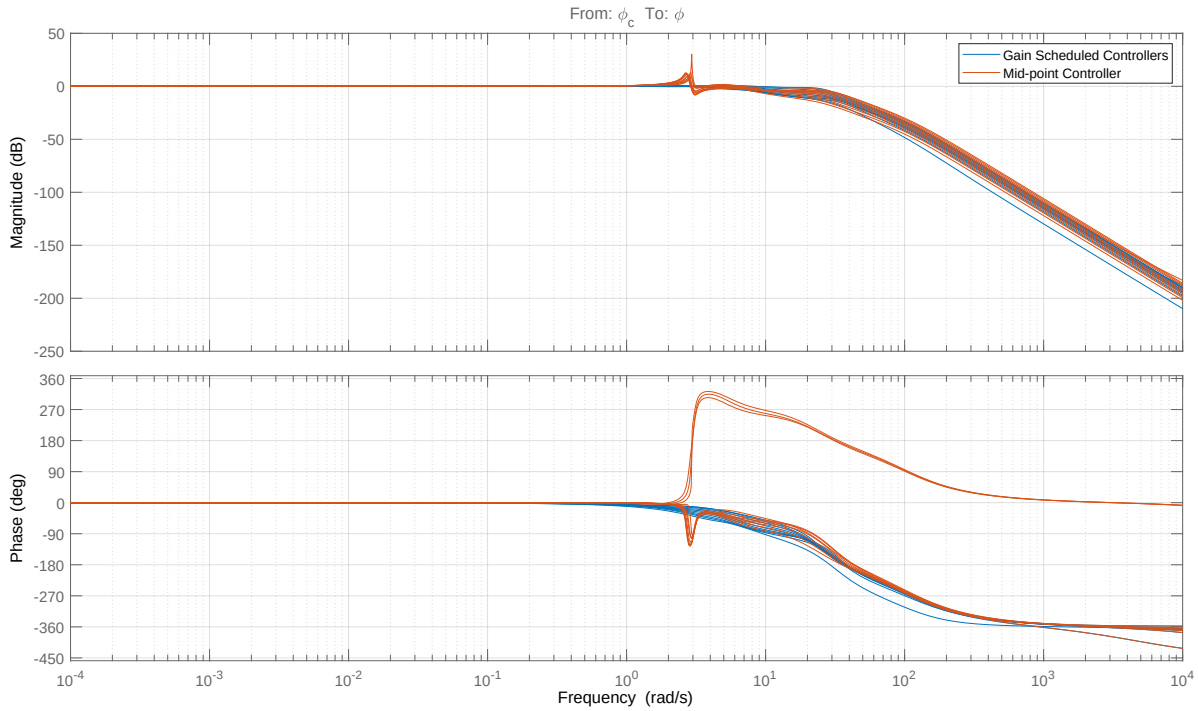


Figure 4.38: Roll angle controller bode plot for $\frac{\phi}{\phi_c}(s)$, comparing a mid-point and gain scheduled controller

K_p as a reference with gains ranging from 0.1 to 6.2 - is clearly non-negligible, so an integrator term is required.

Since we made use of the full order model for $\frac{\phi}{\delta_a}$, we need not be concerned with the reliability of our results in the higher frequency domain as was the case with the pitch angle controller.

We can also note that from Figure 2.20, our maximum expected roll rate, p_{max} over the range of airspeeds considered, is between 1.34 and 4.41 rad/s. These rates would result in a change in ϕ of 1 rad taking between 0.23-0.75 s. From Figure 4.39 we note that our rise times vary from between 0.1 to 0.35 s (where the discrepancy is introduced due to simplified expressions being used to determine p_{max}), indicating that we are likely very close to a completely optimal system. As such, these are quite impressive controller designs and illustrate that near-optimal performance is achievable with a classical controller topology and a more exotic controller need not be considered.

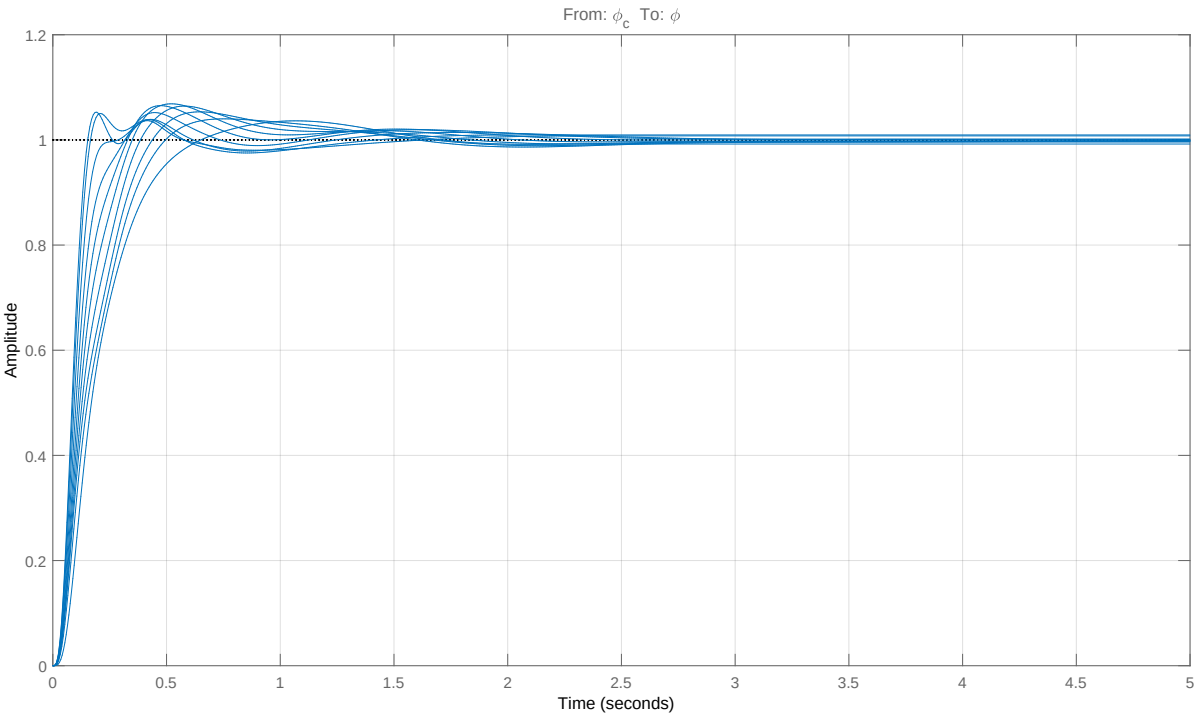


Figure 4.39: Roll angle controller step plot for $\frac{\phi}{\phi_c}(s)$, with mid-point controller omitted due to instabilities

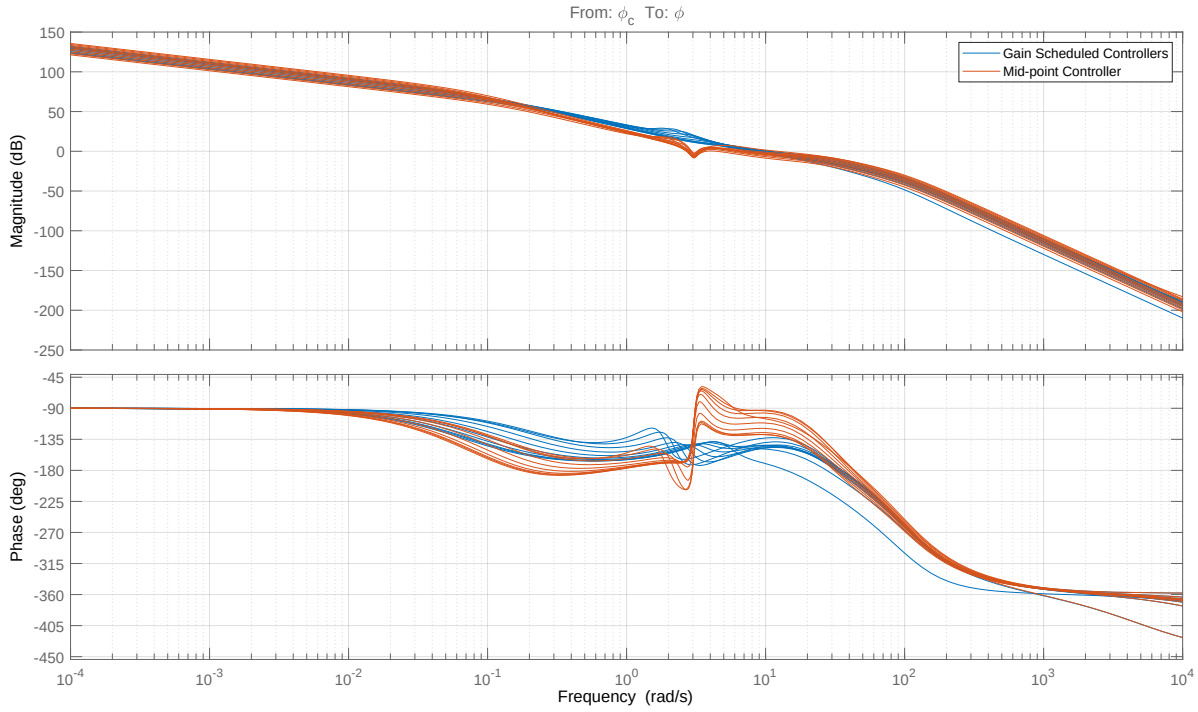


Figure 4.40: Roll angle controller open-loop bode plot for $\frac{\phi}{\phi_c}(s)$, comparing a mid-point and gain scheduled controller

4.3.2 Course Angle Control

Figure 4.33 illustrates the the course angle controller is built around the roll angle controller. We know from Figure 4.38 that we can approximate $\frac{\phi}{\phi_c}(s)$ as unity gain until close to 5 rad/s. As such, as long as we ensure sufficient roll off at this point, we can approximate

$$\frac{\chi}{\phi_c}(s) \approx \frac{g/V_a}{s}$$

Which is used to simplify our plant model in the tuning of the controllers.

Given the simplicity of the plant, we start our tuning algorithms with a standard PI controller topology such that:

$$C_\chi = K_p + K_i \frac{1}{s}$$

This is different to what was done for the altitude controller, and is done in order to determine if a simpler controller can be utilized because we know that our lateral dynamics are significantly lower than the longitudinal dynamics. However, if this is not the case, the PIDF controller can be reverted to. If adequate performance can be provided by a simple PI controller, which is likely the case given the plant, then increasing controller complexity will add very little except for additional implementation overhead.

We then make use of this controller topology, with the tuning goals listed in Listing B.6, in order to design the required controllers. This results in 10 course angle controllers, one for each of the selected values of V_a. The results of this design are depicted in Figures 4.41, 4.42 and 4.43, where a mid-point controller has been used to illustrate if gain scheduling is necessary.
From these results, it should be clear that the primary benefit of the gain scheduled controller is that we have a much lower overshoot in the time domain, with negligible impact on stability margins, or in the frequency domain as a whole. Given that the complexity of the controller has already been reduced by restricting it to be of the PI type, it was decided that the slight increase in controller complexity by making use of gain scheduling was a worthwhile trade-off for the increased performance in the time domain.

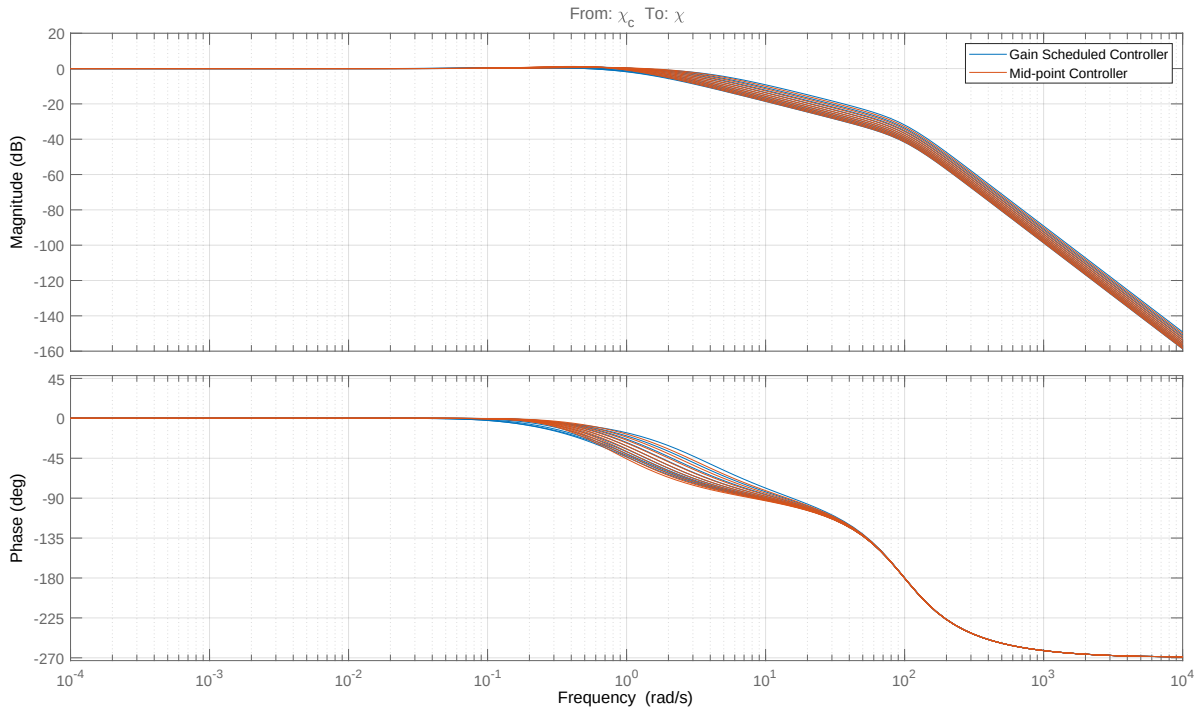


Figure 4.41: Course angle controller bode plot for $\frac{\chi}{\chi_c}(s)$, comparing a mid-point and gain scheduled controller

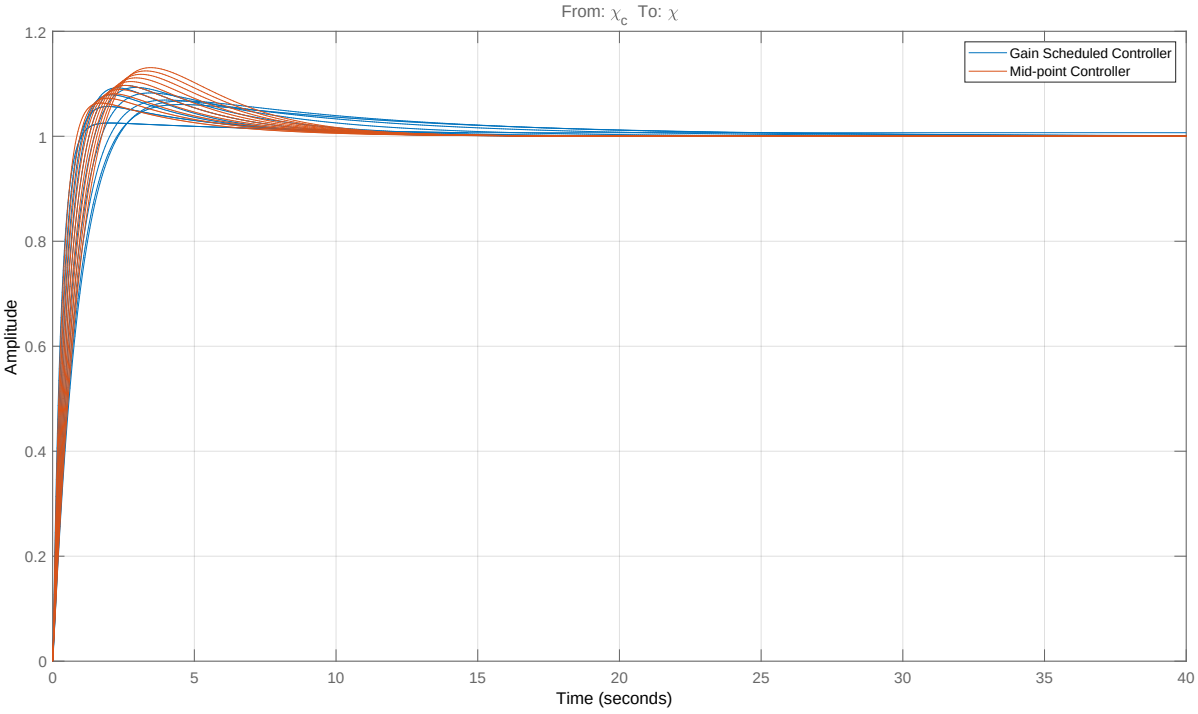


Figure 4.42: Course angle controller step plot for $\frac{\chi}{\chi_c}(s)$, comparing a mid-point and gain scheduled controller

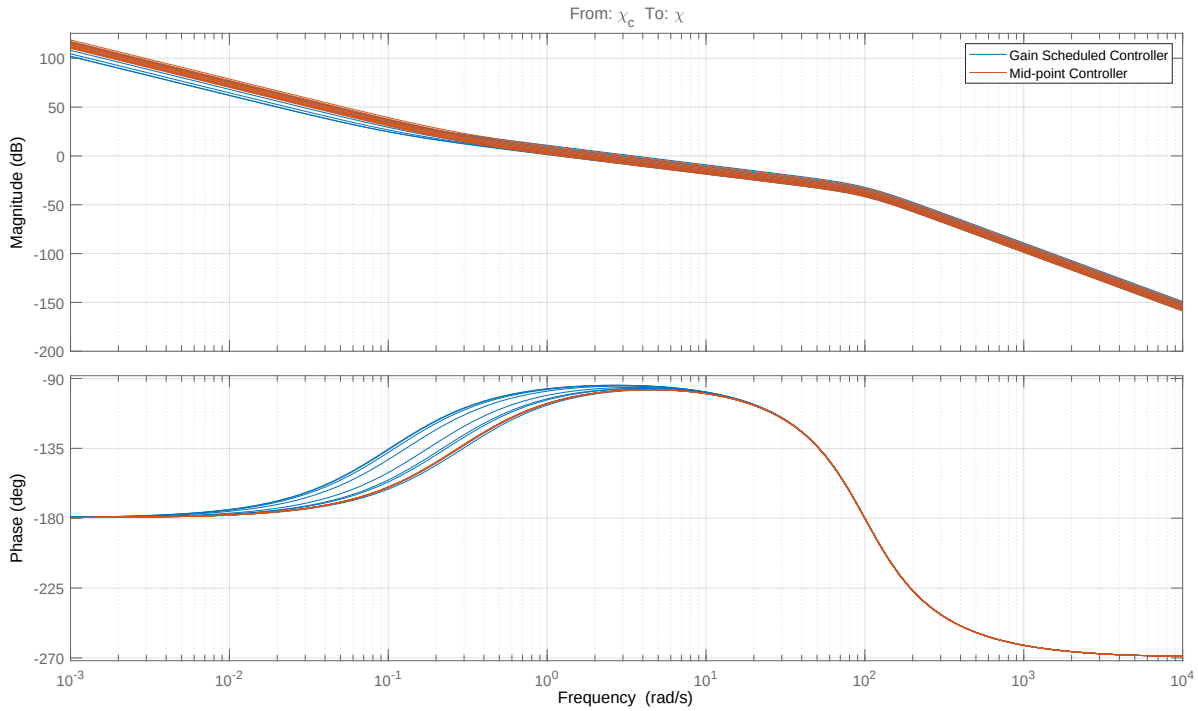


Figure 4.43: Course angle controller open-loop bode plot for $\frac{\chi}{\chi_c}(s)$, comparing a mid-point and gain scheduled controller

4.3.3 Lateral Controller Verification

The only remaining item to note before showing the results of the lateral controller verification is that we make use of the same integrator windup protection scheme as was noted in the longitudinal controller verification section, and as is depicted in Figure 4.24, where we simply "freeze" the integrator on output saturation. The discussion surrounding Figure 4.24 provides more detail on this.

The saturation limits for each of the lateral control parameters are listed in Table 4.3.

Table 4.3: Lateral Controller Saturation Parameters

Parameter	Value	Unit
$\delta_{a_{max}}$	0.523	rad
$\delta_{a_{min}}$	-0.523	rad
ϕ_{max}	0.523	rad
ϕ_{min}	-0.523	rad

The Simulink implementation of the lateral controller verification is depicted in Figure A.31. What is important to note here is that one needs to make use of the longitudinal control system to verify the lateral controllers because of the coordinated turn

methodology. If one did not have a longitudinal controller active, as soon as the aircraft rolled it would start to lose altitude - this is, of course, not the intended behaviour and so we allow our longitudinal controller to reject this disturbance so that we can more accurately review the performance of the lateral controllers. It should also be noted that in this model one can see that the altitude controller does not have any scheduling parameters being fed into it, as was discussed in its design.

The results of the verification run, which makes use of the full 6DoF model, including the stall modelling, are depicted in Figures 4.44, 4.45, 4.46 and 4.47. From Figure 4.44 it is clear that we get good set point tracking of our course angle, χ, with minimal overshoot, even with aggressive commands of 90° changes in course. We note from Figure 4.45 that our roll angle is controlled almost instantly (certainly much faster than the course angle dynamics), meaning that our plant reduction technique for course angle is valid. This is because, from the perspective of the course angle controller:

$$\frac{\phi}{\phi_c}(s) \approx 1$$

From Equation (2.66), restated here, where a zero-wind and zero-sideslip condition is assumed,

$$\dot{\chi}_{max} = \frac{g}{V_a} \tan \phi_{max}$$

we know that given our airspeed, $V_a = 17.3$ m/s, and roll angle command saturation, $\phi_{max} = 0.523$ rad, our maximum course angle angle rate is

$$\dot{\chi}_{max} = 0.327 \text{ rad/s}$$

As such, we expect our optimal response time for a course angle change of 90°, to be 4.8 s. We can see that our rise time (amount of time taken for a parameter to move from 10% - 90% of the command) in Figure 4.44, is 4.5 s, which is expected because we maintain our maximum roll angle for the majority of the coordinated turn manoeuvre. As such, we can see that this system is quantifiably as close to optimal as is physically possible for the airframe.

Figures 4.46 and 4.47 are included to illustrate the operation of the longitudinal controllers in rejecting the disturbance introduced on altitude by rolling the aircraft (which means that the longitudinal controllers need to provide additional lift to keep the altitude constant). From these it can be seen that the altitude lost is less than 1 m, which is less than 1% of our constant commanded altitude of 100 m. This is also the case for the airspeed, where a deviation of less than 1% is seen.

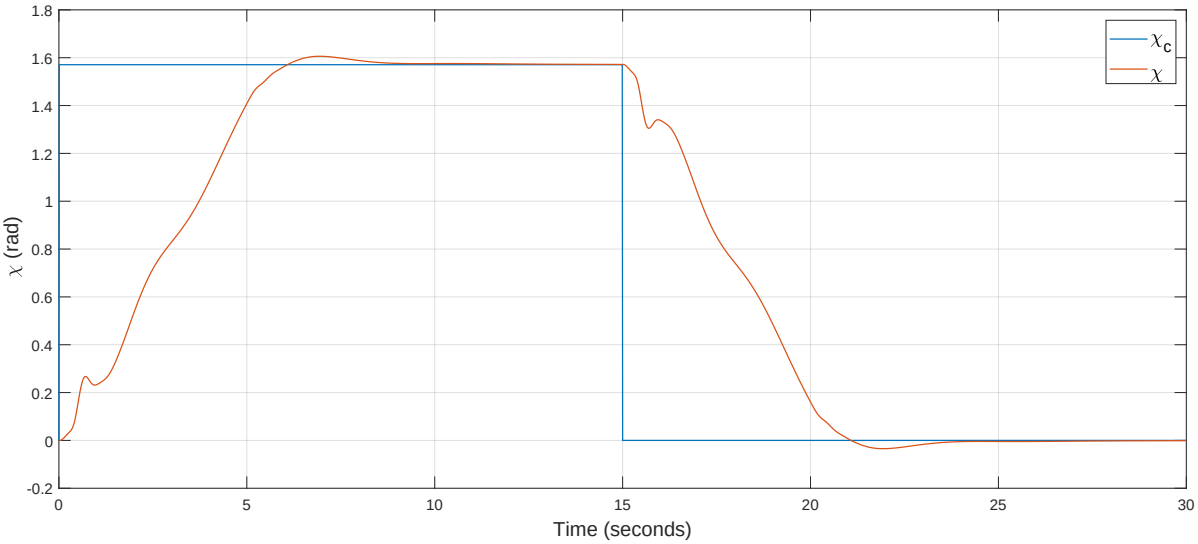


Figure 4.44: Lateral controller verification, χ (rad)

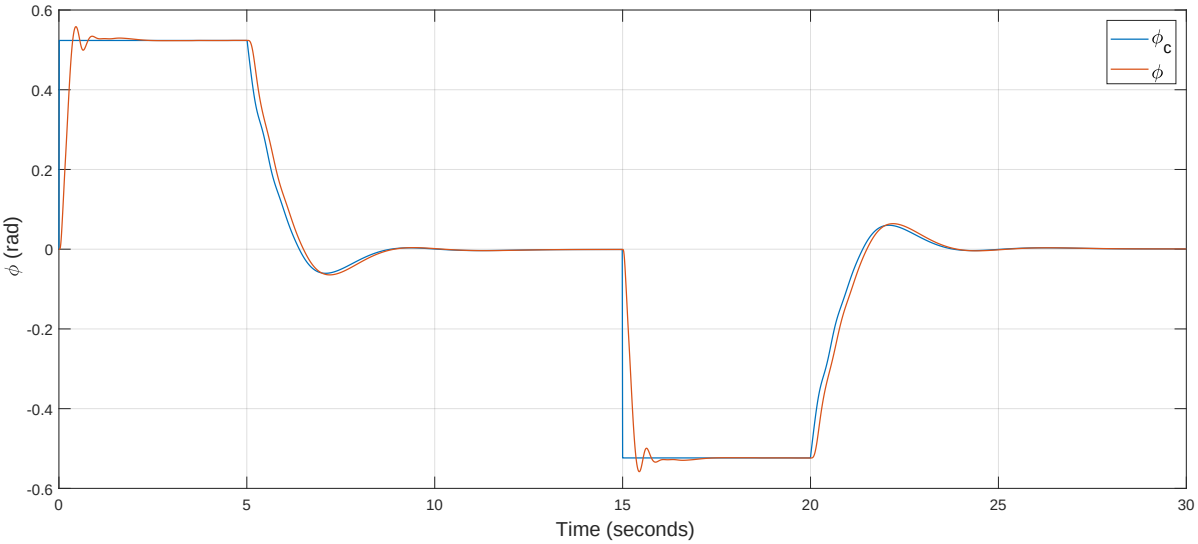


Figure 4.45: Lateral controller verification, ϕ (rad)

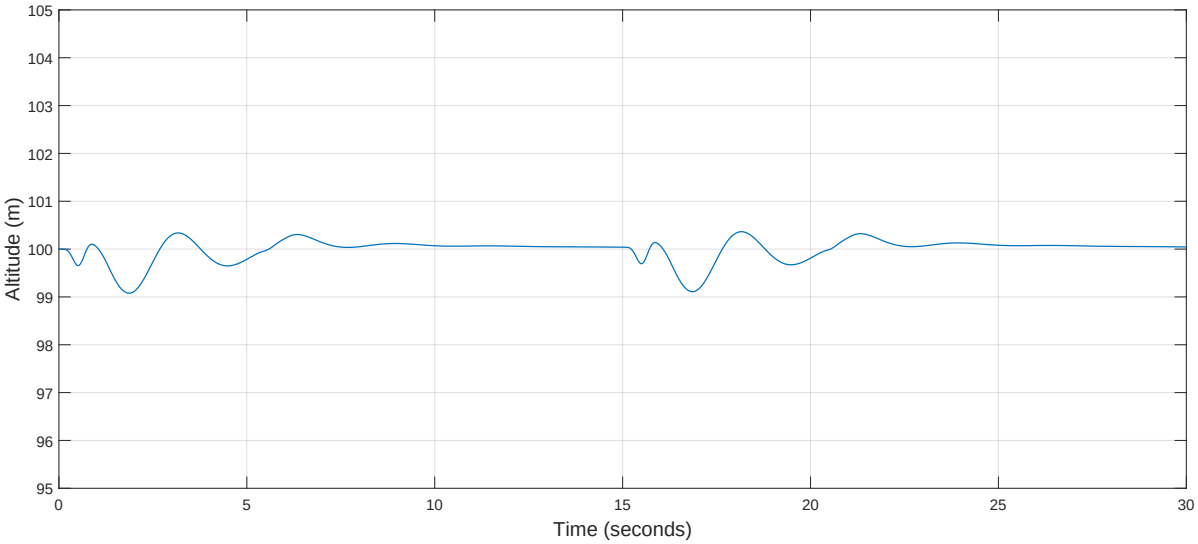


Figure 4.46: Lateral controller verification, h (m)

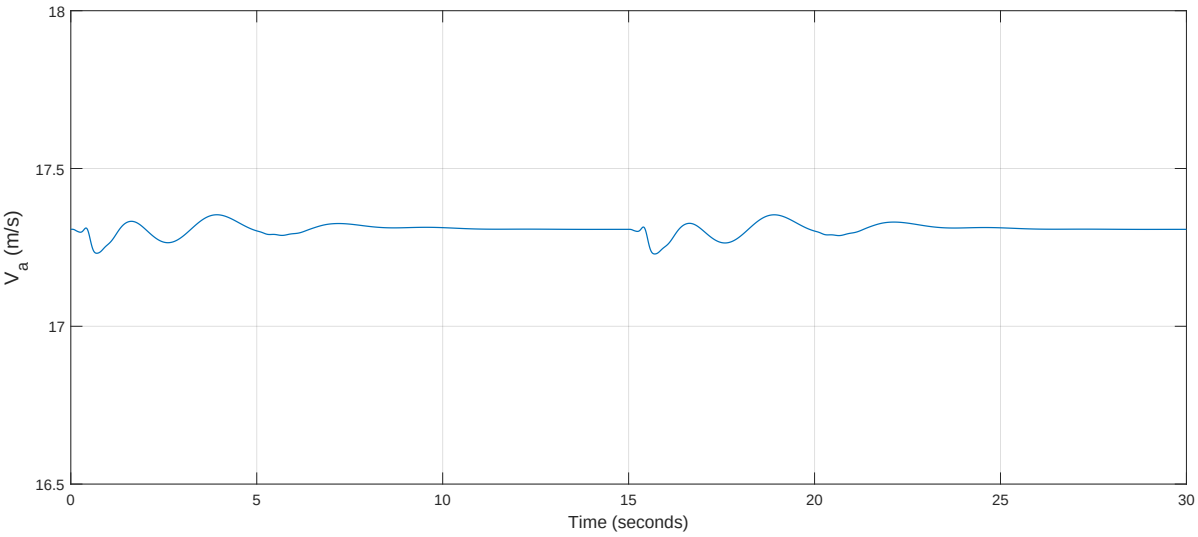


Figure 4.47: Lateral controller verification, V_a (m/s)

4.4 Digital Implementation

In order to eventually implement our control system on a digital platform - usually some form of digital microcontroller - we need to implement the algorithms in the digital domain so that they can be converted to difference equations which can then be written in software directly. This conversion from continuous to digital control is performed very conveniently in the Simulink environment.

Since all of the controllers utilised are built from simple continuous integrators, all that one needs to do is replace these continuous integrator blocks in the model with a discretised variant, select a controller sampling time based on the known system dynamics, and then retune the system using the same tuning goals as in the continuous case, except now with the discretised controllers in the loop. These can then be verified in the same manner as the continuous controllers in order to ensure that there is no significant performance degradation.

4.4.1 Discretisation of Controllers

In order to discretise the controllers, we need to convert all continuous operators, s, into their discrete counterparts, z. To do this, we make use of the bilinear, or Tustin, transform which is stated as:

$$s = \frac{2}{T_s}\frac{z-1}{z+1}$$

All of the controllers selected are of the PI, or PIDF type. Considering the PI type, we can state this in the digital domain as:

$$C_{PI}(z) = K_p + K_i\frac{T_s}{2}\frac{z+1}{z-1}$$

In the case of the PIDF type, where our continuous controller is stated as,

$$C_{PIDF}(s) = K_p + K_i\frac{1}{s} + K_d\frac{sN}{s+N}$$

we cannot directly use the bilinear transform as this creates an algebraic loop for the derivative term. This algebraic loop would result in code generation not being able to be utilized to generate controller implementation code for a digital system, or used for software in the loop (SiL) simulations. In order to get around this and allow for easier implementation, instead of using the bilinear transform for the discretised derivative term, we make use of the simple forward euler conversion stated as:

$$s = \frac{1}{T_s}(z-1)$$

As such,

$$C_{PIDF}(z) = K_p + K_i \frac{T_s}{2} \frac{z+1}{z-1} + K_d \frac{N(z-1)}{z + (NT_s - 1)}$$

Having discretised the controllers to be of the forms depicted previously, the only outstanding decision is to select the sample time, T_s, that is to be used. In order to select the sample time, there are two primary fundamental aspects to consider, as well as two primary practical considerations.

The first of the fundamental aspects is that we know that the effect of discretisation means that we will have fold-over in the frequency domain below the Nyquist frequency, or half of our sampling frequency, $\frac{f_s}{2}$. What this means is that should we have any resonance or unfavourable modes in the frequency space above the fold-over frequency, these will alias back into the frequency space below the fold-over frequency. Subsequently, we need to ensure that we select a sampling rate such that if we consider our frequency response of the system at frequencies above half of the sampling frequency, we have rolled off significantly enough that any aliasing which occurs will not impact our system performance.

The second fundamental consideration is that discretising the system means that we will lose phase margin. This is because if we compare the phase of a continuous system to a digital system, we know that

$$\phi(z) = \phi(s) - \omega T_s$$

where $\phi(z)$ is our phase in a discrete system; $\phi(s)$ is our phase in the corresponding continuous system; ω is the frequency we are interested in; and T_s is the selected sampling time - all in SI units of rad, rad/s and s, respectively. As such, if we consider our phase margin, we find that

$$\phi_{PM}(z) = \phi_{PM}(s) - \omega_{0dB} T_s$$

From this observation we know that our phase margin will always be less in a discrete system, in comparison to the continuous equivalent. As such, we must select T_s so that it is small enough to not cause sufficient loss in phase margin so as to impact the system performance by reducing damping.

Having stated the fundamental considerations one must ensure to keep in mind, we can now discuss the practical considerations alluded to. The first practical consideration one must take into account is that ultimately these digital controllers will need to be implemented on some form of digital computer, usually in the form of a microcontroller. Given the complexity of this type of system, it is likely that the implementation frame-

work will be some kind of real time operating system (RTOS), and the majority of these systems are set up such that the fundamental tick rate of the RTOS is at 1 kHz (or a period of 1 ms). As such, if at all possible, it is convenient to select our sampling time so that we are a reasonable amount above this value. This is so that the system will have sufficient overhead to deal with background tasks and will only be required to call and process the controllers at a reduced rate. We should also aim to have our sampling rate be an integer multiple of 1 ms in order to facilitate easier implementation.

Finally, the last practical consideration to take into account is that the software design to implement the control system is increased in complexity dramatically if one has multiple controllers all running at different sampling times, even if this approach is mathematically entirely sound. As such, we should attempt, wherever possible, to keep as many of our controllers running at the same sample time as possible.

Given the dynamics that we expect from both the sensors and the controllers, as well as what is easily implementable on an embedded system, it was decided that a constant sample time would be used for all controllers and algorithms. This is stated as:

$$T_s = 0.01 \text{ s}$$

It should be noted that since we know from all of the open-loop frequency response plots of the design, these have 0 dB cross-over points of ¡ 10 rad/s. Thus, using 10 rad/s as a worst case, we note that the impact on our phase margin is bounded by:

$$\phi_{PM}(z) \geq \phi_{PM}(s) - 0.1$$

Or, in degrees, we will at worst lose 5° of phase margin due to the discretisation. This is negligible and will not introduce any notable performance degradation in the system.

4.4.2 Digital Longitudinal Controller Verification

The digital longitudinal controller verification is simply a repeat of the tests performed in Section 4.2.5. However these tests are extended to include sensor and filter dynamics, which will be discussed in Chapter 5, so are expected to produce slightly different results, but more realistic ones. The results are depicted in Figures 4.48, 4.49, 4.50 and 4.51, while the implementation which illustrates the use of the sensor models can be found in Figure A.30.

These results illustrate that we get similar performance to that seen from the continuous controllers, as presented in Section 4.2.5, and one can see the discretisation in effect particularly in Figure 4.49. It should also be noted that as per Figure 4.48, the system is implemented such that we allow for a safety factor on our angle of attack in comparison to stall, as was discussed in the Section 4.2.5. For brevity, the results of the digital

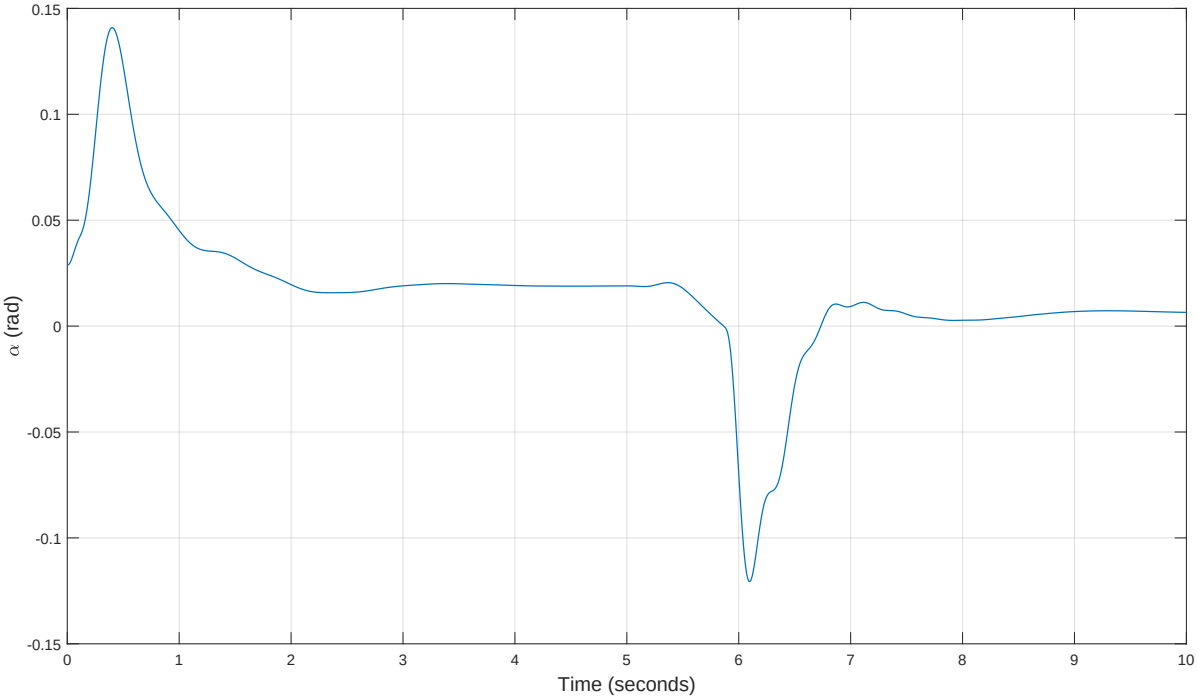


Figure 4.48: Digital longitudinal controller verification, α (rad)

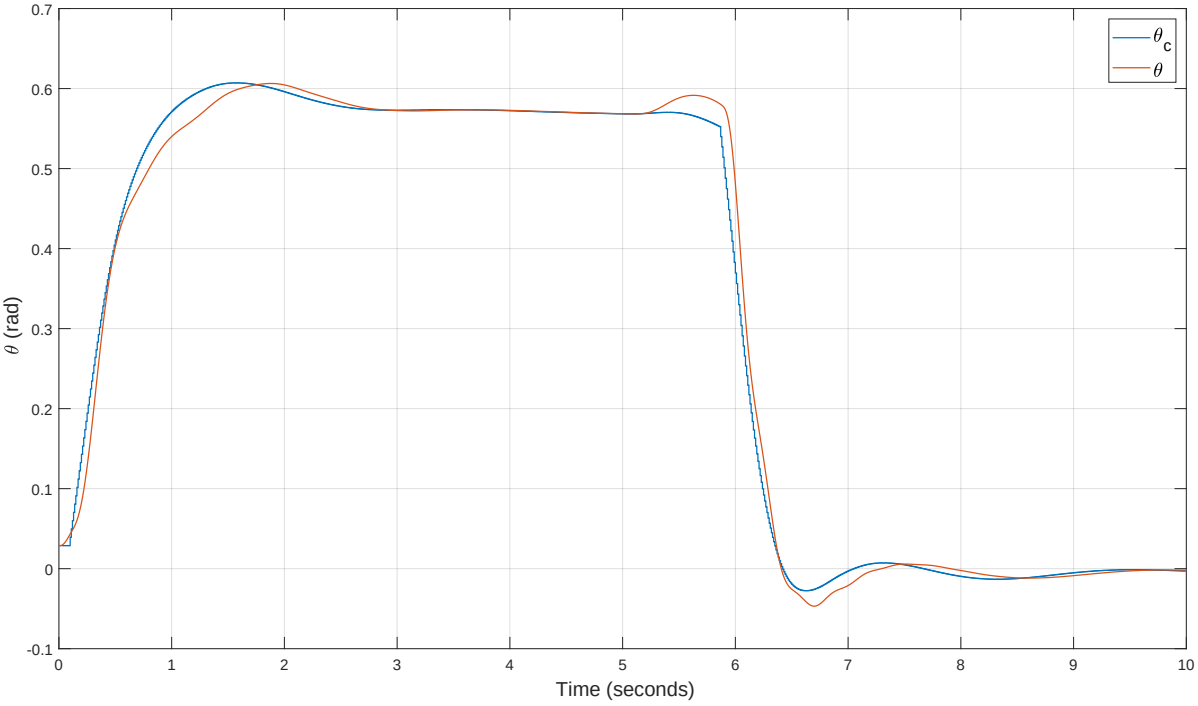


Figure 4.49: Digital longitudinal controller verification, θ (rad)

implementation of airspeed control via pitch angle are not shown, but similar trends are observed.

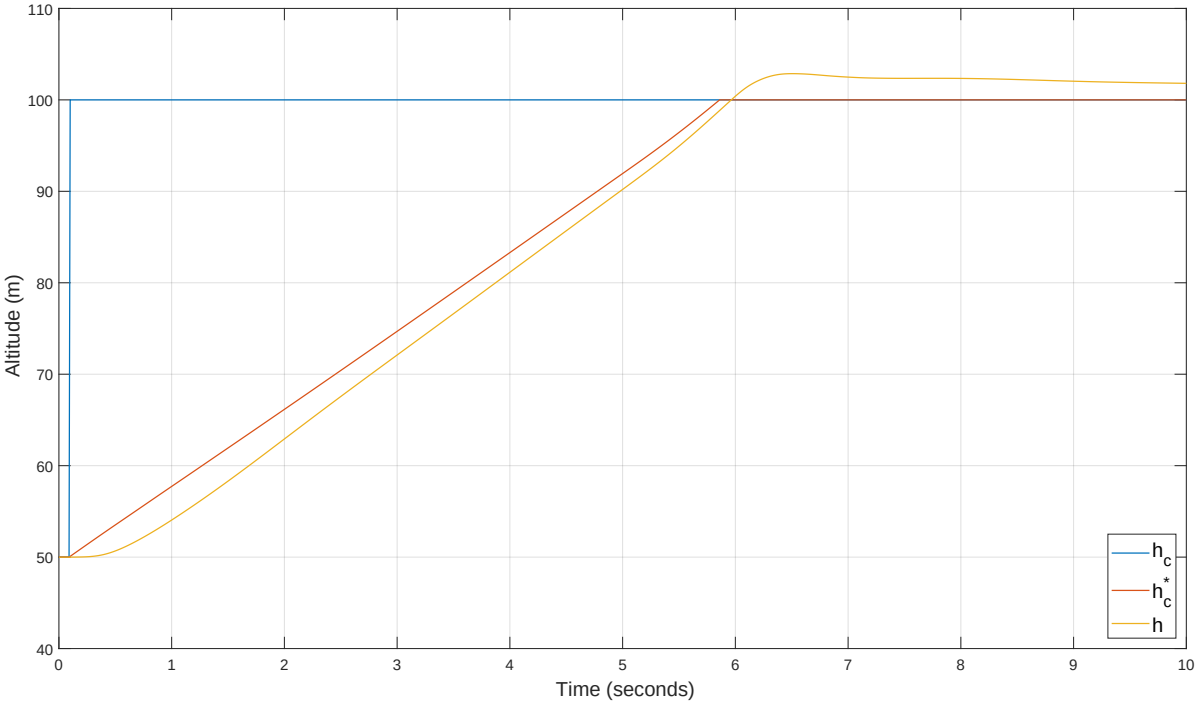


Figure 4.50: Digital longitudinal controller verification, h (m)

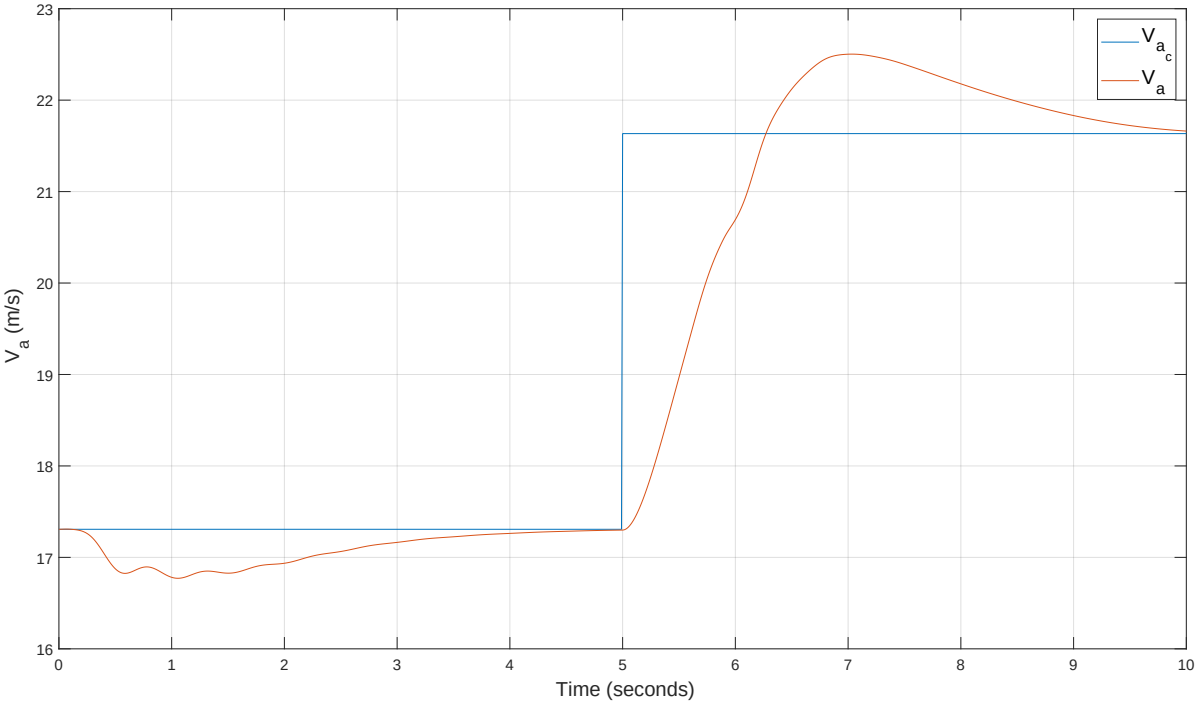


Figure 4.51: Digital longitudinal controller verification, V_a (m/s)

These controllers are deemed to be acceptable conversions from the continuous type and

no adjustments are required.

4.4.3 Digital Lateral Controller Verification

The lateral controller verification is simply a repeat of the tests performed in Section 4.3.3. However, these tests include sensor and filter dynamics, which will be discussed in Chapter 5, so are expected to be produce slightly different results, but more realistic ones. The results are depicted in Figures 4.52, 4.53, 4.54 and 4.55, while the implementation which illustrates the use of the sensor models can be found in Figure A.32.

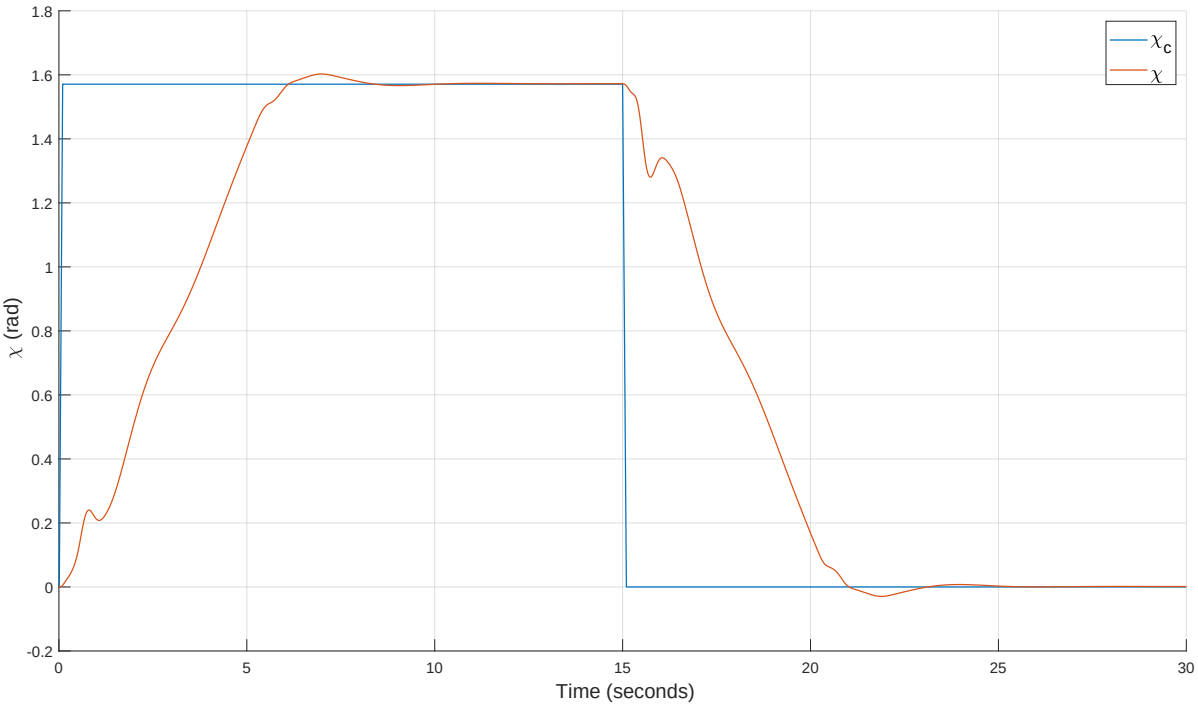


Figure 4.52: Digital lateral controller verification, χ (rad)

These results illustrate that we get similar performance to that seen from the continuous controllers, as presented in Section 4.3.3. One can see the discretisation in effect particularly in Figure 4.53, where it should be noted that we see two sets of sampling time at play because the course angle measurement is only updated at the same rate as our GPS sensor values, which is every 0.2 s, so the course angle controller adjusts its output 20 times before a new measurement is received.

These controllers are deemed to be acceptable conversions from the continuous type and no adjustments are required.

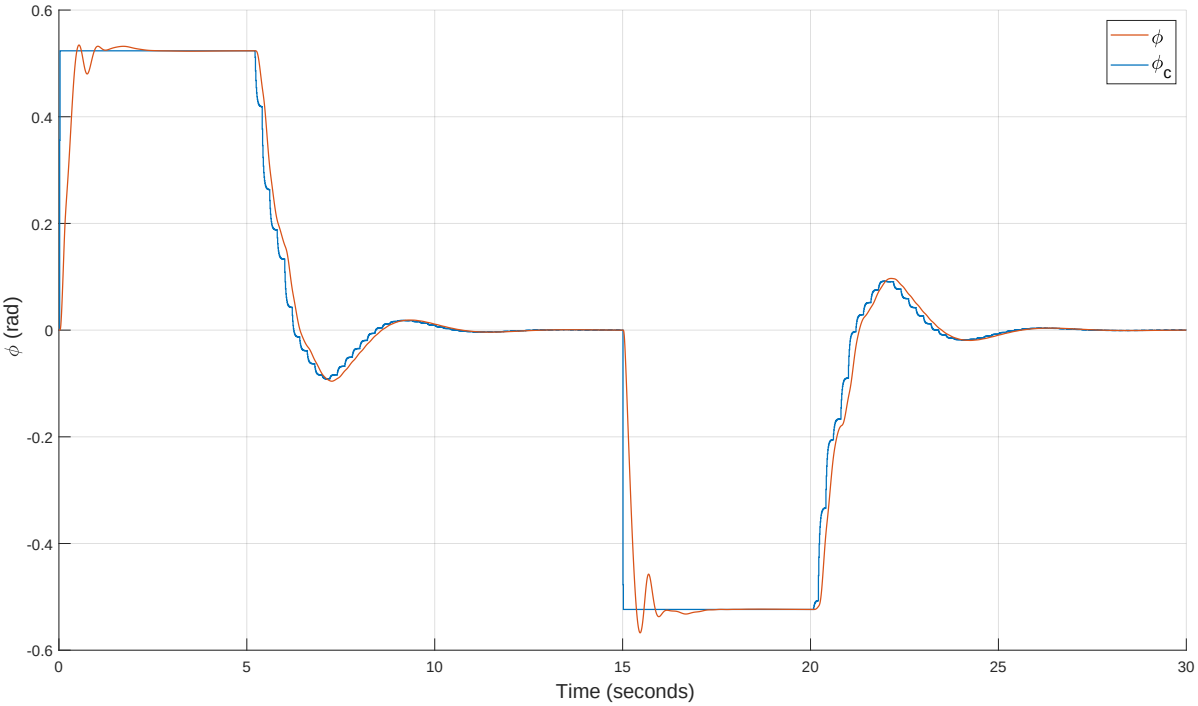


Figure 4.53: Digital lateral controller verification, ϕ (rad)

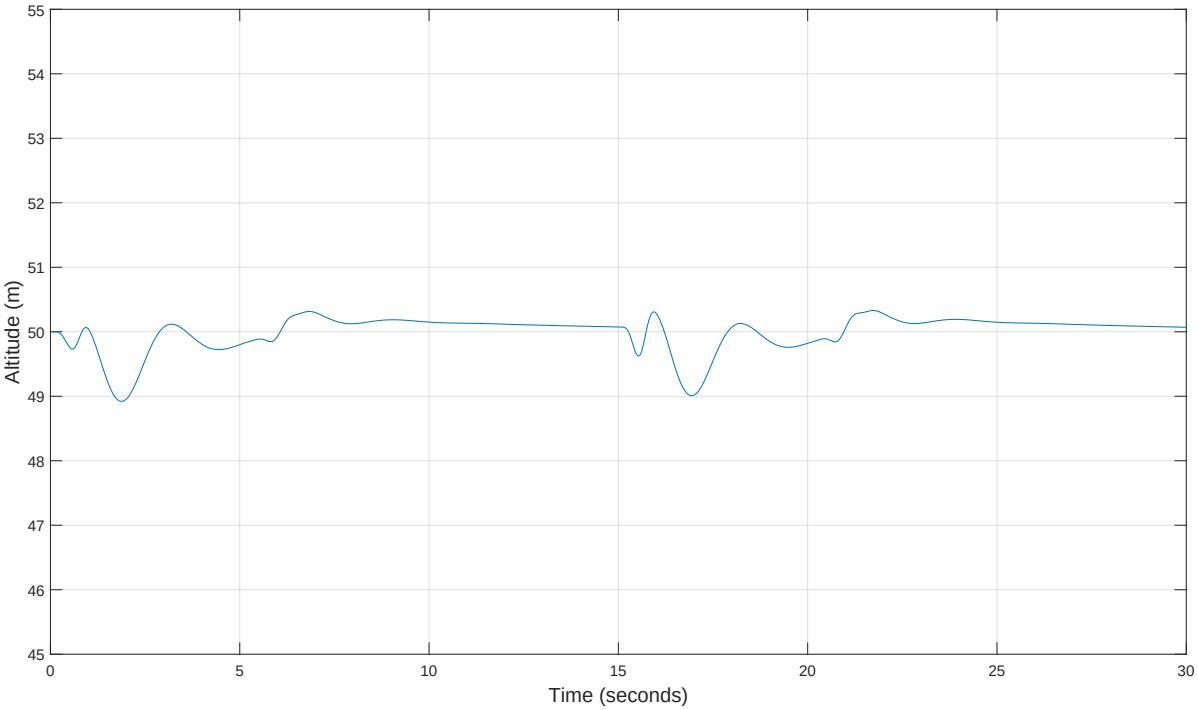


Figure 4.54: Digital lateral controller verification, h (m)

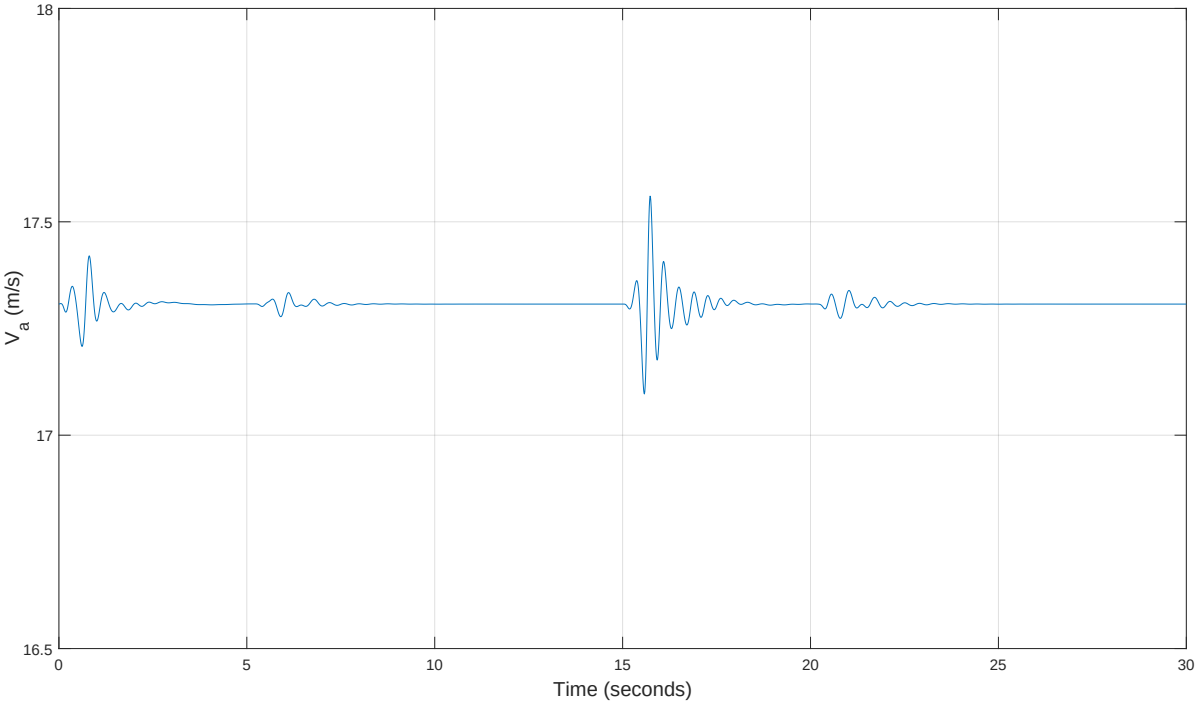


Figure 4.55: Digital lateral controller verification, V_a (m/s)

4.5 Autopilot Verification

The designs presented in the preceding sections are combined to form the full autopilot. The implementation of this system can be found in Figure A.33, which is then masked into a single subsystem which is illustrated in Figure 4.56, where one can see the full input and output requirements for the system, as well as the command inputs.

In order to finalise the verification of the digital controllers which make up the autopilot, we use the full subsystem with arbitrary command inputs in airspeed, V_{a_c}, altitude, h_c, and course angle, χ_c. These inputs have been selected to try and excite as many of the aircraft's modes as possible, as well as be as aggressive as possible so as to ensure that the autopilot works as expected under even the most severe conditions.

The Simulink implementation of the model used to verify the autopilot is depicted in Figure A.34, where it should be noted that the state machine indicated is the one illustrated in Figure A.28. The results of these verification simulations can be seen in Figures 4.57, 4.58, 4.59 and 4.60.

These results illustrate that our autopilot can handle even the most extreme of transients, as we purposefully line up our step commands so as to see how the autopilot handles the combination of control tasks. We do not get much performance degradation compared to the isolated testing done previously, except where we command a step de-

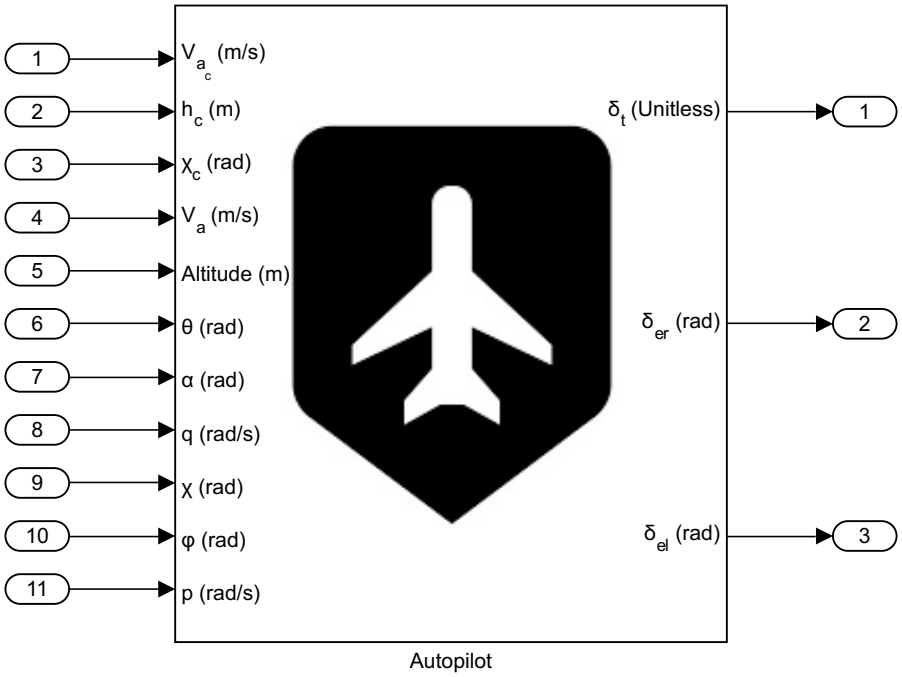


Figure 4.56: Autopilot implementation masked subsystem

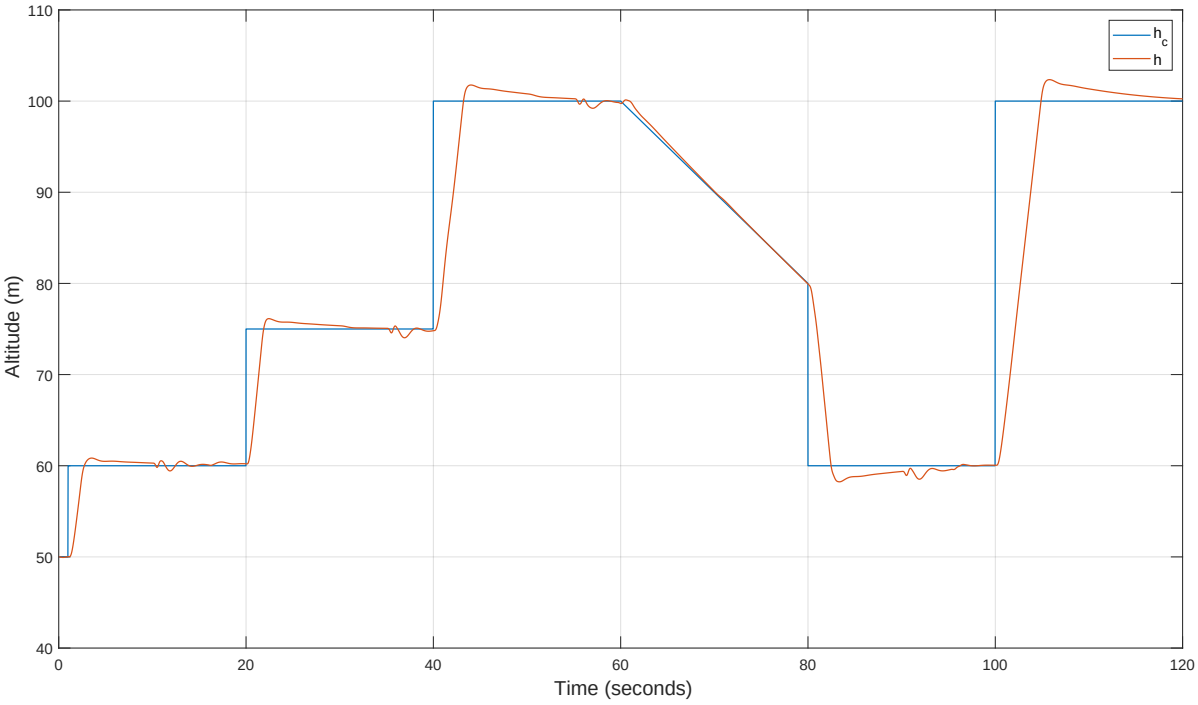


Figure 4.57: Digital autopilot verification, h (m)

scent in altitude and our airspeed increases - this is because even with zero throttle, if we are descending we will gain airspeed with the only force acting to reduce this being drag.

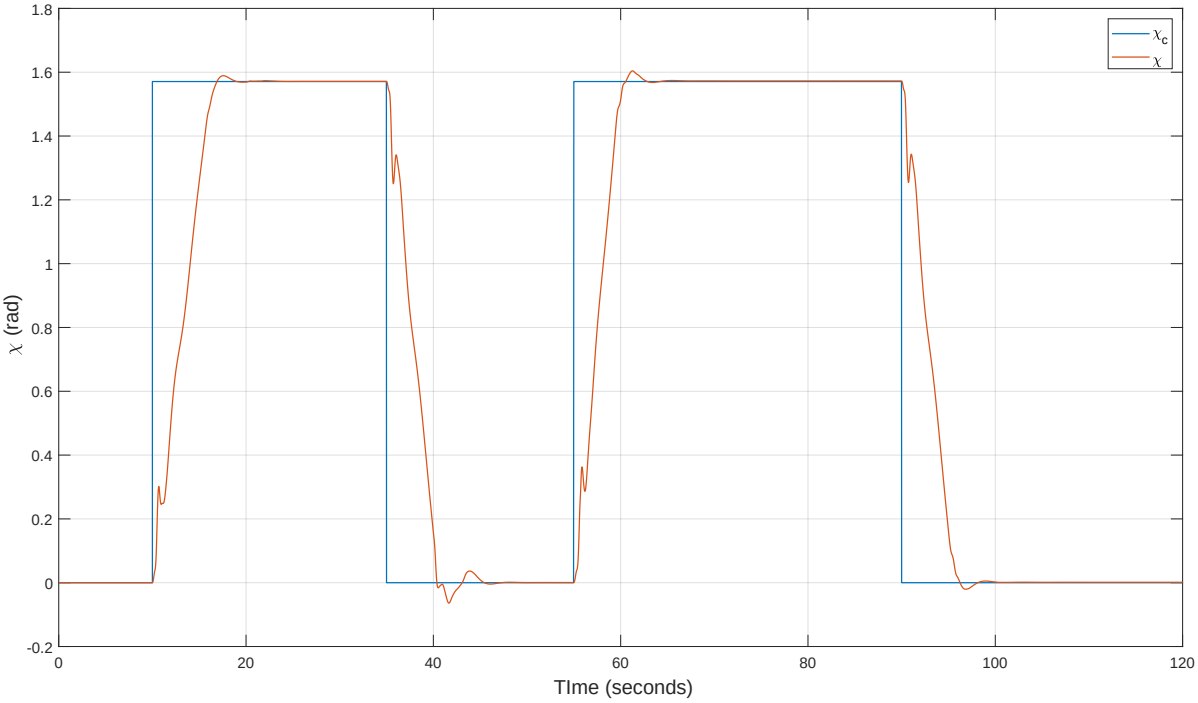


Figure 4.58: Digital autopilot verification, χ (rad)

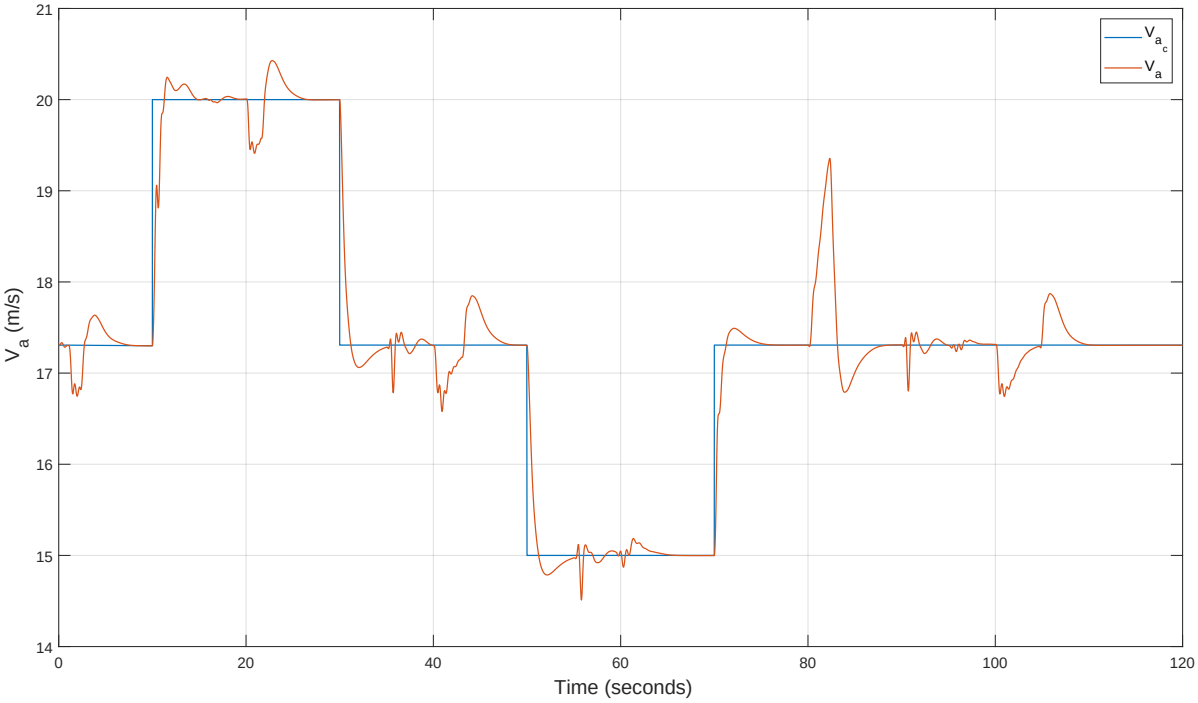


Figure 4.59: Digital autopilot verification, V_a (m/s)

We also note that we show in this test that stall is still possible (albeit only instanta-

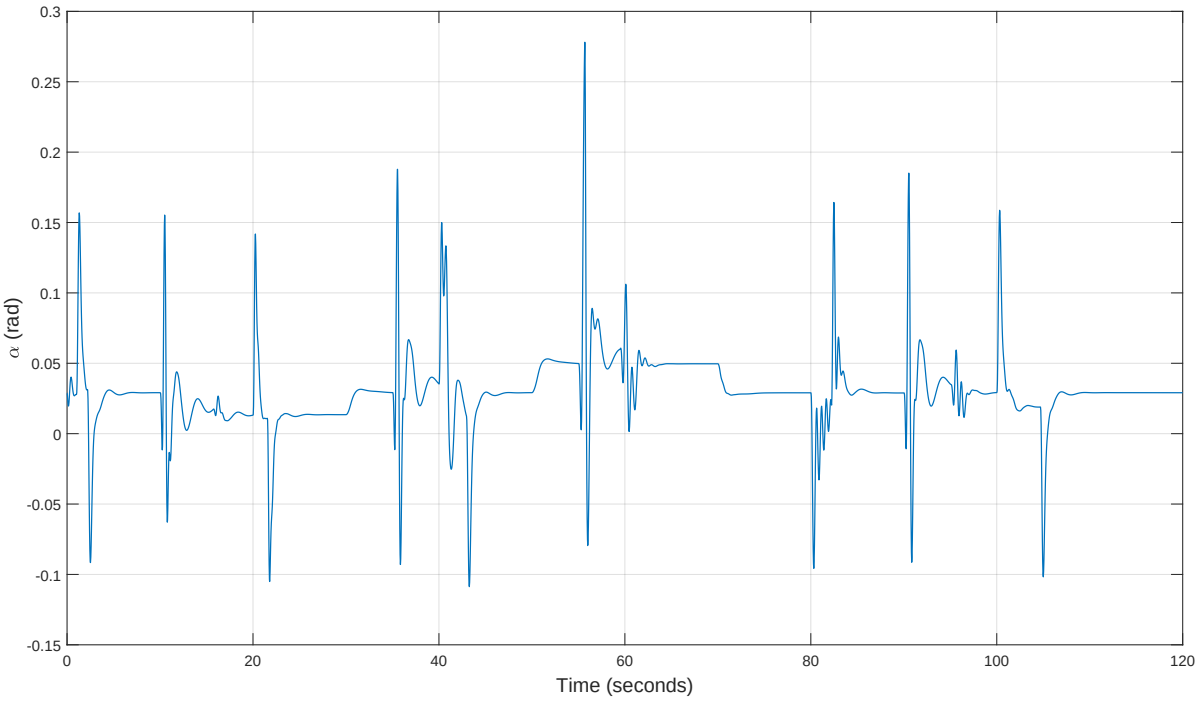


Figure 4.60: Digital autopilot verification, α (rad)

neously), which validates our decision to slow down the altitude controller to be much slower than what is optimal. This is because our 30% safety margin is still reduced to zero in some extreme cases such as the ones presented.

With these results, the autopilot design is complete. However, one should note from Figure A.34 that the aircraft model used is denoted as being of the full state feedback form. What this means is that we have full access to all of the measurements required for feedback into the autopilot, depicted as all but the top three inputs in Figure 4.56. In reality, however, we do not have access to all the states of the aircraft and we will need to estimate and approximate several of them in order to complete the feedback loop and provide the required inputs to the autopilot. This approximation of the system state based on sensor data is known as state estimation, which can be seen in Figure 1.2, and will be discussed in Chapter 5.

CHAPTER 5

State Estimation

As was mentioned in the preceding chapter, the autopilot which has been designed and verified thus far assumes that all of the states of the aircraft, such as the roll and pitch angles, are available for feedback. In reality, sensors which can measure each one of our states are not always available - particularly in the case of angles, which was discussed in Section 3.1.

Subsequently, the purpose of this chapter is to add the state estimation block depicted in Figure 1.2, which will take in measurements from the sensors described in Section 3.1, or $x_m(t)$ from Figure 1.2, and use these to determine an estimate of the current state of the aircraft, $x^*(t)$, which is used by the autopilot, as well as the guidance and navigation laws.

From Section 3.1 we know that we have rate gyroscopes which directly measure angular velocities in the body frame, so we can recover our p, q, and r states by simply low pass filtering the gyroscope outputs. This is also true for our angle of attack, α, which is directly measurable, so can simply be low pass filtered to remove some of the noise from the sensor.

We also derived some mathematical formulae which allow us to model the system state as some other quantity which is measurable by a sensor, as in Equation (3.16), where we convert from our current system airspeed to a measurable value in the form of a differential pressure signal. As such, we know that we can take the measured parameter and convert this back into the original state variable - where in the case of Equation (3.16) we can rearrange our equation such that:

$$V_a^*[k] = \sqrt{\frac{2y_{\Delta P_{V_a}}[k]}{\rho}}$$

Where $V_a^*[k]$ is the estimate of our airspeed at some time step, k, given the differential pressure measurement, $y_{\Delta P_{V_a}}[k]$, at the same time step. All that's been done is a simple rearrangement of the sensor model so as to get back to our original state, and the noise term has been omitted for the sake of simplicity. This method is known as sensor model inversion, and will be utilized in some cases where the state cannot be directly measured, but can be calculated based on a physical value which is measurable.

However, sensor inversion and direct measurements do not cover all of the required cases. This is because neither of these methods take into account the dynamics of the system and as such do not perform well over the full range of flight conditions. Because of this, we will make use of the Extended Kalman Filter in order to create mathematical estimators of some of the signals in the system. This will be done by first considering our attitude estimation, which will allow us to estimate our current roll and pitch angle; after which we will consider how to smooth our GPS measurements so as to reduce the impact of the drift that is introduced in these measurements, as was discussed in Section 3.1. Doing this will allow us to generate better estimates for our inertial position, ground speed, course angle, and heading; as well as the wind speed and direction. These are all important measurements to have in order to provide the feedback required for the autopilot, as well as the guidance and navigation laws, as is depicted in Figure 1.2.

The implementation of the state estimator is depicted in Figure A.35, which has been broken up into four sections:

1. Input Filtering & Sensor Inversion
2. Attitude Estimation
3. GPS Smoothing
4. Output Assignment

Each of these sections will be discussed in the subsections to follow.

5.1 Benchmark Manoeuvre

The benchmark manoeuvre that will be used to quantify the performance of the state estimation is the same as was depicted in Section 3.1.7, in Figures 3.4, 3.5 and 3.6, where we know that our plant responds in the manner depicted in Figures 4.57, 4.58 and 4.59. The sensor measurements derived from Section 3.1.7 will be used as inputs to the state estimator, and our results compared with those generated in Section 4.5, where full state feedback was available.

5.2 Signal Conditioning

As was mentioned in the preamble of this chapter, some of our sensor measurements will be low pass filtered as they can be read directly from sensors. The filters which will be used are simple second-order low pass filters of the type:

$$LPF(s) = \frac{a^2}{(s+a)^2}$$

where a is the bandwidth of the filter in rad/s. This is then converted to a digital filter by making use of the $LPF(z) = \text{c2d}(LPF(s))$ MATLAB function which converts this to the digital domain [57].

For our state estimator, it was decided that low pass filters would be applied to the measurements listed in Table 5.1, where the bandwidth of the filter and the sampling time used for the discretisation are also captured.

Table 5.1: State Estimation Low Pass Filter Parameters

Parameter	**Bandwidth (Hz)**	**Sample Time (s)**
a_x	50	0.01
a_y	50	0.01
a_z	50	0.01
p	50	0.01
q	50	0.01
r	50	0.01
ΔP_{V_a}	5	0.01
ΔP_h	5	0.01
α	50	0.01
χ	5	0.01

5.3 Sensor Model Inversion

As was discussed in the preamble to this chapter, we make use of sensor model inversion to estimate both our airspeed and our altitude, these are done as follows:

$$V_a^*[k] = \sqrt{\frac{2y_{\Delta P_{V_a}}[k]}{\rho}} \tag{5.1}$$

$$h^*[k] = \frac{y_{\Delta P_h}[k]}{\rho g} \tag{5.2}$$

where we note that our sensor measurements are first filtered before being fed into the inversion calculation.

5.4 Extended Kalman Filtering

Certain aircraft states are not easily determined by direct measurement or sensor model inversion, while others are easily measured, but the data contains significant drift over time. In order to counteract both of these limitations, a Kalman Filter [58–62] makes use of the physics of the system in order to estimate what the state of that system would have to be in order to be reading particular sensor values, which are provided as an input. For example, in the continuous case, if we have a continuous state space system defined as

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

we can create a continuous-time observer of the form:

$$\dot{\hat{\mathbf{x}}} = A\hat{\mathbf{x}} + B\mathbf{u} + L(\mathbf{u} - C\hat{\mathbf{x}})$$

where $\hat{\mathbf{x}}$ is the estimated value of $\mathbf{x}$. Defining the observation error as $\tilde{\mathbf{x}} = \mathbf{x} - \hat{\mathbf{x}}$, we find that

$$\dot{\tilde{\mathbf{x}}} = (A - LC)\tilde{\mathbf{x}}$$

which implies that the observation error decays exponentially to zero if L is chosen such that the eigenvalues of $A - LC$ are in the left half of the complex plane. Indeed, this is the premise of the Kalman filter, to select an L such that we can drive our observation error to zero and ensure that $\hat{\mathbf{x}} \to \mathbf{x}$.

An extension of the continuous time, linear Kalman filter is known as the extended Kalman filter (EKF) and this can be stated in its continuous-discrete form as [21]

$$\begin{aligned}\dot{\mathbf{x}} &= f(\mathbf{x}, \mathbf{u}) + \boldsymbol{\xi} \\ \mathbf{y}[n] &= h(\mathbf{x}[n], \mathbf{u}[n]) + \boldsymbol{\eta}[n]\end{aligned}$$

where $\mathbf{y}[n] = \mathbf{y}(t_n)$ is the n^{th} sample of $\mathbf{y}$; $\mathbf{x}[n] = \mathbf{x}(t_n)$ is the n^{th} sample of $\mathbf{x}$; and $\eta[n]$ is the measurement noise at time t_n. We also have that $\boldsymbol{\xi}$ is a zero-mean Gaussian random process with covariance Q, and $\boldsymbol{\eta}[n]$ is a zero-mean Gaussian random variable with covariance R. The random process $\boldsymbol{\xi}$ is called the process noise and represents modelling error and disturbances on the system, while the random variable $\boldsymbol{\eta}$ is called the measurement noise and represents the noise on the sensors. Since we already know the value for R from our sensor modelling, the design of the extended Kalman filter is to tune Q such that our state estimate is as accurate as possible.

In order to implement the Extended Kalman Filter one needs to specify five elements, and then to select Q so as to minimize the error between the estimated state and the actual state. The elements which need to be known are:

1. The state update model, $f(\mathbf{x}[n], \mathbf{u}[n], T_s)$
2. The measurement model, $h(\mathbf{x}[n], \mathbf{u}[n], T_s)$
3. The state update Jacobian with respect to the state, $F^x = \frac{\delta f}{\delta \mathbf{x}}$
4. The measurement Jacobian with respect to the state, $H^x = \frac{\delta h}{\delta \mathbf{x}}$
5. The measurement noise covariance, R

A detailed derivation of the Extended Kalman Filter is beyond the scope of this text. However, we simply note that the previously stated enumerated elements are necessary and will derive what each these are for our particular case. Fortunately one can simply make use of the built in Extended Kalman Filter block in Simulink [63] which requires one to define $f(\mathbf{x}[n], \mathbf{u}[n])$, $h(\mathbf{x}[n], \mathbf{u}[n])$, F^x, H^x, Q, R, and our sampling time, T_s - after which the Extended Kalman Filter algorithm is implemented without any additional input required. One can then iteratively select for Q until an accurate state estimate is possible.

5.4.1 Attitude Estimation

In order to estimate our attitude, or (ϕ, θ), we make use of the non-linear propagation model defined from Equation (2.24) as:

$$\begin{aligned}\dot{\phi} &= p + q\sin\phi\tan\theta + r\cos\phi\tan\theta + \xi_\phi \\ \dot{\theta} &= q\cos\phi - r\sin\phi + \xi_\theta\end{aligned}$$

where we have added the noise terms ξ_ϕ and ξ_θ to model the process noise on p, q, and r, where we assume that these are independent variables with no covariance.

We also make use of our accelerometer model as defined in Section 3.1.3, albeit rearranged to be as a function of measured parameters. As such, we have that

$$\mathbf{y}_{\mathbf{accel}} = \begin{pmatrix} \dot{u} + qw - rv + g\sin\theta \\ \dot{v} + ru - pw - g\cos\theta\sin\phi \\ \dot{w} + pv - qu - g\cos\theta\cos\phi \end{pmatrix} + \boldsymbol{\eta}_{\boldsymbol{accel}} \tag{5.3}$$

However, there is no method for directly measuring $\dot{u}$, $\dot{v}$, $\dot{w}$, u, v, and w. As such, we

will assume that $\dot{u} = \dot{v} = \dot{w} \approx 0$. We also have from Equation (2.6) that:

$$\begin{pmatrix} u \\ v \\ w \end{pmatrix} \approx V_a \begin{pmatrix} \cos\alpha\cos\beta \\ \sin\beta \\ \cos\beta\sin\alpha \end{pmatrix}$$

where we have a measurement for α and V_a, and will make the assumption that $\beta \approx 0$. As such, we have:

$$\begin{pmatrix} u \\ v \\ w \end{pmatrix} \approx V_a \begin{pmatrix} \cos\alpha \\ 0 \\ \sin\alpha \end{pmatrix} \tag{5.4}$$

Substituting Equation (5.4) into Equation (5.3), we obtain:

$$\mathbf{y_{accel}} = \begin{pmatrix} qV_a\sin\alpha + g\sin\theta \\ rV_a\cos\alpha - pV_a\sin\alpha - g\cos\theta\sin\phi \\ -qV_a\cos\alpha - g\cos\theta\cos\phi \end{pmatrix} + \boldsymbol{\eta_{accel}} \tag{5.5}$$

Defining $\mathbf{x} = (\phi, \theta)^T$, $\mathbf{u} = (p, q, r, V_a, \alpha)^T$, $\boldsymbol{\xi} = (\xi_\phi, \xi_\theta)^T$, and $\boldsymbol{\eta_{accel}} = (\eta_{a_x}, \eta_{a_y}, \eta_{a_z})^T$ gives us

$$f(\mathbf{x}, \mathbf{u}) = \begin{pmatrix} p + q\sin\phi\tan\theta + r\cos\phi\tan\theta \\ q\cos\phi - r\sin\phi \end{pmatrix} \tag{5.6}$$

$$h(\mathbf{x}, \mathbf{u}) = \begin{pmatrix} qV_a\sin\alpha + g\sin\theta \\ rV_a\cos\alpha - pV_a\sin\alpha - g\cos\theta\sin\phi \\ -qV_a\cos\alpha - g\cos\theta\cos\phi \end{pmatrix} \tag{5.7}$$

where our state transition and measurement Jacobians are defined as

$$\frac{\delta f}{\delta \mathbf{x}} = \begin{pmatrix} q\cos\phi\tan\theta - r\sin\phi\tan\theta & \frac{q\sin\phi + r\cos\phi}{\cos^2\theta} \\ -q\sin\phi - r\cos\phi & 0 \end{pmatrix} \tag{5.8}$$

$$\frac{\delta h}{\delta \mathbf{x}} = \begin{pmatrix} 0 & g\cos\theta \\ -g\cos\phi\cos\theta & g\sin\phi\sin\theta \\ g\sin\phi\cos\theta & g\cos\phi\sin\theta \end{pmatrix} \tag{5.9}$$

where we have assumed that $\frac{\delta\alpha}{\delta\theta} = \frac{\delta\alpha}{\delta\phi} = 0$.

We also know from our sensor modelling that

$$R = \begin{pmatrix} 0.0015 & 0 & 0 \\ 0 & 0.0015 & 0 \\ 0 & 0 & 0.0015 \end{pmatrix}$$

and in this instance we select Q such that

$$Q = \begin{pmatrix} \text{1E-10} & 0 \\ 0 & \text{1E-10} \end{pmatrix}$$

which was determined iteratively and indicates that our model of the system is a much better predictor of the state than the measurements taken. This will be illustrated in Section 5.5.

The final item to select are the update rates - and one can select different values for both the state transition, as well as the measurement update rates. In the case of all of our attitude measurements, we know our measurement update rate will be the same as that of our required state transition update rate. As such,

$$T_s = 0.01 \text{ s}$$

5.4.2 GPS Smoothing

In order to use our GPS measurements most effectively, it makes sense to extend their use with an EKF. This allows one to have an update rate which is much faster than the GPS sample time of 0.2 s by simply making use of our equations of motion. In other words, we can provide a prediction of the states provided by the GPS in between receiving updates of the actual state from the GPS transceiver. This can be done by using the last measurement received, and the equations of motion, to predict what the state should be at the next time step (some value smaller than 0.2 s). It also allows one to reduce the impact of effects like noise and drift on the measurements. We will make use of the model in [21], which will simply be stated without being derived.

Defining the state as $\mathbf{x} = (p_n, p_e, V_g, \chi, w_n, w_e, \psi)^T$, and the input as $\mathbf{u} = (V_a, q, r, \phi, \theta)^T$,

the non-linear propagation model is given as:

$$f(\mathbf{x}, \mathbf{u}) = \begin{pmatrix} V_g \cos\chi \\ V_g \sin\chi \\ \dot{V}_g \\ \dot{\chi} \\ 0 \\ 0 \\ \dot{\psi} \end{pmatrix} \tag{5.10}$$

where,

$$\dot{\psi} = q\frac{\sin\phi}{\cos\theta} + r\frac{\cos\phi}{\cos\theta} \tag{5.11}$$

$$\dot{V}_g = \frac{(V_a\cos\psi + w_n)(-V_a\dot{\psi}\sin\psi) + (V_a\sin\psi + w_e)(V_a\dot{\psi}\cos\psi)}{V_g} \tag{5.12}$$

$$\dot{\chi} = \frac{g}{V_g}\tan\phi\cos(\chi - \psi) \tag{5.13}$$

and the Jacobian of f, F^x is stated as

$$\frac{\delta f}{\delta \mathbf{x}} = \begin{pmatrix} 0 & 0 & \cos\chi & -V_g\sin\chi & 0 & 0 & 0 \\ 0 & 0 & \sin\chi & V_g\cos\chi & 0 & 0 & 0 \\ 0 & 0 & -\frac{\dot{V}_g}{V_g} & 0 & -\dot{\psi}V_a\sin\psi & \dot{\psi}V_a\cos\psi & \frac{\delta\dot{V}_g}{\delta\psi} \\ 0 & 0 & \frac{\delta\dot{\chi}}{\delta V_g} & \frac{\delta\dot{\chi}}{\delta\chi} & 0 & 0 & \frac{\delta\dot{\chi}}{\delta\psi} \\ 0 & 0 & 0 & 0 & 0 & 0 & \\ 0 & 0 & 0 & 0 & 0 & 0 & \\ 0 & 0 & 0 & 0 & 0 & 0 & \end{pmatrix} \tag{5.14}$$

where,

$$\frac{\delta\dot{V}_g}{\delta\psi} = -\frac{\dot{\psi}V_a(w_n\cos\psi + w_e\sin\psi)}{V_g}$$

$$\frac{\delta\dot{\chi}}{\delta V_g} = -\frac{g}{V_g^2}\tan\phi\cos(\chi - \psi)$$

$$\frac{\delta\dot{\chi}}{\delta\chi} = -\frac{g}{V_g}\tan\phi\sin(\chi - \psi)$$

$$\frac{\delta\dot{\chi}}{\delta\psi} = \frac{g}{V_g}\tan\phi\sin(\chi - \psi)$$

For our measurements, we will make use of the GPS signals for the North and East position, ground speed, and course angle. Since our states are not independent, we make

use of the wind triangle relationship given in Equation (2.8), to create this dependence in the model. Assuming that $\gamma = \gamma_0 = 0$, we can state that

$$V_a \cos\psi + w_n = V_g \cos\chi$$
$$V_a \sin\psi + w_e = V_g \sin\chi$$

From these expressions, we define two pseudo-measurements, which are always equal to zero, as

$$y_{wind_N} = V_a \cos\psi + w_n - V_g \cos\chi \tag{5.15}$$
$$y_{wind_E} = V_a \sin\psi + w_e - V_g \sin\chi \tag{5.16}$$

As such, our measurements are defined as $\mathbf{y_{GPS}} = (y_{GPS_N}, y_{GPS_E}, y_{GPS_{V_g}}, y_{GPS_\chi}, y_{wind_N}, y_{wind_E})^T$, and inputs as $\mathbf{u} = (V_a, q, r, \phi, \theta)^T$. We then define $h(\mathbf{x}, \mathbf{u})$ as:

$$h(\mathbf{x}, \mathbf{u}) = \begin{pmatrix} p_n \\ p_e \\ V_g \\ \chi \\ V_a \cos\psi + w_n - V_g \cos\chi \\ V_a \sin\psi + w_e - V_g \sin\chi \end{pmatrix} \tag{5.17}$$

where the Jacobian of h, H^x, is given by

$$\frac{\delta h}{\delta \mathbf{x}} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & -\cos\chi & V_g \sin\chi & 1 & 0 & -V_a \sin\psi \\ 0 & 0 & -\sin\chi & -V_g \cos\chi & 0 & 1 & V_a \cos\psi \end{pmatrix} \tag{5.18}$$

We also know from our sensor modelling that

$$R = \begin{pmatrix} 4.41\text{E-}02 & 0 & 0 & 0 & 0 & 0 \\ 0 & 4.41\text{E-}02 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1.5625\text{E-}04 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1.5625\text{E-}04 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1\text{E-}12 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1\text{E-}12 \end{pmatrix}$$

where it should be noted that the covariance of our pseudo-measurements (ie. $R(5,5)$ and $R(6,6)$) are assumed to have very little variance as we expect these measurements

to always be zero.

In this instance we select Q such that

$$Q = I_{7\times 7} \begin{pmatrix} 1\text{E-}01 \\ 1\text{E-}01 \\ 1\text{E-}01 \\ 1\text{E-}05 \\ 1\text{E-}01 \\ 1\text{E-}01 \\ 1\text{E-}05 \end{pmatrix}$$

where $I_{7\times 7}$ is a 7x7 identity matrix. Q was determined iteratively.

The final item to select are the update rates - and one can select different values for both the state transition, as well as the measurement update rates. In the case of the GPS smoothing, we know that our measurement update rate is that of the GPS, or 0.2 s, whereas we want our EKF output to be at the rate as our controllers. As such,

$$T_{s_x} = 0.01 \text{ s}$$
$$T_{s_y} = 0.2 \text{ s}$$

5.5 State Estimation Verification

In order to illustrate the impact of the state estimation, we will make use of the same benchmark manoeuvre that has been used throughout this text. As such, we will run the same test as was depicted in Section 4.5, except now instead of having full state feedback, we output only our observable data and use our state estimator in the feedback loop - this is depicted in Figure A.36. The results of this simulation are shown in Figures 5.1, 5.2 and 5.3.
From these results we note that we are still able to track the reference commands, indicating that our state estimation is effective in providing an accurate estimate of our state insofar as that we are still able to control the system. As might be expected, we do observe some degradation in system performance in comparison to the full state feedback case as was covered in Section 4.5. However, given that the reference commands in this benchmark manoeuvre are very aggressive in comparison to what would normally be commanded of the autopilot in a more standard use-case, we expect that this is the absolute worst case and so if the autopilot can still control the aircraft here - it should be able to in all other cases.

In order to illustrate the efficacy of the state estimator itself, we consider our most difficult to estimate parameters, ϕ and θ - the results of which are depicted in Figures 5.4 and 5.5, respectively. We also consider one of the GPS smoothing parameters which is

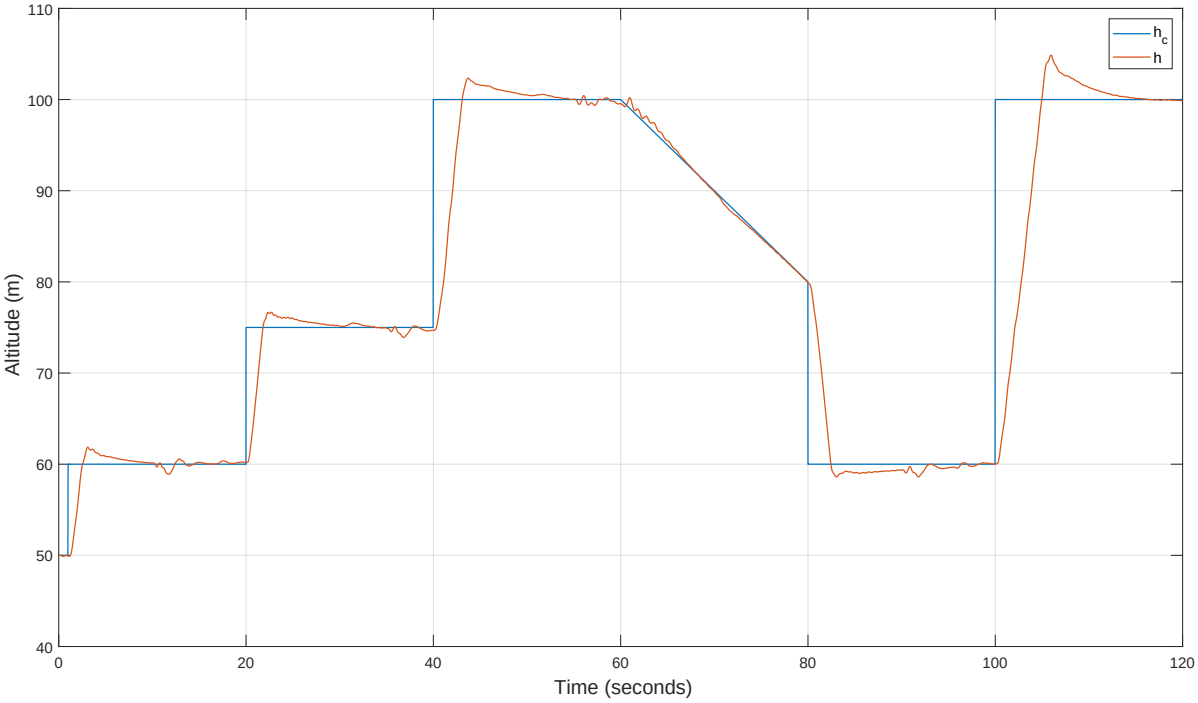


Figure 5.1: Digital autopilot with observable feedback verification, h (m)

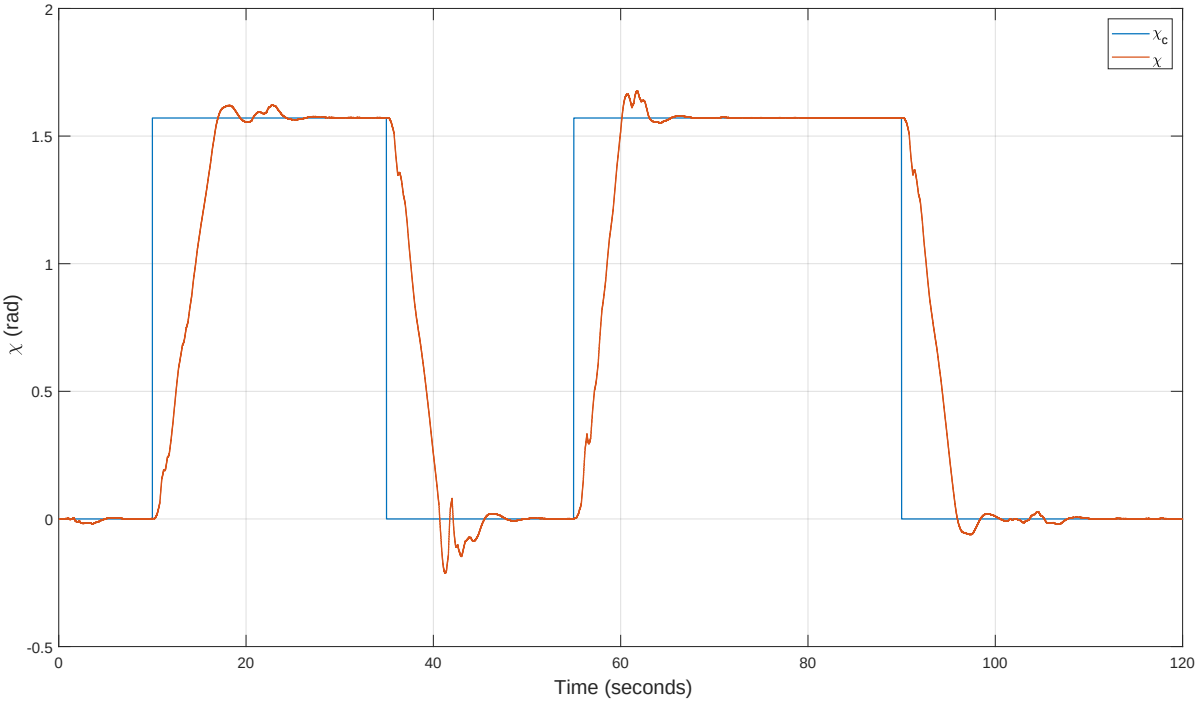


Figure 5.2: Digital autopilot with observable feedback verification, χ (rad)

made use of directly in the autopilot, χ, which is depicted in Figure 5.6. Finally, we look at one one of the easier states to estimate, h, which is depicted in Figure 5.7.

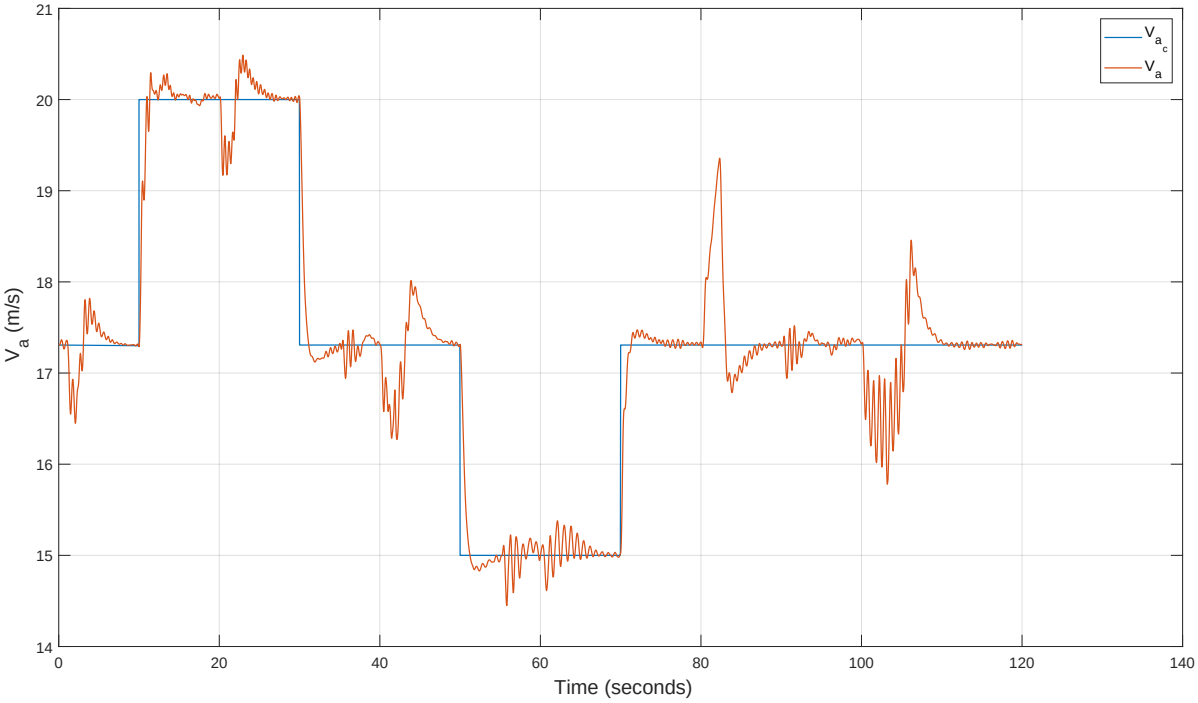


Figure 5.3: Digital autopilot with observable feedback verification, V_a (m/s)

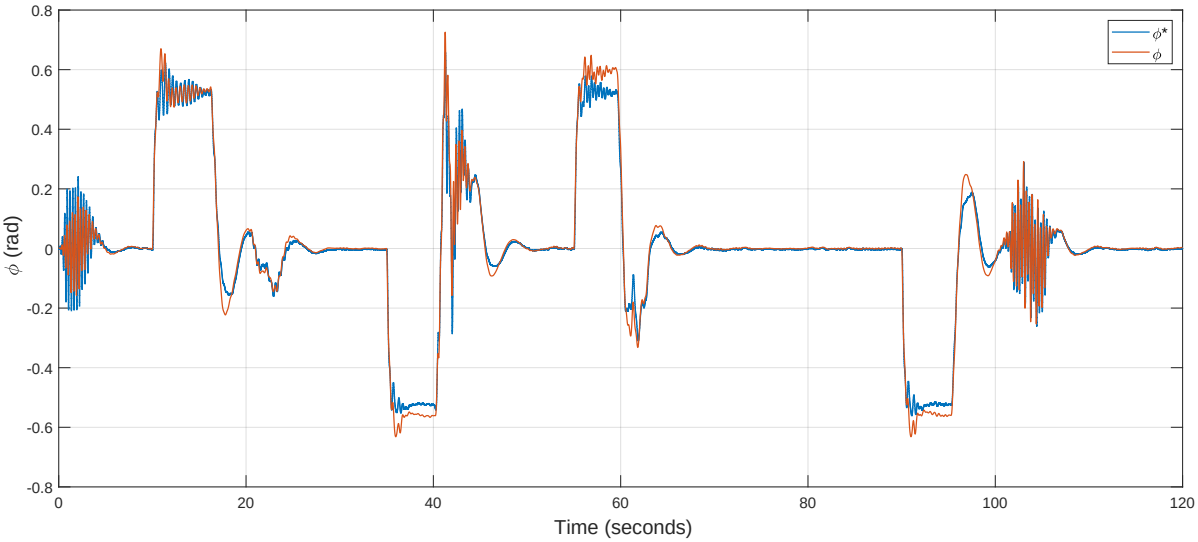


Figure 5.4: State estimation verification, ϕ (rad)

All of these results indicate that while our state estimates are not perfect, they are within reasonable tolerances. Clearly, the error that is introduced still allows one to control the aircraft and as such the state estimator design is deemed as acceptable. Since the benchmark manoeuvre made use of is also quite aggressive, we expect that in the more

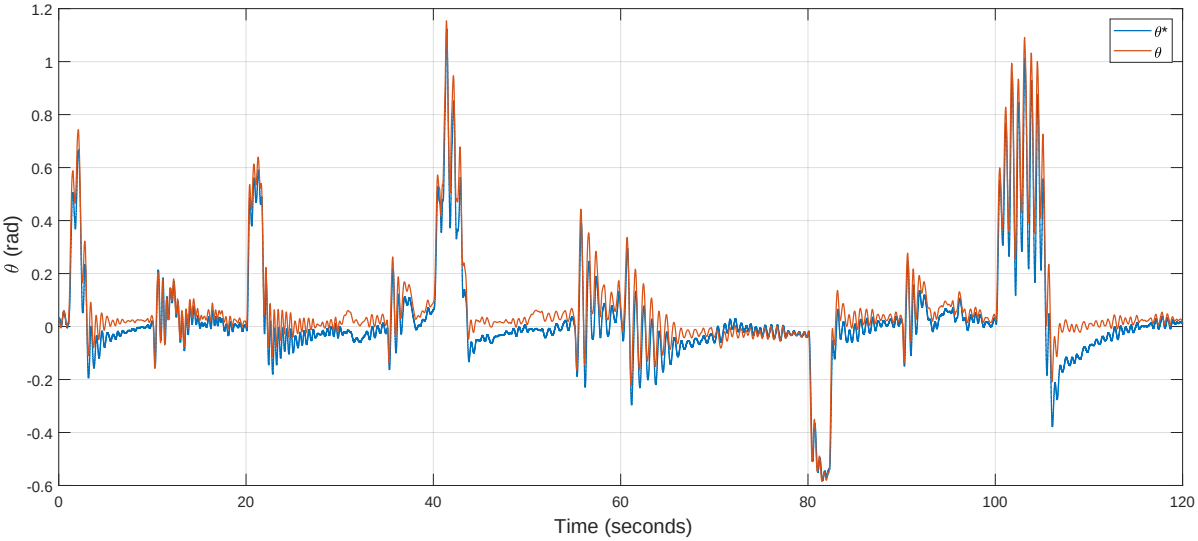


Figure 5.5: State estimation verification, θ (rad)

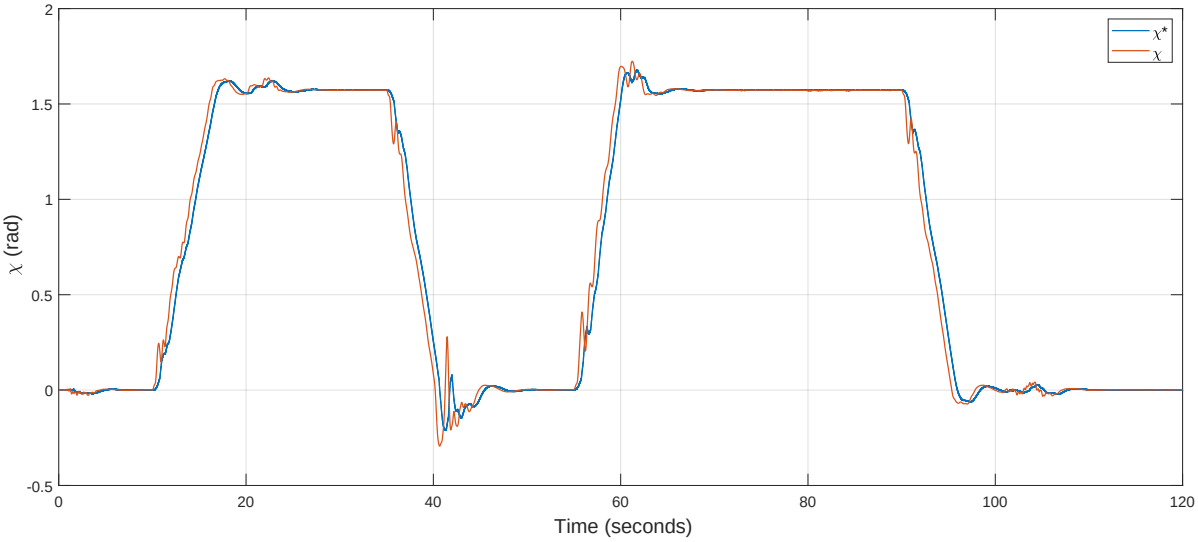


Figure 5.6: State estimation verification, χ (rad)

usual use-case with slower reference commands, our state estimates will likely become more accurate as the estimator will not have to deal with such rapidly varying states.

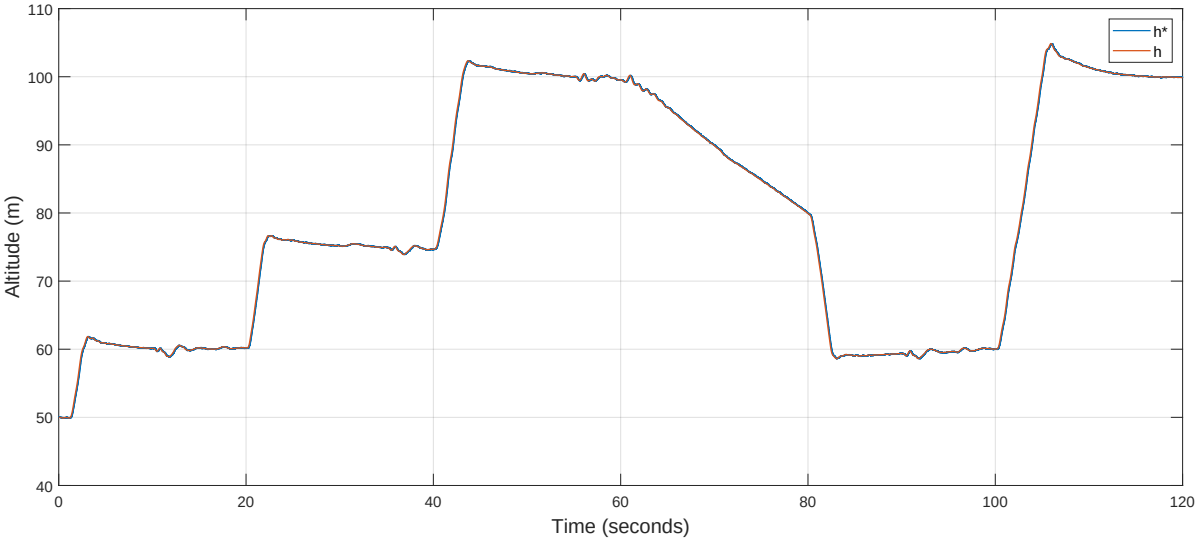


Figure 5.7: State estimation verification, h (m)

Guidance and Navigation

Thus far in the design process we have established an autopilot which can receive an input command of (h_c, V_{a_c}, χ_c), as a function of time, and cause the aircraft to track these commanded values. This has been shown to operate effectively even in the presence of sensor non-linearities and with variations in the plant dynamics relative to the linear models used to tune the controllers. This has been confirmed to work as expected during complicated manoeuvres which have these input commands adjusting simultaneously, or having one input being adjusted while we're part way through a transition in another of the input commands. While this is a critical step in the design process, an operator is unlikely to be in a position where they would provide altitude, airspeed and course angle commands in order to meet their requirements.

The more likely use-case is that they require the aircraft to transition between a set of waypoints, such that

$$\mathcal{W} = \{\mathbf{W_1}, \mathbf{W_2}, \mathbf{W_3}, ..., \mathbf{W_n}\}$$

which are defined as

$$\mathbf{W_n} = (p_n, p_e, p_d)^T$$

where each waypoint represents an inertial position in $\mathbb{R}^3$. This notion was depicted in Figure 1.2, where it can be seen that two primary inputs to the full GNC system are a set of waypoints, as well as an airspeed command.

Subsequently, we need to extend the autopilot with guidance and navigation laws which will allow us to automatically transition between the previously discussed set of waypoints, $\mathcal{W}$, defined by the user during launch and adjusted at any stage during flight from a ground station. As such, we need some way to convert a set of waypoints into commands which our autopilot can track, namely (h_c, V_{a_c}, χ_c), at every time step. These commands should be constructed so that we can accurately follow the waypoint path as required by the user.

In order to accomplish this, we will first establish two sets of algorithms that will allow us to accomplish our two fundamental building blocks of any path management algorithm, straight line and orbit following. This is denoted as the Path Following block in

Figure 1.2.

The process responsible for determining when each algorithm should be used, as well how they should be switched between, is known as path management - which is denoted in Figure 1.2 as the Path Manager block. These algorithms allow us to navigate between waypoints and handle the transition from one line segment connecting the two waypoints we are travelling between to the line segment connecting the current waypoint and the one that follows; and will be detailed after the discussion on the path following algorithms.

Finally, we will make use of the full GNC system - comprising of the autopilot, state estimator, path following algorithms and path management algorithms - to attempt to follow the set of waypoints which were defined as our end goal in both Figures 1.3 and 1.4 in order to verify that our algorithms are capable of handling their required function.

6.1 Straight Line and Orbit Following

Any path can be broken down into segments made up of straight lines and circular orbits. The primary challenge involved in tracking these segments is wind, which can be quite substantial for small aircraft such as this. Subsequently, our algorithms must be developed to work in these conditions. Another limitation is that we have a physical limit on the turn radii that can be accomplished by the system, which imposes a fundamental limit on the spatial frequency of the paths that can possibly be tracked. As such, it is important that the path-tracking algorithms make sure to utilize the full capability of the aircraft.

When considering the idea of trajectory tracking, there is the idea that the vehicle must be at a particular point in space at a particular time, with the location varying as a function of time. With fixed-wing aircraft, the desired position is constantly moving (at the desired ground speed). The approach of tracking a moving point can result in significant problems for small aircraft if disturbances, such as those due to wind, are not properly accounted for. For example, if the aircraft is flying into a strong wind, the progression of the trajectory point must be slowed accordingly. Similarly, if the aircraft is flying downwind, the speed of the tracking point must be increased to keep it from overrunning the desired position. Given that wind disturbances vary and are not easily estimated or predicted, trajectory tracking is a challenging methodology to make use of for small aircraft in anything but calm conditions.

In order to avoid the problems associated with trajectory tracking, we will make use of the concept of path following, where the objective is to be on the path, rather than at a certain point at a particular time. With path following, the time dependence of the problem is removed which greatly reduces the problems introduced by the presence of wind. The work presented in this chapter thus deals with the creation of path following

algorithms, and follows the work presented in [64–66].

It should be noted that airspeed is treated as a command value, so for this section we will assume an airframe moving at constant velocity, V_a.

6.1.1 Straight Line Following

A straight-line path is described by two vectors in $\mathbb{R}^3$ as

$$\mathcal{P}_{line}(\mathbf{r}, \mathbf{q}) = \{\mathbf{x} \in \mathbb{R} : \mathbf{x} = \mathbf{r} + \lambda\mathbf{q}, \lambda \in \mathbb{R}\}$$

where $\mathbf{r} \in \mathbb{R}^3$ is the origin of the path (usually a waypoint), and $\mathbf{q} \in \mathbb{R}^3$ is a unit vector whose direction indicates the desired direction of travel. In order to illustrate this path, one can consider Figure 6.1, which depicts the aircraft travelling in $\mathbb{R}^3$, along with straight-line path $\mathcal{P}_{line}(\mathbf{r}, \mathbf{q})$ with an orthographic view of the scenario. Figure 6.2 depicts this same scenario, but from the top view (or $\mathbf{i^i}$-$\mathbf{j^i}$ plane).

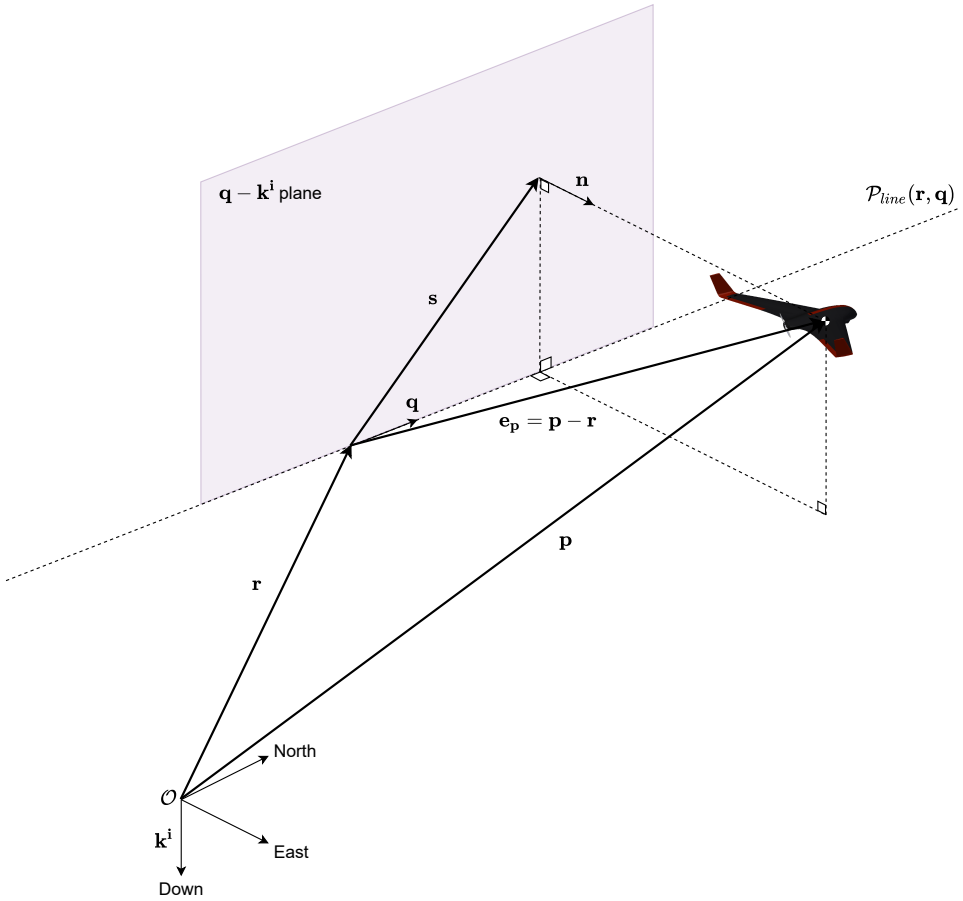


Figure 6.1: Orthographic view of an aircraft travelling relative to the straight-line segment $\mathcal{P}_{line}(\mathbf{r}, \mathbf{q})$

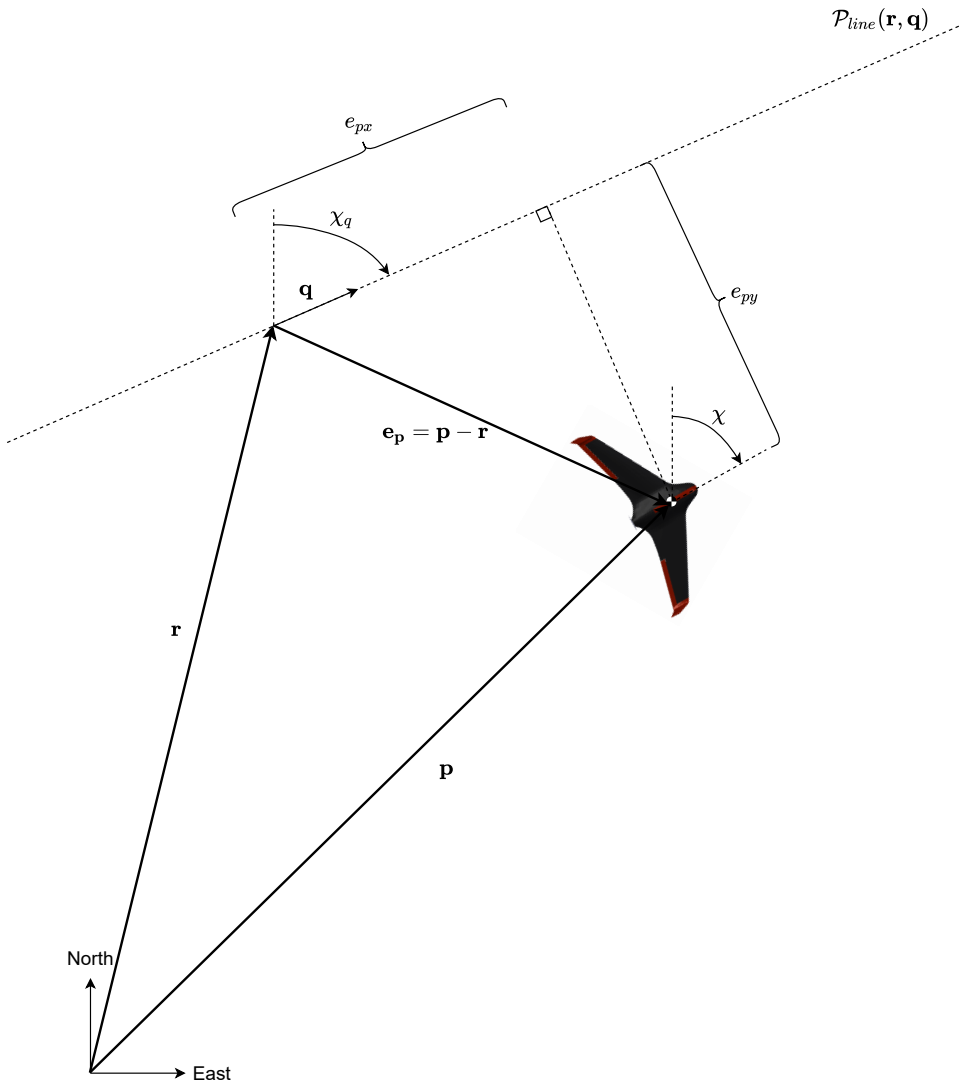


Figure 6.2: Top view of an aircraft travelling relative to the straight-line segment $\mathcal{P}_{line}(\mathbf{r},\mathbf{q})$

Considering Figure 6.2, the course angle of $\mathcal{P}_{line}(\mathbf{r},\mathbf{q})$ is given by

$$\chi_q \triangleq \text{atan2}\left(\frac{q_e}{q_n}\right) \tag{6.1}$$

where $\mathbf{q} = (q_n, q_e, q_d)^T$ expresses the North, East, and Down components of the unit direction vector.

The path-following problem is most easily solved in a frame relative to the straight-line path. Selecting $\mathbf{r}$ as the center of the path frame, with the x-axis aligned with the projection of $\mathbf{q}$ onto the local North-East plane, the z-axis aligned with the inertial

z-axis, and the y-axis selected to create a right-handed coordinated system, we have that

$$\mathcal{R}_i^{\mathcal{P}} \triangleq \begin{pmatrix} \cos\chi_q & \sin\chi_q & 0 \\ -\sin\chi_q & \cos\chi_q & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{6.2}$$

is the transformation from the inertial frame to the path frame, and

$$\mathbf{e_p} = \begin{pmatrix} e_{px} \\ e_{py} \\ e_{pz} \end{pmatrix} \triangleq \mathcal{R}_i^{\mathcal{P}}(\mathbf{p^i} - \mathbf{r^i}) \tag{6.3}$$

is the relative path error expressed in the path frame. The relative error dynamics in the North-East inertial plane, expressed in the path plane, are given by:

$$\begin{aligned} \begin{pmatrix} \dot{e}_{px} \\ \dot{e}_{py} \end{pmatrix} &= \begin{pmatrix} \cos\chi_q & \sin\chi_q \\ -\sin\chi_q & \cos\chi_q \end{pmatrix} \begin{pmatrix} V_g\cos\chi \\ V_g\sin\chi \end{pmatrix} \\ &= V_g \begin{pmatrix} \cos(\chi - \chi_q) \\ \sin(\chi - \chi_q) \end{pmatrix} \end{aligned} \tag{6.4}$$

For path following, we desire to regulate the cross-track error, e_{py}, to zero by commanding the course angle. The relevant dynamics for this are therefore given by

$$\dot{e}_{py} = V_g \sin(\chi - \chi_q) \tag{6.5}$$

Where the lateral straight-line path following problem is to select χ_c so that $e_{py} \to 0$ when χ_q is known.

The geometry needed for the straight-line path following in the longitudinal direction is depicted in Figure 6.1. To calculate the desired altitude, it is necessary to project the relative path error vector onto the vertical plane containing the path direction vector $\mathbf{q}$, as is shown in the figure by $\mathbf{q}$-$\mathbf{k^i}$ plane. We denote the projection of $\mathbf{e}_p$ onto this plane as $\mathbf{s}$. Referring to the details in the $\mathbf{q}$-$\mathbf{k^i}$ plane depicted in Figure 6.3, and using similar triangles, we have the relationship

$$\frac{-s_d}{\sqrt{s_n^2 + s_e^2}} = \frac{-q_d}{\sqrt{q_n^2 + q_e^2}}$$

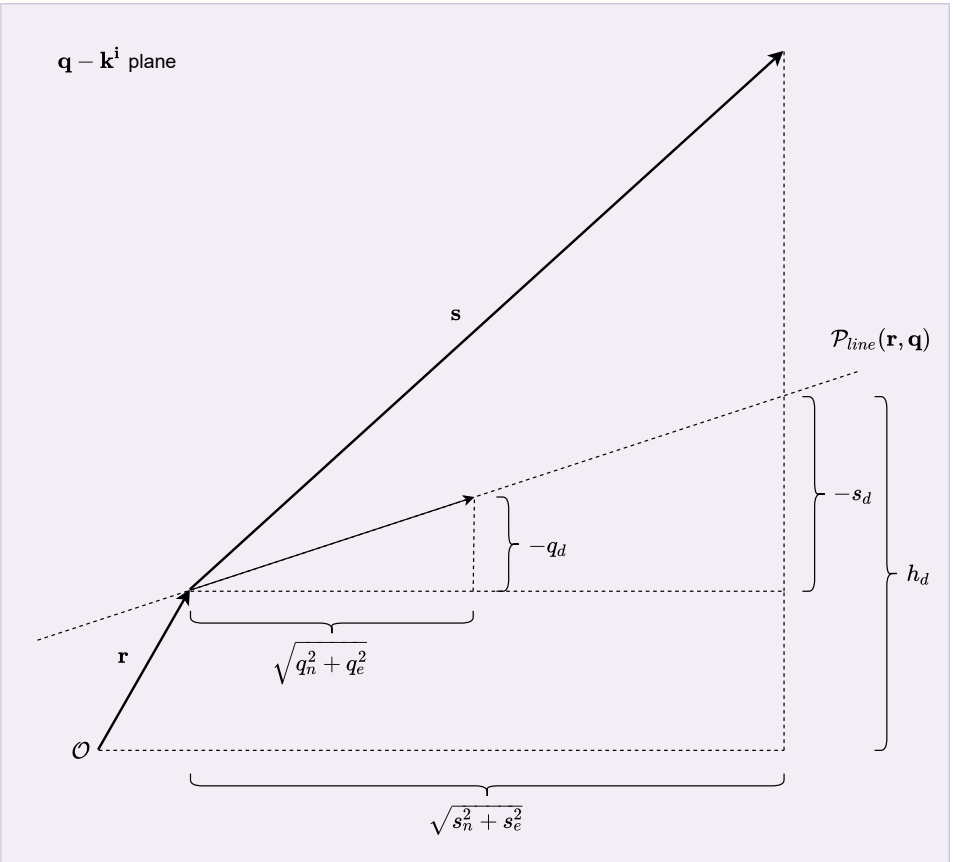


Figure 6.3: Project of the straight-line path following problem onto the $\mathbf{q}$-$\mathbf{k^i}$ plane

The projection $\mathbf{s}$ of the relative error vector is defined as

$$\begin{aligned}\mathbf{s^i} &= \begin{pmatrix} s_n \\ s_e \\ s_d \end{pmatrix} \\ &= \mathbf{e_p^i} - (\mathbf{e_p^i} \cdot \mathbf{n})\mathbf{n}\end{aligned}$$

where $(\mathbf{e_p^i} \cdot \mathbf{n})\mathbf{n}$ is a vector containing all the elements of the relative error vector perpendicular to the $\mathbf{q}$-$\mathbf{k^i}$ plane, which has normal $\mathbf{n}$, and where

$$\mathbf{e_p^i} = \begin{pmatrix} e_{pn} \\ e_{pe} \\ e_{pd} \end{pmatrix} \triangleq \mathbf{p^i} - \mathbf{r^i} = \begin{pmatrix} p_n - r_n \\ p_e - r_e \\ p_d - r_d \end{pmatrix}$$

and unit vector normal to the $\mathbf{q}$-$\mathbf{k^i}$ plane is calculated as

$$\mathbf{n} = \frac{\mathbf{q} \times \mathbf{k^i}}{\|\mathbf{q} \times \mathbf{k^i}\|}$$

From Figure 6.3, the desired altitude for an aircraft at $\mathbf{p}$ following the straight-line path $\mathcal{P}_{line}(\mathbf{r}, \mathbf{q})$, is given by:

$$h_d(\mathbf{r}, \mathbf{p}, \mathbf{q}) = -\left(r_d + \sqrt{s_n^2 + s_e^2}\left(\frac{q_d}{\sqrt{q_n^2 + q_e^2}}\right)\right) \tag{6.6}$$

Which can be used to provide an altitude command for the aircraft if we have access to $(\mathbf{r}, \mathbf{p}, \mathbf{q})$. This will always be the case because we will have both $\mathbf{r}$ and $\mathbf{q}$ from our waypoint path, and $\mathbf{p}$ is simply our inertial position which is provided by the state estimator, as depicted in Figure 1.2.

Having established the problem we are trying to solve for both course angle, Equation (6.5), and altitude, Equation (6.6), we can now develop guidance laws for our straight line path following algorithms which will solve these.

Longitudinal Guidance Strategy

The longitudinal guidance strategy for straight line path following is very simply stated as setting our altitude command, h_c, to the desired altitude established in Equation (6.6). As such:

$$h_c(\mathbf{r}, \mathbf{p}, \mathbf{q}) = h_d(\mathbf{r}, \mathbf{p}, \mathbf{q}) = -\left(r_d + \sqrt{s_n^2 + s_e^2}\left(\frac{q_d}{\sqrt{q_n^2 + q_e^2}}\right)\right) \tag{6.7}$$

Lateral Guidance Strategy

As was stated in Equation (6.5), we need to select a course angle command to feed into our autopilot such that we reduce the cross-tracking error, e_{py}, to zero over time. The strategy in this section will be to construct a desired course angle at every spatial point relative to the straight-line path that in results in the system moving toward the path. This set of desired course angles at every point is a vector-field because the desired course angle specifies a vector (relative to the straight line) with a unity magnitude, but a spatially varying direction.

The objective is to construct the previously mentioned vector field so that when e_{py} is large, the system is directed to approach the path with course angle $\chi^\infty \in (0, \frac{\pi}{2}]$, and so that as e_{py} approaches zero, the desired course angle approaches the course angle of the straight line, χ_q. Toward that end, we define the course angle command, χ_c, which

is fed to the autopilot of the aircraft as

$$\chi_c(e_{py}) = \chi_q - \chi^\infty \frac{2}{\pi} \tan^{-1}(k_{path} e_{py}) \tag{6.8}$$

where k_{path} is a positive constant that influences the rate of transition from χ^∞ to χ_q. Large values of k_{path} result in short, abrupt transitions, while small values cause long, smooth transitions to the desired course.

It should be noted that Equation (6.8) can result in undesirable behaviour if χ_q is computed directly from Equation (6.1), where atan2 returns an angle between $\pm\pi$. This is because of the discontinuity generated at $\pm\pi$, which means that the straight line course angle, χ_q, being expressed as a number slightly smaller than $+\pi$ would result in an aircraft which is currently flying with a course of a number slightly smaller in magnitude than $-\pi$ to bank to the right to try and fly almost 2π rad around to align itself with the desired course angle. Of course, the intended behaviour is to simply bank to the left and travel to a number slightly larger in magnitude than $-\pi$. This scenario is depicted in Figure 6.4.

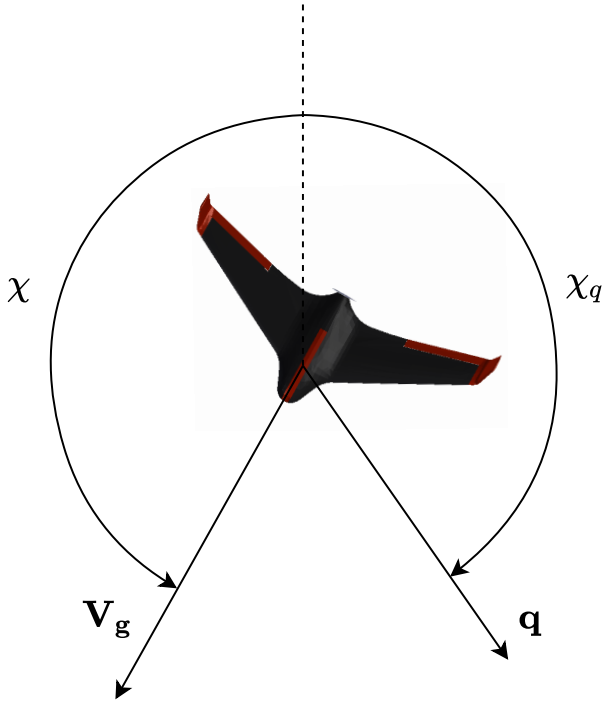


Figure 6.4: Course angle command adjustment to account for current course angle

In order to overcome this, we not not compute χ_q purely from Equation (6.1), but extend this to take the current course angle of the aircraft into account by stating

$$\chi_q = \text{atan2}(q_e, q_n) + 2\pi m \tag{6.9}$$

where $m \in \mathbb{N}$ is selected so that $-\pi \leq \chi_q - \chi \leq \pi$.

In other words, we actively adjust χ_q so that it also switches in sign the moment that our measurement switches in sign. This means that there will never be a discontinuity generated in the error signal between χ_c and χ, and that we will only ever take the most logical route to get to the desired course angle.

The algorithm used to calculate our commanded altitude and course angle using our straight line following algorithm is summarised in Algorithm 2, with the code as implemented shown in Listing B.7, which is made use of in Simulink as shown in Figures A.37 and A.38.

Algorithm 2 $[h_c, \chi_c] = \text{straightLineFollowing}(\mathbf{r}, \mathbf{q}, \mathbf{p}, \chi)$

Compute commanded altitude, h_c, using Equation (6.7)
$\chi_q \leftarrow \text{atan2}(q_e, q_n)$
while $\chi_q - \chi < -\pi$ **do**
    $\chi_q \leftarrow \chi_q + 2\pi$
end while
while $\chi_q - \chi > \pi$ **do**
    $\chi_q \leftarrow \chi_q - 2\pi$
end while
$e_{py} \leftarrow -\sin\chi_q\,(p_n - r_n) + \cos\chi_q\,(p_e - r_e)$
Compute commanded course angle, χ_c, using Equation (6.8)

6.1.2 Orbit Following

An orbit path is described by a center, $\mathbf{c} \in \mathbb{R}^3$, a radius, $\rho \in \mathbb{R}$, and a direction, $\lambda \in \{-1, 1\}$, as

$$\mathcal{P}_{orbit}(\mathbf{c}, \rho, \lambda) = \{\mathbf{r} \in \mathbb{R}^3 : \mathbf{r} = \mathbf{c} + \lambda\rho\,(\cos\varphi, \sin\varphi, 0)^T, \varphi \in [0, 2\pi)\} \tag{6.10}$$

where $\lambda = 1$ signifies a clockwise orbit, and $\lambda = -1$, signifies a counter-clockwise orbit. We assume that the center of the orbit is expressed in inertial coordinates such that $\mathbf{c} = (c_n, c_e, c_d)^T$.

Figure 6.5 shows a top view of an orbital path. The guidance strategy for orbit following is best derived in polar coordinates. We let d be the radial distance from the desired center of the orbit to the aircraft, and let φ be the phase angle of the relative position, as shown in the figure.

As is shown in Figure 6.5, for a clockwise orbit, the desired course angle when the aircraft is located on the orbit is given by $\chi^0 = \varphi + \pi/2$. Similarly, for a counter-clockwise orbit,

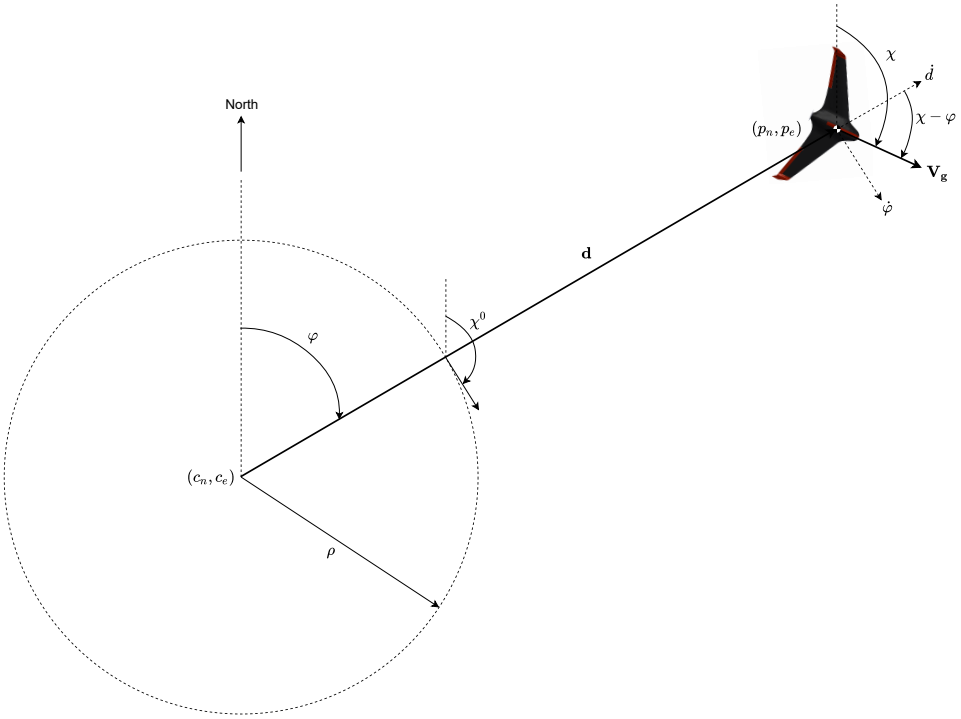


Figure 6.5: Orbit following geometry

the desired angle is given by $\chi^0 = \varphi - \pi/2$. Thus, in general we have

$$\chi^0 = \varphi + \lambda\frac{\pi}{2}$$

The control objective is to drive $d(t)$ to the orbit radius, ρ, and to drive the course angle $\chi(t)$ to χ^0 in the presence of wind.

In order to do this, we make use of a similar strategy to that of course angle following for straight line paths presented in Section 6.1.1. The strategy is to construct a desired course field that moves the aircraft onto the orbit, $\mathcal{P}_{orbit}(\mathbf{c}, \rho, \lambda)$. When the distance between the aircraft and the center of the orbit is large, it is desirable for the system to fly toward the orbit center. In other words, when $d \gg \rho$, the desired course is

$$\chi_d = \chi^0 + \lambda\frac{\pi}{2}$$

and when $d = \rho$, the desired course is $\chi_d = \chi^0$. Thus, a candidate course field is given by

$$\chi_d(d - \rho, \lambda) = \chi^0 + \lambda \tan^{-1}\left(k_{orbit}\left(\frac{d-\rho}{\rho}\right)\right) \tag{6.11}$$

where $k_{orbit} > 0$ is a constant that specifies the rate of transition from $\lambda\pi/2$ to zero. This expression for χ_d is valid for all values of $d > 0$.

As such, the course command for orbit following can be given by

$$\chi^c(t) = \varphi + \lambda \left[\frac{\pi}{2} + \tan^{-1}\left(k_{orbit} \left(\frac{d-\rho}{\rho} \right) \right) \right] \tag{6.12}$$

Similarly to the discussion on the straight line course angle consideration, we need to handle the case where if the angular position in the orbit, φ, is computed to be between $\pm\pi$, then there will be a discontinuity of 2π in the commanded course as the aircraft transitions from $\varphi = \pi$ to $\varphi = -\pi$. To alleviate this problem, φ should be computed as

$$\varphi = \text{atan2}(p_e - c_e, p_n - c_n) + 2\pi m$$

where $m \in \mathbb{N}$ is selected so that $-\pi \leq \varphi - \chi \leq \pi$.

In order to calculate our commanded altitude, h_c, during the orbit-following algorithm, we make use of the geometry shown in Figure 6.6 where we have 3 waypoints, $(\mathbf{W_{i-1}}, \mathbf{W_i}, \mathbf{W_{i+1}})$, which generate two direction vectors, $\mathbf{q_{i-1}}$ and $\mathbf{q_i}$. Our red dashed line illustrates the trajectory we would expect to follow in this scenario - which gives us a method by which we can calculate our altitude command.

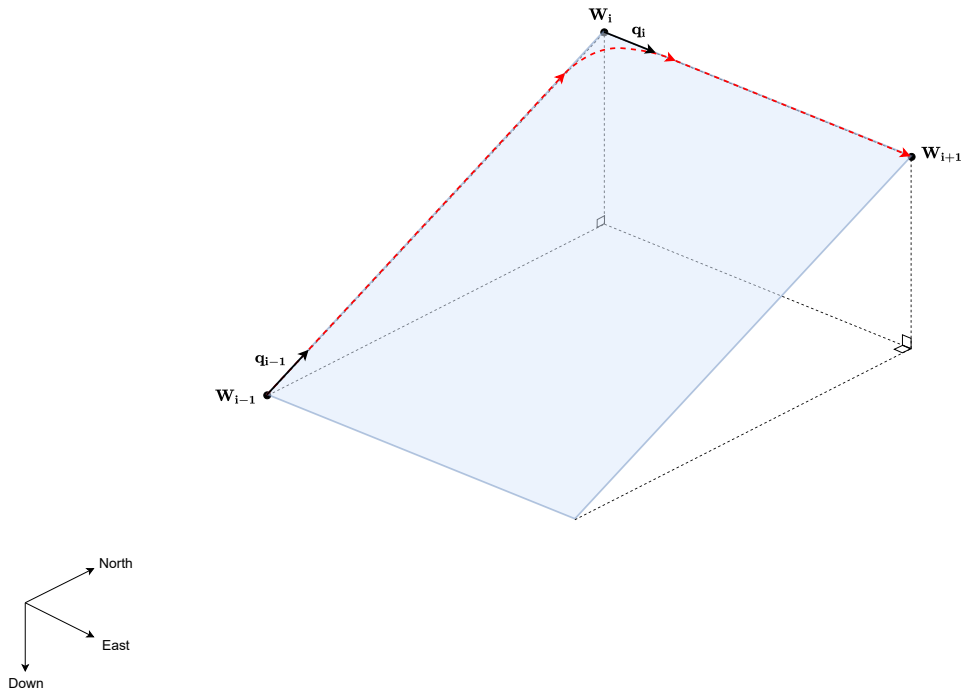


Figure 6.6: Orbit following inclined plane altitude following

This is done by making use of the fact that we would like to remain on the plane denoted in Figure 6.6 during the orbit portion of the trajectory. As such, we construct this plane by first generating the normal to the plane as

$$\mathbf{n} = \frac{\mathbf{q_i} \times \mathbf{q_{i-1}}}{\|\mathbf{q_i} \times \mathbf{q_{i-1}}\|}$$

We then make use of the standard plane equation in the point-normal form as

$$m(x_0 - x) + n(y_0 - y) + o(z_0 - z) = 0$$

In order to calculate our commanded altitude as:

$$h_c = -\left(\frac{n_n(\mathbf{W_{i_n}} - p_n) + n_e(\mathbf{W_{i_e}} - p_e)}{n_d} + \mathbf{W_{i_d}}\right) \tag{6.13}$$

where we have made use of the normal to the plane,

$$\mathbf{n} = \begin{pmatrix} n_n \\ n_e \\ n_d \end{pmatrix}$$

and a point on the plane (either waypoint would work here),

$$\mathbf{W_i} = \begin{pmatrix} W_{i_n} \\ W_{i_e} \\ W_{i_d} \end{pmatrix}$$

as well as our current inertial position in the north and east directions as p_n, and p_e, respectively.

Given that our orbit-following algorithm itself does not have access to some of the information needed to calculate the altitude command, we will make use of the fact that the inertial down position of the orbit center is not used in any calculation and as such can be used by the path management algorithm, discussed in Section 6.2, to provide the commanded altitude as per Equation (6.13). As such,

$$h_c = -c_d \tag{6.14}$$

The orbit following algorithm used is stated in Algorithm 3, with the code as implemented shown in Listing B.8, which is made use of in Simulink as shown in Figures A.37 and A.38.

Algorithm 3 $[h_c, \chi_c] = \text{orbitFollowing}(\mathbf{c}, \rho, \lambda, \mathbf{p}, \chi)$

$h_c = -c_d$
$d \leftarrow \sqrt{(p_n - c_n)^2 + (p_e - c_e)^2}$
$\varphi \leftarrow \text{atan2}(p_e - c_e, p_n - c_n)$
while $\varphi - \chi < -\pi$ **do**
    $\varphi \leftarrow \varphi + 2\pi$
end while
while $\varphi - \chi > \pi$ **do**
    $\varphi \leftarrow \varphi - 2\pi$
end while
Compute commanded course angle, χ_c, using Equation (6.12)

6.2 Path Management

In the previous section we developed strategies for following straight-line paths and circular orbits. The objective of this section is to describe a strategy that combines these straight-line paths and circular orbits to synthesize general classes of paths that are useful for autonomous operation of small unmanned aircraft.

We will discuss how to transition between waypoints by illustrating all of the required geometry to generate an algorithm which can be used to switch between the path following algorithms. In general this will result in a system which takes a set of waypoints, the current inertial position, and the minimum turning radius currently possible, as inputs and converts these into the required commands to select between Algorithm 2 and Algorithm 3. This algorithm will then provide the path following algorithms with all required inputs to generate our altitude and course angle commands which will be provided to the previously designed autopilot. This work primarily follows that presented in [67], albeit with some notable deviations such as the altitude command during circular orbits. This is stated as the Path Manager in Figure 1.2.

6.2.1 Waypoint Following with Fillets

We define a set of waypoints as

$$\mathcal{W} = \{\mathbf{W_1}, \mathbf{W_2}, ...\mathbf{W_n}\} \tag{6.15}$$

where $\mathbf{W_i} = (W_{n_i}, W_{e_i}, W_{d_i})^T \in \mathbb{R}^3$. In this section we address the problem of switching from one waypoint segment to another. Consider the scenario shown in Figure 6.7 that depicts an aircraft tracking the straight-line segment $\overline{\mathbf{W_{i-1}W_i}}$.

Intuitively, when the aircraft reaches $\mathbf{W_i}$, we desire to switch the guidance algorithm so that it will track the straight-line segment $\overline{\mathbf{W_iW_{i+1}}}$. One possible strategy is to switch

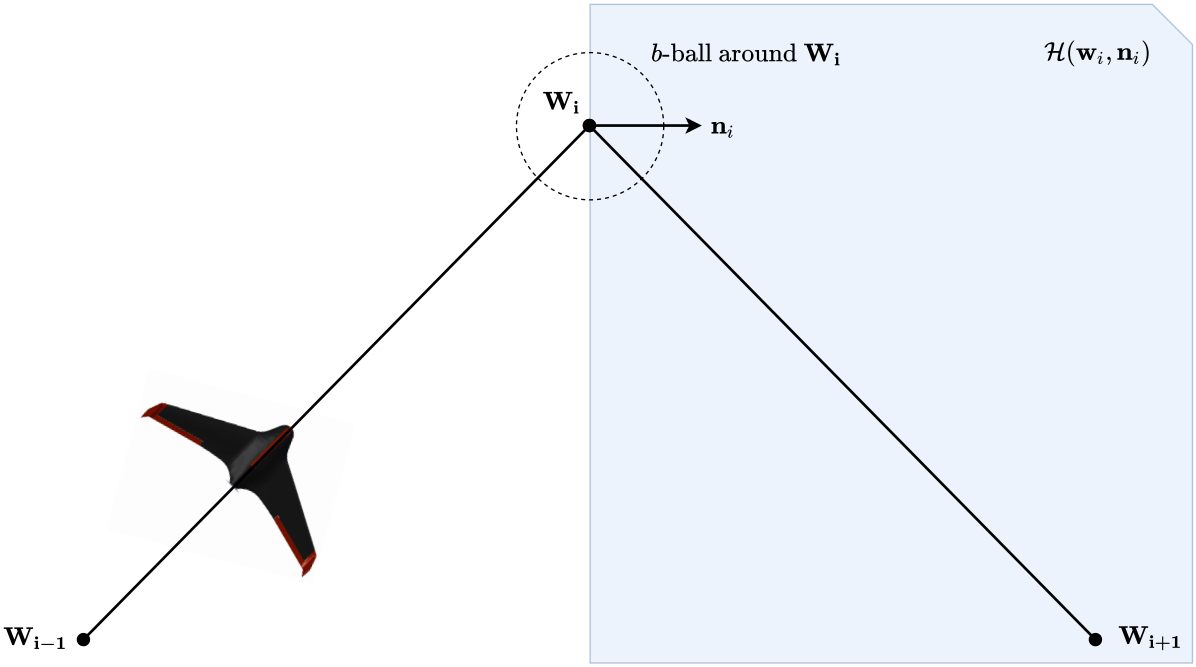


Figure 6.7: b-ball switching between waypoint segments

when the UAV enters a sphere, or "ball", around $\mathbf{W_i}$. In other words, the guidance algorithm would switch at the first time instant when

$$\|\mathbf{p}(t) - \mathbf{W_i}\| \leq b$$

where b is the radius of the sphere, and $\mathbf{p}(t)$ is the location of the aircraft. However, if there are disturbances like wind, if b is chosen too small, or if the segment from $\mathbf{W_{i-1}}$ to $\mathbf{W_i}$ is short and the tracking algorithm has not had time to converge, then the aircraft might never enter the b-ball around $\mathbf{W_i}$.

A better approach, and one that is not sensitive to tracking error, is to use a half-plane switching criteria. Given a point $\mathbf{r} \in \mathbb{R}^3$ and a normal vector $\mathbf{n} \in \mathbb{R}^3$, we define the half plane

$$\mathcal{H}(\mathbf{r}, \mathbf{n}) \triangleq \{\mathbf{p} \in \mathbb{R}^3 : (\mathbf{p} - \mathbf{r})^T \mathbf{n} \geq 0\}$$

Referring to Figure 6.7, we define the unit vector pointing in the direction of the line $\overline{\mathbf{W_i W_{i+1}}}$ as

$$\mathbf{q_i} \triangleq \frac{\mathbf{W_{i+1}} - \mathbf{W_i}}{\|\mathbf{W_{i+1}} - \mathbf{W_i}\|} \tag{6.16}$$

and accordingly, the unit normal to the 3-D half plane that separates the line $\overline{\mathbf{W_{i-1} W_i}}$

from the line $\overline{\mathbf{W_i W_{i+1}}}$ is given by

$$\mathbf{n_i} \triangleq \frac{\mathbf{q_{i-1}} + \mathbf{q_i}}{\|\mathbf{q_{i-1}} + \mathbf{q_i}\|}$$

The aircraft tracks the straight-line path from $\mathbf{W_{i-1}}$ to $\mathbf{W_i}$, until it enters $\mathcal{H}(\mathbf{w_i}, \mathbf{n_i})$, at which point it will track the straight-line path from $\mathbf{W_i}$ to $\mathbf{W_{i+1}}$. The benefit of this approach is that it is extremely simple to implement and it ensures that the aircraft reaches the waypoint before transitioning to the next straight-line path. However, the paths generated provide neither a smooth, nor a balanced, transition between the line segments connecting waypoints and thus we need to extend this in order to provide this functionality.

The extension to this which is made use of in this text is to insert a fillet between the line segments, as is shown in Figure 6.8. A disadvantage of the paths generated by this algorithm is that one does not pass through the waypoint $\mathbf{W_i}$, which is sometimes desirable. However, it is assumed for this project that the line segments carry a higher importance than the waypoints, so smoothly transitioning between the line segments connecting the waypoints is more desirable than flying through the waypoints themselves.

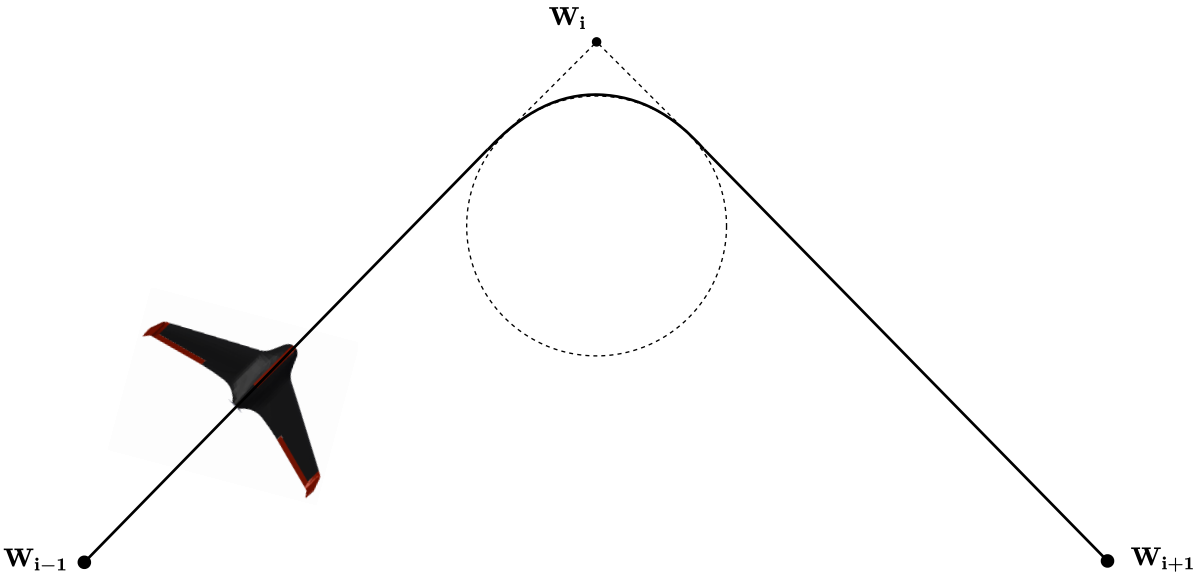


Figure 6.8: Fillet insertion between line segments $\overline{\mathbf{W_{i-1}W_i}}$ and $\overline{\mathbf{W_i W_{i+1}}}$

The geometry near the transition between the line segments is illustrated in Figure 6.9. With the unit vector $\mathbf{q_i}$ aligned with the line between waypoints $\mathbf{W_i}$ and $\mathbf{W_{i+1}}$, as per Equation (6.16), the angle between $\overline{\mathbf{W_{i-1}W_i}}$ and $\overline{\mathbf{W_i W_{i+1}}}$ is given by

$$\varrho \triangleq \cos^{-1}(-\mathbf{q_{i-1}}^T \mathbf{q_i}) \tag{6.17}$$

If the radius of the fillet is R, as is shown in the figure, then the distance between the waypoint $\mathbf{W_i}$ and the location where the fillet intersects the line $\overline{\mathbf{W_i W_{i+1}}}$ is $R/\tan\frac{\varrho}{2}$, and the distance between $\mathbf{W_i}$ and the edge of the fillet circle along the bisector of ϱ is given by $R/\sin\frac{\varrho}{2} - R$.

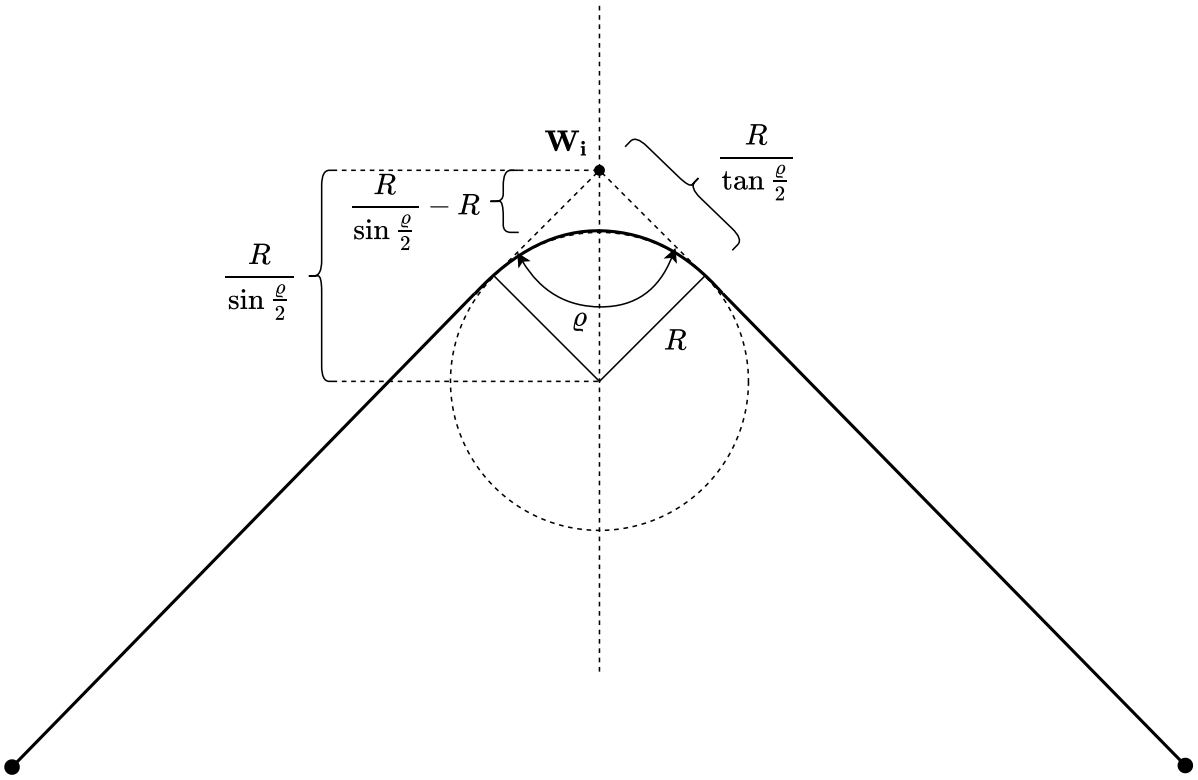


Figure 6.9: Geometry of fillet insertion between line segments $\overline{\mathbf{W_{i-1}W_i}}$ and $\overline{\mathbf{W_i W_{i+1}}}$

In order to implement the fillet manoeuvre using the path-following algorithms described in the previous section, we will follow the straight-line segment $\overline{\mathbf{W_{i-1}W_i}}$ until entering the half plane $\mathcal{H}_1$, as is shown in Figure 6.10. The right-handed circular orbit of radius R is then followed until entering the half plane $\mathcal{H}_2$, as is shown in the figure, at which point the straight-line segment $\overline{\mathbf{W_i W_{i+1}}}$ is followed.

As shown in Figure 6.10, the center of the fillet circle is given by

$$\mathbf{c} = \mathbf{W_i} - \left(\frac{R}{\sin\frac{\varrho}{2}}\right)\frac{\mathbf{q_{i-1}} - \mathbf{q_i}}{\|\mathbf{q_{i-1}} - \mathbf{q_i}\|}$$

Similarly, the half plane $\mathcal{H}_1$ is defined by the location

$$\mathbf{r_1} = \mathbf{W_i} - \left(\frac{R}{\tan\frac{\varrho}{2}}\right)\mathbf{q_{i-1}}$$

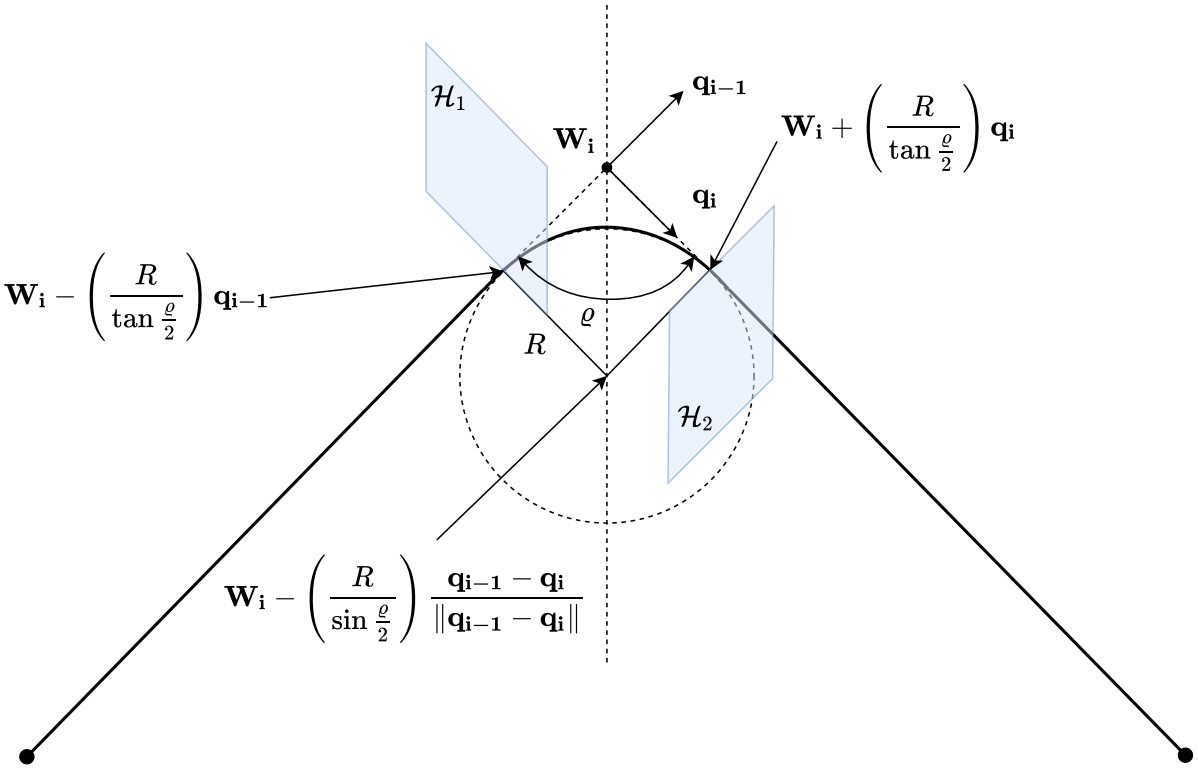


Figure 6.10: Geometry of fillet insertion between line segments $\overline{\mathbf{W_{i-1}W_i}}$ and $\overline{\mathbf{W_iW_{i+1}}}$ indicating half planes needed for the path following algorithm

and the normal vector $\mathbf{q_{i-1}}$.

The half plane $\mathcal{H}_2$ is defined by the location

$$\mathbf{r_1} = \mathbf{W_i} + \left(\frac{R}{\tan\frac{\varrho}{2}}\right)\mathbf{q_i}$$

and the normal vector $\mathbf{q_i}$.

The algorithm for manoeuvring along the waypoint path $\mathcal{W}$ using fillets to smooth between the straight-line segments is given by Algorithm 4.

Algorithm 4 [flag, $\mathbf{r}$, $\mathbf{q}$, $\mathbf{c}$, ρ, λ]= pathManager_followWaypointsFillet($\mathcal{W}, \mathbf{p}, R$)

if New waypoint path $\mathcal{W}$ is received **then**
    Use $\mathbf{p}$ to find the closest waypoint to current position, $\mathbf{W_x}$, i = x - 1
    state = 1
end if
$\mathbf{q_{i-1}} \leftarrow \frac{\mathbf{W_i} - \mathbf{W_{i-1}}}{\|\mathbf{W_i} - \mathbf{W_{i-1}}\|}$
$\mathbf{q_i} \leftarrow \frac{\mathbf{W_{i+1}} - \mathbf{W_i}}{\|\mathbf{W_{i+1}} - \mathbf{W_i}\|}$

7: $\varrho \leftarrow \cos^{-1}(-\mathbf{q_{i-1}}^T \mathbf{q_i})$
8: **if** state == 1 **then**
9: flag $\leftarrow 1$
10: $\mathbf{r} \leftarrow \mathbf{W_{i-1}}$
11: $\mathbf{q} \leftarrow \mathbf{q_{i-1}}$
12: $\mathbf{z} \leftarrow \mathbf{W_i} - \left(\frac{R}{\tan(\varrho/2)}\right)\mathbf{q_{i-1}}$
13: **if** $\mathbf{p} \in \mathcal{H}(\mathbf{z}, \mathbf{q_{i-1}})$ **then**
14: state $\leftarrow 2$
15: **end if**
16: **else if** state == 2 **then**
17: flag $\leftarrow 2$
18: $\mathbf{c} \leftarrow \mathbf{W_i} - \left(\frac{R}{\sin\frac{\varrho}{2}}\right)\frac{\mathbf{q_{i-1}} - \mathbf{q_i}}{\|\mathbf{q_{i-1}} - \mathbf{q_i}\|}$
19: Calculate h_c with Equation (6.13)
20: $c(3) \leftarrow -h_c$
21: $\rho \leftarrow R$
22: $\lambda \leftarrow \text{sign}(q_{i-1_N} q_{i_E} - q_{i-1_E} q_{i_N})$
23: $\mathbf{z} \leftarrow \mathbf{W_i} + \left(\frac{R}{\tan\frac{\varrho}{2}}\right)\mathbf{q_i}$
24: **if** $\mathbf{p} \in \mathcal{H}(\mathbf{z}, \mathbf{q_i})$ **then**
25: $i \leftarrow (i+1)$
26: **if** $i > N$ **then**
27: $i \leftarrow 2$
28: **end if**
29: state $\leftarrow 1$
30: **end if**
31: **end if**

The if statement in line 1 is used to determine if any waypoints in our waypoint path, $\mathcal{W}$ have changed. If so, we make use of our current position to calculate which is the waypoint we should be travelling to next, with the assumption that if we switch waypoint paths, we do not want to start from the beginning of the defined path, but simply travel to the next closest waypoint on the new path and then carry on as normal. We then calculate the unit vectors $\mathbf{q_{i-1}}$ and $\mathbf{q_i}$, followed by the fillet angle, ϱ, in lines 5-7.

A state of 1 in the state machine indicates that we are following a straight-line segment (which we also treat as the default state if the waypoint path has changed, as well as at startup), and as such the path manager commands the aircraft to follow the straight-line path along $\overline{\mathbf{W_{i-1}W_i}}$, which is parametrized by $\mathbf{r} = \mathbf{W_{i-1}}$ and $\mathbf{q} = \mathbf{q_{i-1}}$, which are assigned in lines 10 and 11. Lines 12-15 test to see if the aircraft has transitioned into the half plane shown as $\mathcal{H}_1$ in Figure 6.10. If it has, then the state machine is updated to a state of 2.

A state of 2 indicates that the aircraft is commanded to follow the orbit the defines the fillet. The center, radius, and direction of the orbit are assigned in lines 18-22, where

it should be noted that we make use of Equation (6.13) to ensure that we stay on the plane defined by $\mathbf{q_i}$, $\mathbf{q_{i-1}}$ and $\mathbf{W_i}$ while following the orbit. Lines 23-30 test to see if the aircraft has transitioned into the half plane shown as $\mathcal{H}_2$ in Figure 6.10. If it has, then the waypoint pointer is incremented, and the state machine is switched back to a state of 1 as we now command the aircraft to follow the line segment $\overline{\mathbf{W_i W_{i+1}}}$. If we have reached the last waypoint, we simply cycle back around so that we will fly a closed-loop and continue along this indefinitely unless instructed otherwise, this is handled in lines 26-28.

The implementation code for this algorithm is shown in Listing B.9, with the masked implementation of the full guidance and navigation system being illustrated in Figure 6.11.

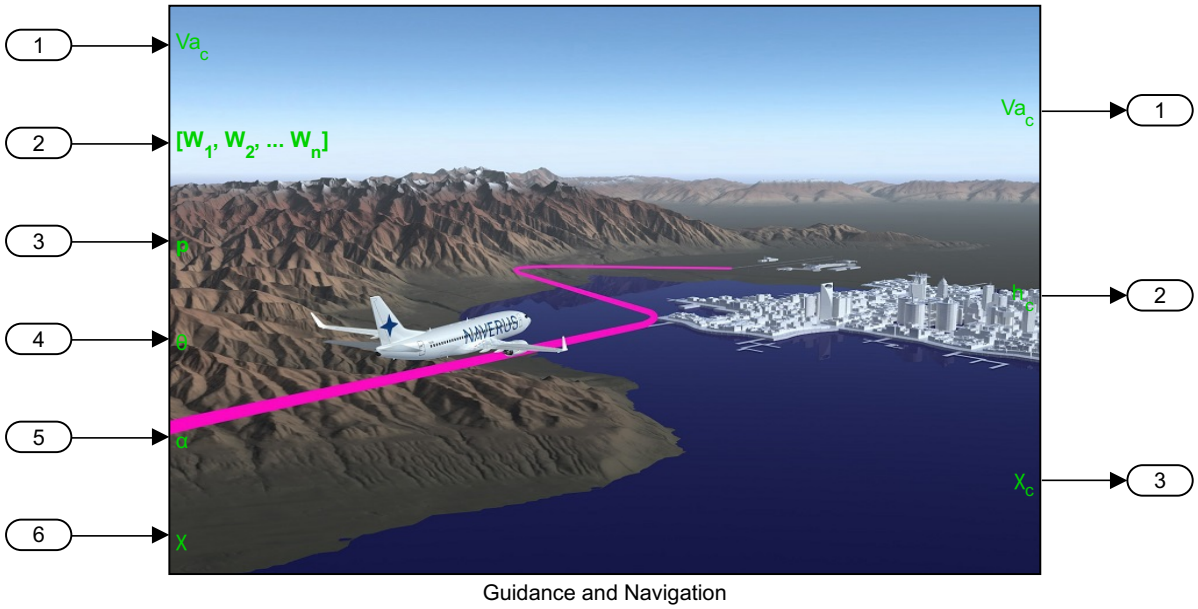


Figure 6.11: Masked guidance and navigation system

The unmasked variant of this is shown in Figure A.39, where it should be noted that the only item not yet discussed is the turning radius, R, which is one of the required inputs to the path manager. This value is calculated from Equation (2.63) as

$$R = SF\frac{V_g^2 \cos\gamma}{g\tan\left(0.5\phi_{max}\right)}$$

where R is the turning radius achievable given our current ground speed, V_g, and flight path angle, γ, where we know our maximum bank angle, ϕ_{max}, as defined in our controller design - where we use half of this value due to needing to first bank and then reduce to zero once the turn is complete. The value of SF is a safety factor which is applied to the turn radius so as to extend it and not specify course angles which require

near-optimal performance for the aircraft to achieve. The illustration of this calculation is depicted by its Simulink implementation in Figure A.39.

All of the required parameters for the full guidance and navigation system depicted in Figure 6.11 are provided in Table 6.1. Where it should be noted that the only item not discussed is the update rate at which the system will run. Similarly to the autopilot, it was decided to keep this consistent with all other systems in the design so this was set to update the commands to the autopilot at 0.01 s.

Table 6.1: Guidance and Navigation Design Parameters

Parameter	**Value**	**Unit**
T_s	0.01	s
χ_∞	1.5708	rad
k_{path}	0.01	Unitless
k_{orbit}	0.08	Unitless
SF	1.2	Unitless

6.3 Guidance and Navigation Verification

In order to verify the guidance and navigation algorithms previously developed, we make use of the set of waypoints depicted in Figure 1.3, with a starting trim condition at $h = 100$ m and $V_a = 17.3$ m/s (ie. we will not verify takeoff in this section). With this in mind we can see how our full system, including the autopilot and all non-linearities, operates with the inclusion of the guidance and navigation system - albeit in calm, windless conditions. This is depicted in in Figures 6.12, 6.13 and 6.14, and the Simulink model used to run these tests is provided in Figure A.40.

Figure 6.12 illustrates the system performance in an orthographic view so that we can view how the system responds from a three dimensional perspective. From this view its clear that we get accurate tracking and our path planning algorithm behaves as expected and generates fillets between the waypoint line segments. We also note that we get a nice co-planar response when transitioning along an inclined plane. This can most readily be seen in the case of the transition between line segments $\overline{\mathbf{W_6W_7}}$ and $\overline{\mathbf{W_7W_8}}$.

Figure 6.13 illustrates the system performance in the lateral case, depicting our $\mathbf{i^i}$-$\mathbf{j^i}$ plane. From this we can see how well the aircraft is able to transition between line segments, ignoring the effects of altitude. From the figure we are able to accurately transition between waypoint line segments, with very low deviation from the line segment in the transient. Where there is deviation, the system always corrects itself and positions itself back on the line segment as expected.

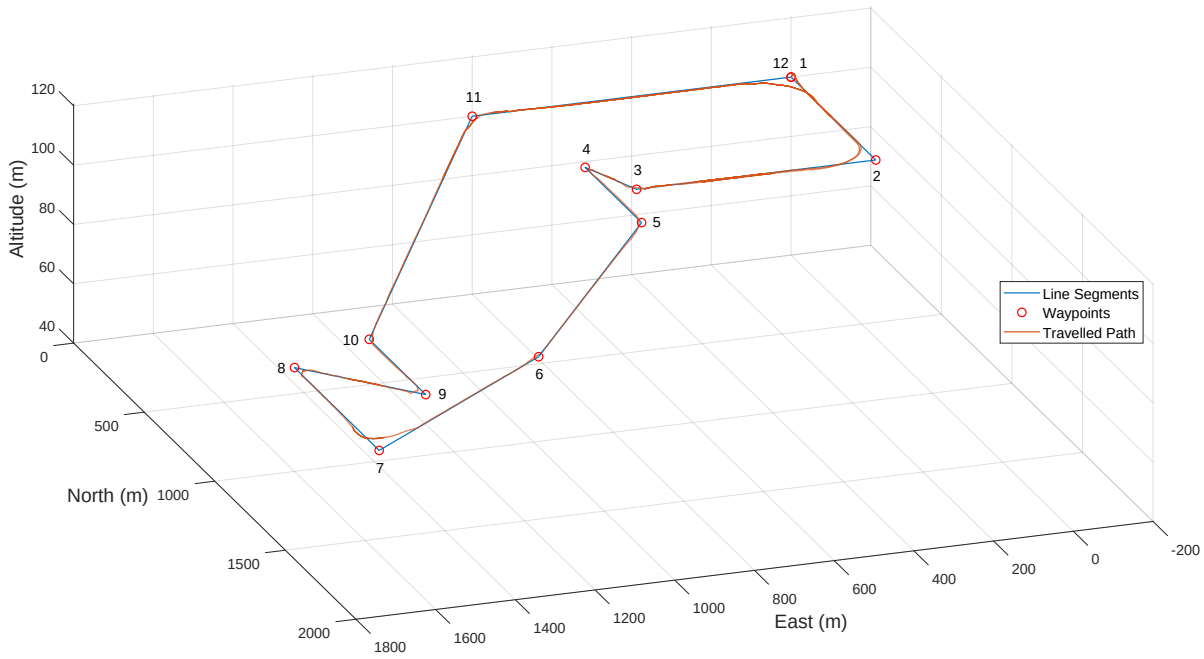


Figure 6.12: Guidance and navigation system verification, orthographic view

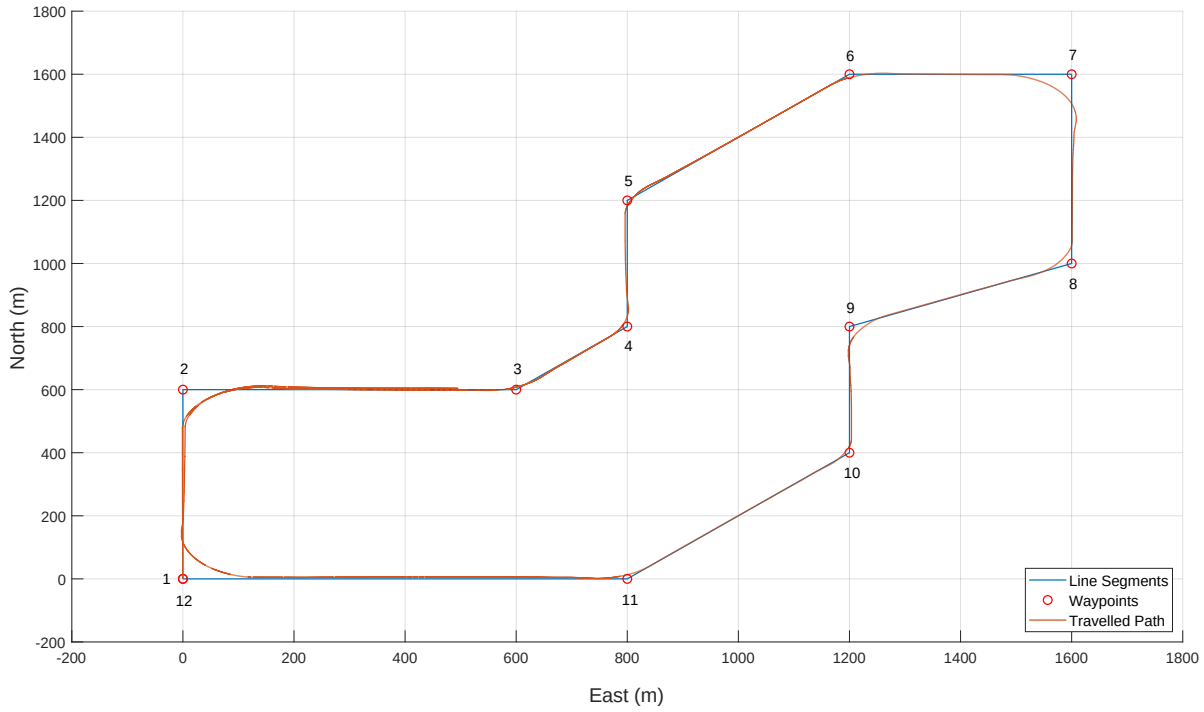


Figure 6.13: Guidance and navigation system verification, $\mathbf{i}^{\mathbf{i}}$-$\mathbf{j}^{\mathbf{i}}$ plane view

Figure 6.14 illustrates the system performance in the longitudinal case in one direction, depicting the $\mathbf{j}^{\mathbf{i}}$-$\mathbf{k}^{\mathbf{i}}$ plane. From this we can see that we get good altitude tracking, with very low deviation from the commanded line segments, and whenever deviation is

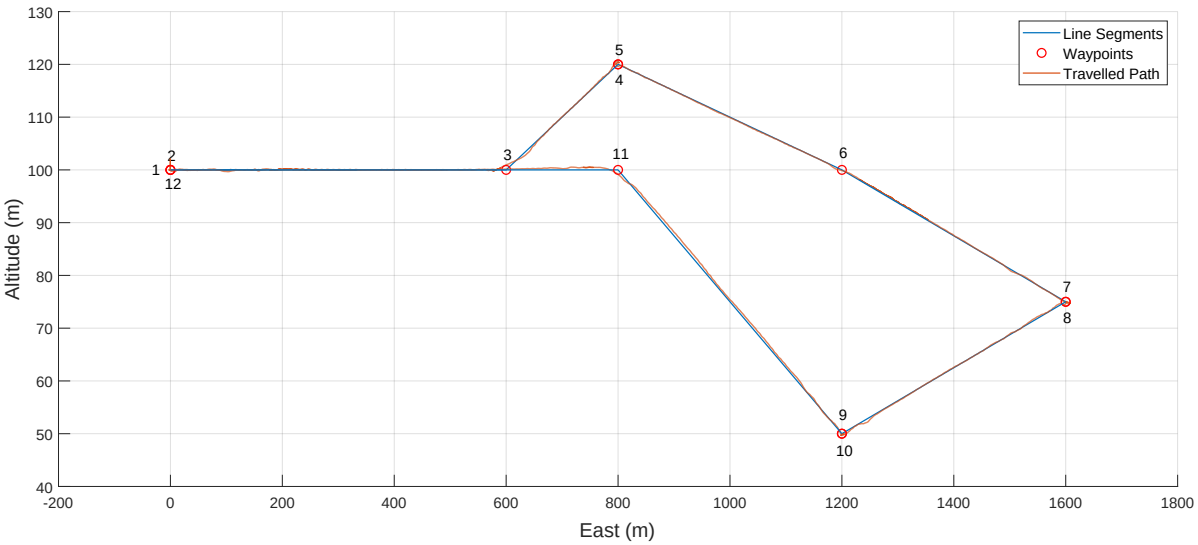


Figure 6.14: Guidance and navigation system verification, $\mathbf{j^i}$-$\mathbf{k^i}$ plane view

present, the system is forced back to the required value as expected.

With this, the full system depicted in Figure 1.2 has now been designed, optimised, and tested, under various conditions. As such, the design portion of this dissertation is now over, with work after this point being focused on system integration and verification.

CHAPTER 7

System Integration and Verification

The final step in the design process is to integrate all of the elements into a single digital implementation. This is very easily done with Simulink's code generation tool - which will generate all of the required .c and .h files, as well as build this into an executable file that can be called by an S-function within Simulink [68]. This is all automated with no additional input required, as long as care has been taken throughout the development process to ensure that one has created models that do not contain any errors or algebraic loops which would not be implementable in a standalone system.

In this project, what this means is that we can reduce the autopilot, state estimator and guidance and navigation algorithms to a single system which could be implemented on an embedded microcontroller. Given that this will only be tested on a host computer (as opposed to target hardware - which would be the final embedded flight system), this kind of testing is known as Software in the Loop (SiL) and ensures that one has software which can be executed to generate the required functionality. As such, it creates confidence in the algorithms generated, but does not guarantee correct operation on a target system as there might be hardware-specific problems that cause undesirable behaviour. Regardless of the limitations, this is a critical step in showing that the digital flight control system behaves as expected when executed as a binary.

This implementation is illustrated in Figure 7.1, which is the system which will be used in place of the autopilot, state estimator and guidance and navigation systems in all of the verification simulations performed in this chapter. It should be noted that the masked subsystem image used is that of the Pixhawk 2, and the reason for this is because this Guidance Navigation and Control (GNC) system could be implemented on an embedded platform such as the Pixhawk should one wish to perform processor in the loop (PiL), hardware in the loop (HiL), or flight tests with the algorithms developed and software generated with Simulink Code Generator.

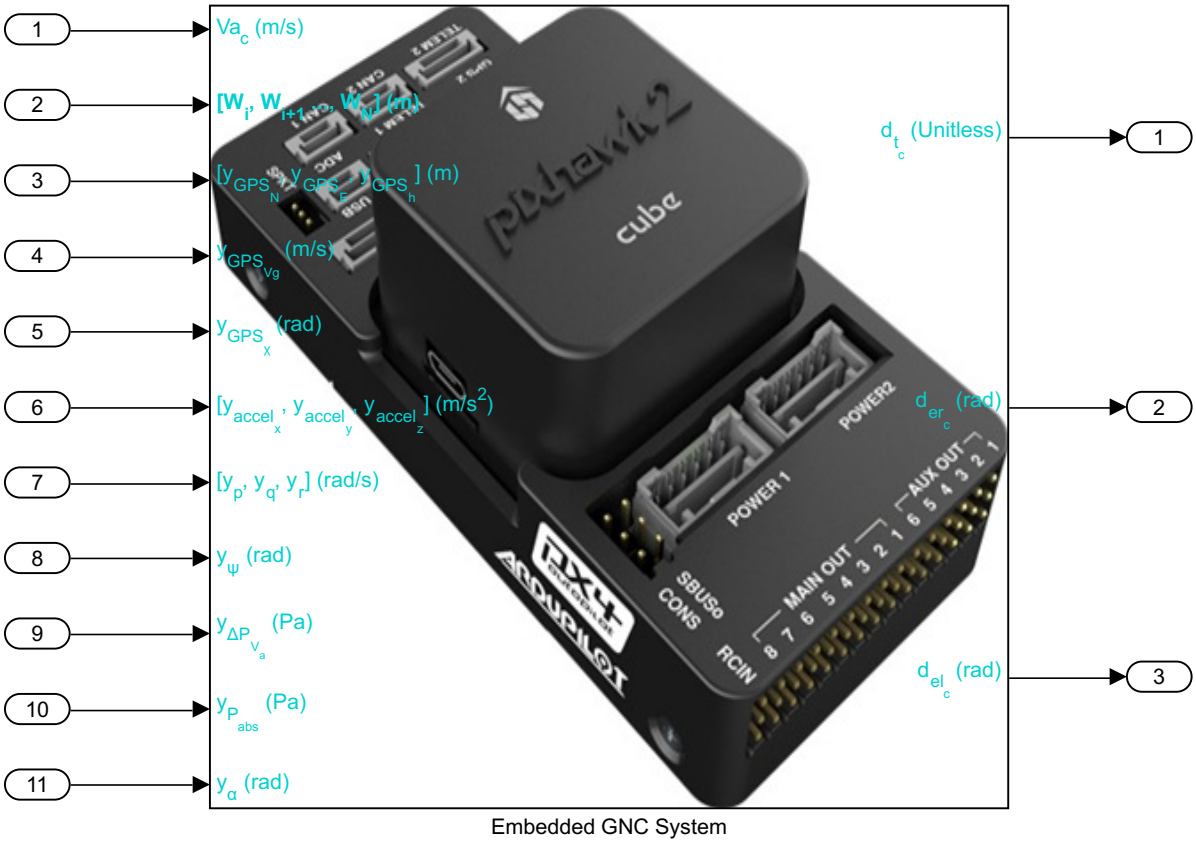


Figure 7.1: SiL implementation of the full Guidance, Navigation and Control (GNC) system

7.1 Benchmark Flight Paths

In order to review how effective our GNC system is, we make use of the flight paths depicted in Figure 1.3 and Figure 1.4. We run both cases individually in order to illustrate the performance with two different sets of waypoints. These are run with the full non-linear aircraft, actuator and sensor models in order to illustrate the performance of the system with the highest fidelity model available. In order to perform a like-for-like comparison with the results of our system prior to code generation (ie. as in Figures 6.12, 6.13 and 6.14, we do not include take-off functionality when looking at the waypoint path defined in Figure 1.3. However, this is included when considering the path defined in Figure 1.4 in order to illustrate that we are able to handle a take-off condition where the flight starts at 2 m above the ground.

The Simulink model used for these tests is depicted in Figure A.41.

7.1.1 Waypoint Path

In order to illustrate that our system operates as expected and as per our previous results, we compare the results generated in Figures 7.2, 7.3 and 7.4 to those in Figures 6.12, 6.13 and 6.14 where we can do a like-for-like comparison between the systems. These results from compiled, code-generated software, with the other set from our design model. From these figures we see that we get the expected performance, with no deviation from the design model variant.

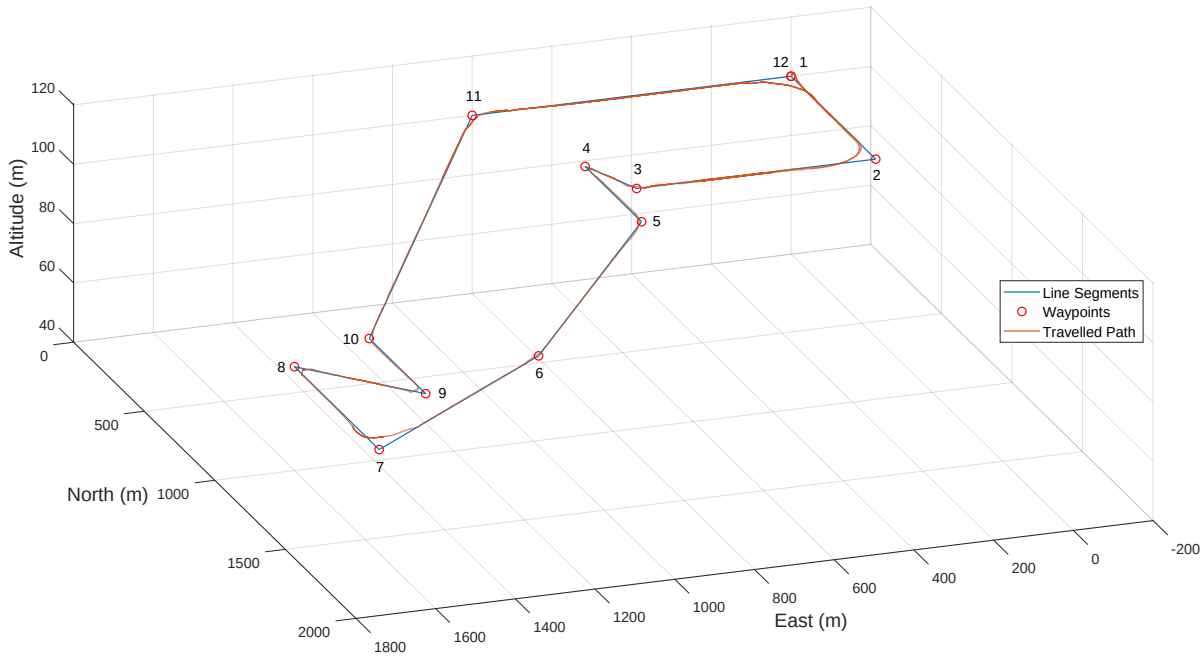


Figure 7.2: GNC system verification, orthographic view

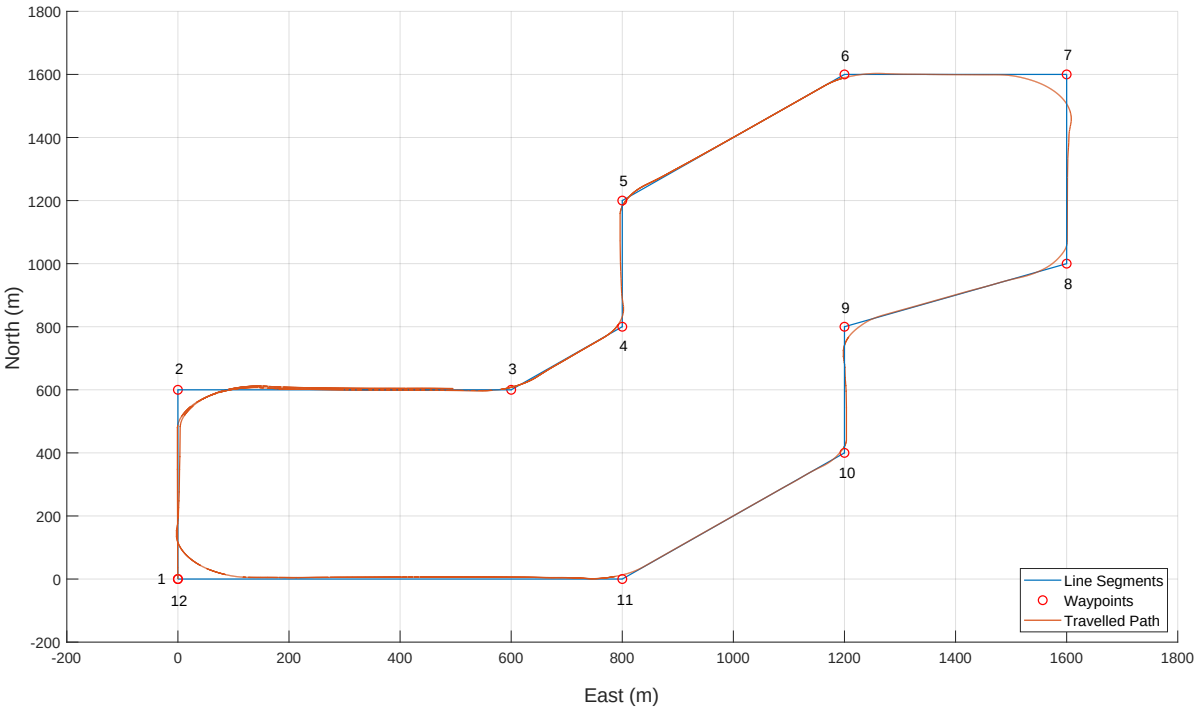


Figure 7.3: GNC system verification, $\mathbf{i^i}$-$\mathbf{j^i}$ plane view

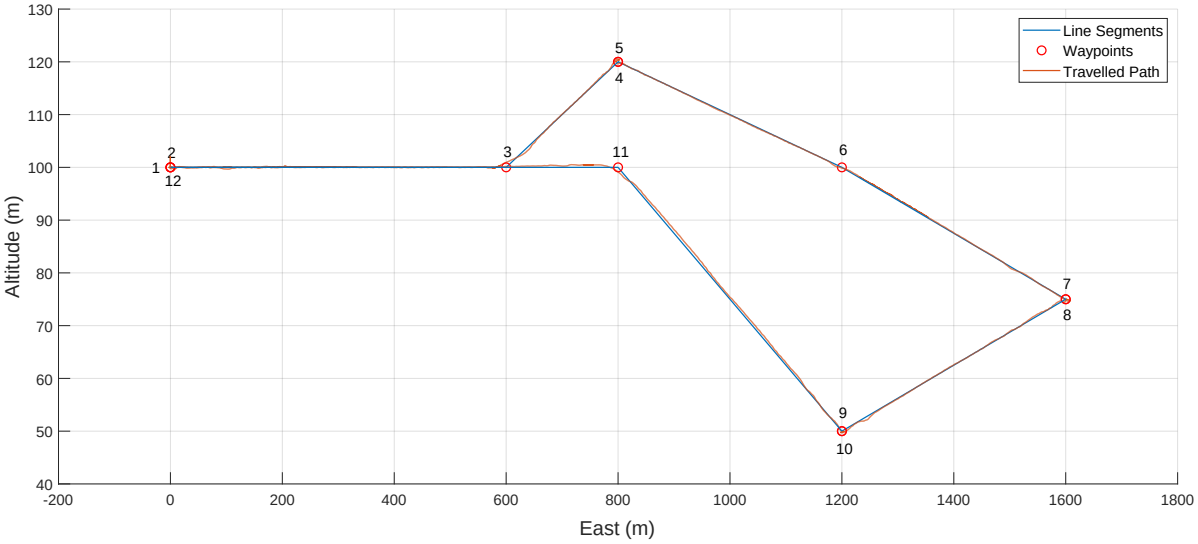


Figure 7.4: GNC system verification, $\mathbf{j^i}$-$\mathbf{k^i}$ plane view

7.1.2 Alternate Waypoint Path

The alternate waypoint path is depicted in Figure 1.4 and is simply another set of waypoints to ensure that we get adequate performance with some fairly difficult transitions. The results of this simulation are shown in Figures 7.5, 7.6 and 7.7.

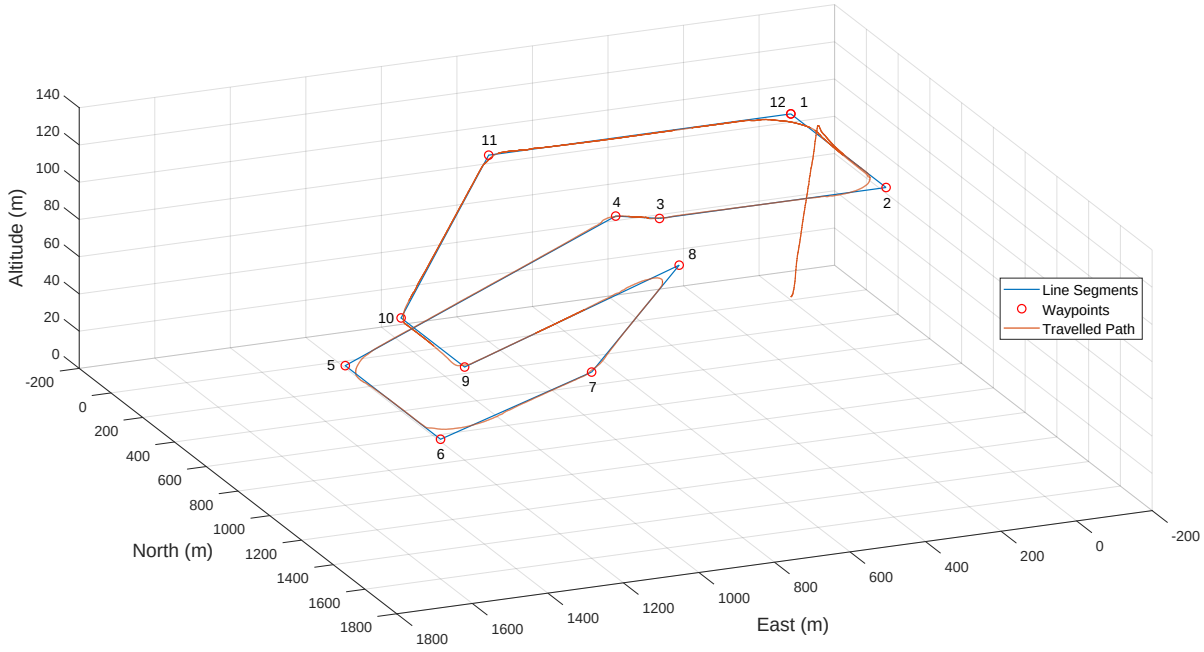


Figure 7.5: GNC system verification, alternate waypoints, orthographic view

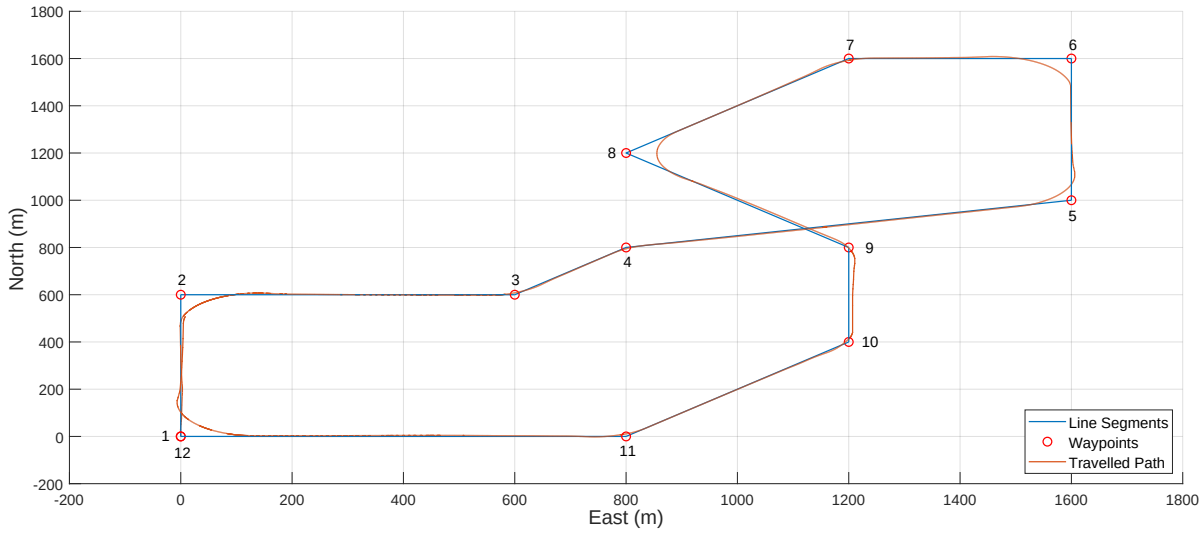


Figure 7.6: GNC system verification, alternate waypoints, $\mathbf{i^i}$-$\mathbf{j^i}$ plane view

From these results it should be clear that our GNC system behaves as expected with minimal error seen for all transitions. A particularly difficult manoeuvre worth highlighting is that of the transition between line segments $\overline{\mathbf{W_7W_8}}$ and $\overline{\mathbf{W_8W_9}}$, where the aircraft is initially ascending and flying in a south-westerly direction along $\overline{\mathbf{W_7W_8}}$ before having to simultaneously turn to fly in a south-easterly direction, as well as to start descending as is required for line segment $\overline{\mathbf{W_7W_8}}$. This transition fully shows the benefit of the co-planar altitude algorithm that was developed in Figure 6.6 and Equation (6.13) - as we remain co-planar with the plane defined by $\overline{\mathbf{W_7W_8}}$ and $\overline{\mathbf{W_7W_8}}$ throughout the

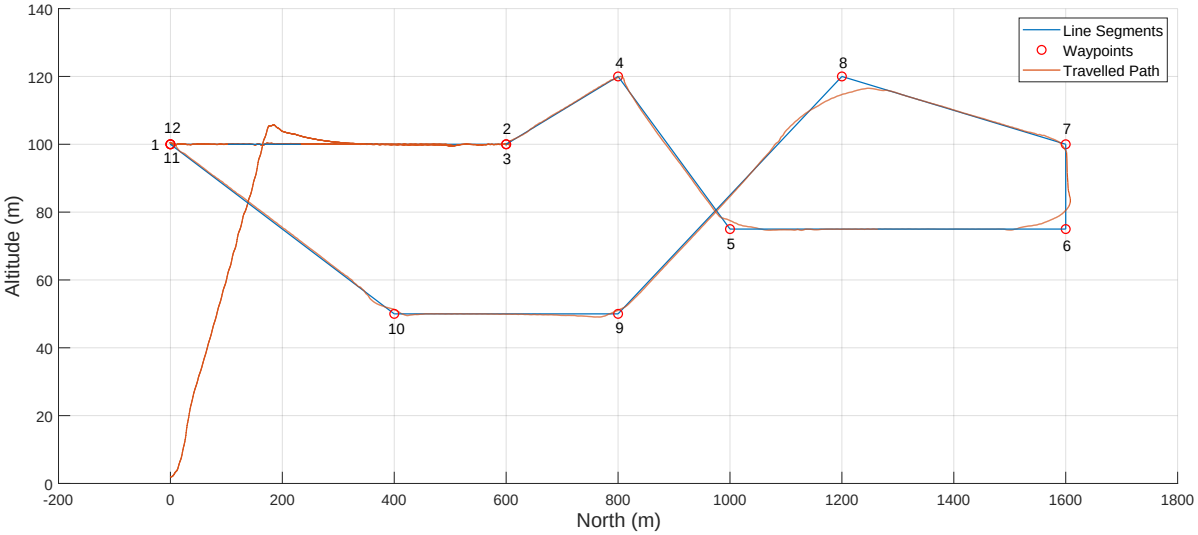


Figure 7.7: GNC system verification, alternate waypoints, $\mathbf{i^i}$-$\mathbf{k^i}$ plane view

manoeuvre.

Another aspect shown in these results is our take-off algorithm, which has not been illustrated as yet in the text. This simply shows that we are able to start from a low altitude (2 m above ground level in this instance) and gain altitude so as to track our desired waypoint path. This is the expected behaviour and as such our take-off algorithm is shown to operate as expected.

7.2 Wind Inclusion

We extend the previous section by adding the effect of wind, as was discussed in Section 2.5. This allows us to determine if our GNC system is still able to perform as expected under quite severe wind conditions. Again, this is done for both sets of waypoints individually as depicted in Figure 1.3 and Figure 1.4. In order to illustrate performance in different wind conditions, we consider two separate cases. Namely $(w_N, w_E, w_D) = (5, 5, 0)$ and $(w_N, w_E, w_D) = (0, -7, 0)$ - along with the turbulence model which acts in all directions, as was discussed in Section 2.5. It should be noted that these wind conditions are selected because they, in combination with the turbulence model, results in a fresh breeze [13] condition as was discussed in Section 1.3. This is a significant wind condition in that results in wind speeds in excess of 50% of our commanded airspeed, so should the aircraft be shown to behave well under these conditions, it will indicate a very robust GNC system.

7.2.1 Southwesterly Fresh Breeze

The south-westerly fresh breeze applied is of the form $(w_N, w_E, w_D) = (5, 5, 0)$ m/s. This is a fairly significant wind condition, and will make up a large fraction of the desired airspeed. As such, we expect significant performance degradation - but we do expect to be able to withstand the condition and still track the desired path for both sets of waypoints. The applied wind condition is depicted in Figure 7.8, where components in each inertial direction are shown, as well as the total wind magnitude. The total magnitude is clearly of the magnitude of a fresh breeze, which is defined as having a magnitude of between 8.0-10.8 m/s, and in some places even exceeds this. It should also be clear from the figure that the wind is made up of both a steady state condition (ie. $(w_N, w_E, w_D) = (5, 5, 0)$ m/s), as well as a turbulence model, which adds filtered white noise on top of these values as was discussed in Section 2.5. The same random number generator seed is used for all simulations so that we can be sure that the exact same wind conditions, as a function of time, are being made use of in each case. The results of this are shown for both waypoint configurations depicted in Figure 1.3 and Figure 1.4.

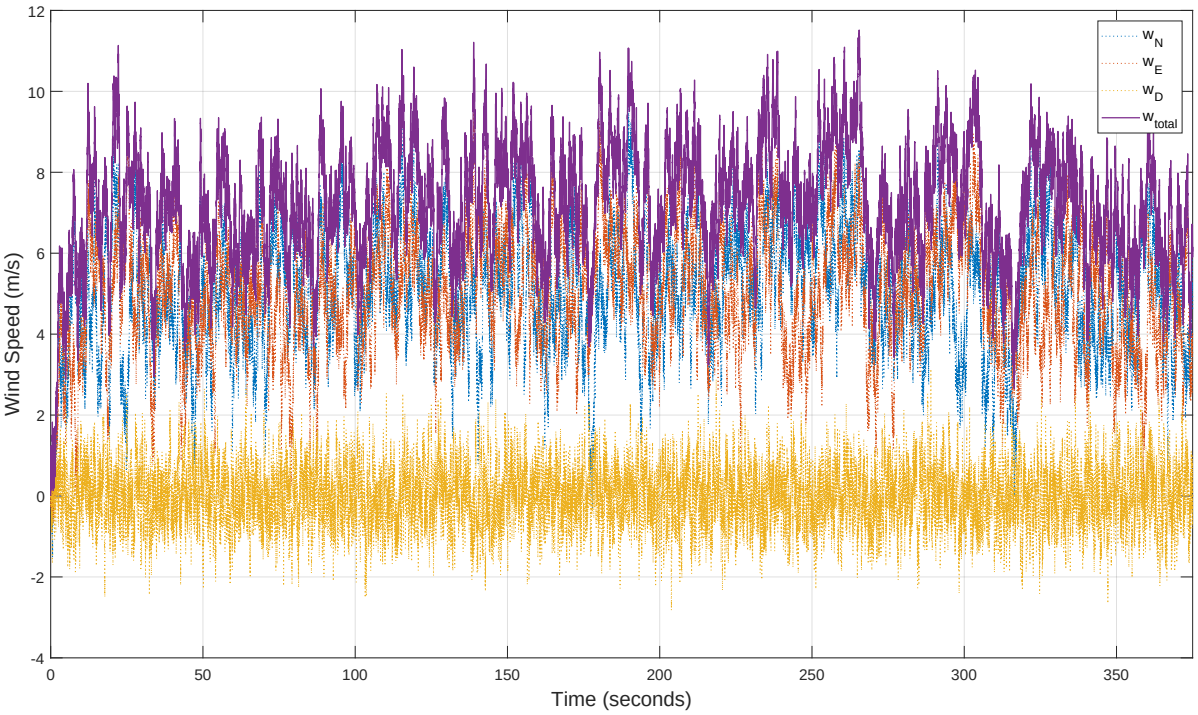


Figure 7.8: GNC system verification, southwesterly fresh breeze wind magnitude

Standard Waypoint Configuration

The results of the south-westerly wind condition depicted in Figure 7.8 for the standard waypoint configuration shown in Figure 1.3 are provided in Figures 7.9, 7.10 and 7.11. These results illustrate the expected performance degradation, particularly in the altitude control space. This is to be expected as controlling against fast-transient updrafts

is virtually impossible other than to reject larger steady state offsets. We note that this results in particularly poor performance along line segment $\overline{\mathbf{W_8W_9}}$, where it is noted that this occurred at approximately 290 s into the simulation; which lines up with the large transient seen in Figure 7.8 at this time point. The system does, however, recover from this and accurately continues to track the waypoint path as expected.

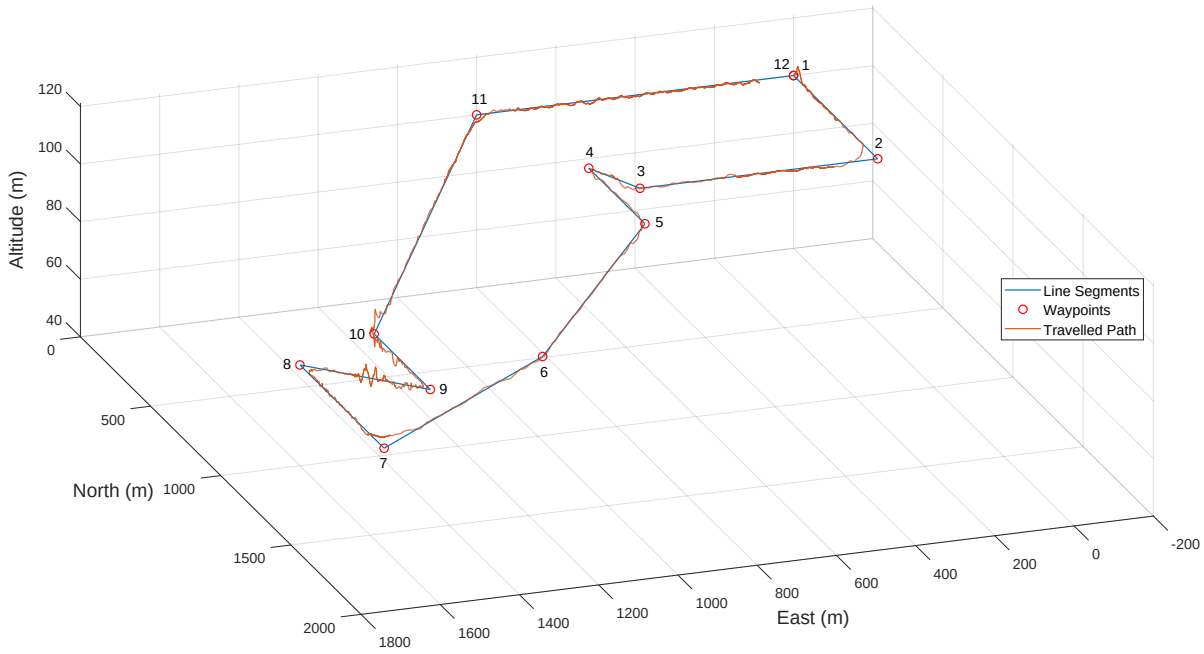


Figure 7.9: GNC system verification, southwesterly fresh breeze, orthographic view

It can also be seen from the results depicted that we do transition to line segment $\overline{\mathbf{W_1W_2}}$ at the end of the verification run. This is expected because at the end of the test we are flying into the wind while travelling along $\overline{\mathbf{W_{11}W_{12}}}$ and so expect to traverse this more slowly than in previous tests. Indeed, what this illustrates is that we spend more time flying in both a southerly, as well as a westerly direction than in a northerly, or and easterly direction because we are slowed down by wind being applied in these directions indicating that we must be flying into a headwind more so than flying in a tailwind.

Given that our commanded airspeed is 17.3 m/s, and the wind magnitude in places reaches values of 66% of this, our performance is quite exemplary under the circumstances and the GNC system is shown to be very robust even in severe conditions.

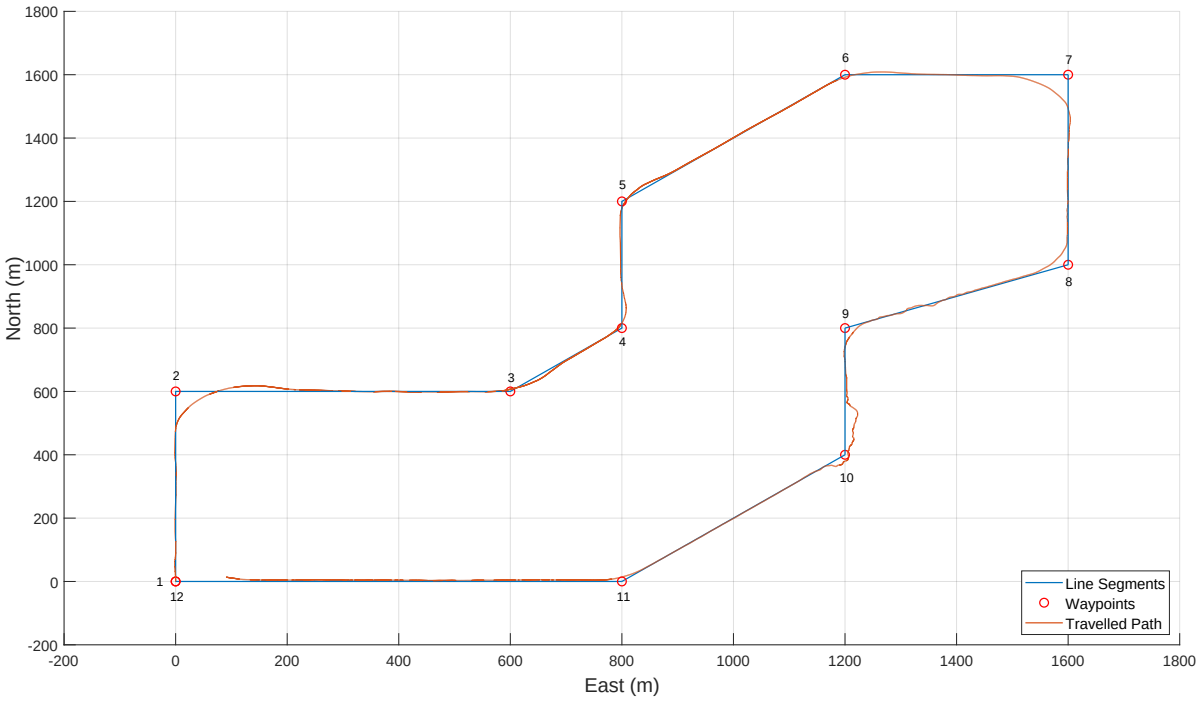


Figure 7.10: GNC system verification, southwesterly fresh breeze, $\mathbf{i}^{\mathbf{i}}$-$\mathbf{j}^{\mathbf{i}}$ plane view

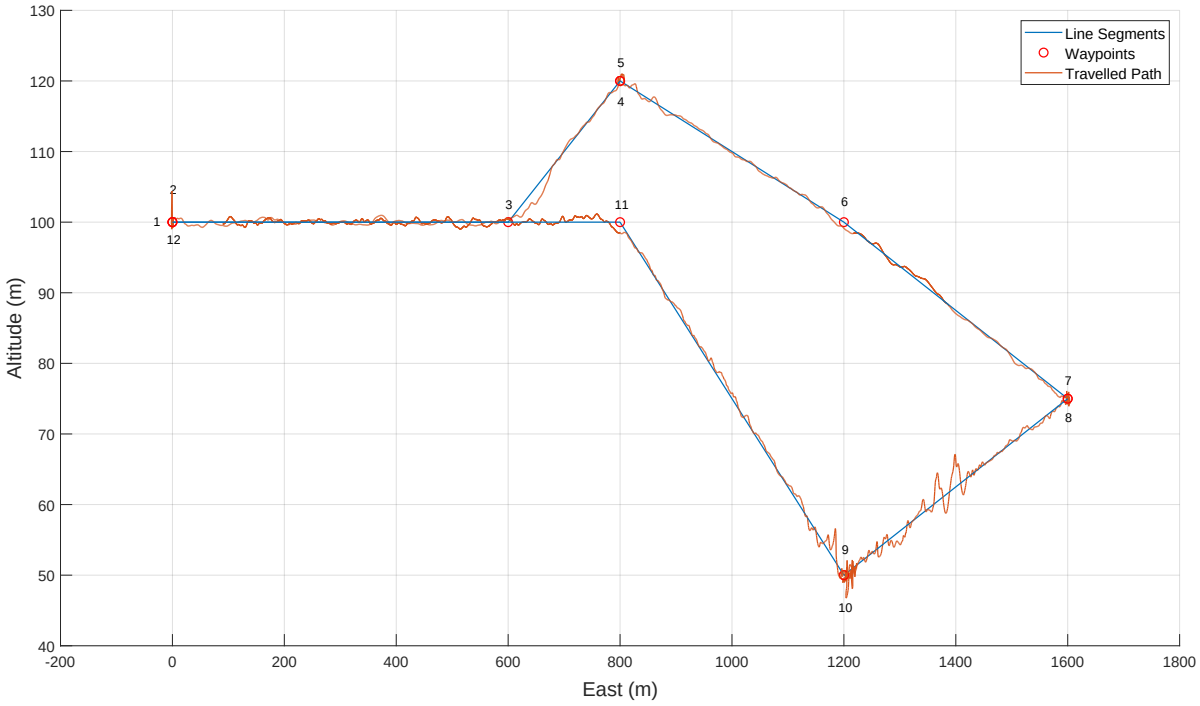


Figure 7.11: GNC system verification, southwesterly fresh breeze, $\mathbf{j}^{\mathbf{i}}$-$\mathbf{k}^{\mathbf{i}}$ plane view

Alternate Waypoint Configuration

The results of the south-westerly wind condition depicted in Figure 7.8 for the alternate waypoint configuration shown in Figure 1.4 are provided in Figures 7.12, 7.13 and 7.14. From these results we see a similar performance degradation as in the standard waypoint configuration, but we also note that we are still able to accurately track our waypoints. It should be noted in particular that our previously discussed difficult transition between $\overline{\mathbf{W_7W_8}}$ and $\overline{\mathbf{W_8W_9}}$ is still managed well even under these fairly extreme wind conditions. This is a strong indicator of the robustness of the system. We also note that in the time duration specified for the test (375 s), we do not get back to our starting waypoint. This, again, is expected because we spend a lot more time with a headwind than a tailwind in this simulation. These results indicate that the GNC system is robust and behaves as expected even in the presence of relatively severe wind and turbulence.

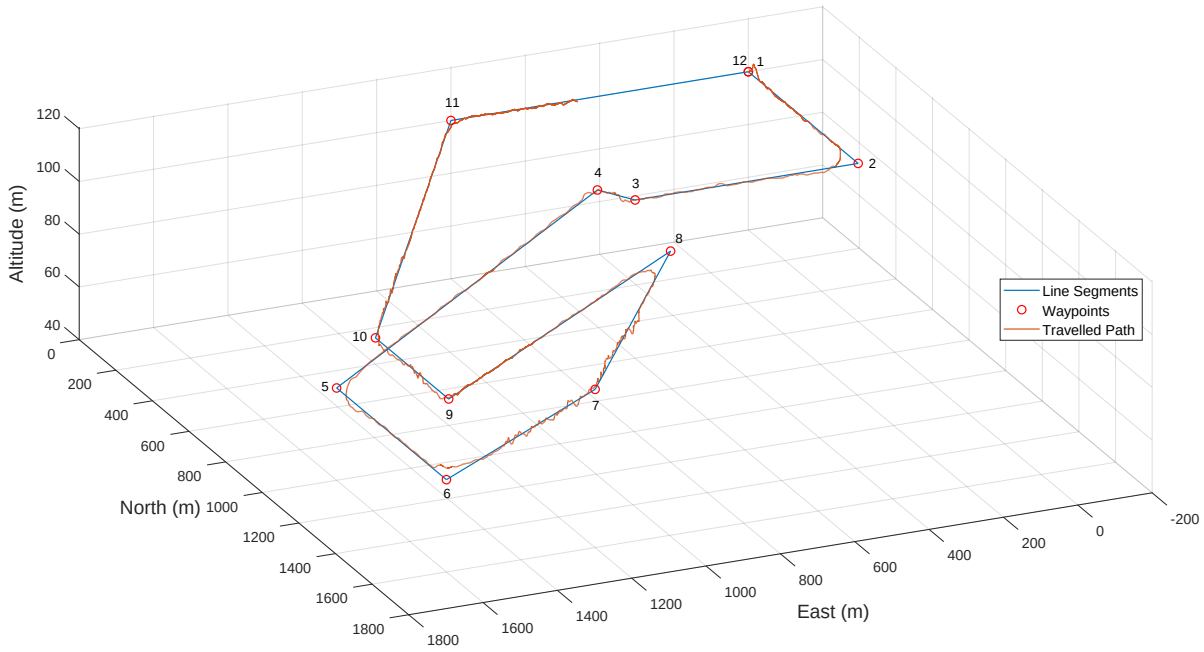


Figure 7.12: GNC system verification, alternate waypoints, southwesterly fresh breeze, orthographic view

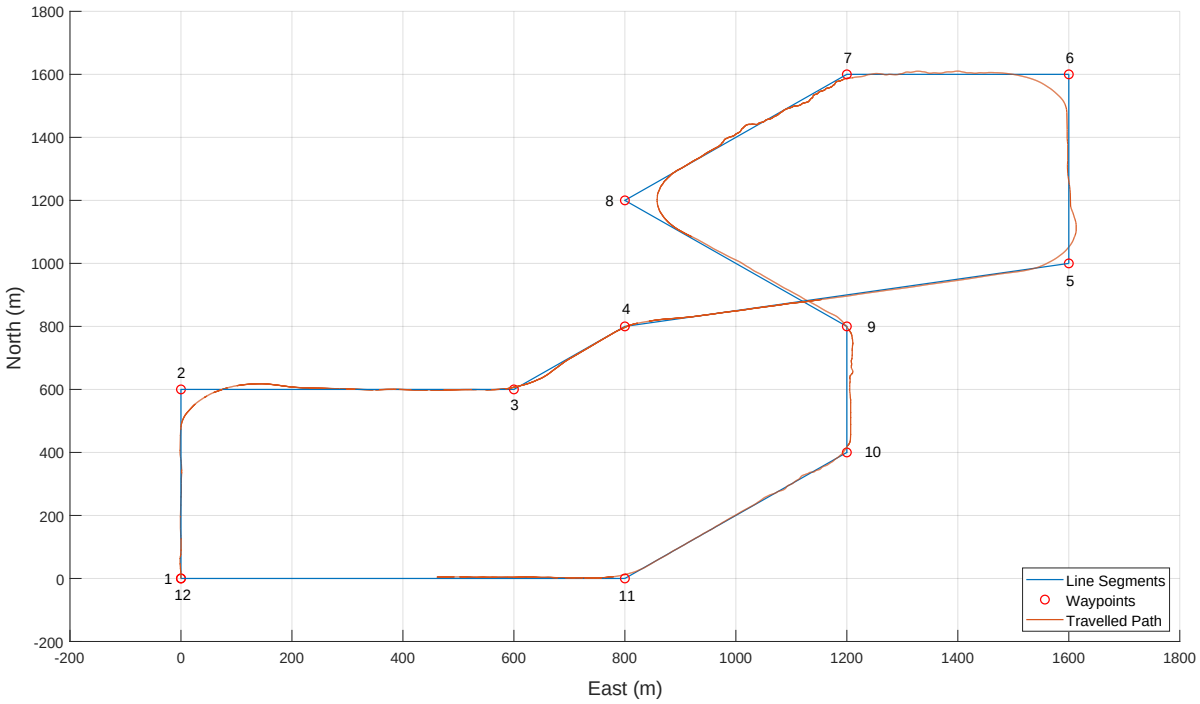


Figure 7.13: GNC system verification, alternate waypoints, southwesterly fresh breeze, $\mathbf{i}^{\mathbf{i}}$-$\mathbf{j}^{\mathbf{i}}$ plane view

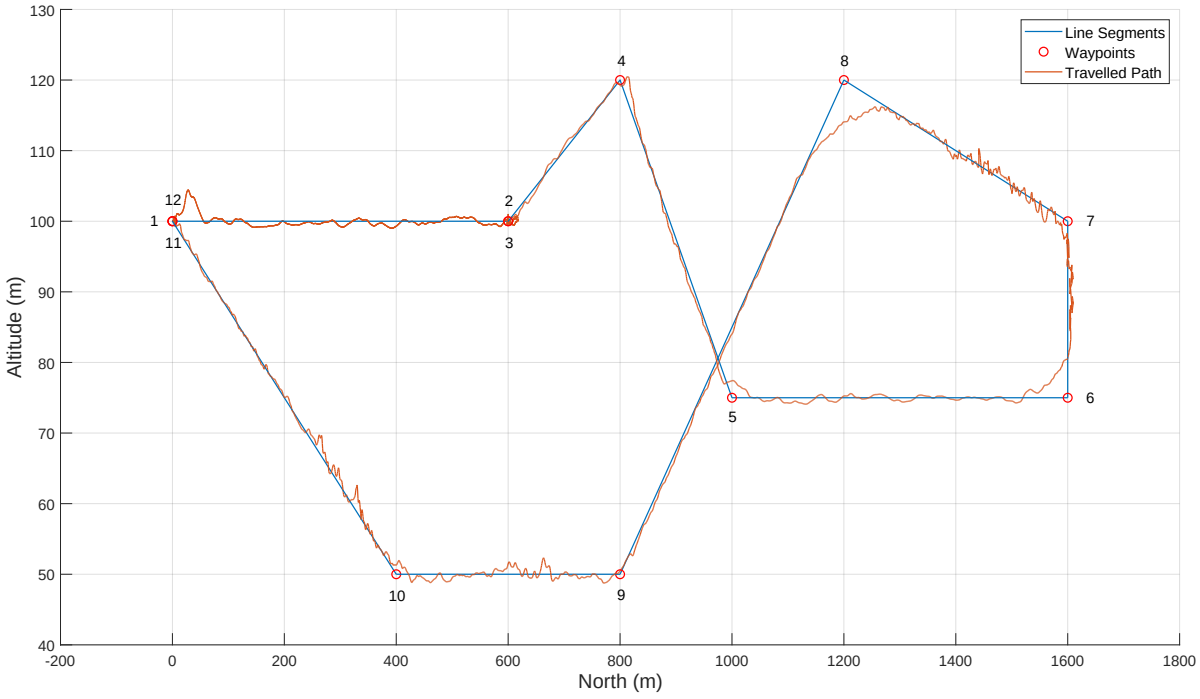


Figure 7.14: GNC system verification, alternate waypoints, southwesterly fresh breeze, $\mathbf{i}^{\mathbf{i}}$-$\mathbf{k}^{\mathbf{i}}$ plane view

7.2.2 Easterly Fresh Breeze

The easterly fresh breeze applied is of the form $(w_N, w_E, w_D) = (0, -7, 0)$ m/s. This is a fairly significant wind condition, and will make up a large fraction of the desired airspeed. As such, we expect significant performance degradation - but we do expect to be able to withstand the condition and still track the desired path for both sets of waypoints. The applied wind condition is depicted in Figure 7.15, where components in each inertial direction are shown, as well as the total wind magnitude. The total magnitude is clearly of the magnitude of a fresh breeze, which is defined as having a magnitude of between 8.0-10.8 m/s, and in some places even exceeds this. It should also be clear from the figure that the wind is made up of both a steady state condition (ie. $(w_N, w_E, w_D) = (0, -7, 0)$ m/s), as well as a turbulence model, which adds filtered white noise on top of these values as was discussed in Section 2.5. The same random number generator seed is used for all simulations so that we can be sure that the exact same wind conditions, as a function of time, are being made use of in each case. The results of this are shown for both waypoint configurations depicted in Figure 1.3 and Figure 1.4

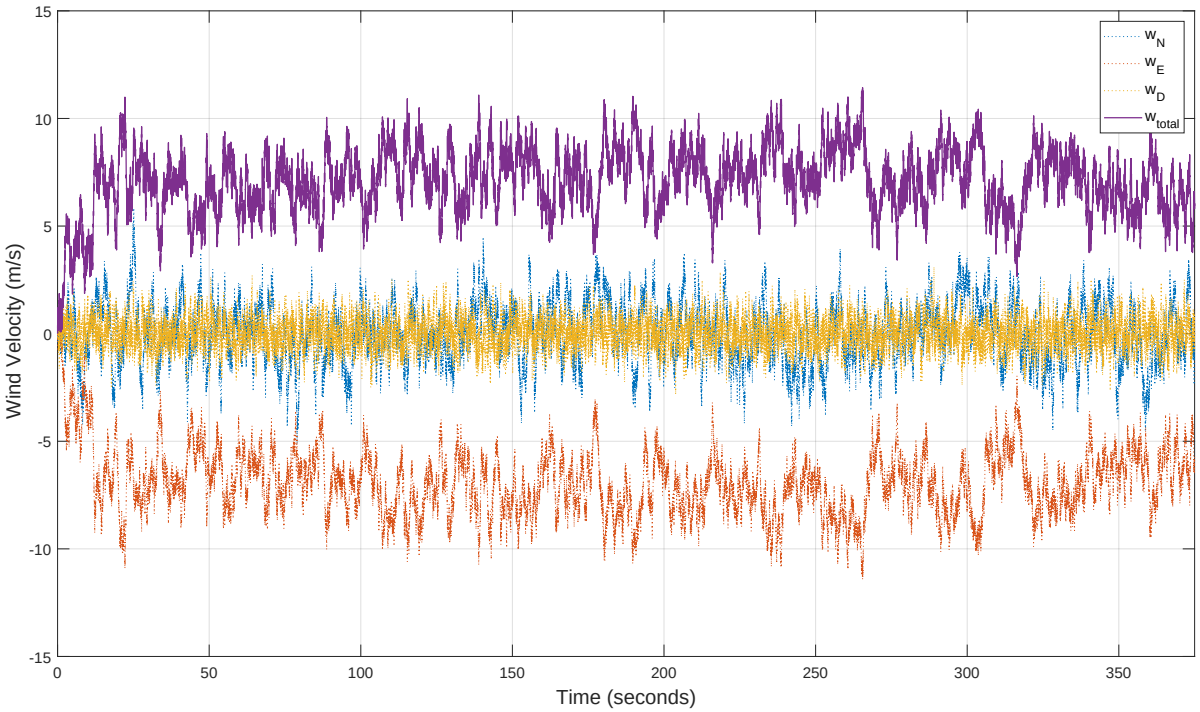


Figure 7.15: GNC system verification, easterly fresh breeze wind magnitude

Standard Waypoint Configuration

The results of the easterly wind condition depicted in Figure 7.15 for the standard waypoint configuration shown in Figure 1.3 are provided in Figures 7.16, 7.17 and 7.18. These results illustrate the expected performance degradation, particularly in the altitude control space as in the southwesterly wind case. It can also be seen from this

simulation that we now are able to start again along our line segment $\overline{\mathbf{W_1W_2}}$, this is expected because we now have a tailwind in the west direction (an easterly wind means that the wind direction is to the west) and no headwind in the south direction and so expect to traverse the waypoint path faster.

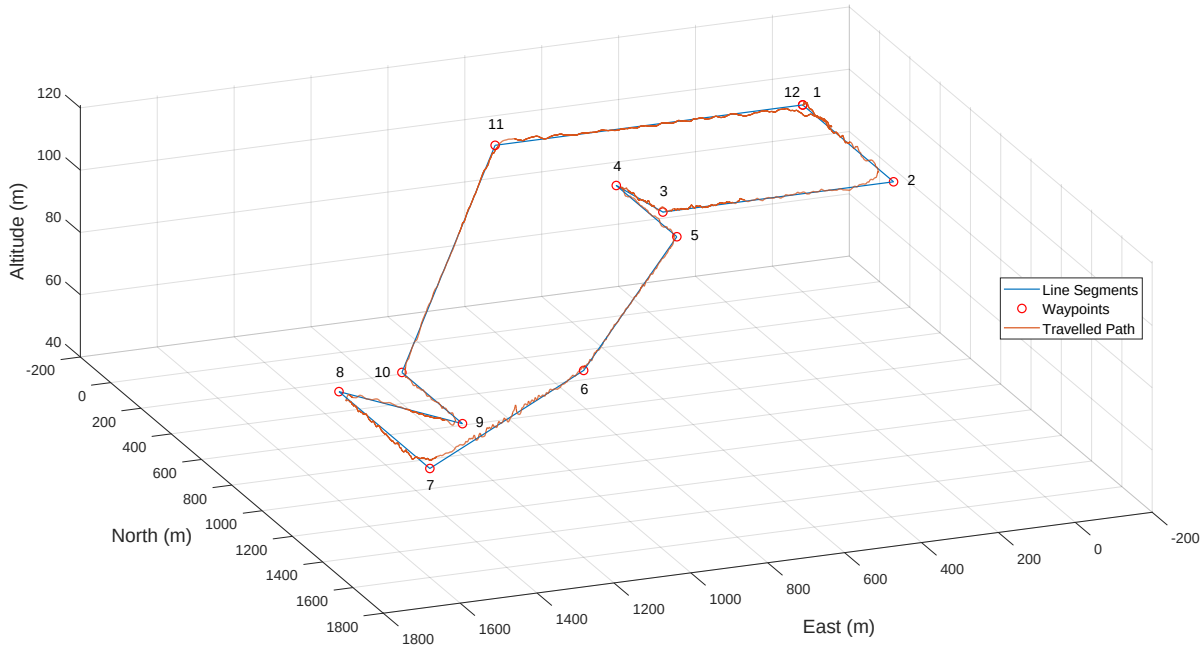


Figure 7.16: GNC system verification, easterly fresh breeze, orthographic view

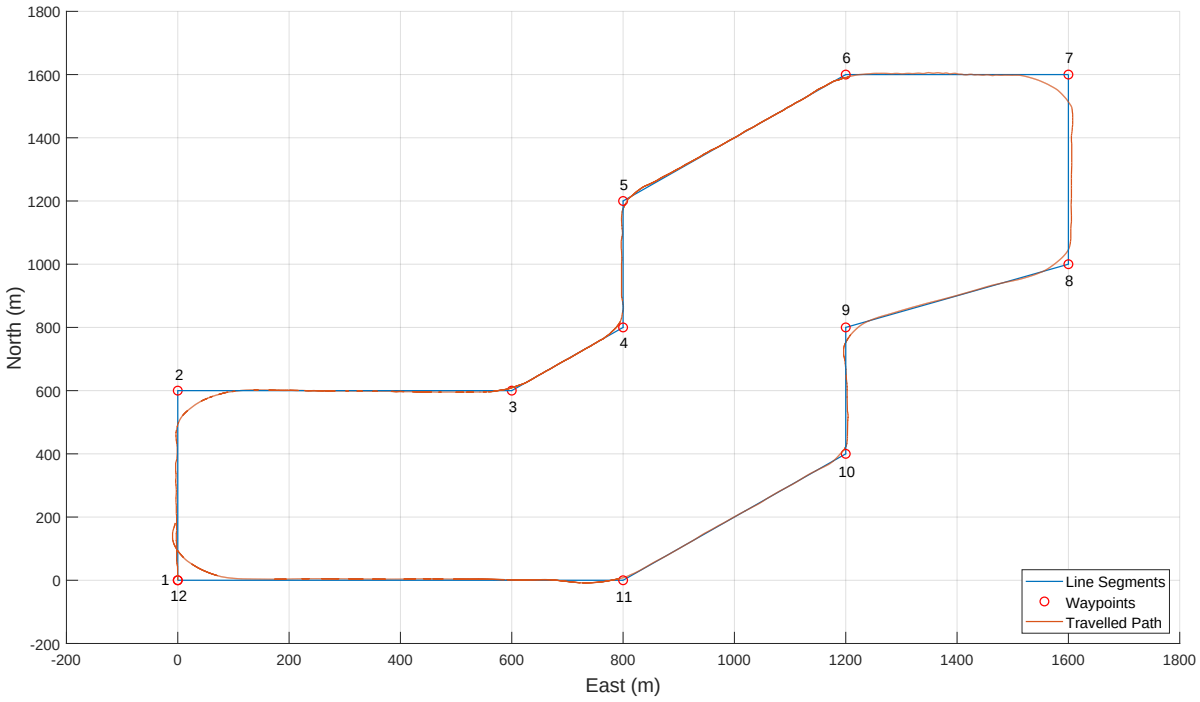


Figure 7.17: GNC system verification, easterly fresh breeze, $\mathbf{i}^{\mathbf{i}}$-$\mathbf{j}^{\mathbf{i}}$ plane view

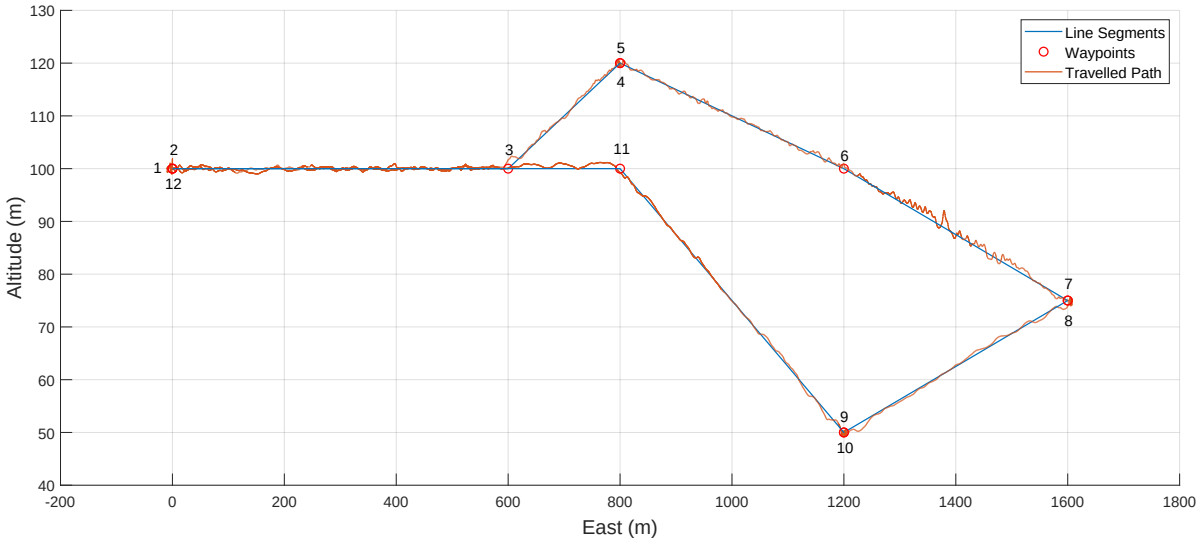


Figure 7.18: GNC system verification, easterly fresh breeze, $\mathbf{j}^{\mathbf{i}}$-$\mathbf{k}^{\mathbf{i}}$ plane view

Alternate Waypoint Configuration

The results of the easterly wind condition depicted in Figure 7.15 for the alternate waypoint configuration shown in Figure 1.4 are provided in Figures 7.19, 7.20 and 7.21. From these results we see a similar performance degradation as in the standard waypoint configuration, but we also note that we are still able to accurately track our waypoints. It should be noted in particular that our previously discussed difficult transition be-

tween $\overline{\mathbf{W_7W_8}}$ and $\overline{\mathbf{W_8W_9}}$ is still managed well even under these fairly extreme wind conditions. This, like the verification runs before it, also illustrates robust performance in severe conditions.

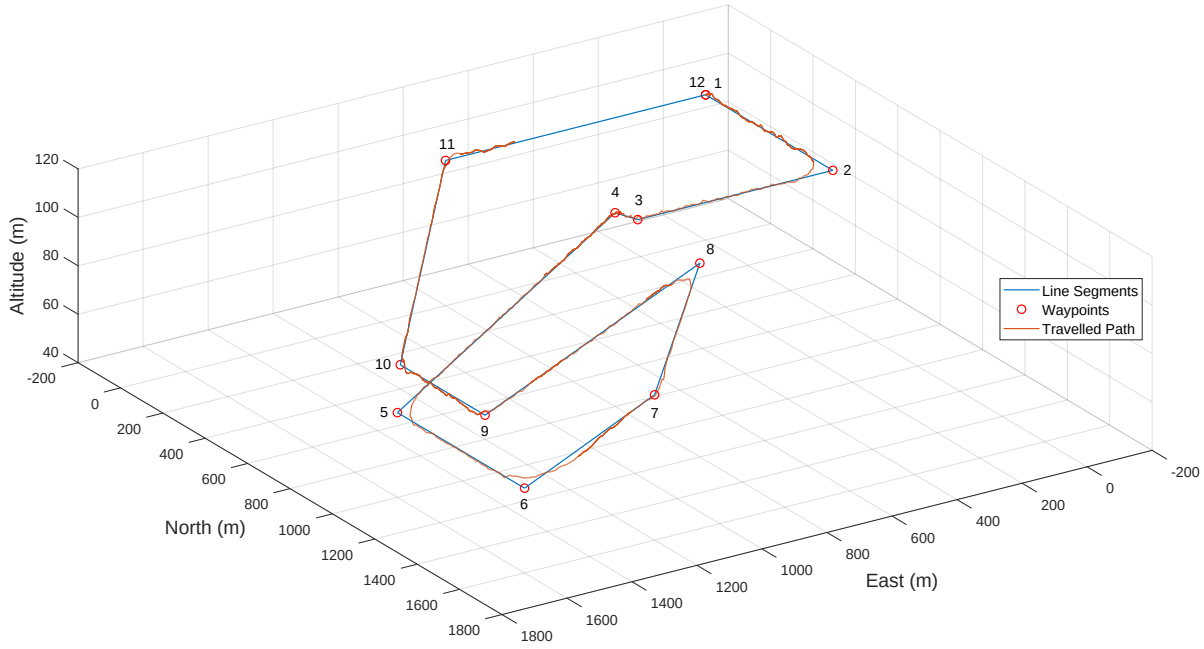


Figure 7.19: GNC system verification, alternate waypoints, easterly fresh breeze, orthographic view

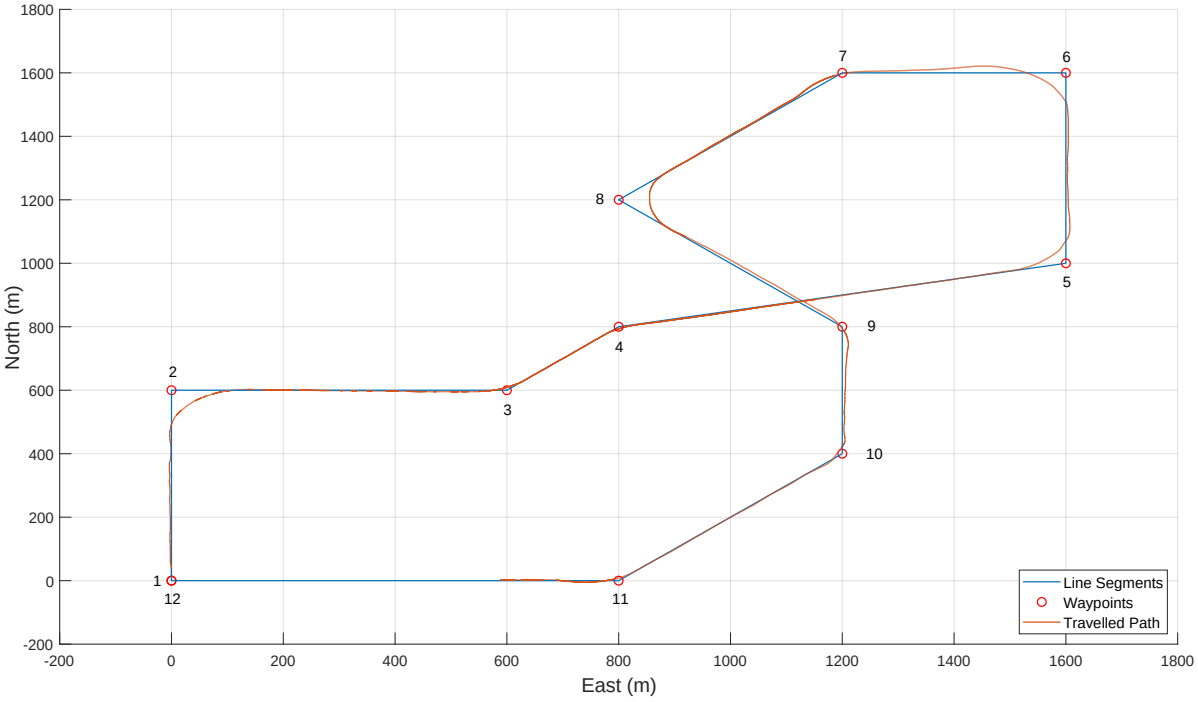


Figure 7.20: GNC system verification, alternate waypoints, easterly fresh breeze, $\mathbf{i}^{\mathbf{i}}$-$\mathbf{j}^{\mathbf{i}}$ plane view

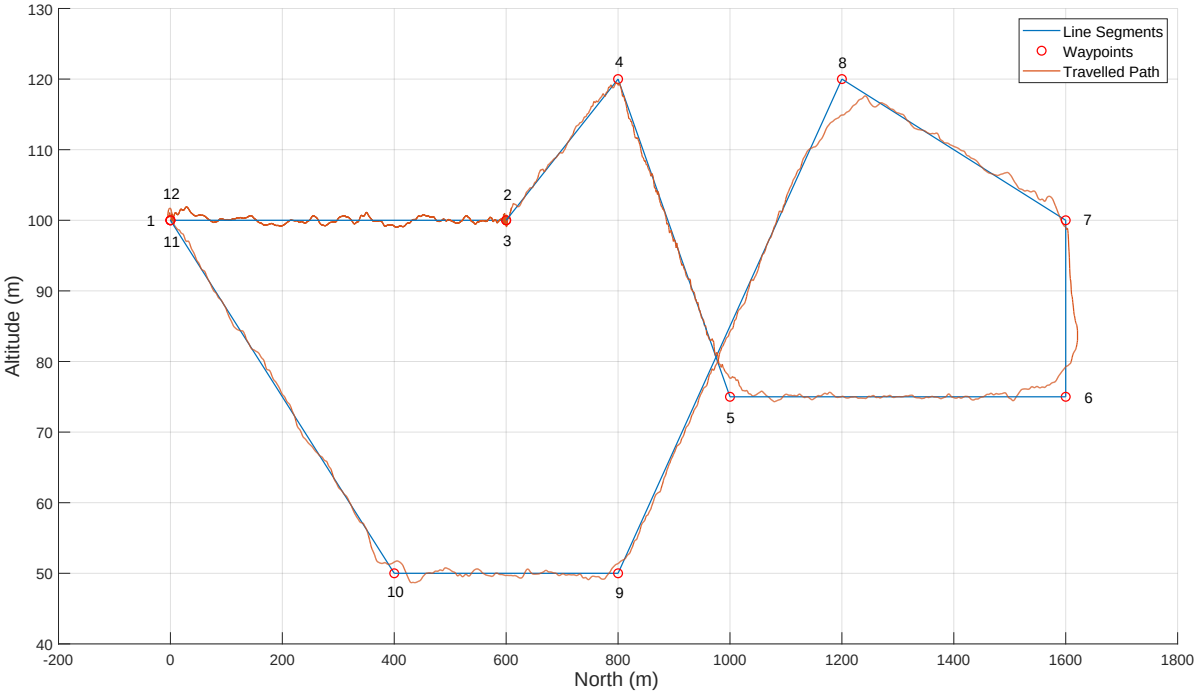


Figure 7.21: GNC system verification, alternate waypoints, easterly fresh breeze, $\mathbf{i}^{\mathbf{i}}$-$\mathbf{k}^{\mathbf{i}}$ plane view

7.3 Waypoint Path Switching

One of the system requirements stated in section 1.3 was to allow for switching of waypoints during flight, this section aims to illustrate this functionality. This is done for two conditions to illustrate that it works as expected in various cases, while flying with the initial set of waypoints as defined in Figure 1.3, and the set of waypoints which are switched to as in Figure 1.4. Wind effects are included, but only the south-westerly $(w_N, w_E, w_D) = (5, 5, 0)$ m/s model is used for the sake of brevity.

7.3.1 Switching between $\mathbf{W_4}$ and $\mathbf{W_5}$

The first switch over is done while flying between $\mathbf{W_4}$ and $\mathbf{W_5}$, although closer to $\mathbf{W_4}$. We expect that when we do this, we will switch to $\mathbf{W_5}$ of the alternate waypoints as our current target, which means that we expect to perform a bank to the east so as to align with $\overline{\mathbf{W_4W_5}}$ of the alternate waypoints as depicted in Figure 1.4. We should then follow the path defined by the alternate waypoints for the remainder of the simulation.

The results of this waypoint switch are depicted in Figures 7.22 and 7.23. These results clearly illustrate our expected functionality whereby we transition smoothly between the previous $\overline{\mathbf{W_4W_5}}$ line segment, which we were tracking prior to the switch, to the alternate waypoint line segment $\overline{\mathbf{W_4W_5}}$. After this we continue along the alternate waypoint path as expected with the same performance as was seen previously. This illustrates that our waypoint switching algorithm as illustrated in Algorithm 4 behaves correctly when receiving a new waypoint path mid-flight.

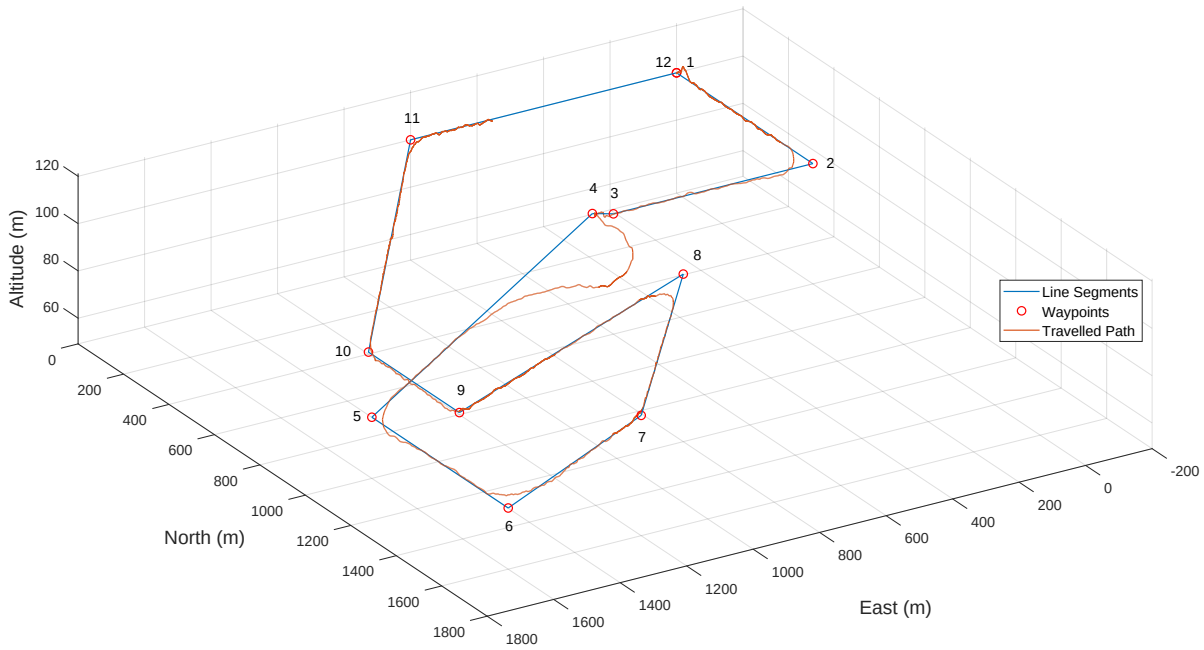


Figure 7.22: GNC system verification, waypoint switching between $\mathbf{W_4}$ and $\mathbf{W_5}$, orthographic view

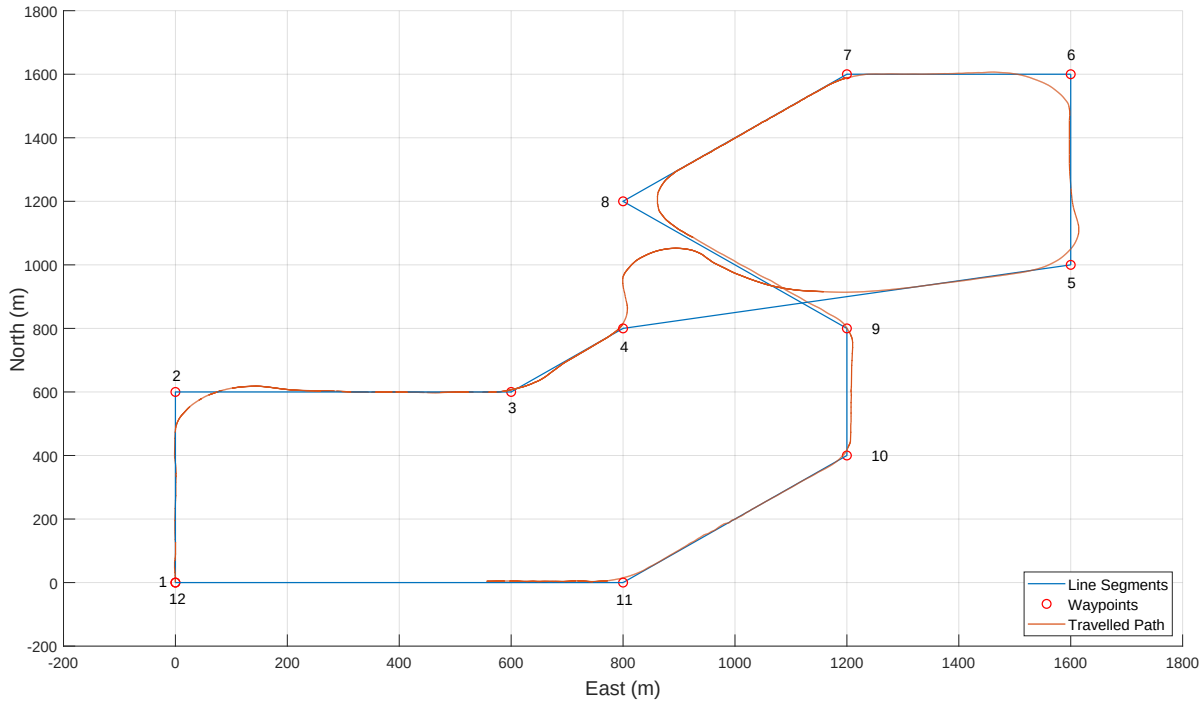


Figure 7.23: GNC system verification, waypoint switching between $\mathbf{W_4}$ and $\mathbf{W_5}$, $\mathbf{i^i}$-$\mathbf{j^i}$ plane view

7.3.2 Switching between $\mathbf{W_6}$ and $\mathbf{W_7}$

The second switch over is done while flying between $\mathbf{W_6}$ and $\mathbf{W_7}$, although closer to $\mathbf{W_6}$. We expect that when we do this, we will switch to $\mathbf{W_7}$ of the alternate waypoints as our current target, which so happens to be the same as $\mathbf{W_6}$ of the previous path. This means that we expect to perform a bank to turn the aircraft around completely so as to align with $\overline{\mathbf{W_6}\mathbf{W_7}}$ of the alternate waypoints as depicted in Figure 1.4. We should then follow the waypoints defined by this path for the remainder of the simulation.

The results of this waypoint switch are depicted in Figures 7.24 and 7.25. These results clearly illustrate our expected functionality whereby we transition smoothly between the previous $\overline{\mathbf{W_6}\mathbf{W_7}}$ line segment we were tracking, to the alternate waypoint line segment $\overline{\mathbf{W_7}\mathbf{W_8}}$. After this, we continue along the alternate waypoint path as expected with the same performance as was seen previously. This illustrates that our waypoint switching algorithm as illustrated in Algorithm 4 behaves correctly when receiving a new waypoint path mid-flight, even if that new waypoint set requires the aircraft to double back on itself.

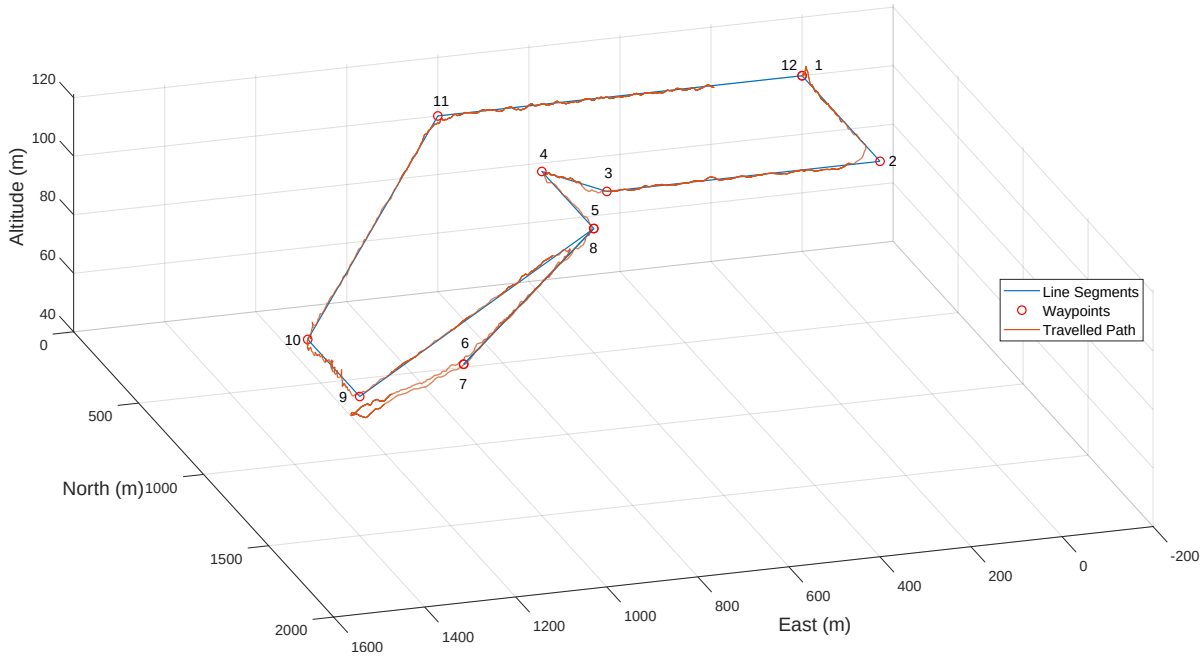


Figure 7.24: GNC system verification, waypoint switching between $\mathbf{W_6}$ and $\mathbf{W_7}$, orthographic view

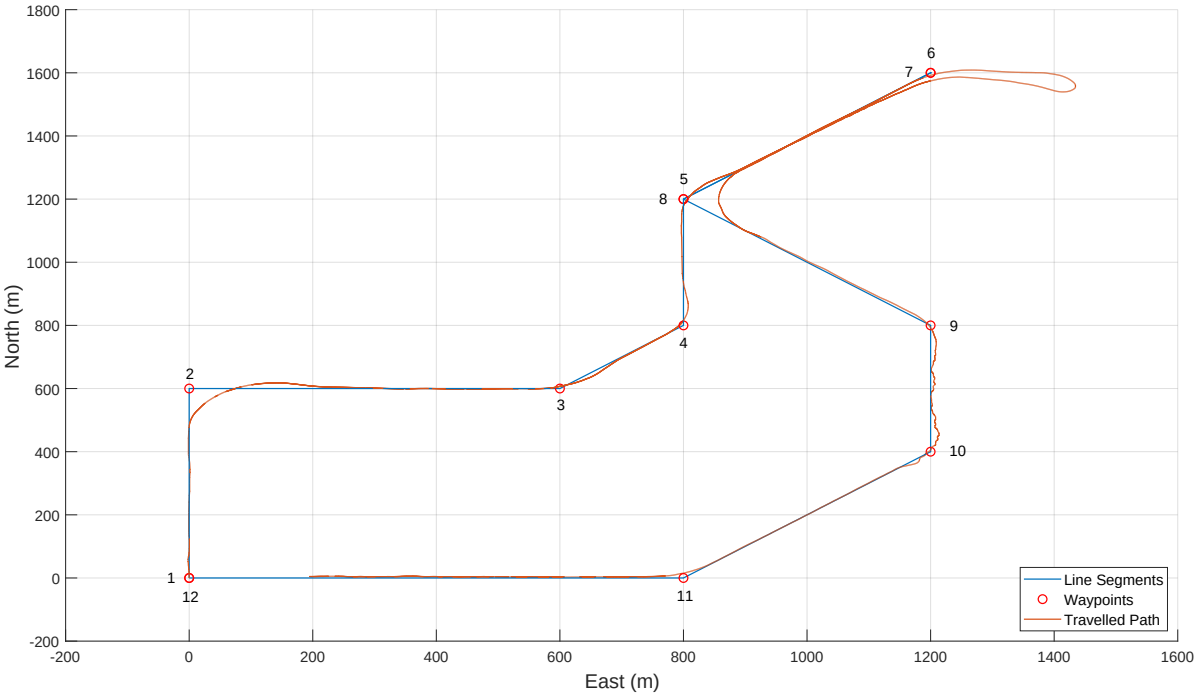


Figure 7.25: GNC system verification, waypoint switching between $\mathbf{W_6}$ and $\mathbf{W_7}$, $\mathbf{i^i}$-$\mathbf{j^i}$ plane view

Future Work and Project Extensions

Having now completely designed, optimised and verified the full GNC system for the Skywalker X8 aircraft, as was originally illustrated in Figure 1.2, we can consider how this work might be improved upon and identify several interesting extensions which could use this work as a base. This is made quite simple as the reader has been provided with the full set of design tools, which are all fully commented and reusable, so any future work need not repeat anything discussed in this text unless required.

In order to facilitate this discussion, we will firstly consider all of the identified limitations and improvements which can be made to the work presented in this text - of which all have been discussed in the preceding chapters, but which are captured here to collate these discussions. This will be done on a design-element-by-design-element basis, using the system architecture presented in Figure 1.2 as a basis for how the elements of the design are split.

After this, we will consider several examples of extension points which are not discussed in this text but which this work could serve as the basis for. Some of these ideas are alluded to in the introduction to this dissertation as work which has already been done, but additional research areas will be identified and some first steps proposed.

8.1 Limitations and Improvements

The design presented in this text is by no means perfect and several additional steps will be recommended as to how this can be taken from simulations on a host computer to implementation on a physical flight computer. However, great lengths have been gone to to ensure that the process followed is logically sound and that any gaps which do exist have been explained and are unlikely to result in any undesirable behaviour should this system be implemented in its current state. With that said, there are various aspects which one could improve on in each of the chapters which were presented previously - and all of these items have already been discussed in their respective chapters. The following should simply serve as a summary of all of these discussions, perhaps with slightly more detail provided in some cases.

8.1.1 Plant Modelling

The plant model presented is a full, non-linear 6DoF model which should prove to be quite high fidelity in comparison to the physical system as it includes many non-linear effects such as a fairly advanced stall model. It also includes the ability to be impacted by atmospheric effects, which allows one to determine the system robustness to disturbances in general. However, some elements can certainly be improved upon.

Euler angles, while most appropriate in an academic setting - such as this dissertation - due to them lending themselves well to intuitive discussions, can be moved away from and quaternions made use of instead. This will allow for much more computationally efficient designs of the autopilot and guidance laws, which will aid in implementation as it will reduce processor load on the flight computer, and also minimise the singularity handling which has to be done (ie. divisions by zero are eliminated when making use of quaternions). A good introduction to quaternions is provided in [69], while applications of this to flight systems can be found in [33].

While all of the aerodynamic coefficients used in this text were determined from both wind tunnel testing and CFD simulation, thus leaving us with the highest fidelity model likely available in the literature, one can always improve on these coefficients to vary as a function of other system parameters. For example, the propeller on the aircraft will cause a change in the boundary layer across the wings in the areas close to it - this will thus have an impact on their aerodynamic performance, which will vary as a function of the propeller speed. As such, there is a potential that one could parametrise the aerodynamic coefficients to include the impact of propeller speed. This is true in other areas too, such as the impact of turbulence generated from flow separation when the aircraft has a sideslip angle that is too great in magnitude. As such, the aerodynamic model can always be improved - it just depends on the particular application as to how far one is required to go until their model is good enough.

Wind modelling for this project was included, but is by no means exhaustive. Other wind modelling effects can be examined, such as wind shear [70], wind gusts [71], and perhaps even alternative wind turbulence models other than the Dryden model made use of [34]. One could also consider the impact of modelling wind as a vector field with varying magnitude and direction as a function of spatial location, and how this might impact the aircraft. This might become particular relevant if the aircraft is flying through a static obstacle field - which is one of the extension points while will be mentioned in Section 8.2 - of tightly spaced objects which have a localised impact on the atmospheric conditions surrounding them.

8.1.2 Sensors and Actuators

The sensor and actuator models presented in this text are far beyond what one might find in a standard autopilot design text. However, they are still by no means exhaustive,

with a much more comprehensive model being presented in Figure 3.1. If one requires more accurate sensor models, the recommendation would be to review [44] which provides a good deal of insight on how to model sensors quite extensively.

At some point, however, one will require physical hardware in order to get the most accurate models because certain information is simply not made available by the manufacturers and needs to be determined experimentally. Should this be an option, the recommendation would then be to follow the approach of computing the Allan variance in order to characterise the sensor [72]. This is a method of representing the root mean square (RMS) random-drift error as a function of averaging time. The Allan variance method can be used to determine the characteristics of the underlying random processes that give rise to the data noise, but will require having a physical sensor available to characterise.

This thinking also extends to the actuators, where both small and large signal responses can be experimentally determined and more accurate models made use of from the resultant system identification [73].

8.1.3 Autopilot Design

The autopilot design presented is fairly extensive from a classical design perspective and very little could likely be done in that domain to improve the performance achieved. However, this is not the case for all of the controllers, with the airspeed control via pitch angle controller being a prime example. Should another application require a faster responding "glider-mode" controller, this system could likely be revisited as not much time was spent on ensuring optimal response. The remainder of the controllers, however, were constantly compared to what is the physical limit of the airframe and constantly found to be close to these values and can thus categorically not be improved on.

However, the altitude controller proved more challenging to make optimal because of trying to avoid stall being induced by the pitch angle controller. As such, it might warrant further investigation into whether a more exotic or advanced controller topology - which, for example, might explicitly rate limit the pitch angle command as part of its design, as opposed to the current implicit limit based on controller bandwidth - could be found to optimise this further. It should be noted that this did not prove problematic in any of the waypoint paths considered, so would only become necessary if one required absolutely optimal altitude control or very rapid, fine tolerance control over the pitch angle of the aircraft.

8.1.4 State Estimation

The state estimation provided in this text is the industry-standard approach to dealing with determining unmeasurable states, as well as to increasing the sampling time and smoothing of GPS data. As such, the methodology itself likely does not require any

refinement. What could be refined, however, would be to develop an algorithm which iteratively solves for each of the elements of the process noise covariance matrix, or Q. This can be done in many ways, but the recommendation would be to consider making use of a gradient descent optimisation algorithm which would attempt to solve for each element in Q simultaneously. This is done by perturbing them and reviewing the impact on the resultant error between the actual and the estimated state, and using the perturbation directions that result in a decrease in the error to iteratively optimise the respective element. This would eventually result in a Q which causes the least error overall between actual and estimated states.

One could, of course, provide specific states with additional weighting if one, for example, required a much more accurate estimate of pitch angle than they did of roll angle. This should ideally be done with quite extensive data which covers the full range of expected measurements and with as much of the expected system dynamics and sensor noise/non-linearity in the data as possible so as to get the most broadly applicable result.

8.1.5 Guidance and Navigation

Guidance and navigation laws are, almost by definition, application specific. The methodology selected in this dissertation is a fairly generic approach which should be broadly applicable to many use-cases. However, there are cases where waypoints have a higher value than the line segments connecting them. For example, consider a medication transport aircraft which is required to drop a package at a specific location in $\mathbb{R}^3$ - this kind of system would clearly be required to transition through a commanded waypoint at which it needs to make the drop, and as such its guidance laws would be developed with moving through waypoints carrying a much higher weighting than smooth transitions between line segments. As such, the path manager could be redesigned with different design criteria in mind should it be so required; but any kind of path can be constructed with a combination of straight lines and circular orbits, so the path following algorithms should not need adjusting other than perhaps tuning their design parameters to be more or less aggressive.

Another addition one could make to the path manager presented is the ability to asynchronously flag certain waypoints as loiter points. This would mean that the aircraft would transition to this waypoint and then simply travel in a circular orbit of constant altitude - defined by the waypoint altitude - around the point until commanded otherwise. All of the constituent elements needed to accomplish this are already in place. One would simply need to add to the path manager so that if a waypoint is flagged for loitering, it should provide the center point of the orbit as the waypoint itself, as opposed to the center of the fillet circle. This would be done once the half plane $\mathcal{H}_1$, as presented in Figure 6.10, is reached. This was not implemented in this text as it did not add anything to the discussion, but is an extremely useful feature to have for any guidance and navigation system which is required to be made use of in practice.

8.1.6 System Integration and Verification

The system integration and verification done for this project far exceeds the verification that would normally be done for a system of this nature in an academic setting. In most cases, one would simply design the controllers and discretise them, and treat all simulation thereafter as being as close to reality as required. This, however, is not the case here and this work extends on that by including preliminary software in the loop simulation. This is done to ensure that compiled, generated C-code (which can be implemented on an embedded system) is able to achieve the same functionality as is achieved with the design models made use of.

However, the next step in this process would be to conduct processor in the loop testing - where code is run on the intended target hardware, but all sensor and plant modelling is still performed on the host and the simulation likely does not proceed in real time. The purpose of this is to ensure that any errors that might be introduced in transitioning from the host to the target architecture are identified and remedied before moving forward. For example, the target might not have access to a floating point unit (FPU), and would either have to make use of software emulation or would be constrained to fixed point operation. If this was the case we might have to adjust how the code is generated, or even how the controllers are designed, so as to facilitate this additional constraint.

Once processor in the loop testing is complete, one could conduct hardware in the loop testing. This is where our plant is modelled in a real time simulation environment, and we run our flight control software on a full embedded platform such as the Pixhawk. Our sensor and actuator data can either be communicated to the flight computer over a serial link or as physical electrical signals, depending on the real time simulation platform made use of. Indeed, hardware in the loop testing is already supported as a simulation mode for the PX4 firmware [74], so this would reduce development effort substantially.

In the associated code repository for this project, an item which was not discussed in the body of the text was also implemented. This makes use of the Gazebo environment on a Linux virtual machine to act as an isolated simulation environment in which one can perform any testing required. This is one of the environments supported by the PX4 firmware [74], and as such should reduce development time significantly if this area is explored.

Gazebo has its own physics engine, which was extended as part of this project to include aerodynamic forces and moments modelling for the Skywalker X8 aircraft (which is easily adjustable for any other airframe or configuration). This simulation platform can also be set up to communicate with MATLAB/Simulink on the root machine via the Robot Operating System (ROS) communication channel and this can both be used for testing the controllers using the external physics engine and aerodynamics model - which would test for controller robustness - or just as a simple 3D animation tool with all the

states being written directly from Simulink. An example of this communications link is provided in the associated Simulink project, and future work can certainly leverage this functionality to serve as an excellent starting point for any extensions to the research presented in this dissertation.

The code repository which contains the Gazebo plugins developed, as well as all resources needed to have a flying Skywalker X8 aircraft modelled in the environment can be found on Github [75], and the aircraft in the modelling environment is depicted in Figure 8.1.

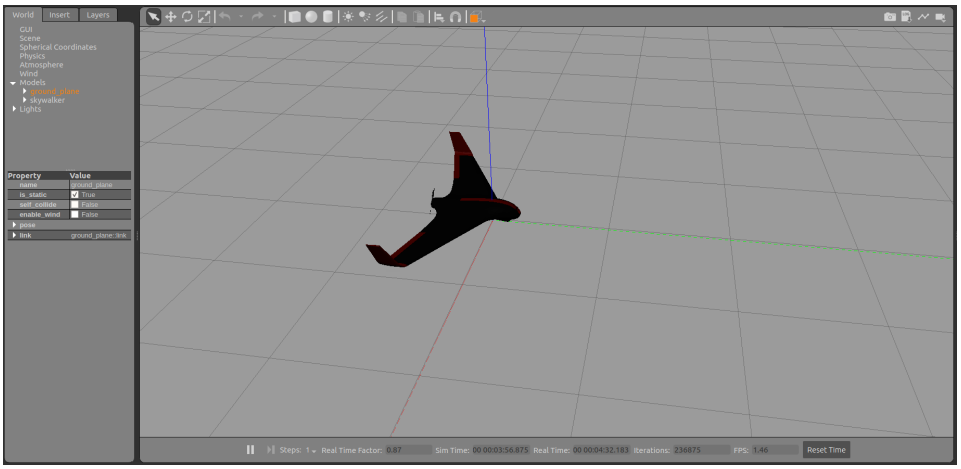


Figure 8.1: Skywalker X8 as modelled in the Gazebo simulation environment

8.2 Extension Points

Having discussed all of the limitations that exist, as well as the improvements one could make on the work presented in this dissertation specifically, the next topic to consider is how this research might be extended. In other words, if we treat this work as a base, how might one build onto it in order to establish surrounding architectures which make use of the full GNC system presented herein, or make adjustments to the already developed system so as to accomplish some other task.

From the perspective of the plant, one could quite easily extend the current model by including additional functionality such as an under-slung load, or an alternate propeller configuration - one does get a vertical take-off and landing (VTOL) version of the Skywalker X8 airframe, for example. Given that this would change the system dynamics, the full autopilot would need to be adjusted. However, the tools and processes developed in this text are such that retuning of the controllers to generate an autopilot that is optimal for the new system dynamics would be made substantially easier as all of the tools and processes used are modular and well documented. This would enable one's focus being put entirely on the currently unmodelled dynamics, as opposed to first needing to

address the base system.

Having a well-defined and modelled system, as well as a full autopilot available, means that one can extend this work quite easily by simply adjusting how one interacts with the path manager. Computer vision research is a prime candidate for this, as instead of providing the path manager with a set of user-defined waypoints, these waypoints can be determined on-board, or at the ground station, by using image processing techniques. This would allow for the aircraft to, for example, move through either a static, or a dynamic, obstacle field. An example of this might be for the aircraft to move through some tightly packed buildings. One can extend this to adjusting the path manager to also control for aircraft airspeed, as we have shown in the discussion surrounding Figure 2.19 that airspeed is linked to the minimum possible turn radius, which impacts the spatial frequency of what our path planning algorithm could accomplish. This idea is depicted in Figure 8.2, which illustrates how a set of waypoints to avoid a set of static obstacles might be generated.

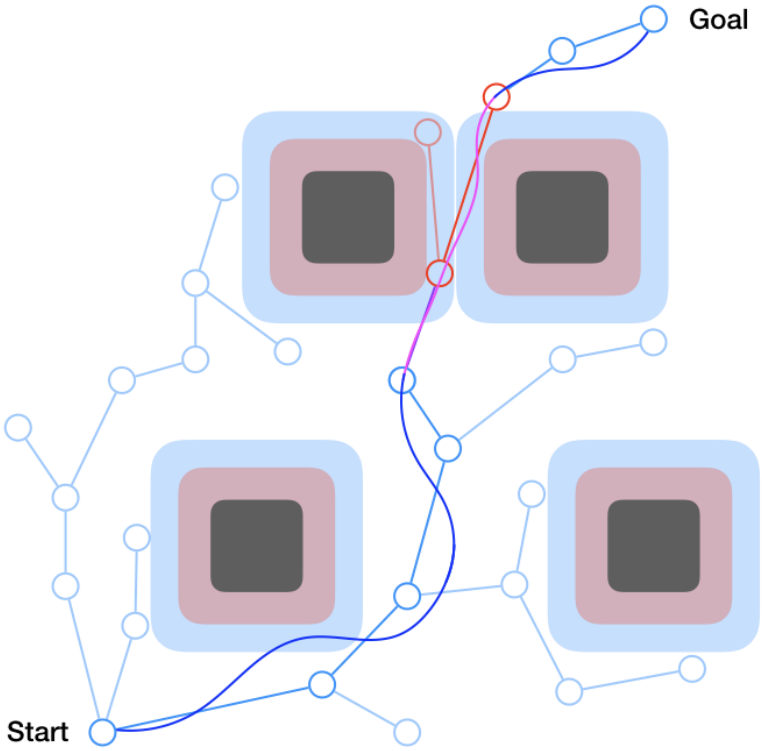


Figure 8.2: Static obstacle avoidance path planning [76]

One could extend the presented system in another direction by considering fields such as area mapping, where an external system could be developed which generates a set of

waypoints that most optimally covers an area. With this being developed and making use of an airframe with access to sensor technologies such as LiDAR, ultrasonic and camera, a three-dimensional map of this area can be generated. Fixed wing aircraft are ideal for these types of applications in outdoor environments and for larger areas, as they generally have much longer flight times than rotor-based aircraft. An example of the kinds of maps generated using these methods is depicted in Figure 8.3.

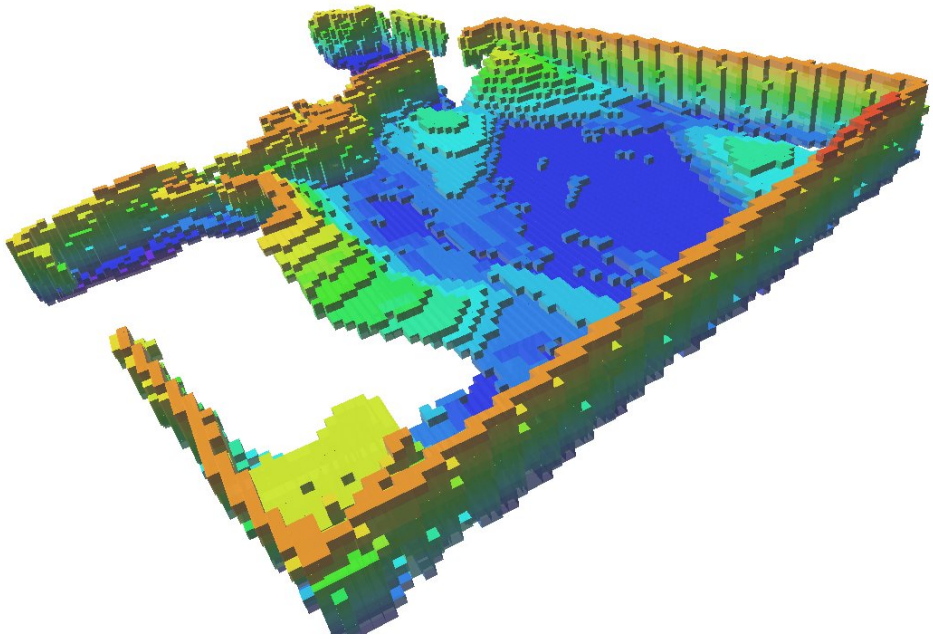


Figure 8.3: Area mapping [77]

Extensions such as these are fairly complicated in and of themselves and have all of their own challenges. As such, having a solid foundation that already has been shown to work well in simulation, under even the most severe of conditions, means that the focus can be put on the actual objectives of the research being done, and not first having to develop a working base system. Having access to all of the tools and processes which were used to generate the underlying autopilot and guidance laws is also invaluable. This is because should any adjustments need to be made, all of the tools used, as well as the insight and justification for all decisions, are available in both this text, as well as the accompanying MATLAB code itself.

It should also be noted that the Gazebo model alluded to in the discussion surrounding Figure 8.1 is an excellent starting point for any extensions to this work. This is because it will allow for a fully isolated 3D simulation environment where one could simulate their obstacle avoidance algorithms, as depicted in Figure 8.2, or review how effective their area mapping flight path calculations are, as depicted in Figure 8.3, amongst many other application areas. One of the key reasons for selecting this simulation environment is that its usage is extremely broad, with research having been done with it in topics

ranging from obstacle avoidance [78–80], to area mapping [81], and even to formation flight [82, 83], so it would serve as an ideal candidate for the research areas suggested.

CHAPTER 9

Conclusion

As was stated at the start of this text: there is very little literature that exists on the development of the autopilot and guidance laws upon which some of the higher level research conducted with this airframe is based. It is primarily found that the majority of the literature is based on generic, commercially available, autopilot systems. While there is nothing inherently incorrect about this approach, it does impose some potentially severe constraints on the conclusions one could draw if their proposed methodology does not operate as expected. This is because most commercial autopilots are developed to be broadly applicable to many airframes, with very little tuning being done to optimise them for a specific airframe - except for some basic calibration. As such, should one develop a higher level system based on these lower level autopilots, it might simply be the case that while the higher level design developed in the specific research operates correctly, the underlying autopilot is simply not optimal enough to accomplish what is required of it - even if the aircraft itself would be able to with a better tuned GNC system.

This work fills this gap by providing details not only of an already working system which was developed and tested throughout this text, but also on how to develop a well-designed, well-tuned autopilot, guidance and navigation system. Due to this, the work presented can now be extended without any concern that the underlying system is not properly optimised, and should any system built on top of this not operate as expected, one can more conclusively state that the error lies with either the system being developed - as opposed to the underlying GNC system - or perhaps the selection of an inappropriate airframe.

The filling of the gap in the current literate by developing a well-optimised GNC system for the Skywalker X8 airframe was accomplished in this text by illustrating its adherence to the design goals set out in Section 1.3. We required the system to be able to track two sets of waypoints, as depicted in Figures 1.3 and 1.4. This was illustrated in Section 7.1.1 and Section 7.1.2, respectively. We then also required that we have the ability to switch between sets of waypoints and handle this transition from one set to the next gracefully, whereby we treat the closest waypoint of the new set as the last waypoint reached, and the one which follows that as the next waypoint to travel toward. This functionality was illustrated in Section 7.3. Finally, the remaining design goal was to ensure that all of this functionality was available even in the presence of relatively severe wind. This was illustrated in Section 7.2, where wind speeds were sometimes in excess of 60% of the aircraft's airspeed, thus illustrating strong performance in even the most severe conditions. As such, all of our primary design goals were met

and verified entirely - with the full design and optimisation process discussed in this text.

The secondary goal of this research was to establish a strong foundation for future work to be done. In order to facilitate this, all of the tools and processes made use of were ensured to be well-documented and reusable, as well as broadly applicable to many use-cases and not completely constrained to this particular application. All of the software, models and methods used are made freely available to the reader - with the full set of MATLAB scripts and Simulink models utilized being provided [14], and several Gazebo packages and models, as discussed in Section 8.2, also made available [75]. Having access to all of the underlying resources used throughout this text, as well as the text itself illustrating the design process which was followed to establish a final design which meets all of the required design goals, should prove invaluable as if any adjustments need to made for another application this should not prove too challenging.

Continuing from this, several potential improvements and extension points to this work are presented in Section 8.1 and Section 8.2, along with various references and suggestions of starting points provided. This should further facilitate making use of this work as a foundation for future projects and as such illustrates that our secondary goal is readily met.

The purpose of this dissertation was to document the design and optimisation of a full suite of GNC algorithms for a small UAV, the Skywalker X8. This process was fully described, and all of the design goals set out were met. This marks the successful conclusion of this research.

References

[1] K. Gryte, R. Hann, M. Alam, J. Rohac, T. Johansen, and T. Fossen, "Aerodynamic Modeling of the Skywalker X8 Fixed-Wing Unmanned Aerial Vehicle", Jun. 2018, pp. 826–835. DOI: 10.1109/ICUAS.2018.8453370.

[2] R. Farhadi, V. Kortunov, A. Molchanov, and T. Solianyk, "Estimation of the lateral aerodynamic coefficients for skywalker x8 flying wing from real flight-test data", *Acta Polytechnica*, vol. 58, p. 77, Apr. 2018. DOI: 10.14311/AP.2018.58.0077.

[3] K. Gryte, "High Angle of Attack Landing of an Unmanned Aerial Vehicle", M.S. thesis, Norwegian University of Science and Technology, 2015.

[4] J. Smith, J. Su, C. Liu, and W.-H. Chen, "Disturbance Observer Based Control with Anti-Windup Applied to a Small Fixed Wing UAV for Disturbance Rejection", *Journal of Intelligent & Robotic Systems*, vol. 88, no. 2, pp. 329–346, Dec. 2017, ISSN: 1573-0409. DOI: 10.1007/s10846-017-0534-5. [Online]. Available at: https://doi.org/10.1007/s10846-017-0534-5.

[5] K. Gryte, S. Mathisen, T. Johansen, and T. Fossen, "Non-linear Model Predictive Control for Longitudinal and Lateral Guidance of a Small Fixed-Wing UAV in Precision Deep Stall Landing", Jan. 2016. DOI: 10.2514/6.2016-0512.

[6] R. Farhadi, "Robust PID Control Tuning for the Uncertain Nonlinear Dynamic Model of the Unmanned Aerial Vehicle", *Electronics and Control Systems*, vol. 3, Dec. 2017. DOI: 10.18372/1990-5548.53.12143.

[7] J. Ryan, A. Hubbard, J. Todd, R. Carr, J. Box, P. Christoffersen, T. Holt, and N. Snooke, "Repeat UAV photogrammetry to assess calving front dynamics at a large outlet glacier draining the Greenland Ice Sheet", *The Cryosphere Discussions*, vol. 8, Mar. 2014. DOI: 10.5194/tcd-8-2243-2014.

[8] J. Ryan, A. Hubbard, J. Box, J. Todd, P. Christoffersen, R. Carr, T. Holt, and N. Snooke, "UAV photogrammetry and Structure from Motion to assess calving dynamics at Store Glacier, a large outlet draining the Greenland ice sheet", *The Cryosphere*, vol. 9, pp. 1–11, Jan. 2015. DOI: 10.5194/tc-9-1-2015.

[9] P. Dvořák, J. Müllerová, T. Bartaloš, and J. Brůna, "Unmanned aerial vehicles for alien plant species detection and monitoring", *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. XL-1/W4, pp. 83–90, Aug. 2015. DOI: 10.5194/isprsarchives-XL-1-W4-83-2015.

[10] B. Witte, R. Singler, and S. Bailey, "Development of an Unmanned Aerial Vehicle for the Measurement of Turbulence in the Atmospheric Boundary Layer", *Atmosphere*, vol. 8, p. 195, Oct. 2017. DOI: 10.3390/atmos8100195.

[11] J. Paredes, J. Gonzalez, C. Saito, and A. Flores, "Multispectral imaging system with UAV integration capabilities for crop analysis", Jun. 2017, pp. 1–4. DOI: 10.1109/GRSS-CHILE.2017.7996009.

[12] "Pixhawk 4", [Online]. Available at: https://docs.px4.io/v1.9.0/en/flight_controller/pixhawk4.html (Last accessed: 09/15/2019).

[13] "Beaufort wind force scale", [Online]. Available at: https://www.metoffice.gov.uk/weather/guides/coast-and-sea/beaufort-scale (Last accessed: 09/04/2019).

[14] "Code Repository for the Guidance, Navigation & Control of a Small, Unmanned Blended Wing Body Aircraft Dissertation", [Online]. Available at: https://github.com/davidvanwyk/SkywalkerX8 (Last accessed: 09/27/2019).

[15] J. H. Blakelock, *Automatic Control of Aircraft and Missiles*, 2nd ed. New York: John Wiley & Sons, 1991.

[16] R. C. Nelson, *Flight Stability and Automatic Control*, 2nd ed. Boston, MA: McGraw-Hill, 1998.

[17] B. L. Stevens and F. L. Lewis, *Aircraft Control and Simulation*, 2nd ed. Hoboken, NJ: John Wiley & Sons, Inc., 2003.

[18] T. R. Yechout, S. L. Morris, D. E. Bossert, and W. F. Hallgren, *Introduction to Aircraft Flight Mechanics*. AIAA Education Series, American Institute of Aeronautics and Astronautics, 2003.

[19] D. T. Greenwood, *Principles of Dynamics*, 2nd ed. Englewood Cliffs, NJ: Prentice Hall, 1988.

[20] M. D. Shuster, "Survey of attitude representations", *Journal of the Astronautical Sciences*, vol. 41, pp. 439–517, Oct. 1993.

[21] R. W. Beard and T. W. McLain, *Small Unmanned Aircraft: Theory and Practice*. Princeton, NJ: Princeton University Press, 2012.

[22] H. Goldstein, *Classical Mechanics*. Cambridge, MA: Addison-Wesley, 1951.

[23] A. V. Rao, *Dynamics of Particles and Rigid Bodies: A Systematic Approach*. Cambridge University Press, 2006.

[24] J. B. Marion, *Classical Dynamics of Particles and Systems*, 2nd ed. New York: Academic Press, 1970.

[25] W. E. Wiesel, *Spaceflight Dynamics*, 2nd ed. New York: McGraw Hill, 1997.

[26] J. R. Wertz, *Spacecraft Attitude Determination and Control*. Dordrecth, Netherlands: Kluwer Academic Publishers, 1978.

[27] J. Roskam, *Airplane Flight Dynamics and Automatic Flight Controls, Parts I & II*. Lawrence, KS: DARcorporation, 1998.

[28] M. V. Cook, *Flight Dynamics Principles*. New York: John Wiley & Sons, Inc., 1997.

[29] R. F. Stengel, *Flight Dynamics*. Princeton, NJ: Princeton University Press, 2004.

[30] W. M. Spong and W. Vidyasagar, *Robot Dynamics and Conrol*. New York: John Wiley & Sons, Inc., 1989.

[31] K. S. Fu, R. C. Gonzalez, and C. S. G. Lee, *Robotics: Control, Sensing, Vision and Intelligence*. New York: McGraw-Hill, 1987.

[32] "Simulink Engine Interaction with C S-Functions", [Online]. Available at: https://www.mathworks.com/help/simulink/sfg/how-the-simulink-engine-interacts-with-c-s-functions.html (Last accessed: 09/08/2019).

[33] W. F. Phillips, *Mechanics of Flight*, 2nd ed. New Jersey: Wiley, 2010.

[34] "Dryden Wind Turbulence Model (Continuous)", [Online]. Available at: https://www.mathworks.com/help/aeroblks/drydenwindturbulencemodelcontinuous.html (Last accessed: 09/25/2019).

[35] R. Rysdyk, "Course and Heading Changes in Significant Wind", *Journal of Guidance Control Dynamics*, vol. 33, pp. 1311–1312, Jul. 2010. DOI: 10.2514/1.48934.

[36] "atmoscoesa", [Online]. Available at: https://www.mathworks.com/help/aerotbx/ug/atmoscoesa.html (Last accessed: 09/25/2019).

[37] "COESA Atmosphere Model", [Online]. Available at: https://www.mathworks.com/help/aeroblks/coesaatmospheremodel.html (Last accessed: 09/25/2019).

[38] "findop", [Online]. Available at: https://www.mathworks.com/help/slcontrol/ug/findop.html?searchHighlight=findop&s_tid=doc_srchtitle (Last accessed: 09/25/2019).

[39] "CAP 722: Unmanned Aircraft System Operations in UK Airspace - Guidance & Policy", *CAA*, 2019.

[40] "FAA Federal Regulations, Title 14: Aeronautics and Space, Part 107-Small Unmanned Aircraft Systems", *FAA*, 2018.

[41] "International Standard Atmosphere (ISA)", [Online]. Available at: https://www.skybrary.aero/index.php/International_Standard_Atmosphere_(ISA) (Last accessed: 09/25/2019).

[42] "linearize", [Online]. Available at: https://www.mathworks.com/help/slcontrol/ug/linearize.html (Last accessed: 09/25/2019).

[43] "linspace", [Online]. Available at: https://www.mathworks.com/help/matlab/ref/linspace.html (Last accessed: 09/25/2019).

[44] "imuSensor: IMU Simulation Model", [Online]. Available at: https://www.mathworks.com/help/fusion/ref/imusensor-system-object.html (Last accessed: 09/14/2019).

[45] J. Rankin, "GPS and differential GPS: An error model for sensor simulation", *Position, Locatio, and Navigation Symposium*, pp. 260–266, 1994.

[46] *GPS-AIDED INERTIAL NAVIGATION SYSTEM*, VN-200 GPS/INS, VECTORNAV.

[47] R. Figliola and D. Beasley, *Theory and Design for Mechanical Measurements*. New York: John Wiley & Sons, 2006.

[48] *MPU-6000/MPU-6050 Product Specification*, MPU-6000/MPU-6050, Rev. 3.4, Invensense, Aug. 2013.

[49] *MS4525DO Data Sheet*, MS4525DO, TE Connectivity, Apr. 2019.

[50] *BMP280 Digital Pressure Sensor*, BMP280, Rev. 1.14, Bosch Sensortec, May 2015.

[51] *Smart Miniature Vane SMV-1*, SMV-1, Release 2, Simtec Buergel AG, Mar. 2017.

[52] B. Biju and S. Pradeep, "Automatic Landing System Design using Feedback Linearization Method", vol. 1, May 2007, ISBN: 978-1-62410-017-8. DOI: 10.2514/6.2007-2733.

[53] "Team Corally - CS-3007 HV High Speed Mini Servo - High Voltage - Coreless Motor - Titanium Gear - Ball Beared - Full Alloy Case", [Online]. Available at: https://corally.com/Servos/Team-Corally-CS-3007-HV-High-Speed-Mini-Servo-High-Voltage-Coreless-Motor-Titanium-Gear-Ball-Beared-Full-Alloy-Case/ (Last accessed: 09/15/2019).

[54] W. Chen, *The Electrical Engineering Handbook*. London: Elsevier Academic Press, 2004.

[55] "balred", [Online]. Available at: https://www.mathworks.com/help/control/ref/balred.html (Last accessed: 09/26/2019).

[56] "Stateflow: Model and simulate decision logic using state machines and flow charts", [Online]. Available at: https://www.mathworks.com/products/stateflow.html (Last accessed: 09/27/2019).

[57] "c2d", [Online]. Available at: https://www.mathworks.com/help/control/ref/c2d.html (Last accessed: 09/26/2019).

[58] R. E. Kalman, "A New Approach to Linear Filtering and Prediction Problems", *Journal of Basic Engineering*, vol. 82, no. 1, p. 35, 1960. DOI: 10.1115/1.3662552. [Online]. Available at: https://doi.org/10.1115%2F1.3662552.

[59] F. L. Lewis, *Optimal Estimation: With an Introduciton to Stochastic Control Theory*. New York: John Wiley & Sons, 1986.

[60] A. Gelb, *Applied Optimal Estimation*. Cambridge, MA: MIT Press, 1974.

[61] B. D. O. Anderson and J. B. Moore, *Linear Optimal Control*. Englewood Cliffs, NJ: Prentice Hall, 1971.

[62] R. G. Brown, *Introduction to Random Signal Analysis and Kalman Filtering*. New York: John Wiley & Sons, Inc., 1983.

[63] "Extended Kalman Filter", [Online]. Available at: https://www.mathworks.com/help/control/ref/ekf_block.html (Last accessed: 09/26/2019).

[64] R. W. Beard, "Embedded UAS autopilot and sensor systems", *Encyclopedia of Aerospace Engineering*, pp. 4799–4814, 2010.

[65] D. R. Nelson, D. B. Barber, T. W. McLain, and R. W. Beard, "Vector Field Path Following for Miniature Air Vehicles", *IEEE Transactions on Robotics*, vol. 23, pp. 519–529, 2007.

[66] D. Nelson, D. Barber, T. McLain, and R. Beard, "Vector field path following for small unmanned air vehicles", Jul. 2006, pp. 5788–5794. DOI: 10.1109/ACC.2006.1657648.

[67] E. Anderson, R. Beard, and T. McLain, "Real-time dynamic trajectory smoothing for unmanned air vehicles", *Control Systems Technology, IEEE Transactions on*, vol. 13, pp. 471–477, Jun. 2005. DOI: 10.1109/TCST.2004.839555.

[68] "Generate C Code from Simulink Model", [Online]. Available at: https://www.mathworks.com/help/dsp/ug/generate-c-code-from-simulink-model.html (Last accessed: 09/27/2019).

[69] J. B. Kuipers, *Quaternions and Rotation Sequences: A Primer with Applications to Orbits, Aerospace, and Virtual Reality*. Princetion, NJ: Princeton University Press, 1999.

[70] "Wind Shear Model", [Online]. Available at: https://www.mathworks.com/help/aeroblks/windshearmodel.html (Last accessed: 09/27/2019).

[71] "Discrete Wind Gust Model", [Online]. Available at: https://www.mathworks.com/help/aeroblks/discretewindgustmodel.html (Last accessed: 09/27/2019).

[72] N. El-Sheimy, H. Hou, and X. Niu, "Analysis and modeling of inertial sensors using Allan variance", *IEEE Transactions on instrumentation and measurement*, vol. 57, no. 1, pp. 140–149, 2007.

[73] "Create linear and nonlinear dynamic system models from measured input-output data", [Online]. Available at: https://www.mathworks.com/products/sysid.html (Last accessed: 09/27/2019).

[74] "Hardware in the Loop Simulation (HITL)", [Online]. Available at: https://dev.px4.io/v1.9.0/en/simulation/hitl.html (Last accessed: 09/27/2019).

[75] "Gazebo package and model files for the Skywalker X8", [Online]. Available at: https://github.com/davidvanwyk/gazebo (Last accessed: 09/27/2019).

[76] "FaSTrack: Ensuring Safe Real-Time Navigation of Dynamic Systems", [Online]. Available at: http://ompl.kavrakilab.org/blog.html (Last accessed: 09/24/2019).

[77] A. Rajendraprakash, "Terrain mapping near the vehicle, SLAM and global map building for lunar rover", en, G2 Pro gradu, diplomityö, 2013-08-26, pp. 67+7. [Online]. Available at: http://urn.fi/URN:NBN:fi:aalto-201311017767.

[78] A. Budiyanto, A. Cahyadi, T. B. Adji, and O. Wahyunggoro, "UAV obstacle avoidance using potential field under dynamic environment", in *2015 International Conference on Control, Electronics, Renewable Energy and Communications (ICCEREC)*, IEEE, 2015, pp. 187–192.

[79] N. Kumar, Z. Vamossy, and Z. Szabó-Resch, "Robot obstacle avoidance using bumper event", in *2016 IEEE 11th International Symposium on Applied Computational Intelligence and Informatics (SACI)*, IEEE, 2016, pp. 485–490.

[80] S. Hrabar, "3D path planning and stereo-based obstacle avoidance for rotorcraft UAVs", in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, 2008, pp. 807–814.

[81] N. Habibie, A. M. Nugraha, A. Z. Anshori, M. A. Ma'sum, and W. Jatmiko, "Fruit mapping mobile robot on simulated agricultural area in Gazebo simulator using simultaneous localization and mapping (SLAM)", in *2017 International Symposium on Micro-NanoMechatronics and Human Science (MHS)*, IEEE, 2017, pp. 1–7.

[82] Y. Fu, X. Wang, L. Huan, and H. Zhu, "Multi-UAV formation control method based on modified artificial physics", in *2016 Chinese Control and Decision Conference (CCDC)*, IEEE, 2016, pp. 2523–2529.

[83] A. A. A. Rizqi, A. I. Cahyadi, and T. B. Adji, "Path planning and formation control via potential function for uav quadrotor", in *2014 International Conference on Advanced Robotics and Intelligent Systems (ARIS)*, IEEE, 2014, pp. 165–170.

APPENDIX A

Simulink Models

All Simulink models are available in the models folder with the root directory found at [14].

A.1 Plant

A.1.1 Six Degree of Freedom Model

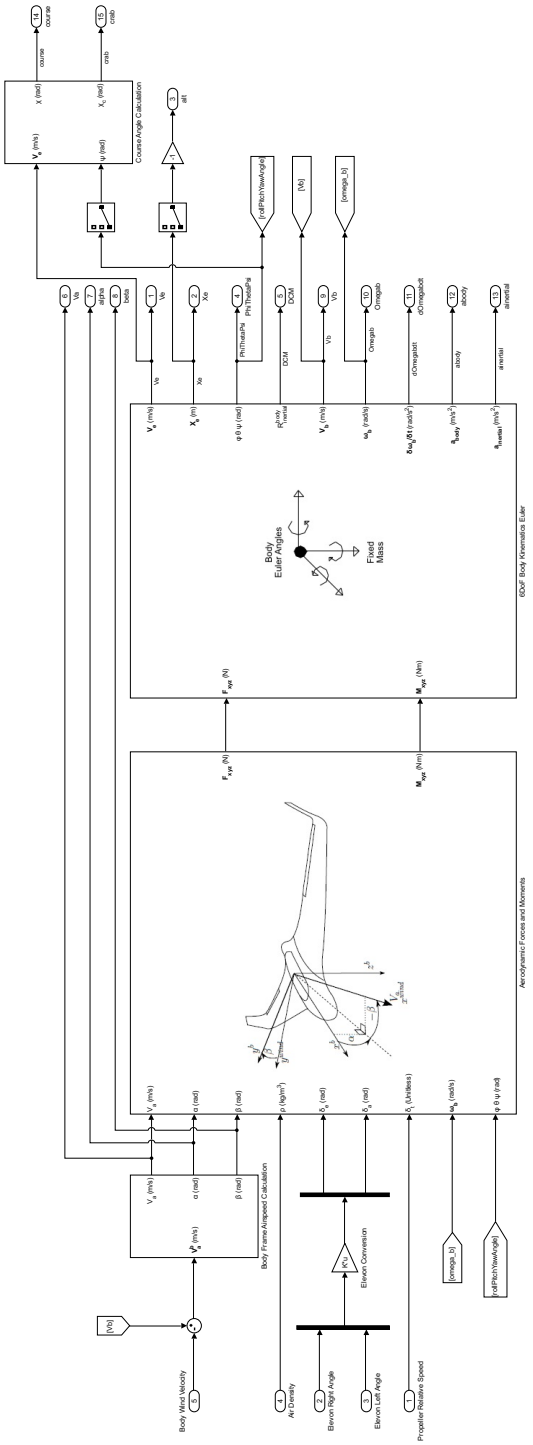


Figure A.1: Simulink implementation of the full Skywalker X8 aircraft model

A.1.2 Atmospheric Model

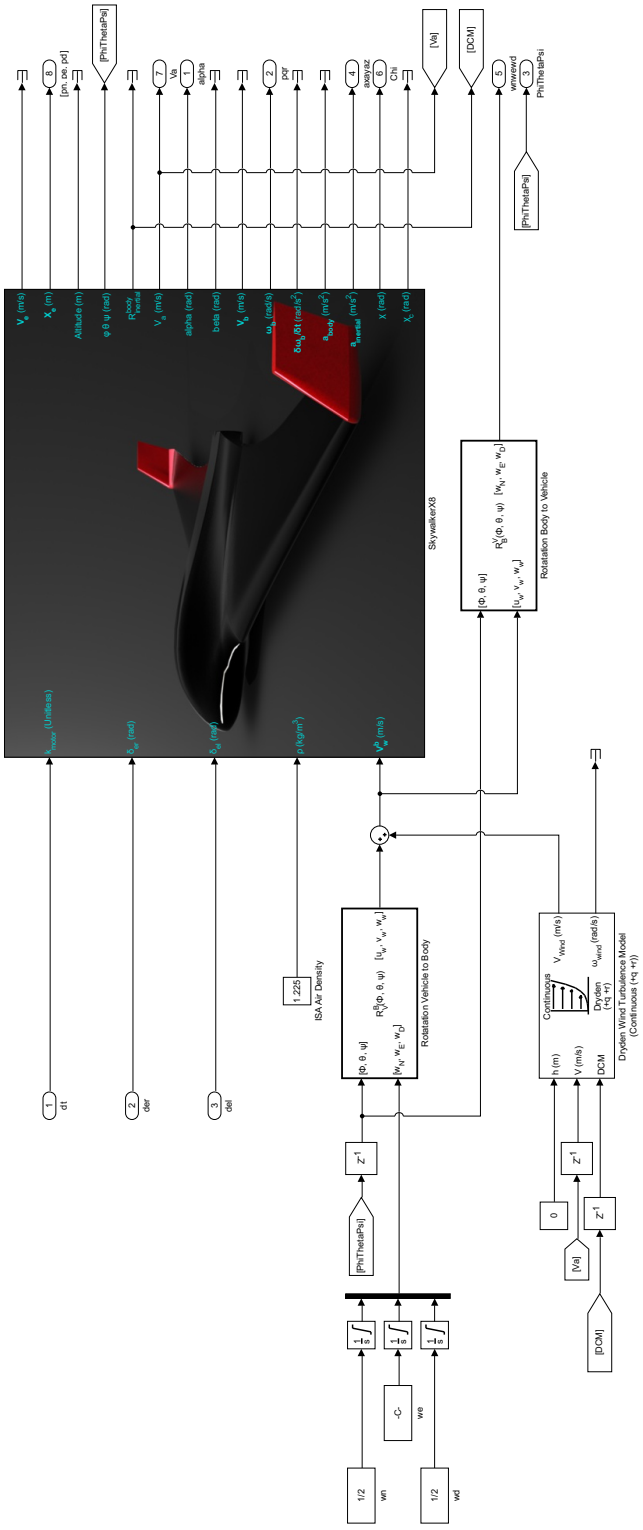


Figure A.2: Skywalker X8 aircraft model including wind modelling

A.1.3 Flight Envelope Generation Model

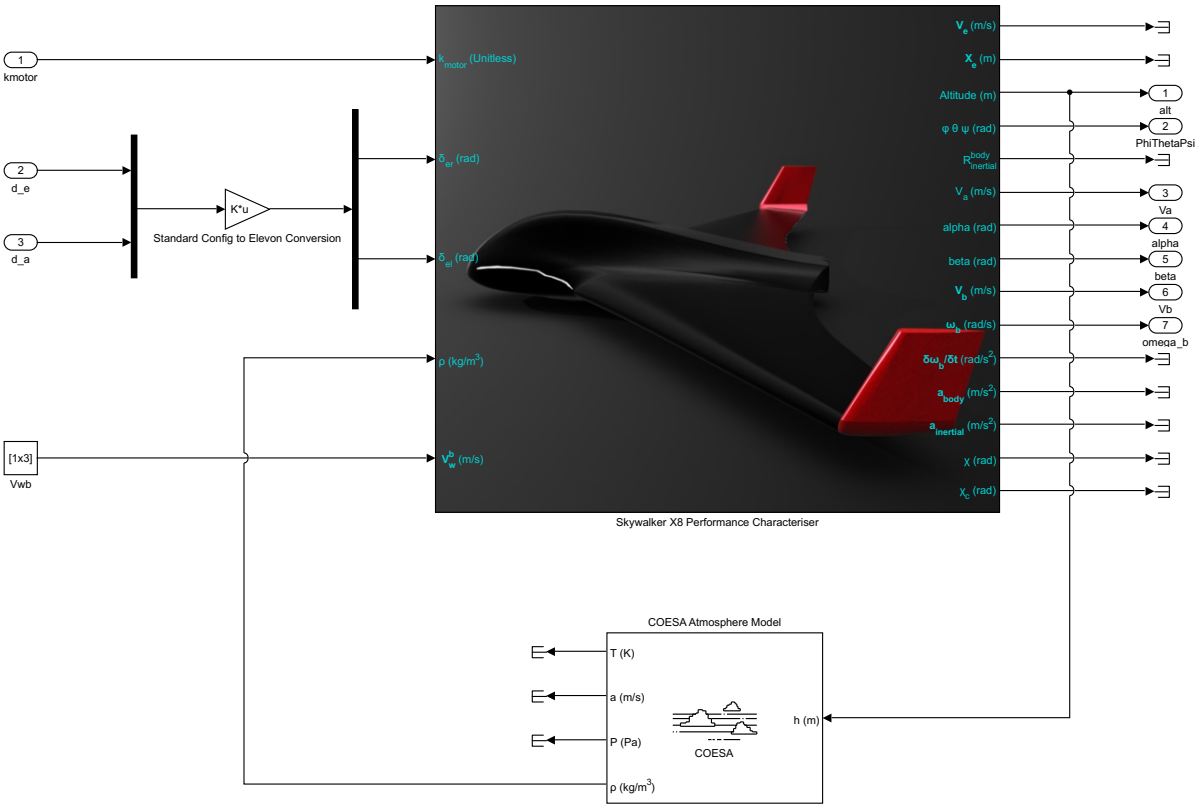


Figure A.3: Simulink implementation of the performance characterization model

A.2 Sensors

A.2.1 GPS

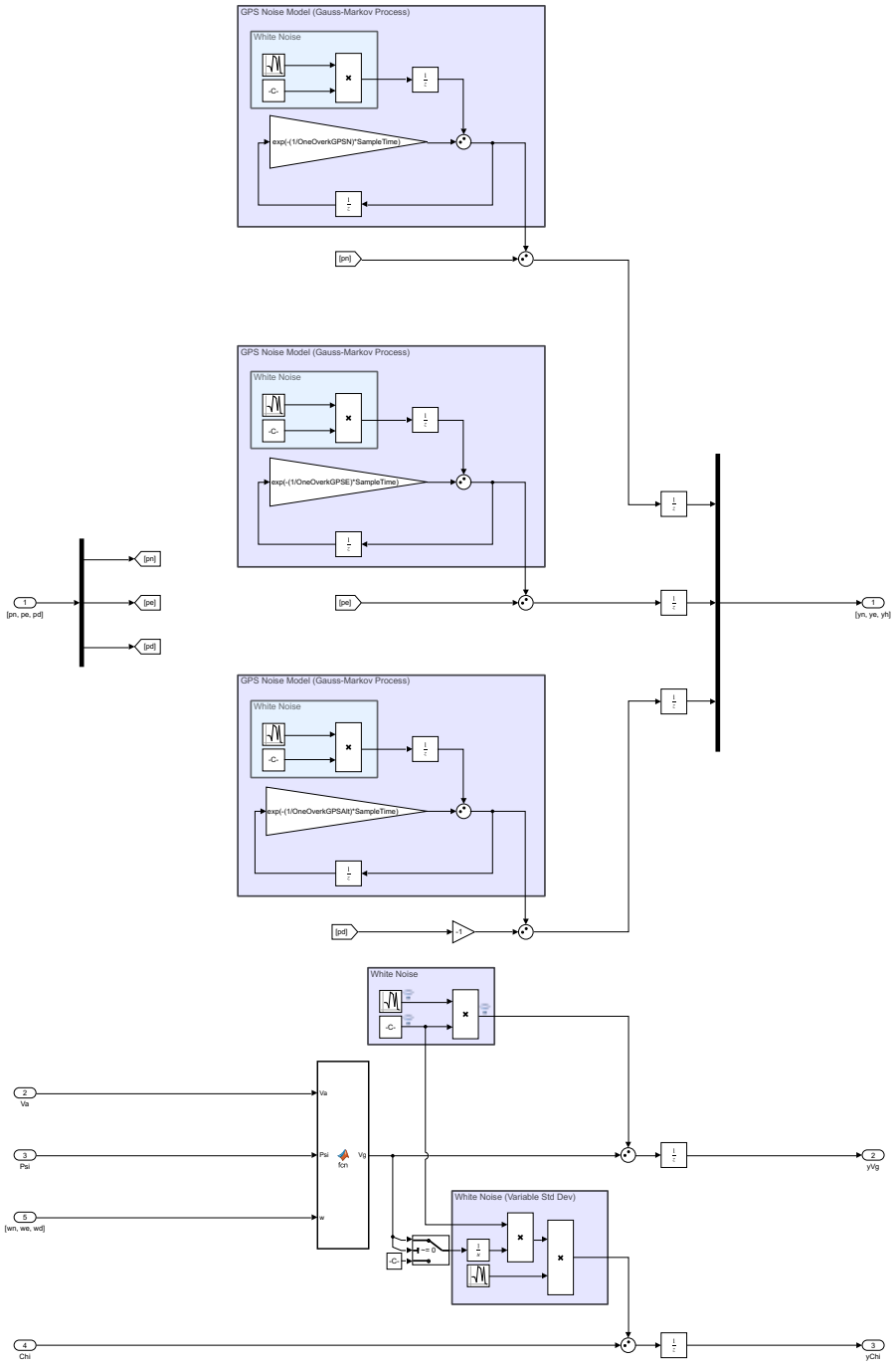


Figure A.4: GPS Sensor implementation

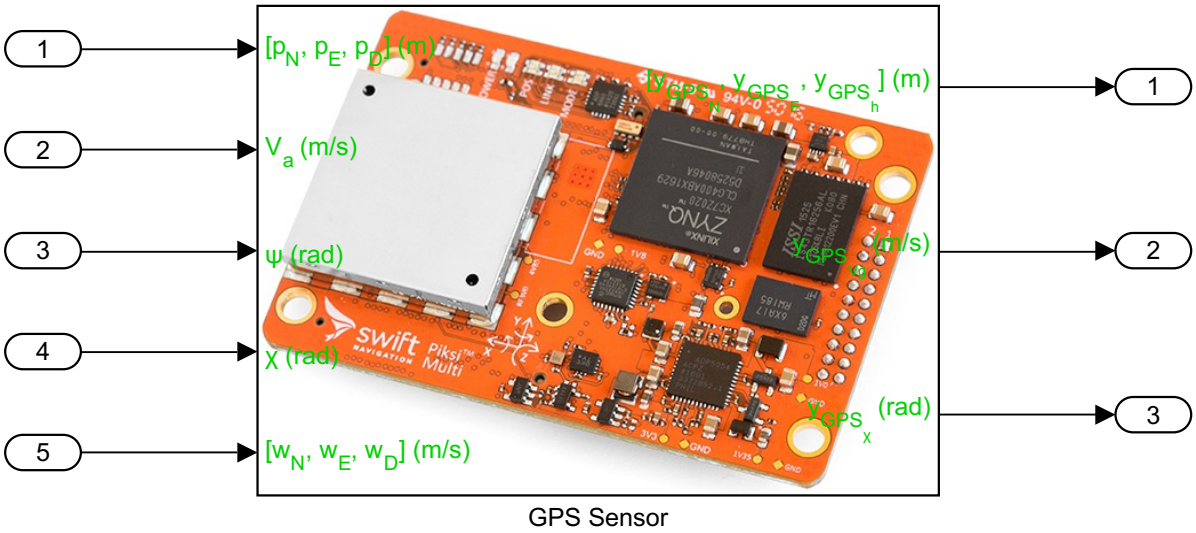


Figure A.5: GPS Sensor masked subsystem

A.2.2 9-Axis IMU

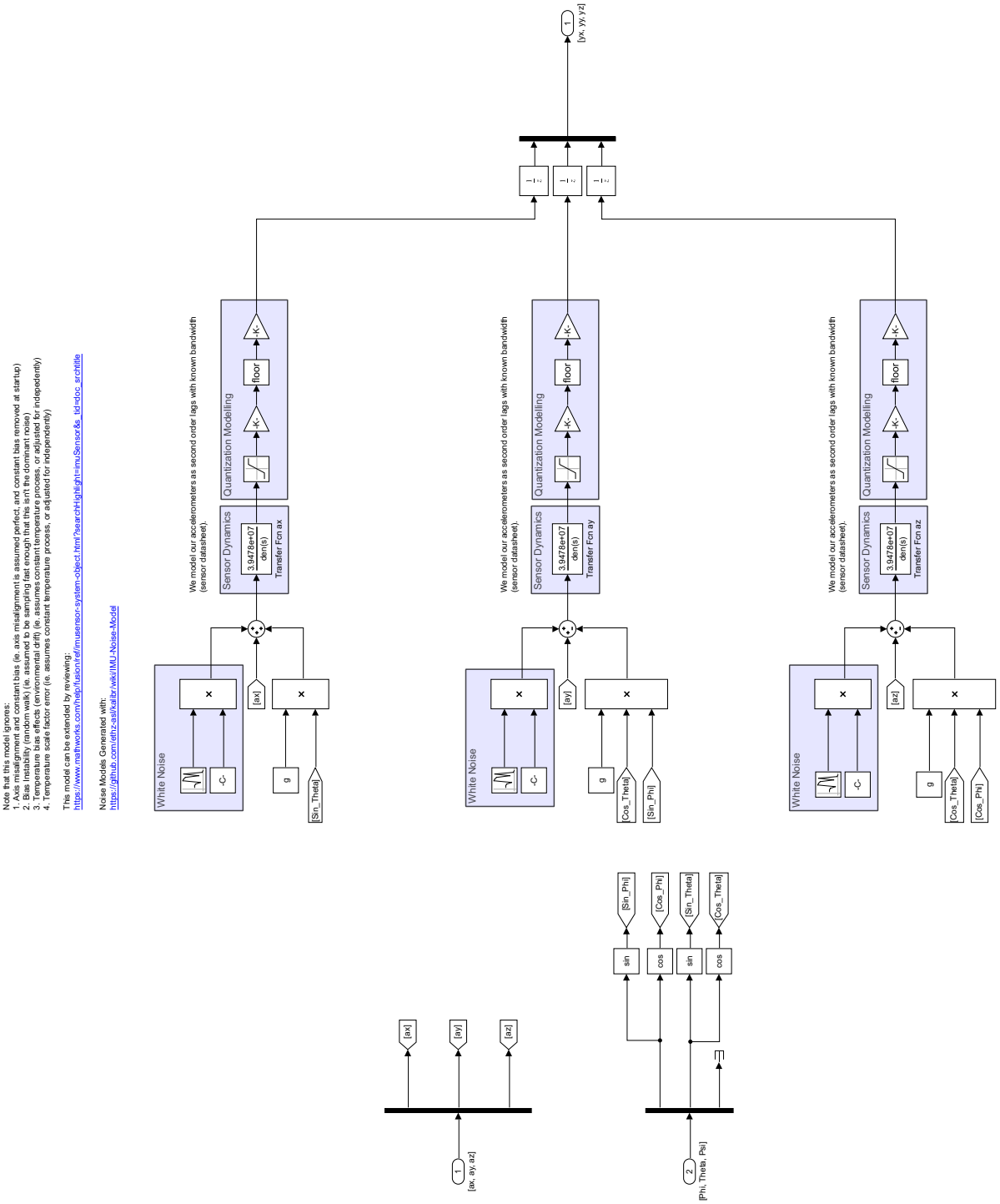


Figure A.6: 3-Axis Accelerometer sensor model implementation

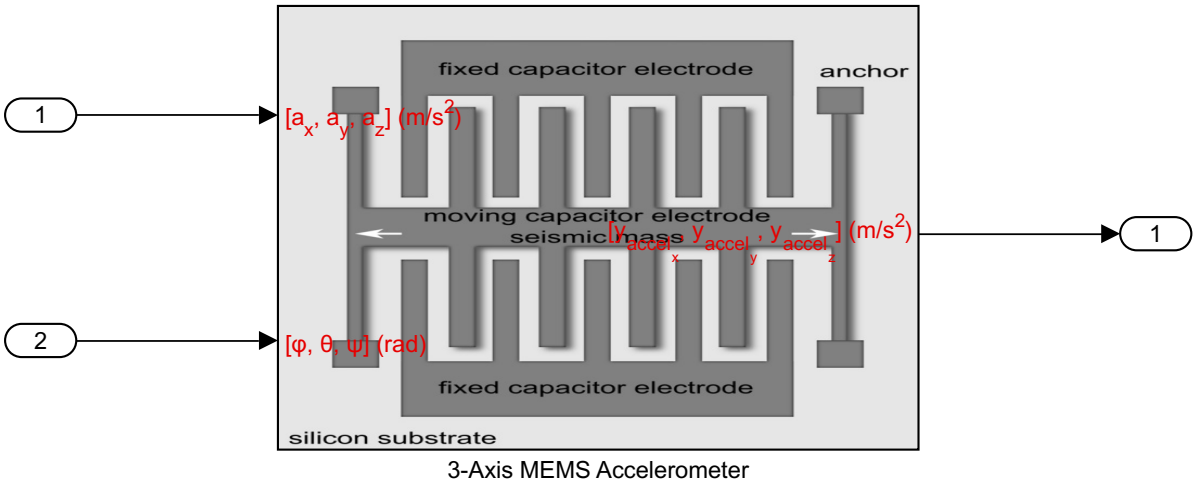
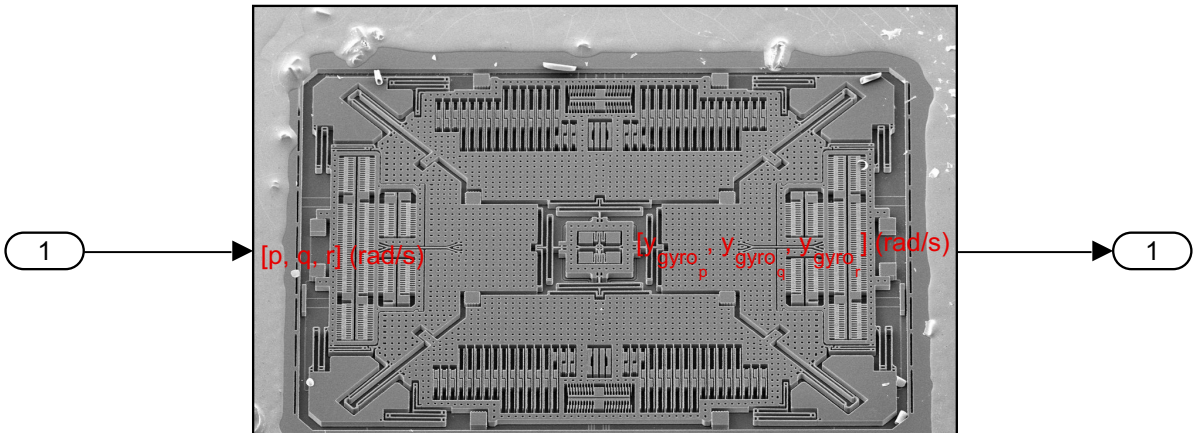


Figure A.7: 3-Axis Accelerometer masked subsystem

Figure A.8: 3-Axis Gyroscope sensor model implementation



Note that this model ignores:
1. Bias Instability (random walk) (ie. assumed to be sampling fast enough that this isn't the dominant noise)
2. Temperature bias effects (environmental drift) (ie. assumes constant temperature process, or adjusted for independently)
3. Temperature scale factor error (ie. assumes constant temperature process, or adjusted for independently)

This model can be extended by reviewing:
https://www.mathworks.com/help/fusion/ref/imusensor-system-object.html?searchHighlight=imuSensor&s_tid=doc_srchtitle

Noise Model Generated with:
https://github.com/ethz-asl/kalibr/wiki/IMU-Noise-Model

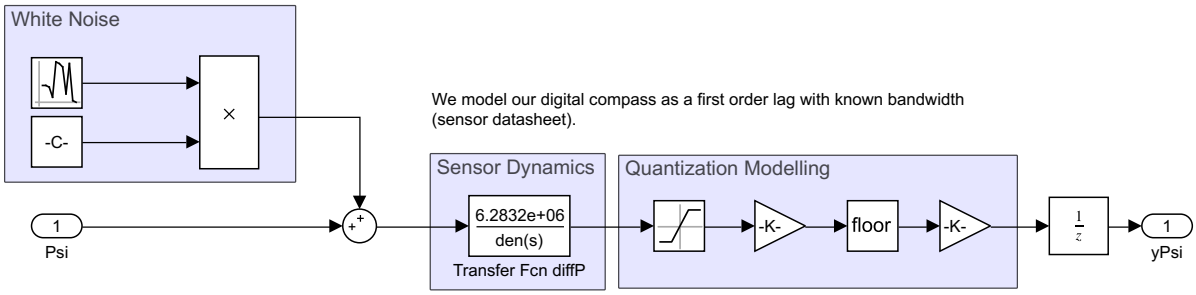


Figure A.10: Magnetometer Compass sensor model implementation

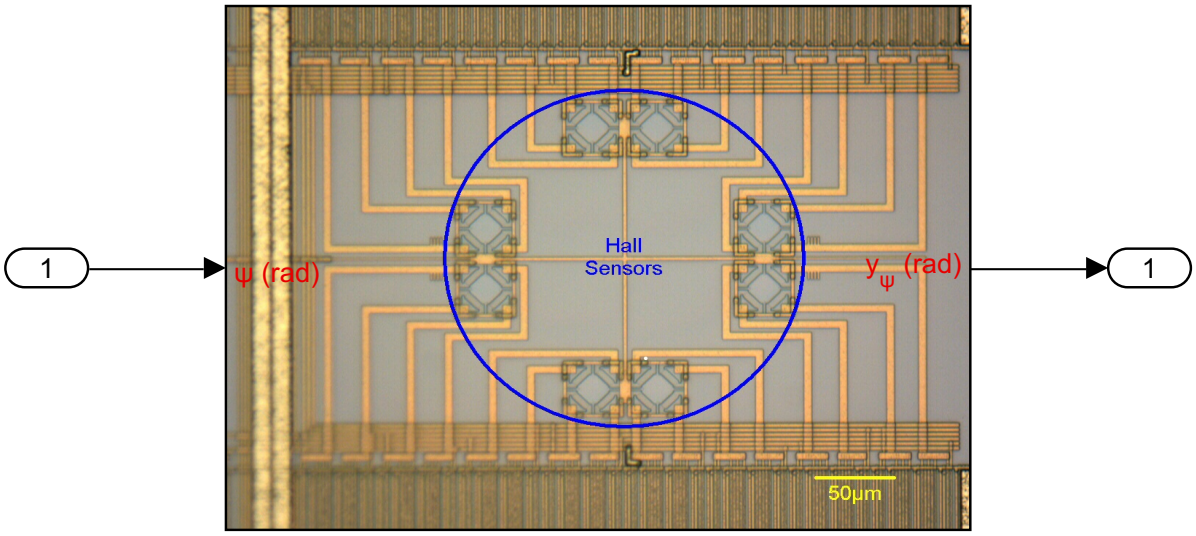


Figure A.11: Magnetometer Compass masked subsystem

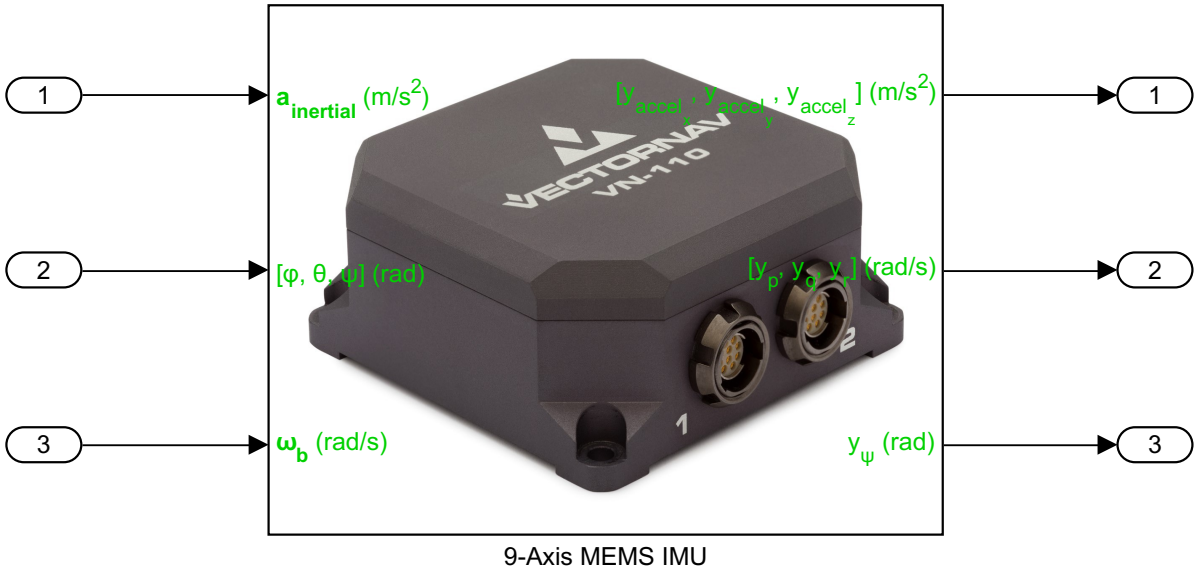


Figure A.12: 9-Axis MEMS IMU masked subsystem

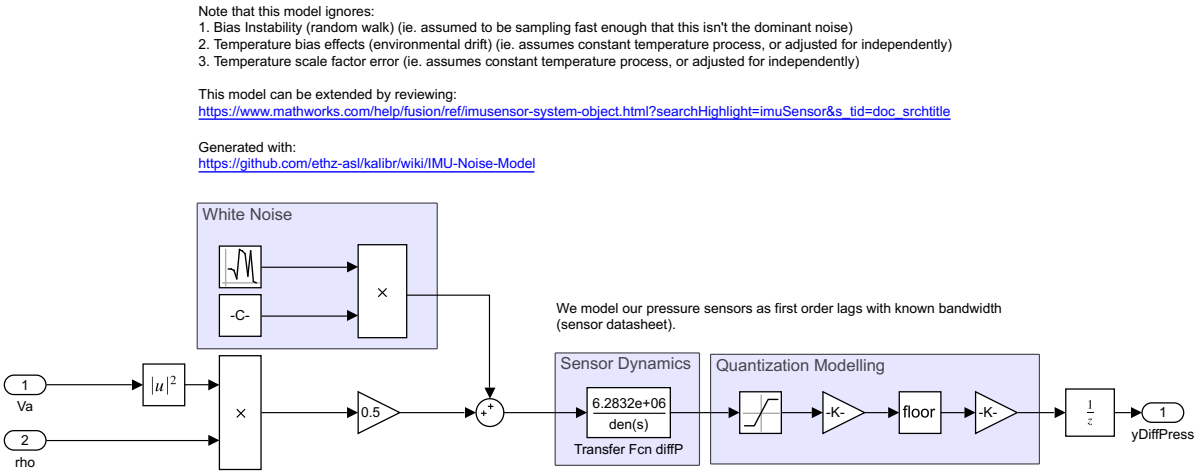


Figure A.13: Airspeed Sensor model

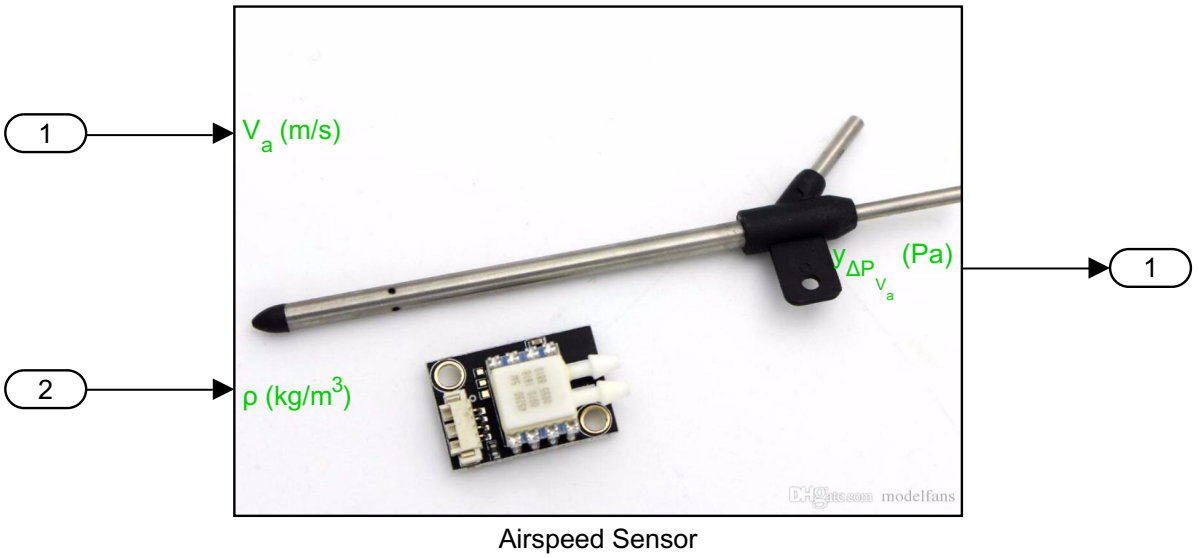


Figure A.14: Airspeed Sensor model masked subsystem

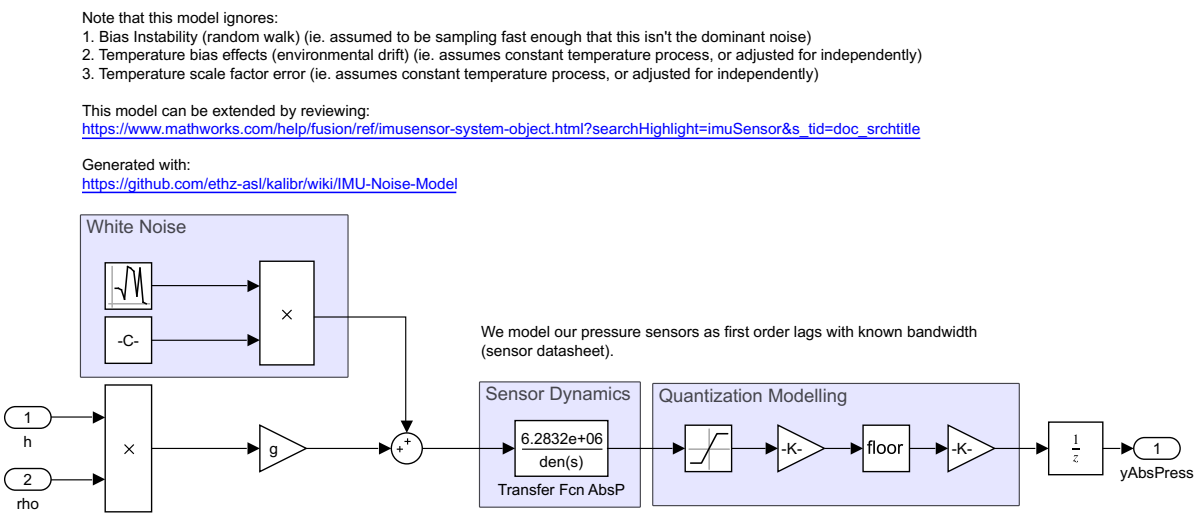


Figure A.15: Barometric Altitude Pressure Sensor model

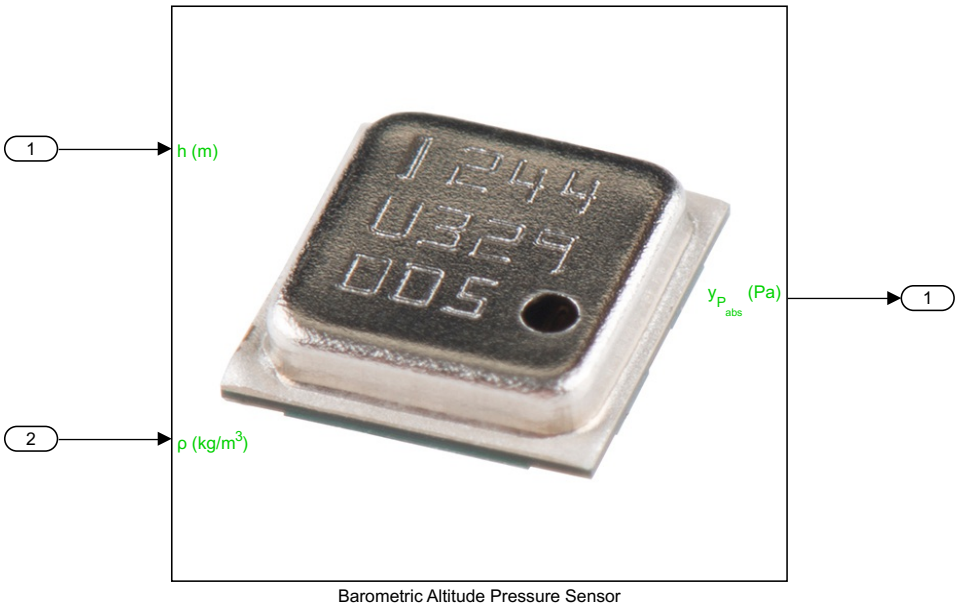


Figure A.16: Barometric Altitude Pressure Sensor model masked subsystem

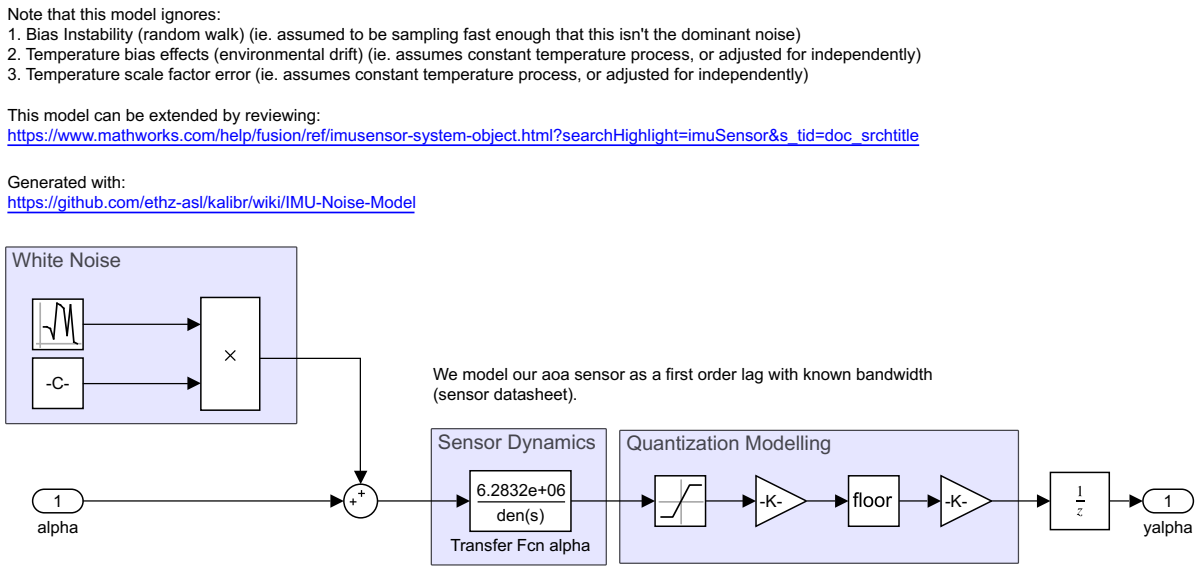


Figure A.17: Angle of Attack Sensor model

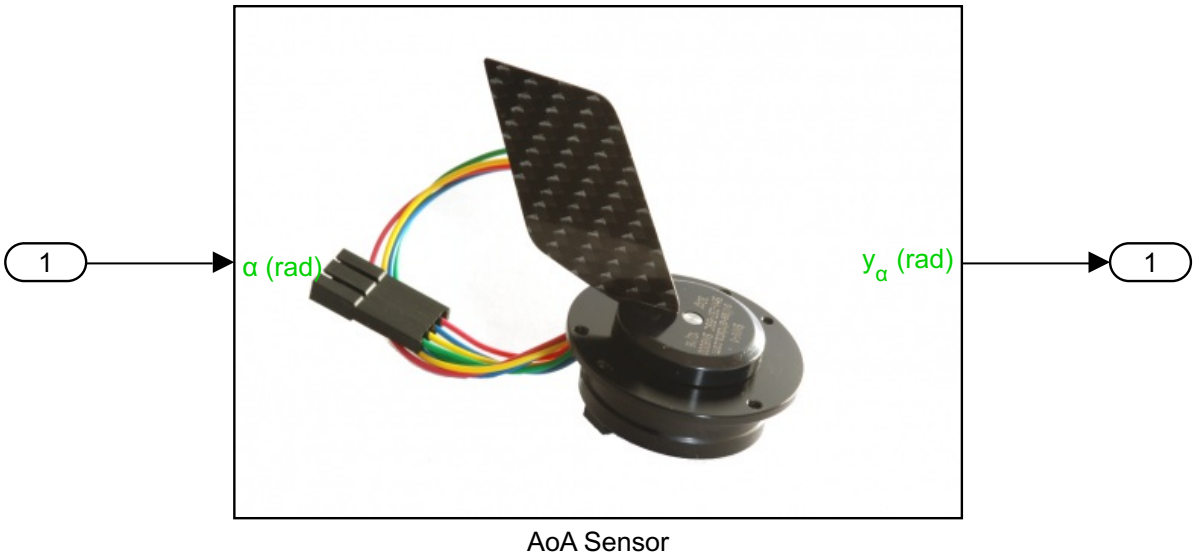


Figure A.18: Angle of Attack Sensor model masked subsystem

A.2.6 Sensor Suite

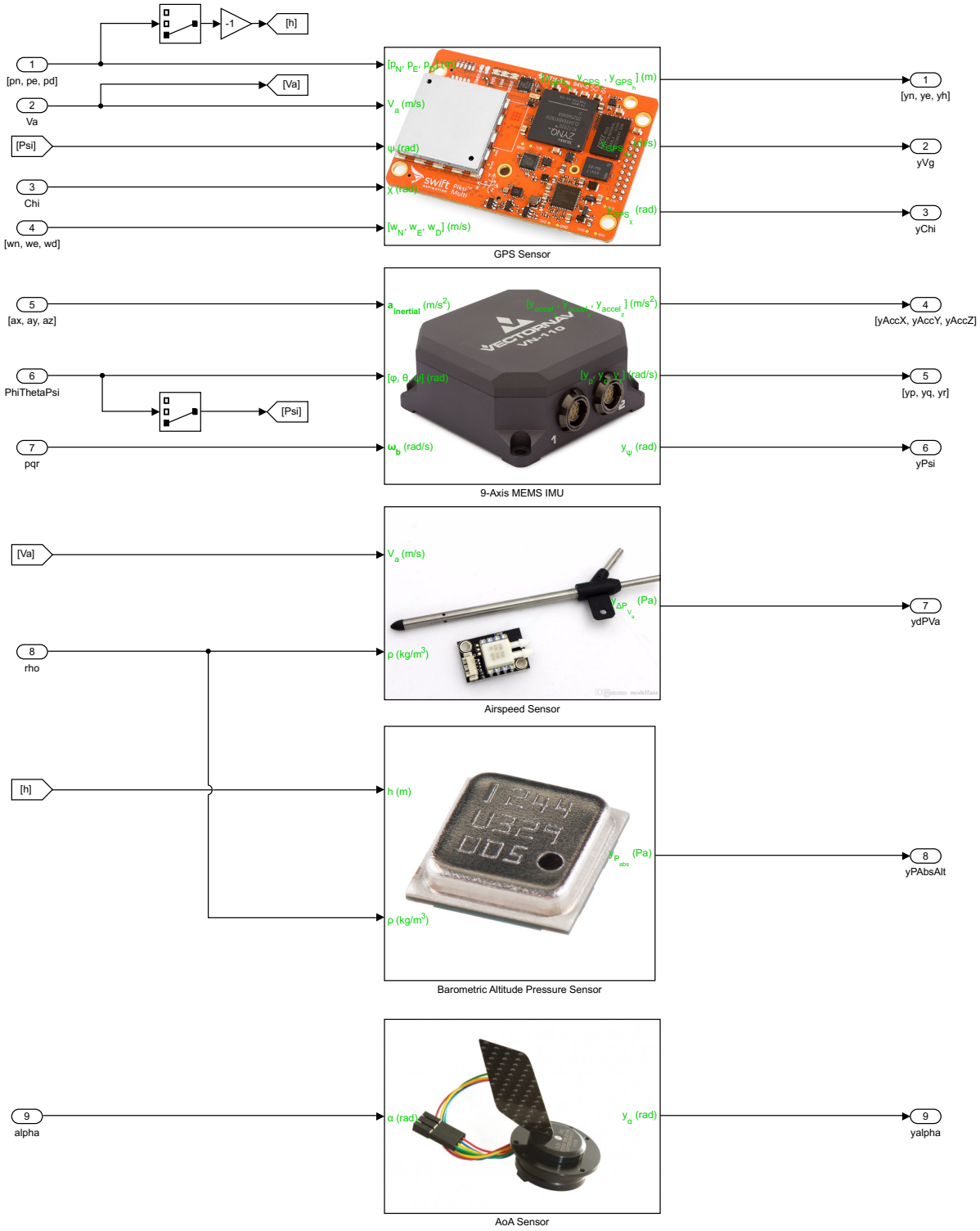


Figure A.19: Full sensor suite model

A.2.7 Sensor Suite Verification

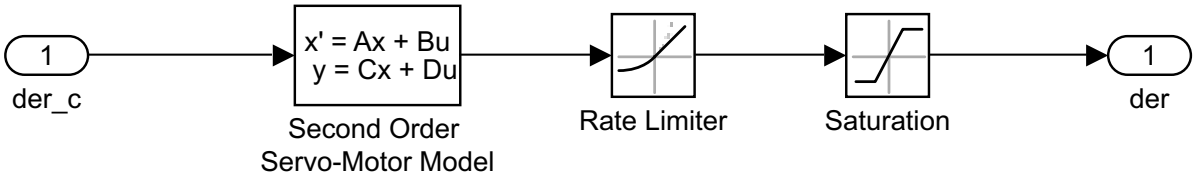


Figure A.21: Servo-Motor actuator model

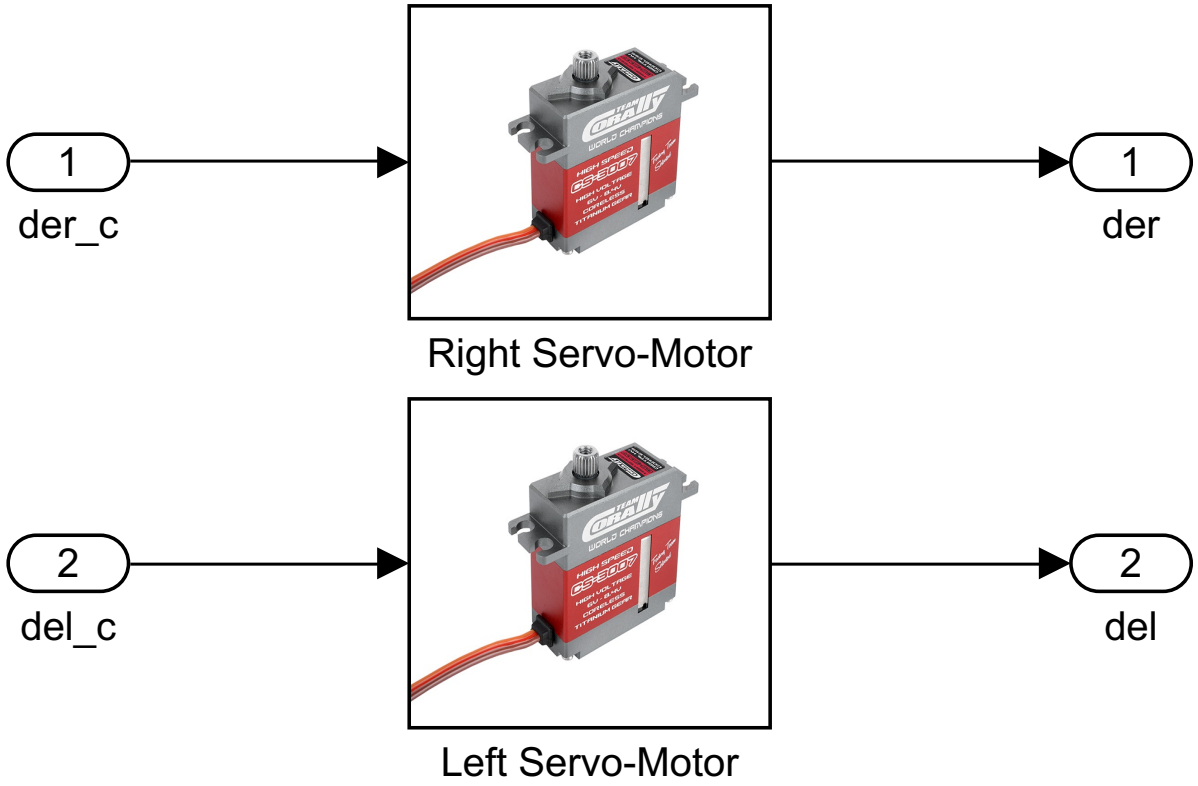


Figure A.22: Servo-Motor actuator model masked subsystem

A.3.2 Propeller Motor

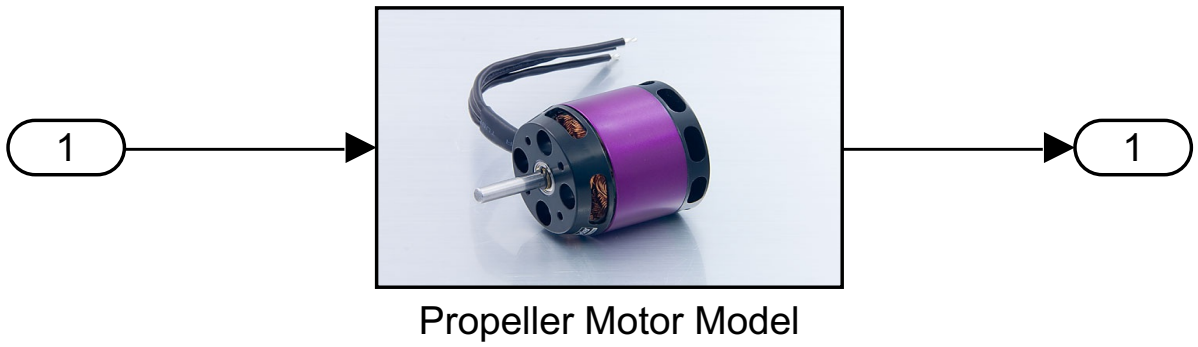


Figure A.23: Propeller Motor actuator model masked subsystem

A.4 Gain Scheduled Controllers

A.4.1 Longitudinal Controllers

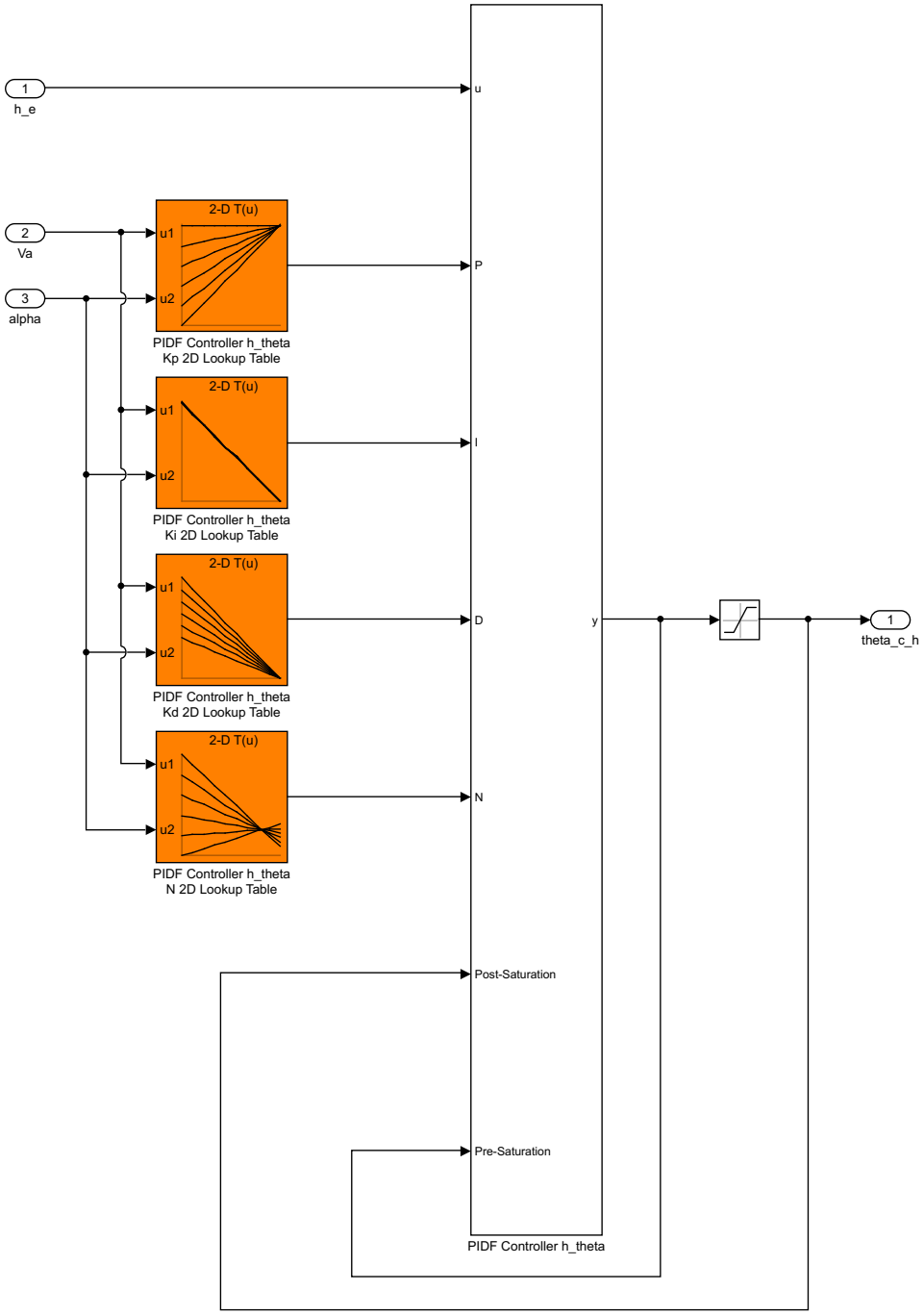


Figure A.24: Standard longitudinal gain scheduled controller as a function of (V_a, α), using the altitude controller as an example

A.4.2 Lateral Controllers

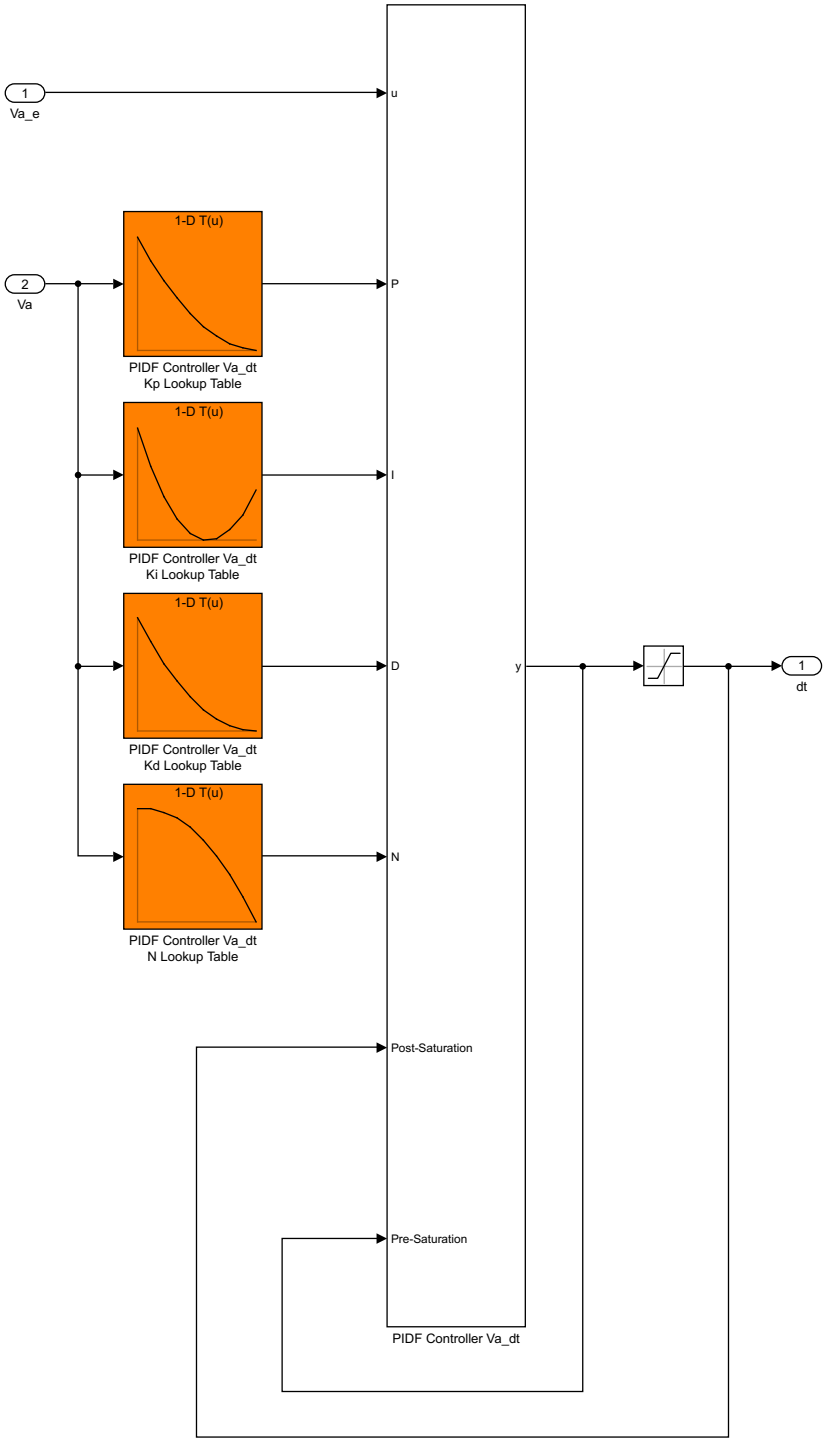


Figure A.25: Standard lateral gain scheduled controller as a function of V_a, using the airspeed controller as an example

A.5 Longitudinal Controller Tuning Model

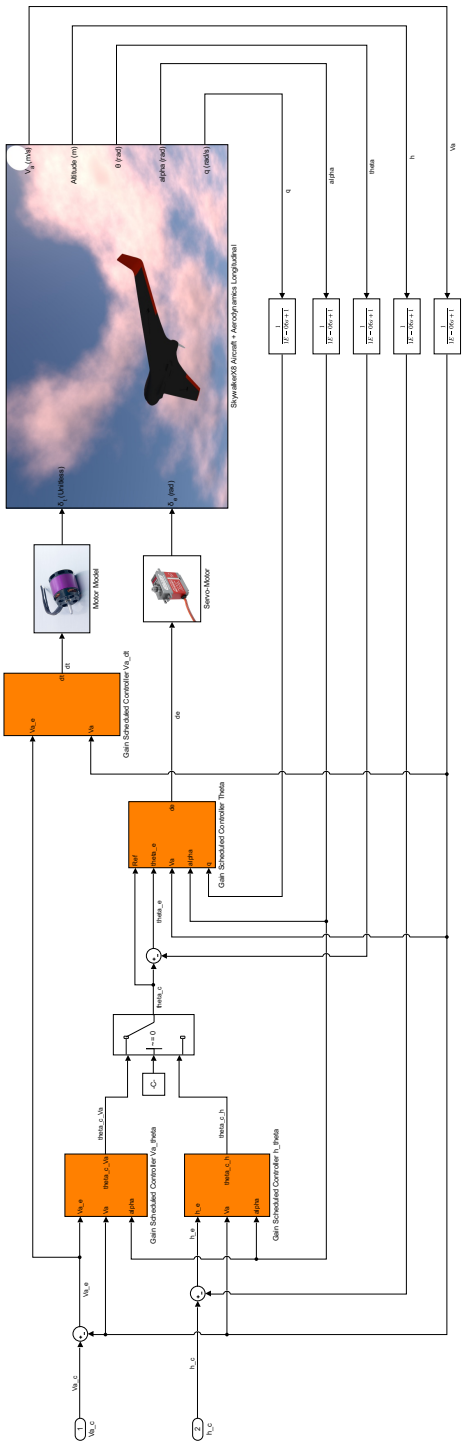


Figure A.26: Longitudinal control system tuning model

A.6 Lateral Controller Tuning Model

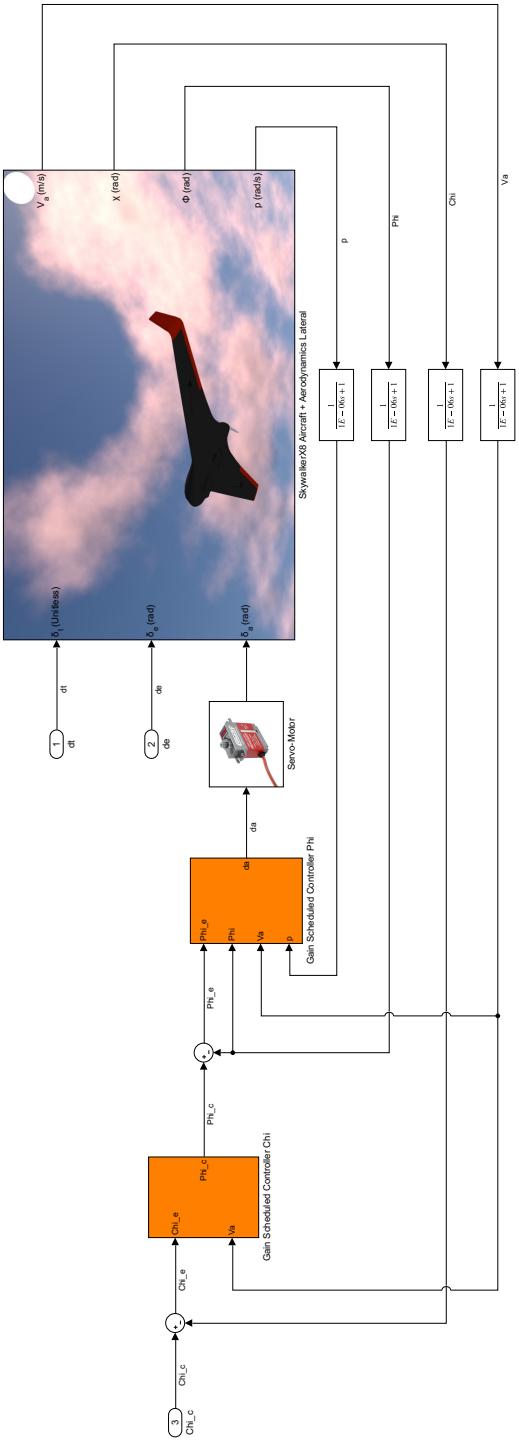


Figure A.27: Lateral control system tuning model

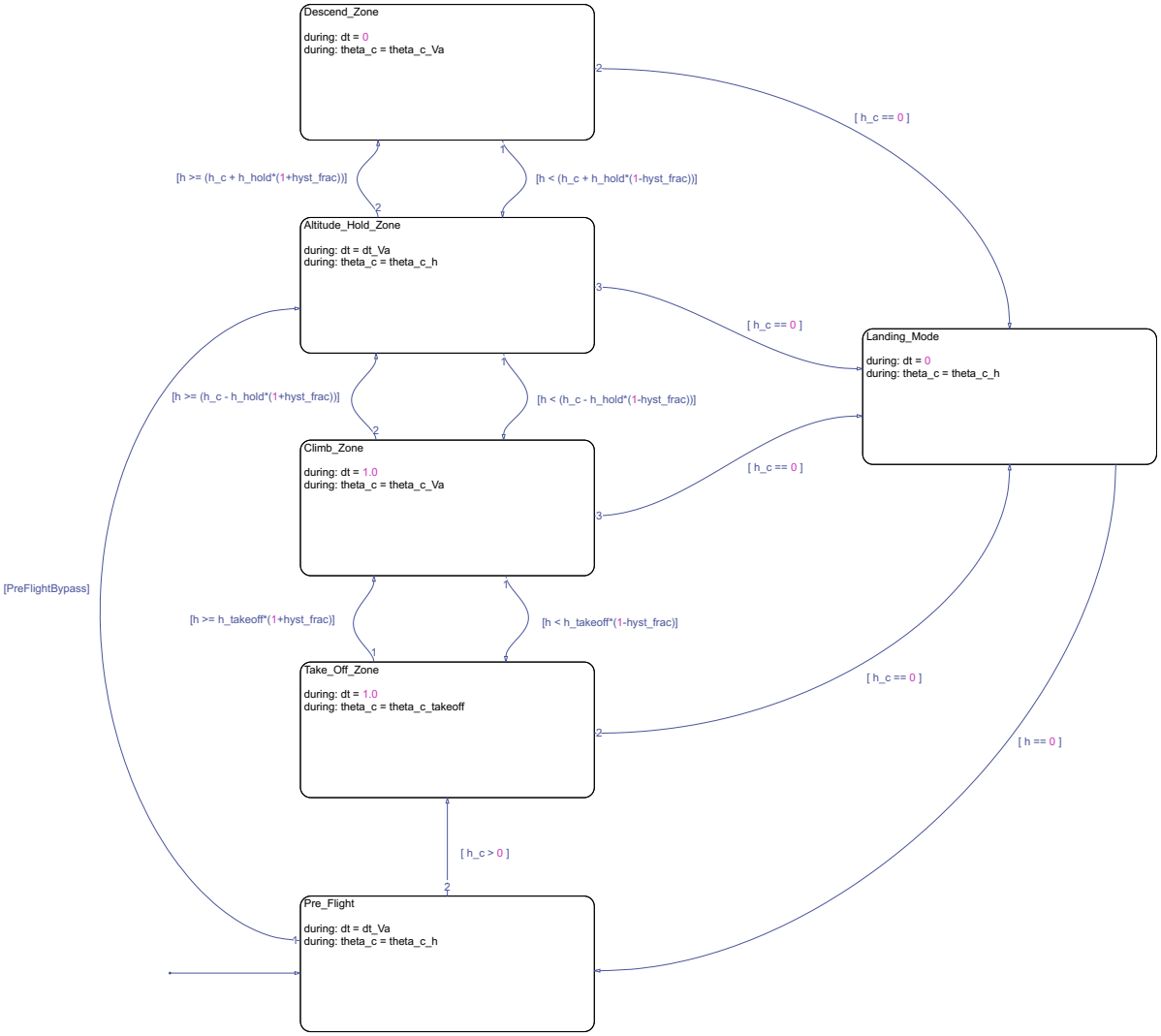


Figure A.28: Longitudinal State Machine implementation in Stateflow

A.8 Longitudinal Control Verification

A.8.1 Continuous Controllers

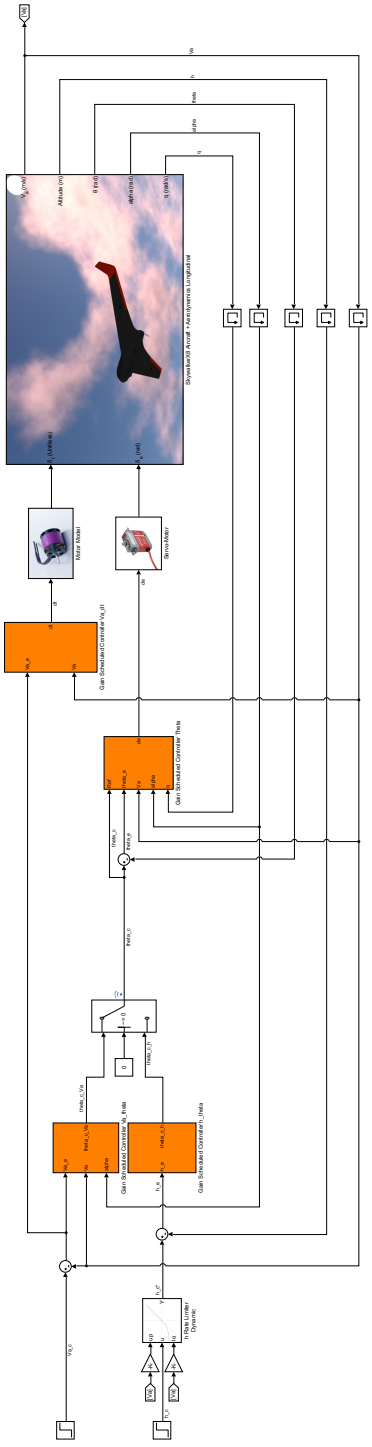


Figure A.29: Longitudinal controller verification implementation

A.8.2 Discrete Controllers

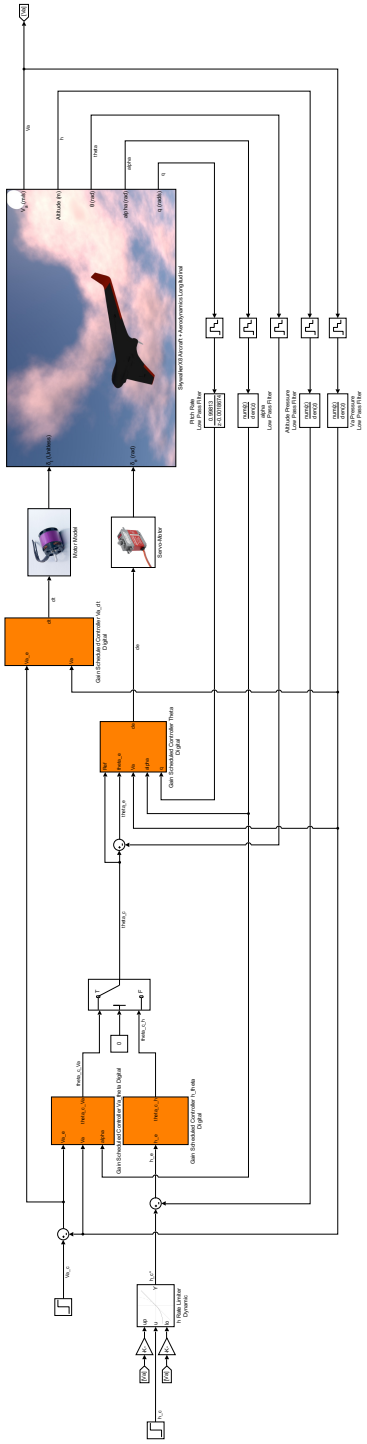


Figure A.30: Digital longitudinal controller verification implementation

A.9 Lateral Control Verification

A.9.1 Continuous Controllers

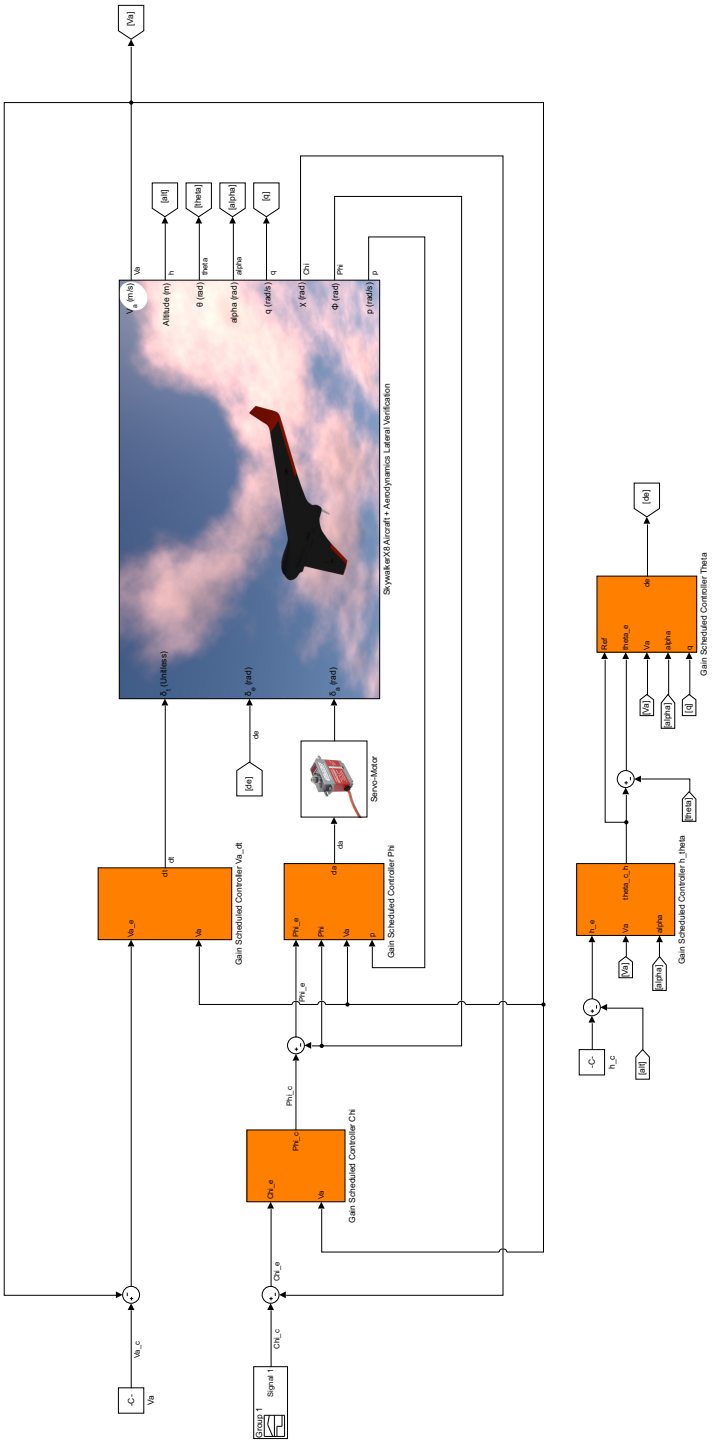


Figure A.31: Lateral controller verification implementation

A.9.2 Discrete Controllers

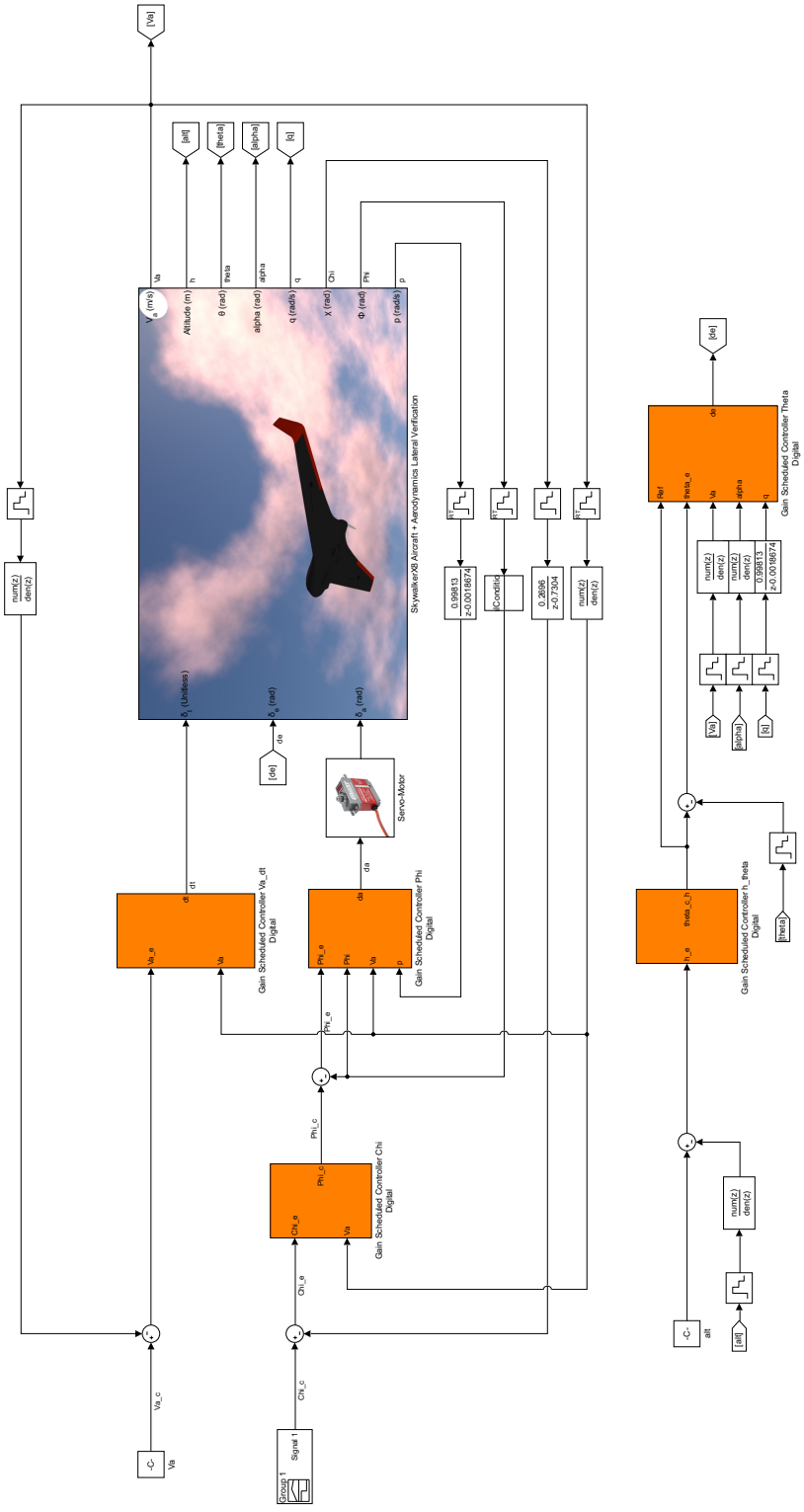


Figure A.32: Digital lateral controller verification implementation

A.10 Autopilot

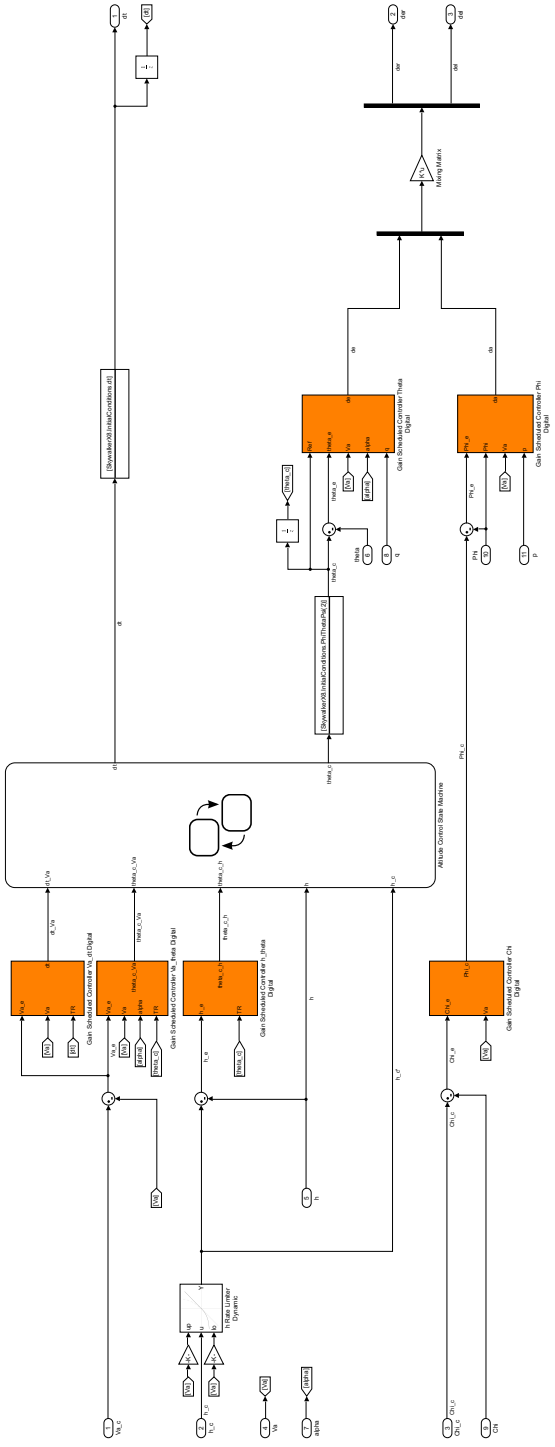


Figure A.33: Autopilot implementation

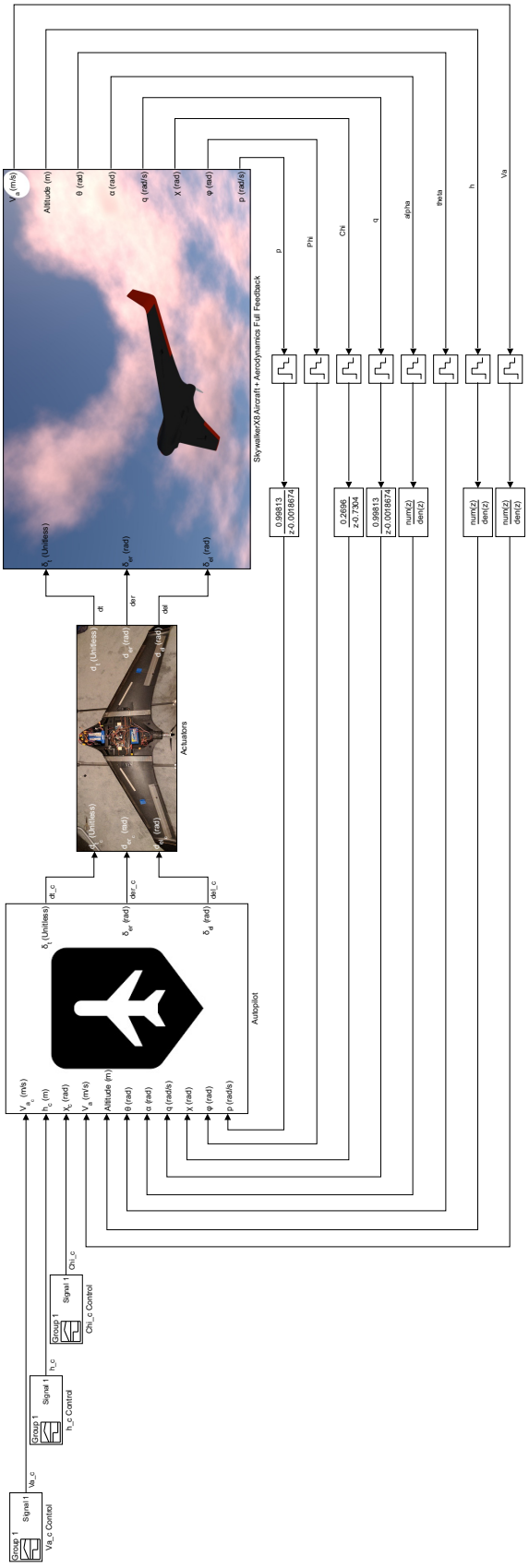


Figure A.34: Autopilot implementation verification

A.11 State Estimation

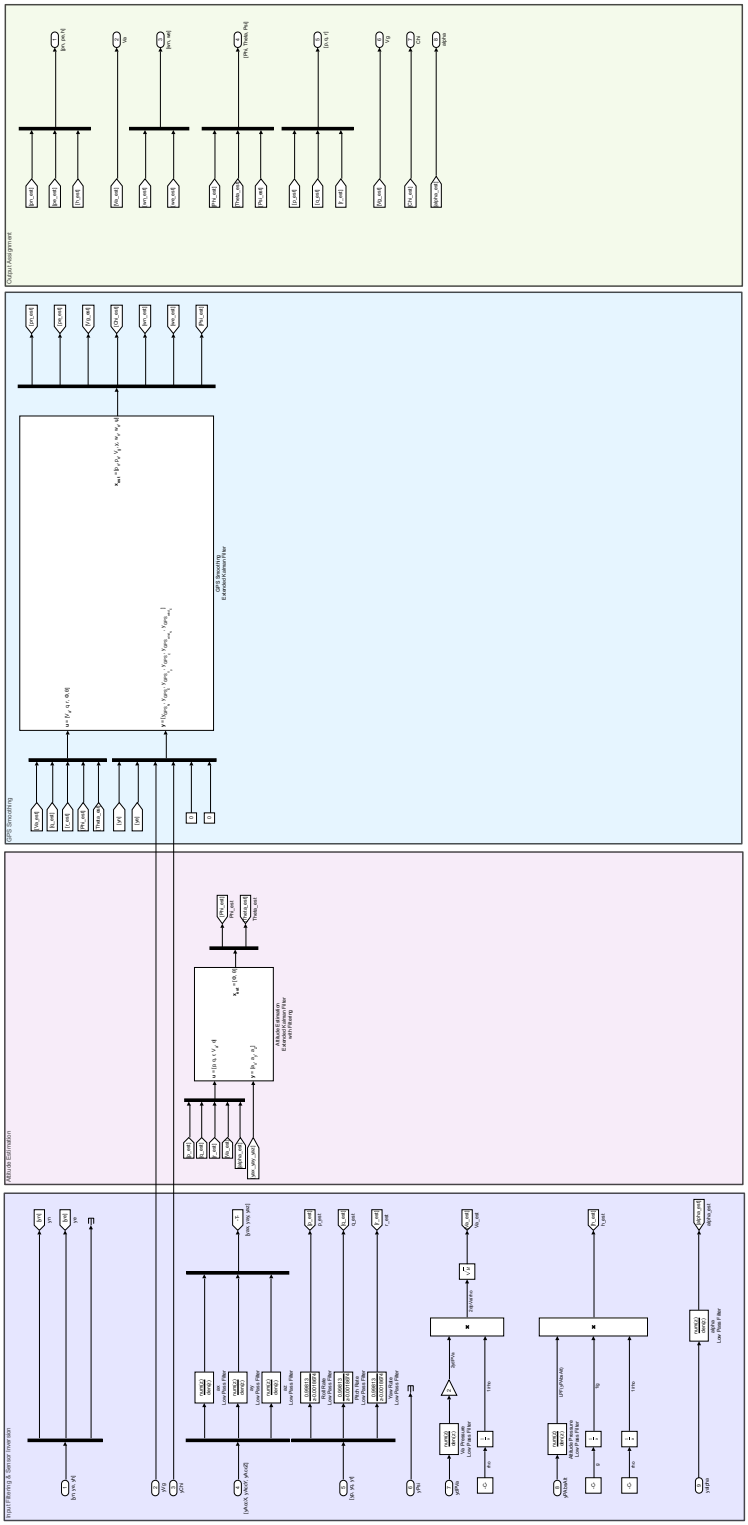


Figure A.35: State Estimation implementation

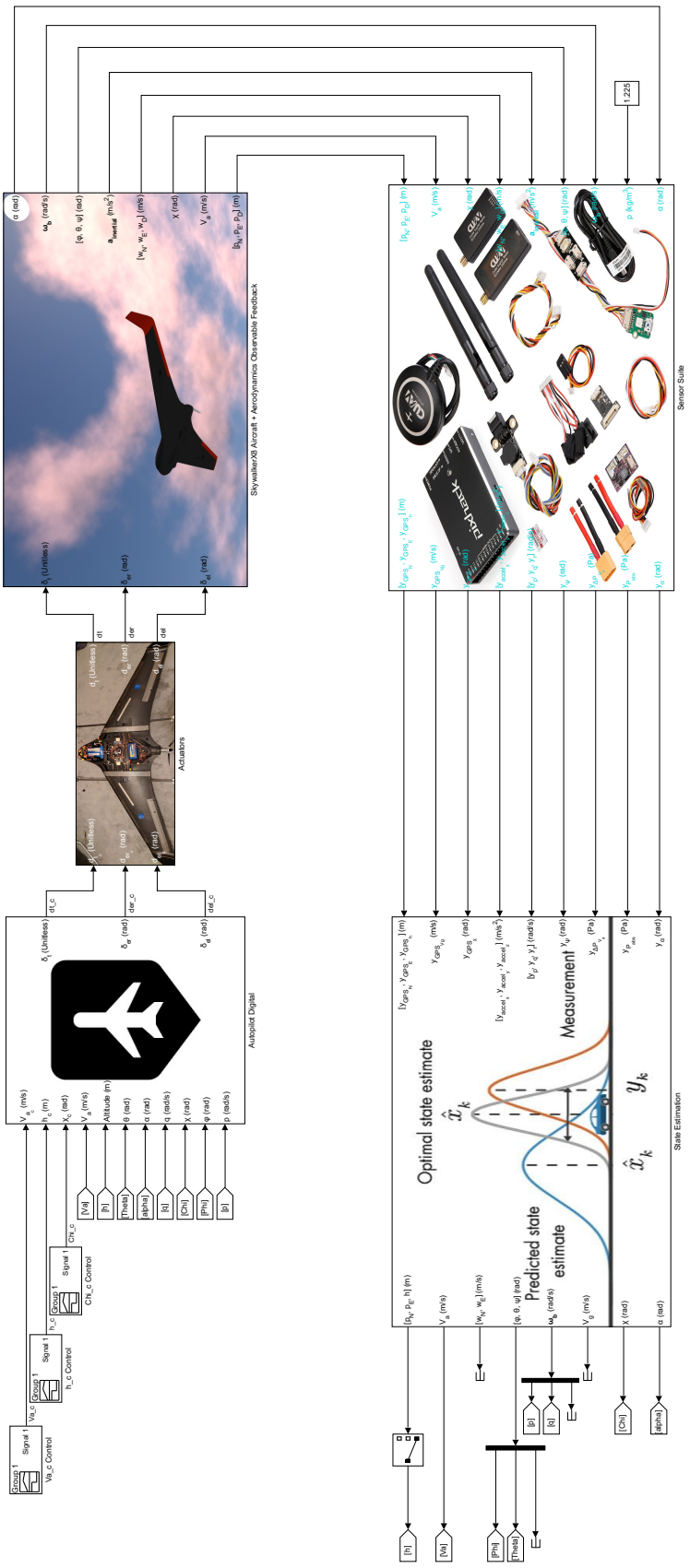


Figure A.36: State Estimation verification

A.12 Guidance and Navigation

A.12.1 Path Following

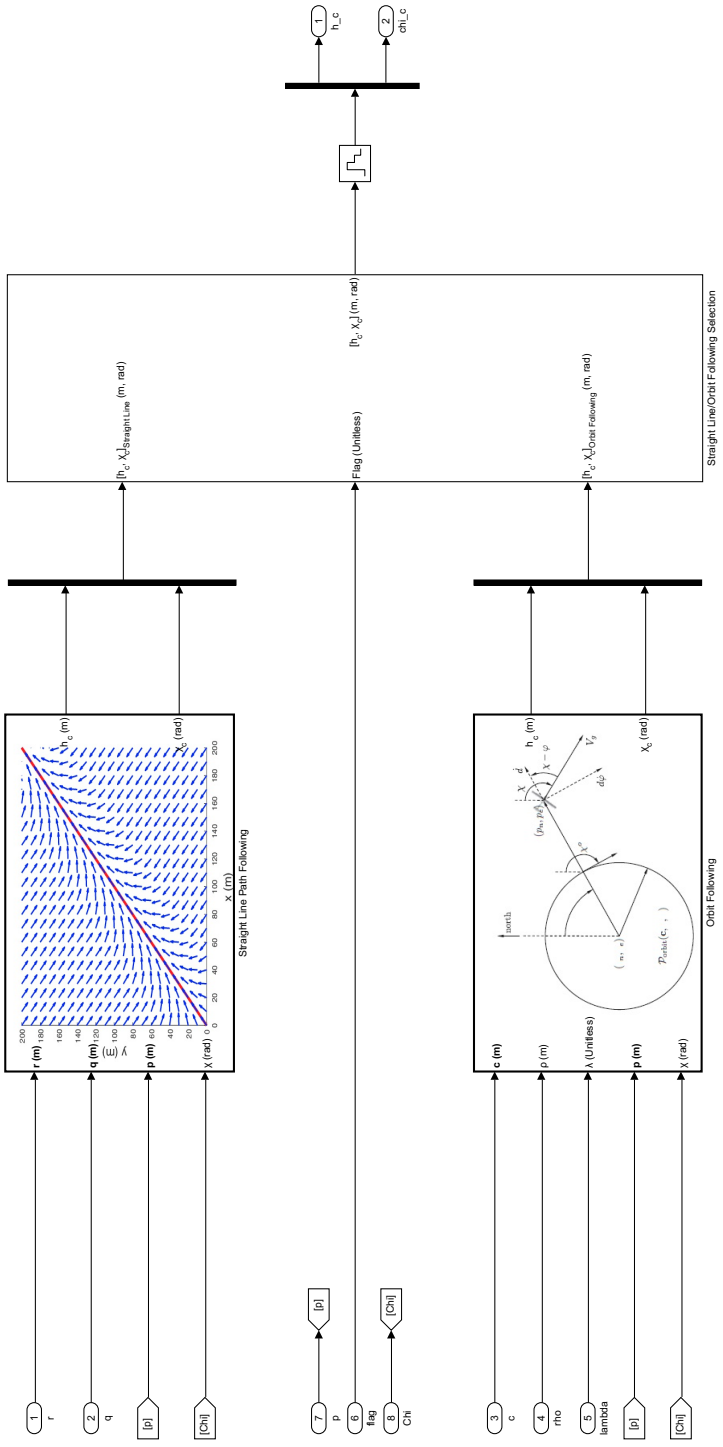


Figure A.37: Path following algorithm implementation

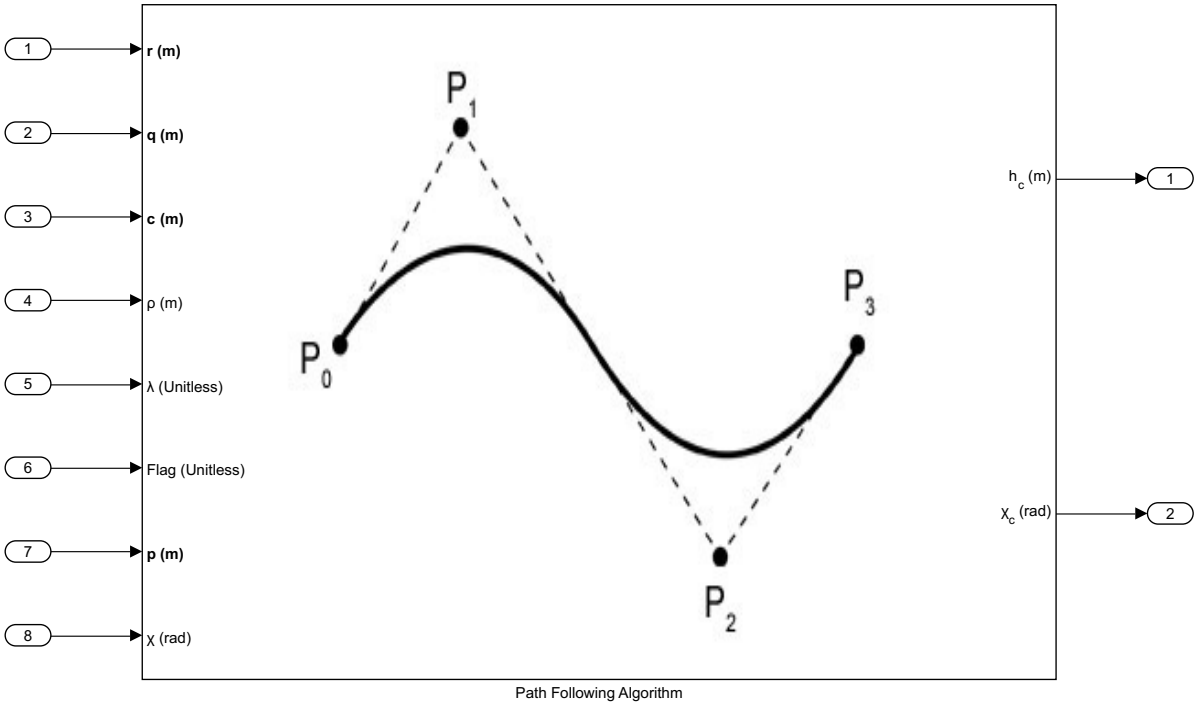


Figure A.38: Masked path following algorithm implementation

A.12.2 Path Management

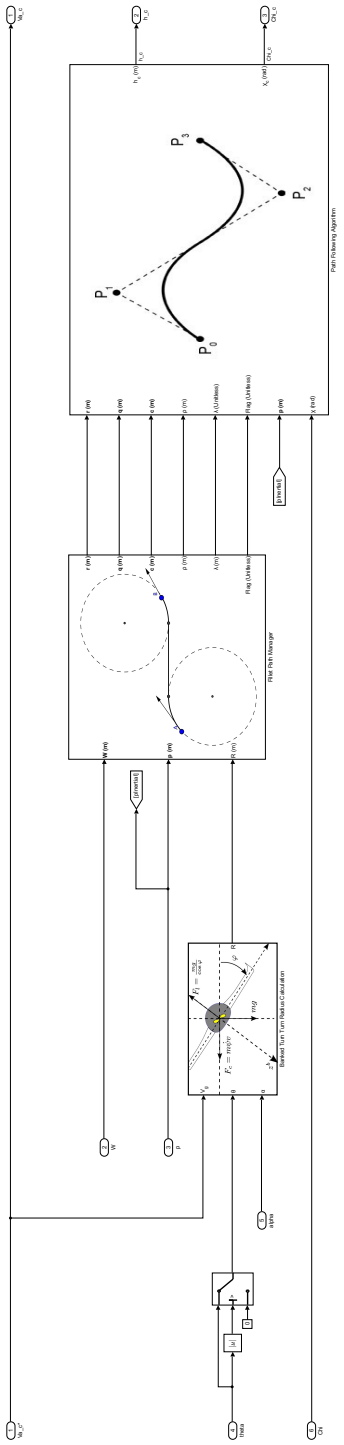


Figure A.39: Path management shown in the context of the full guidance and navigation system

A.12.3 Guidance and Navigation Verification

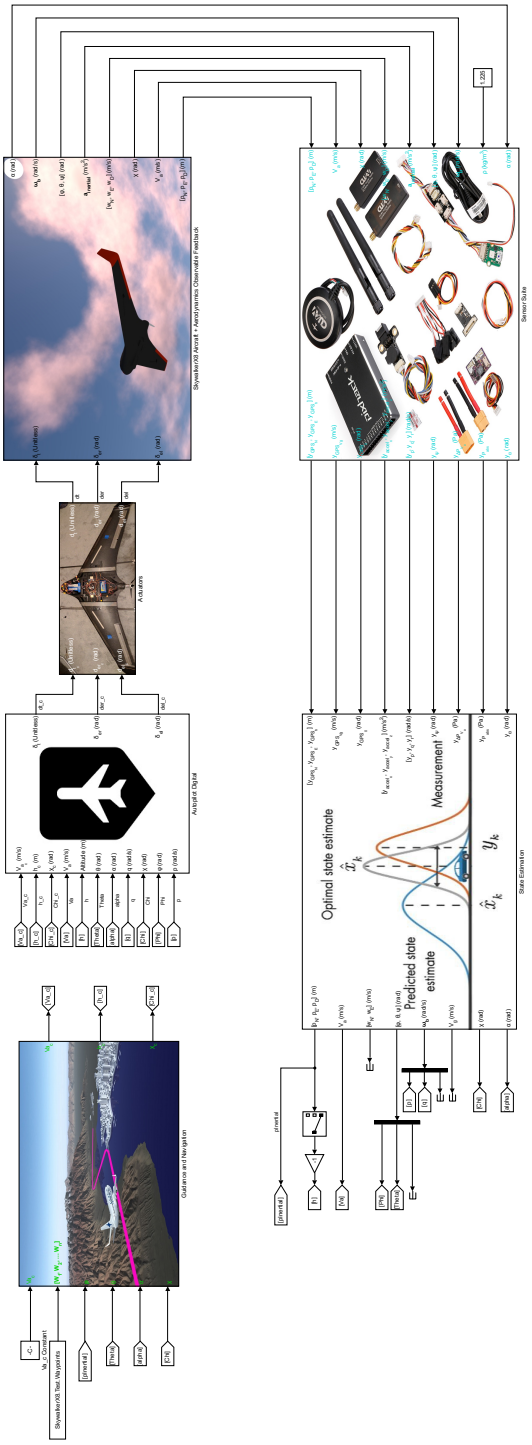


Figure A.40: Guidance and navigation verification model implementation

A.13 Guidance Navigation and Control Software in the Loop

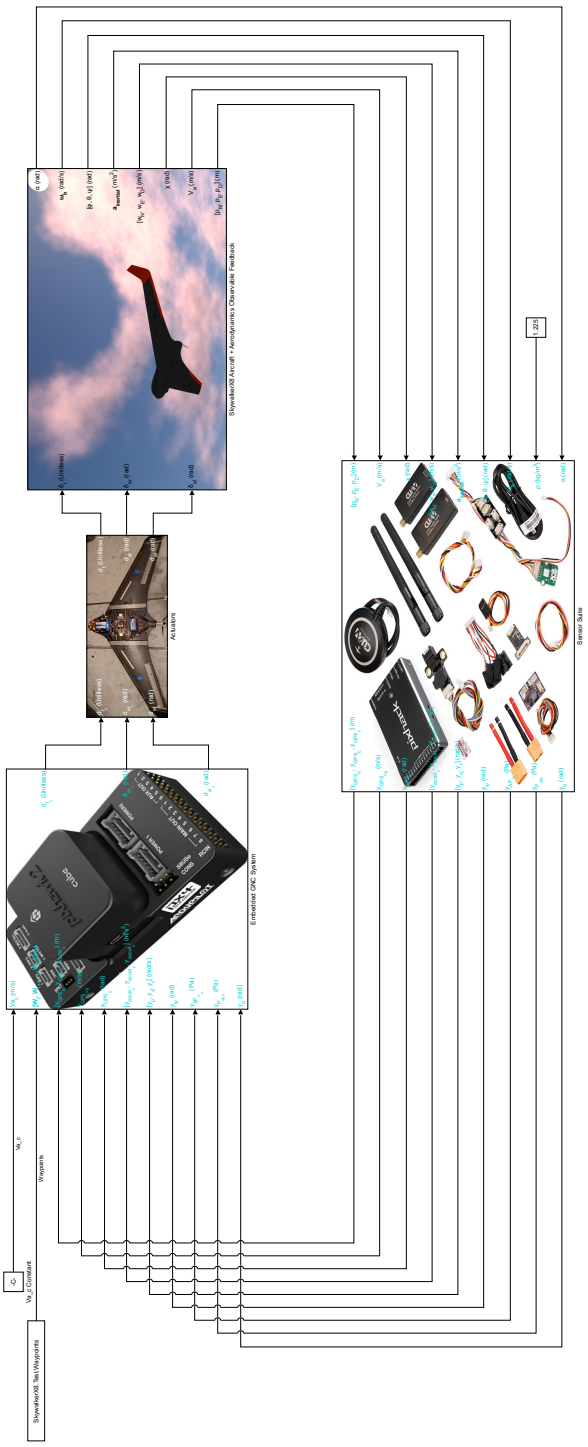


Figure A.41: Guidance, navigation and control implementation verification model

APPENDIX B

MATLAB Scripts

All MATLAB scripts are available in the scripts folder with the root directory found at [14].

B.1 Longitudinal Controller Tuning Goals

B.1.1 Pitch Angle Controller

Listing B.1: Pitch angle controller tuning goals, SkywalkerX8_Longitudinal_ControlDesign_VaAlpha.m

```
%% Theta Controller %%

% The next controller we'd like to tune will be the pitch ...
    controller.
% Thus, we create a 2D tunable surface and initialise the values ...
    based on
% sisotool at midpoint Va and alpha - keeping our gain limit in mind...
     (which
% will be calculated later on).

disp('Theta Control Loop Design Started')

% We can see from sisotool that having a PIDF controller is more ...
    beneficial
% than the normal PI controller. We get the following gains from ...
    sisotool
% with a midpoint controller (Va, alpha) = (Va(5), alpha(3)). We ...
    then do
% our tuning and update this with the tuned midpoint controller from...
     that
% (can be repeated as many times as the user would like) from time ...
    to time.

KpInit = 0.1096;
KiInit = 0.3107;
KdInit = -0.3911;
NInit = 0.3323;

Kp2DOFInit = -1.0415;
Kd2DOFInit = 0.8578;
N2DOFInit = 0.9124;

Kp = tunableSurface('Kp', KpInit, TuningGrid, ShapeFcn);
Ki = tunableSurface('Ki', KiInit, TuningGrid, ShapeFcn);
Kd = tunableSurface('Kd', KdInit, TuningGrid, ShapeFcn);
N = tunableSurface('N', NInit, TuningGrid, ShapeFcn);

Kp2DOF = tunableSurface('Kp2DOF', Kp2DOFInit, TuningGrid, ShapeFcn);
Kd2DOF = tunableSurface('Kd2DOF', Kd2DOFInit, TuningGrid, ShapeFcn);
N2DOF = tunableSurface('N2DOF', N2DOFInit, TuningGrid, ShapeFcn);

tunedBlocks = {'PIDF Controller theta Kp 2D Lookup Table',...
               'PIDF Controller theta Ki 2D Lookup Table',...
               'PIDF Controller theta Kd 2D Lookup Table',...
```

```
                'PIDF Controller theta N 2D Lookup Table',...
                'PDF Controller theta 2DOF Kp 2D Lookup Table',...
                'PDF Controller theta 2DOF Kd 2D Lookup Table',...
                'PDF Controller theta 2DOF N 2D Lookup Table'};

thetaPlantSubValue = [tf(0) SkywalkerX8.Control.Longitudinal...
                      .LinearizedPlantModels.De2VaReducedModels;...
                      tf(0) SkywalkerX8.Control.Longitudinal...
                      .LinearizedPlantModels.De2AltReducedModels;...
                      tf(0) SkywalkerX8.Control.Longitudinal...
                      .LinearizedPlantModels.De2ThetaReducedModels;...
                          ...
                      tf(0) tf(0);...
                      tf(0) SkywalkerX8.Control.Longitudinal...
                      .LinearizedPlantModels.De2qReducedModels];

thetaPlantSubName = plantSubName;

thetaPlantSub = struct('Name', thetaPlantSubName, 'Value', ...
    thetaPlantSubValue);

switchSub = [tf(0) tf(0) tf(0)];
algBreakSub = tf(1);

controllerVaDtName = [sys '/Gain Scheduled Controller Va_dt'];
controllerVaDtValue = [tf(0) tf(0)];

controllerVaDtSub = struct('Name', controllerVaDtName, 'Value', ...
    controllerVaDtValue);

controllerVaThetaName = [sys '/Gain Scheduled Controller Va_theta'];
controllerVaThetaValue = [tf(0) tf(0) tf(0)];

controllerVaThetaSub = struct('Name', controllerVaThetaName, 'Value'...
    , controllerVaThetaValue);

controllerHThetaName = [sys '/Gain Scheduled Controller h_theta'];
controllerHThetaValue = [tf(0) tf(0) tf(0)];

controllerHThetaSub = struct('Name', controllerHThetaName, 'Value', ...
    controllerHThetaValue);

controllerSwitchName = [sys '/Switch'];
controllerSwitchValue = switchSub;

controllerSwitchSub = struct('Name', controllerSwitchName, 'Value', ...
    controllerSwitchValue);

thetaClampSwitchName = [sys '/Gain Scheduled Controller Theta/PI ...
    Controller Theta/Clamp Switch'];
thetaClampSwitchValue = [tf(0) tf(0) tf(1)];

```

```
82 thetaClampSwitchSub = struct('Name', thetaClampSwitchName, 'Value', ...
       thetaClampSwitchValue);
83
84 algebraicLoopBreakValue = algBreakSub;
85
86 algebraicLoopBreak1Name = [sys '/Alg Break 1'];
87 algebraicLoopBreak2Name = [sys '/Alg Break 2'];
88 algebraicLoopBreak3Name = [sys '/Alg Break 3'];
89 algebraicLoopBreak4Name = [sys '/Alg Break 4'];
90 algebraicLoopBreak5Name = [sys '/Alg Break 5'];
91
92 algebraicLoopBreak1Sub = struct('Name', algebraicLoopBreak1Name, '...
       Value', algebraicLoopBreakValue);
93 algebraicLoopBreak2Sub = struct('Name', algebraicLoopBreak2Name, '...
       Value', algebraicLoopBreakValue);
94 algebraicLoopBreak3Sub = struct('Name', algebraicLoopBreak3Name, '...
       Value', algebraicLoopBreakValue);
95 algebraicLoopBreak4Sub = struct('Name', algebraicLoopBreak4Name, '...
       Value', algebraicLoopBreakValue);
96 algebraicLoopBreak5Sub = struct('Name', algebraicLoopBreak5Name, '...
       Value', algebraicLoopBreakValue);
97
98 BlockSubs = [thetaPlantSub;...
99              controllerVaDtSub;...
100             controllerVaThetaSub;...
101             controllerHThetaSub;...
102             controllerSwitchSub;...
103             thetaClampSwitchSub;...
104             algebraicLoopBreak1Sub;...
105             algebraicLoopBreak2Sub;...
106             algebraicLoopBreak3Sub;...
107             algebraicLoopBreak4Sub;...
108             algebraicLoopBreak5Sub];
109
110 % Create slTuner interface for tuning of tunedBlocks within the ...
       context of
111 % sys, after substituting out the blocks defined in BlockSubs, with ...
       the
112 % options given in LinOpt.
113
114 ST0 = slTuner(sys, tunedBlocks, BlockSubs, LinOpt);
115
116 ST0.Ts = 0; % We want continuous-time linearizations
117
118 % Add all relevant points of interest (POI) that we will examine in ...
       the
119 % model.
120
121 POI = {'SkywalkerX8_Longitudinal_Control/theta_c'
122        'SkywalkerX8_Longitudinal_Control/theta_e'
123        'SkywalkerX8_Longitudinal_Control/de'
```

```
124         'SkywalkerX8_Longitudinal_Control/Gain Scheduled Controller ...
               Theta/DF Controller Theta/Sum/de*_e'
125         'SkywalkerX8_Longitudinal_Control/Gain Scheduled Controller ...
               Theta/DF Controller Theta/Product1/Nde*_e'
126         'SkywalkerX8_Longitudinal_Control/Gain Scheduled Controller ...
               Theta/DF Controller Theta/Integrator/de*'
127         'SkywalkerX8_Longitudinal_Control/Switch/theta_c'
128         'SkywalkerX8_Longitudinal_Control/Gain Scheduled Controller ...
               Theta/de'
129         'SkywalkerX8_Longitudinal_Control/Gain Scheduled Controller ...
               Theta/PI Controller Theta/POut'
130         'SkywalkerX8_Longitudinal_Control/SkywalkerX8 Aircraft + ...
               Aerodynamics Longitudinal/Va'
131         'SkywalkerX8_Longitudinal_Control/SkywalkerX8 Aircraft + ...
               Aerodynamics Longitudinal/h'
132         'SkywalkerX8_Longitudinal_Control/SkywalkerX8 Aircraft + ...
               Aerodynamics Longitudinal/theta'
133         'SkywalkerX8_Longitudinal_Control/SkywalkerX8 Aircraft + ...
               Aerodynamics Longitudinal/alpha'
134         'SkywalkerX8_Longitudinal_Control/SkywalkerX8 Aircraft + ...
               Aerodynamics Longitudinal/q'};
135
136 ST0.addPoint(POI);
137
138 ST0.addOpening('theta_c');
139 ST0.addOpening('SkywalkerX8_Longitudinal_Control/SkywalkerX8 ...
       Aircraft + Aerodynamics Longitudinal/Va');
140 ST0.addOpening('SkywalkerX8_Longitudinal_Control/SkywalkerX8 ...
       Aircraft + Aerodynamics Longitudinal/h');
141 ST0.addOpening('SkywalkerX8_Longitudinal_Control/SkywalkerX8 ...
       Aircraft + Aerodynamics Longitudinal/alpha');
142
143 % We know we need to constrain what the maximum value can be for Kp ...
       due to
144 % actuator saturation.
145
146 % We'll make the assumption that theta will not exceed 60 degrees, ...
       and we
147 % know our maximum elevator deflection. We will assume our maximum
148 % generated command from the outer loop will not be an instant swing...
        - so
149 % we assume 50% of the expected maximum theta. We also take this
150 % opportunity to set our saturation limits.
151
152 thetaMax = 60*pi/180*0.5;
153 deMax = 2*15*pi/180; %x2 because our mixing matrix essentially ...
       halves this across 2 actuators
154
155 SkywalkerX8.Control.Longitudinal.thetaController...
156     .upperSaturationNegativeDe = deMax;
157 SkywalkerX8.Control.Longitudinal.thetaController...
158     .lowerSaturationNegativeDe = -deMax;
```

```
159
160 % so we can generate a
161 % limit on gain from these 2 values:
162
163 thetaGainLimit = deMax/thetaMax;
164
165 % and we assume this will only really be commanded if we are in trim...
      (ie.
166 % we don't need to double it because of going from - max to + max)
167
168 thetaPGainLimitInput = 'SkywalkerX8_Longitudinal_Control/Sum/theta_e...
    ';
169 thetaPGainLimitOutput = 'SkywalkerX8_Longitudinal_Control/Gain ...
    Scheduled Controller Theta/de';
170
171 R1 = TuningGoal.Gain(thetaPGainLimitInput, thetaPGainLimitOutput, ...
    thetaGainLimit);
172 R1.Focus = [1E-02 1E02]; %Only focus on this frequency band
173
174 % Next we know that we need to limit our derivative in terms of ...
    bandwidth.
175 % We do this with the associated filter (ie. we constrain the gain ...
    of "N",
176 % the F in the PIDF controller). This needs to be limited to <13 rad...
    /s, as
177 % this is the rate limit of the actuator.
178
179 NGainLimit = 13;
180
181 NGainLimitInput = 'SkywalkerX8_Longitudinal_Control/Gain Scheduled ...
    Controller Theta/DF Controller Theta/Sum/de*_e';
182 NGainLimitOutput = 'SkywalkerX8_Longitudinal_Control/Gain Scheduled ...
    Controller Theta/DF Controller Theta/Product1/Nde*_e';
183
184 NGainLimitLoopOpening = {'SkywalkerX8_Longitudinal_Control/Gain ...
    Scheduled Controller Theta/DF Controller Theta/Sum/de*_e',...
185                          'SkywalkerX8_Longitudinal_Control/Gain ...
                                Scheduled Controller Theta/DF Controller...
                                 Theta/Product1/Nde*_e'};
186
187 R2 = TuningGoal.Gain(NGainLimitInput, NGainLimitOutput, NGainLimit);
188 R2.Openings = NGainLimitLoopOpening;
189 R2.Focus = [1E-02 1E02]; %Only focus on this frequency band
190
191 % Next we know that our general response should have the standard ...
    gain
192 % margin of 6 dB, and a 40 deg phase margin should provide adequate ...
    damping,
193 % while still ensuring as rapid a response as possible (and around a...
     10% OS
194 % for a 2nd order system). So we assume this
195
```

```
196 gainMargin = 6; % >6 dB gain margin ideally
197 phaseMargin = 40; % 70 deg phase margin would provide adequate ...
       damping 1/sqrt(2) ≤ zeta < 1 - but we are fine with this being a ...
       bit lower in the interests of getting better response if possible
198 R3 = TuningGoal.Margins('SkywalkerX8_Longitudinal_Control/...
       SkywalkerX8 Aircraft + Aerodynamics Longitudinal/theta', ...
       gainMargin, phaseMargin);
199 R3.Focus = [1E-02 1E02]; %Only focus on this frequency band
200
201 % Next we know that we want our crossover frequency to be at ≤ 3 rad...
       /s, as
202 % per our linearization where we identified that our reduced order ...
       models
203 % are only accurate up until approximately 3 rad/s, so we try and ...
       cut off
204 % the plant prior to this so that our reduced order models are ...
       accurate and
205 % the inaccuracies are minimized by the rolloff. We also try and aim...
        for a
206 % 40 dB/decade rolloff after the crossover frequency and allow a ...
       tolerance
207 % of half a decade (ie. 2.5 -> 3.5 rad/s crossover) at the ...
       crossover.
208
209 LS = frd([100 1 0.0001],[0.03 3.0 300]);
210 R4 = TuningGoal.LoopShape('theta', LS, 0.5);
211
212 % Next we know that we want our altitude loop to be a simple first ...
       order
213 % lag. We also know that the desirable response time is what it ...
       would take
214 % for the system flying at its current airspeed (Va) at the maximum
215 % expected theta (assumed 60 degrees). As such, we want a variable ...
       tuning
216 % goal as a function of Va. We will slow this down by 5% in order to
217 % account for the need to first establish the desired theta (ie. Va ...
       at
218 % theta = 60 degrees would be our optimal solution assuming theta = ...
       60
219 % could be achieved instantly). We will make this specification for ...
       rise
220 % time, which is approximately 2.2*tau (or tau = trise/2.2)
221
222 % We also know the theta loop needs to be adequately far enough away...
        in
223 % bandwidth so that it has minimal impact on the altitude loop. As ...
       such, we
224 % specify a bandwidth separation in terms of a faster rise time. We ...
       also
225 % allow 5% overshoot because this is an internal loop so overshoot ...
       is not
226 % much of an issue for us.
```

```
227
228 altMax = 122*0.1; %Small signal design goal
229 tauScalar = 1.05;
230 riseTimeToTau = 1/2.2;
231 bandwidthSeparation = 20; %Small signal design goal
232 tauValues = zeros(nVa, nalpha);
233 OSValues = zeros(nVa, nalpha);
234
235 for i = 1:nVa
236
237     optimalTimeToAltitude = altMax/(...
          SkywalkerX8.Control.Longitudinal.SchedulingVariables.VaArray(...
          i)*sin(thetaMax));
238     tauValues(i, :) = optimalTimeToAltitude*tauScalar*riseTimeToTau/...
          bandwidthSeparation;
239     OSValues(i, :) = 5;
240
241 end
242
243 FH = @(tau, OS) TuningGoal.StepTracking('theta_c', 'theta', tau, OS)...
      ;
244
245 R5 = varyingGoal(FH, tauValues, OSValues);
246
247 thetac2DeGainLimit = deMax/thetaMax;
248
249 thetac2DeGainLimitInput = 'SkywalkerX8_Longitudinal_Control/Sum/...
      theta_e';
250 thetac2DeGainLimitOutput = 'SkywalkerX8_Longitudinal_Control/Gain ...
      Scheduled Controller Theta/PI Controller Theta/POut';
251
252 thetac2DeGainLimitLoopOpening = {thetac2DeGainLimitInput,...
253                                  thetac2DeGainLimitOutput};
254
255 R6 = TuningGoal.Gain(thetac2DeGainLimitInput, ...
      thetac2DeGainLimitOutput, thetac2DeGainLimit);
256 R6.Openings = thetac2DeGainLimitLoopOpening;
257 R6.Focus = [1E-02 1E02]; %Only focus on this frequency band
258
259 ST0.setBlockParam('PIDF Controller theta Kp 2D Lookup Table', Kp);
260 ST0.setBlockParam('PIDF Controller theta Ki 2D Lookup Table', Ki);
261 ST0.setBlockParam('PIDF Controller theta Kd 2D Lookup Table', Kd);
262 ST0.setBlockParam('PIDF Controller theta N 2D Lookup Table', N);
263
264 ST0.setBlockParam('PDF Controller theta 2DOF Kp 2D Lookup Table', ...
      Kp2DOF);
265 ST0.setBlockParam('PDF Controller theta 2DOF Kd 2D Lookup Table', ...
      Kd2DOF);
266 ST0.setBlockParam('PDF Controller theta 2DOF N 2D Lookup Table', ...
      N2DOF);
267
268 STTheta = systune(ST0, [R1, R3, R6], [R2, R4, R5], sysTuneOpts);
```

```
269
270 % Get values from ST and write them to our desired variables
271 % Note that TF here stands for transfer function and simply captures...
       the
272 % values with the correct structure so that later they can be used ...
       for
273 % block subs.
274
275 tableValues = getBlockValue(STTheta);
276 SkywalkerX8.Control.Longitudinal.thetaController.KpThetaTF = ...
       tableValues.PIDF_Controller_theta_Kp_2D_Lookup_Table;
277 SkywalkerX8.Control.Longitudinal.thetaController.KiThetaTF = ...
       tableValues.PIDF_Controller_theta_Ki_2D_Lookup_Table;
278 SkywalkerX8.Control.Longitudinal.thetaController.KdThetaTF = ...
       tableValues.PIDF_Controller_theta_Kd_2D_Lookup_Table;
279 SkywalkerX8.Control.Longitudinal.thetaController.NThetaTF = ...
       tableValues.PIDF_Controller_theta_N_2D_Lookup_Table;
280
281 SkywalkerX8.Control.Longitudinal.thetaController.KpTheta2DOFTF = ...
       tableValues.PDF_Controller_theta_2DOF_Kp_2D_Lookup_Table;
282 SkywalkerX8.Control.Longitudinal.thetaController.KdTheta2DOFTF = ...
       tableValues.PDF_Controller_theta_2DOF_Kd_2D_Lookup_Table;
283 SkywalkerX8.Control.Longitudinal.thetaController.NTheta2DOFTF = ...
       tableValues.PDF_Controller_theta_2DOF_N_2D_Lookup_Table;
284
285 SkywalkerX8.Control.Longitudinal.thetaController.KpTheta = squeeze(...
       tableValues.PIDF_Controller_theta_Kp_2D_Lookup_Table);
286 SkywalkerX8.Control.Longitudinal.thetaController.KiTheta = squeeze(...
       tableValues.PIDF_Controller_theta_Ki_2D_Lookup_Table);
287 SkywalkerX8.Control.Longitudinal.thetaController.KdTheta = squeeze(...
       tableValues.PIDF_Controller_theta_Kd_2D_Lookup_Table);
288 SkywalkerX8.Control.Longitudinal.thetaController.NTheta = squeeze(...
       tableValues.PIDF_Controller_theta_N_2D_Lookup_Table);
289
290 SkywalkerX8.Control.Longitudinal.thetaController.KpTheta2DOF = ...
       squeeze(tableValues.PDF_Controller_theta_2DOF_Kp_2D_Lookup_Table)...
       ;
291 SkywalkerX8.Control.Longitudinal.thetaController.KdTheta2DOF = ...
       squeeze(tableValues.PDF_Controller_theta_2DOF_Kd_2D_Lookup_Table)...
       ;
292 SkywalkerX8.Control.Longitudinal.thetaController.NTheta2DOF = ...
       squeeze(tableValues.PDF_Controller_theta_2DOF_N_2D_Lookup_Table);
293
294 % Get scheduling parameters and write them to our desired variables
295 % Note thase these are based on our basis function.
296
297 TGS = getBlockParam(STTheta);
298 SkywalkerX8.Control.Longitudinal.thetaController.BasisFunction = ...
       TGS.PIDF_Controller_theta_Kp_2D_Lookup_Table.BasisFunctions;
299 SkywalkerX8.Control.Longitudinal.thetaController...
300     .KpThetaParameterizedGains = getData(...
          TGS.PIDF_Controller_theta_Kp_2D_Lookup_Table);
```

```
SkywalkerX8.Control.Longitudinal.thetaController...
    .KiThetaParameterizedGains = getData(...
        TGS.PIDF_Controller_theta_Ki_2D_Lookup_Table);
SkywalkerX8.Control.Longitudinal.thetaController...
    .KdThetaParameterizedGains = getData(...
        TGS.PIDF_Controller_theta_Kd_2D_Lookup_Table);
SkywalkerX8.Control.Longitudinal.thetaController...
    .NThetaParameterizedGains = getData(...
        TGS.PIDF_Controller_theta_N_2D_Lookup_Table);

SkywalkerX8.Control.Longitudinal.thetaController...
    .KpTheta2DOFParameterizedGains = getData(...
        TGS.PDF_Controller_theta_2DOF_Kp_2D_Lookup_Table);
SkywalkerX8.Control.Longitudinal.thetaController...
    .KdTheta2DOFParameterizedGains = getData(...
        TGS.PDF_Controller_theta_2DOF_Kd_2D_Lookup_Table);
SkywalkerX8.Control.Longitudinal.thetaController...
    .NTheta2DOFParameterizedGains = getData(...
        TGS.PDF_Controller_theta_2DOF_N_2D_Lookup_Table);

disp('Theta Control Loop Design Complete')
```

B.1.2 Altitude Controller

Listing B.2: Altitude controller tuning goals, SkywalkerX8_Longitudinal_ControlDesign_VaAlpha.m

```
%% Alt Control Outer Loop %%

disp('Alt Control Loop Design Started')

%We get our initial values from siso tool with a midpoint controller...
    (Va,
%alpha) = (Va(5), alpha(3)). We also know that our reduced model is
%essentially a scaled integration based on Va and our current theta....
    As
%such, we can simplify this controller drastically to a simple P ...
   controller
%and then constrain our gain to limit actuator saturation.

KpInit = 0.0168;
KiInit = 0.005;
KdInit = -0.0263;
NInit = 0.1880;

Kp = tunableSurface('Kp', KpInit, TuningGrid, ShapeFcn);
Ki = tunableSurface('Ki', KiInit, TuningGrid, ShapeFcn);
Kd = tunableSurface('Kd', KdInit, TuningGrid, ShapeFcn);
N = tunableSurface('N', NInit, TuningGrid, ShapeFcn);

```

```
tunedBlocks = {'PIDF Controller h_theta Kp 2D Lookup Table',...
               'PIDF Controller h_theta Ki 2D Lookup Table',...
               'PIDF Controller h_theta Kd 2D Lookup Table',...
               'PIDF Controller h_theta N 2D Lookup Table'};

% By ensuring that we keep our bandwidth lower than 2-3 x that of ...
    the theta
% controller we can trivialise our plant to De2Alt/De2Theta (ie. ...
    Theta to
% Alt) because we can assume theta_c to theta is unity gain.

% We can get theta_c to alt as the sub model and then we can simply ...
    replace
% the theta controller with feedthrough an open up q and theta. This...
     is
% better.

minRealTolerance = 1E-05;

%While getting the transfer function from theta_c to alt is ...
    certainly
%possible, it contains higher order dynamics we don't really care ...
    about. As
%such, we use the simplified linear model of Va/s when looking at ...
    theta to
%altitude and use this as the plant sub. The intention here is to ...
    make
%tuning easier.

thetaCToAlt = minreal(tf(getIOTransfer(STTheta, 'theta_c', 'h')), ...
    minRealTolerance, false);

thetaCToAltEstimate = idss(zeros(1, 1, nVa, nalpha));

for i = 1:nVa

    thetaCToAltEstimate(:, :, i, :) =  tf(...
        SkywalkerX8.Control.Longitudinal.SchedulingVariables.VaArray(...
        i), [1 0]);

end

altPlantSubValue = [tf(0) tf(0);...
                    tf(0) thetaCToAltEstimate;...
                    tf(0) tf(0);...
                    tf(0) tf(0);...
                    tf(0) tf(0)];

altPlantSubName = plantSubName;

altPlantSub = struct('Name', altPlantSubName, 'Value', ...
    altPlantSubValue);
```

```
61
62 controllerThetaSubValue = [tf(0),...
63                          tf(1),...
64                          tf(0),...
65                          tf(0),...
66                          tf(0)];
67
68 controllerThetaSubName = [sys '/Gain Scheduled Controller Theta'];
69
70 controllerThetaSub = struct('Name', controllerThetaSubName, 'Value',...
       controllerThetaSubValue);
71
72 switchSub = [tf(0) tf(0) tf(1)]; %This changes from previous because...
       we now want our theta_alt controller to go through
73 algBreakSub = tf(1);
74
75 controllerVaDtName = [sys '/Gain Scheduled Controller Va_dt'];
76 controllerVaDtValue = [tf(0) tf(0)];
77
78 controllerVaDtSub = struct('Name', controllerVaDtName, 'Value', ...
       controllerVaDtValue);
79
80 controllerVaThetaName = [sys '/Gain Scheduled Controller Va_theta'];
81 controllerVaThetaValue = [tf(0) tf(0) tf(0)];
82
83 controllerVaThetaSub = struct('Name', controllerVaThetaName, 'Value'...
       , controllerVaThetaValue);
84
85 controllerSwitchName = [sys '/Switch'];
86 controllerSwitchValue = switchSub;
87
88 controllerSwitchSub = struct('Name', controllerSwitchName, 'Value', ...
       controllerSwitchValue);
89
90 altClampSwitchName = [sys '/Gain Scheduled Controller h_theta/PIDF ...
       Controller h_theta/Clamp Switch'];
91 altClampSwitchValue = [tf(0) tf(0) tf(1)];
92
93 altClampSwitchSub = struct('Name', altClampSwitchName, 'Value', ...
       altClampSwitchValue);
94
95 algebraicLoopBreakValue = algBreakSub;
96
97 algebraicLoopBreak1Name = [sys '/Alg Break 1'];
98 algebraicLoopBreak2Name = [sys '/Alg Break 2'];
99 algebraicLoopBreak3Name = [sys '/Alg Break 3'];
100 algebraicLoopBreak4Name = [sys '/Alg Break 4'];
101 algebraicLoopBreak5Name = [sys '/Alg Break 5'];
102
103 algebraicLoopBreak1Sub = struct('Name', algebraicLoopBreak1Name, '...
       Value', algebraicLoopBreakValue);
```

```
algebraicLoopBreak2Sub = struct('Name', algebraicLoopBreak2Name, '...
    Value', algebraicLoopBreakValue);
algebraicLoopBreak3Sub = struct('Name', algebraicLoopBreak3Name, '...
    Value', algebraicLoopBreakValue);
algebraicLoopBreak4Sub = struct('Name', algebraicLoopBreak4Name, '...
    Value', algebraicLoopBreakValue);
algebraicLoopBreak5Sub = struct('Name', algebraicLoopBreak5Name, '...
    Value', algebraicLoopBreakValue);

BlockSubs = [altPlantSub;...
             controllerThetaSub;...
             controllerVaDtSub;...
             controllerVaThetaSub;...
             controllerSwitchSub;...
             altClampSwitchSub;...
             algebraicLoopBreak1Sub;...
             algebraicLoopBreak2Sub;...
             algebraicLoopBreak3Sub;...
             algebraicLoopBreak4Sub;...
             algebraicLoopBreak5Sub];

% Create slTuner interface for tuning of tunedBlocks within the ...
    context of
% sys, after substituting out the blocks defined in BlockSubs, with ...
    the
% options given in LinOpt.

ST0 = slTuner(sys, tunedBlocks, BlockSubs, LinOpt);

ST0.Ts = 0; % We want continuous-time linearizations

% Add all relevant points of interest (POI) that we will examine in ...
    the
% model.

POI = {'SkywalkerX8_Longitudinal_Control/h_c'
       'SkywalkerX8_Longitudinal_Control/h_e'
       'SkywalkerX8_Longitudinal_Control/theta_c_h'
       'SkywalkerX8_Longitudinal_Control/de'
       'SkywalkerX8_Longitudinal_Control/Gain Scheduled Controller ...
          h_theta/PIDF Controller h_theta/Sum/theta_c*_e'
       'SkywalkerX8_Longitudinal_Control/Gain Scheduled Controller ...
          h_theta/PIDF Controller h_theta/Product1/theta_c*'
       'SkywalkerX8_Longitudinal_Control/Gain Scheduled Controller ...
          h_theta/PIDF Controller h_theta/POut'
       'SkywalkerX8_Longitudinal_Control/Gain Scheduled Controller ...
          h_theta/PIDF Controller h_theta/IOut/1'
       'SkywalkerX8_Longitudinal_Control/SkywalkerX8 Aircraft + ...
          Aerodynamics Longitudinal/Va'
       'SkywalkerX8_Longitudinal_Control/SkywalkerX8 Aircraft + ...
          Aerodynamics Longitudinal/h'
```

```
142         'SkywalkerX8_Longitudinal_Control/SkywalkerX8 Aircraft + ...
               Aerodynamics Longitudinal/theta'
143         'SkywalkerX8_Longitudinal_Control/SkywalkerX8 Aircraft + ...
               Aerodynamics Longitudinal/alpha'
144         'SkywalkerX8_Longitudinal_Control/SkywalkerX8 Aircraft + ...
               Aerodynamics Longitudinal/q'};
145
146 ST0.addPoint(POI);
147
148 ST0.addOpening('SkywalkerX8_Longitudinal_Control/SkywalkerX8 ...
       Aircraft + Aerodynamics Longitudinal/Va');
149 ST0.addOpening('SkywalkerX8_Longitudinal_Control/SkywalkerX8 ...
       Aircraft + Aerodynamics Longitudinal/theta');
150 ST0.addOpening('SkywalkerX8_Longitudinal_Control/SkywalkerX8 ...
       Aircraft + Aerodynamics Longitudinal/alpha');
151 ST0.addOpening('SkywalkerX8_Longitudinal_Control/SkywalkerX8 ...
       Aircraft + Aerodynamics Longitudinal/q');
152
153 % We know we need to constrain what the maximum value can be for Kp ...
       due to
154 % actuator saturation.
155
156 % We'll make the assumption that theta will not exceed 60 degrees, ...
       and we
157 % know our maximum altitude is 122 m by law - so let's assume 10% of...
        that
158 % will be the largest seen error signal (small signal). We also take...
        this
159 % opportunity to set our saturation limits.
160
161 thetaMax = 60*pi/180;
162 altMax = 122*0.1;
163
164 SkywalkerX8.Control.Longitudinal.altController.upperSaturationThetaC...
        = thetaMax;
165 SkywalkerX8.Control.Longitudinal.altController.lowerSaturationThetaC...
        = -thetaMax;
166
167 % so we can generate a
168 % limit on gain from these 2 values:
169
170 altGainLimit = thetaMax/altMax;
171
172 % and we assume this will only really be commanded if we are in trim
173
174 altPGainLimitInput = 'SkywalkerX8_Longitudinal_Control/Sum2/h_e';
175 altPGainLimitOutput = 'SkywalkerX8_Longitudinal_Control/Gain ...
       Scheduled Controller h_theta/PIDF Controller h_theta/POut';
176
177 R1 = TuningGoal.Gain(altPGainLimitInput, altPGainLimitOutput, ...
       altGainLimit);
178 R1.Openings = {altPGainLimitInput,...
```

```
179                 altPGainLimitOutput};
180 R1.Focus = [1E-02 1E02];
181
182 % Next we know that our general response should have the standard ...
      gain
183 % margin of 6 dB, and a 60 deg phase margin forces a response that ...
      does not
184 % have a resonance peak, but we accept slightly lower in the ...
      interest of
185 % better performance,
186 % while still ensuring as rapid a response as possible (and around a...
       5% OS
187 % for a 2nd order system). So we assume this
188
189 gainMargin = 6; % >6 dB gain margin ideally
190 phaseMargin = 45; % 45 deg phase margin would provide adequate ...
      damping
191 R2 = TuningGoal.Margins('SkywalkerX8_Longitudinal_Control/...
      SkywalkerX8 Aircraft + Aerodynamics Longitudinal/h', gainMargin, ...
      phaseMargin);
192 R2.Focus = [1E-02 1E02];
193
194 % Next we know that we want our crossover frequency to be at 1.0 rad...
      /s, as
195 % from our previous design we know our cutoff frequency was at
196 % approximately 10 rad/s, so we can reduce the inner-loop to a unity...
       gain if
197 % we band limit this outer loop to 1/10th the cutoff frequency of ...
      the inner
198 % loop.
199
200 LS = frd([1000000 1 0.0001],[0.01 1.0 100]);
201 R3 = TuningGoal.LoopShape('h', LS, 0.5);
202
203 % Next we know that we want our altitude loop to be a simple first ...
      order
204 % lag. We also know that the desirable response time is what it ...
      would take
205 % for the system flying at its current airspeed (Va) at the maximum
206 % expected theta (assumed 60 degrees). As such, we want a variable ...
      tuning
207 % goal as a function of Va. We will slow this down by 5% in order to
208 % account for the need to first establish the desired theta (ie. Va ...
      at
209 % theta = 60 degrees would be our optimal solution assuming theta = ...
      60
210 % could be achieved instantly). We will make this specification for ...
      rise
211 % time, which is approximately 2.2*tau (or tau = trise/2.2)
212
213 tauScalar = 1.05;
214 riseTimeToTau = 1/2.2;
```

```
215 bandwidthSeparation = 2;
216 tauValues = zeros(nVa, nalpha);
217 OSValues = zeros(nVa, nalpha);
218
219 for i = 1:nVa
220
221     optimalTimeToAltitude = altMax/(...
            SkywalkerX8.Control.Longitudinal.SchedulingVariables.VaArray(...
            i)*sin(thetaMax));
222     tauValues(i, :) = optimalTimeToAltitude*tauScalar*riseTimeToTau/...
            bandwidthSeparation;
223     OSValues(i, :) = 0; %0% overshoot is ideal if possible
224
225 end
226
227 FH = @(tau, OS) TuningGoal.StepTracking('h_c', 'h', tau, OS);
228
229 R4 = varyingGoal(FH, tauValues, OSValues);
230
231 altFFControllerPGainLimitInput = 'SkywalkerX8_Longitudinal_Control/...
        h_c';
232 altFFControllerPGainLimitOutput = 'SkywalkerX8_Longitudinal_Control/...
        Gain Scheduled Controller h_theta/theta_c_h';
233
234 R5 = TuningGoal.Gain(altFFControllerPGainLimitInput, ...
        altFFControllerPGainLimitOutput, altGainLimit);
235 R5.Openings = 'SkywalkerX8_Longitudinal_Control/SkywalkerX8 Aircraft...
         + Aerodynamics Longitudinal/h';
236 R5.Focus = [1E-02 1E02];
237
238 % Next we know that we need to limit our derivative in terms of ...
        bandwidth.
239 % We do this with the associated filter (ie. we constrain the gain ...
        of "N",
240 % the F in the PIDF controller). This needs to be limited to ...
        approximately
241 % 6.5 rad/s (half of the theta control loop filter) to not cause ...
        issues with the inner loop.
242
243 NGainLimit = 6.5;
244
245 NGainLimitInput = 'SkywalkerX8_Longitudinal_Control/Gain Scheduled ...
        Controller h_theta/PIDF Controller h_theta/Sum/theta_c*_e';
246 NGainLimitOutput = 'SkywalkerX8_Longitudinal_Control/Gain Scheduled ...
        Controller h_theta/PIDF Controller h_theta/Product1/theta_c*';
247
248 NGainLimitLoopOpening = {'SkywalkerX8_Longitudinal_Control/Gain ...
        Scheduled Controller h_theta/PIDF Controller h_theta/Sum/theta_c*...
        _e',...
249                          'SkywalkerX8_Longitudinal_Control/Gain ...
                                Scheduled Controller h_theta/PIDF ...
                                Controller h_theta/Product1/theta_c*'};
```

```
250
251 R6 = TuningGoal.Gain(NGainLimitInput, NGainLimitOutput, NGainLimit);
252 R6.Openings = NGainLimitLoopOpening;
253 R6.Focus = [1E-02 1E02]; %Only focus on this frequency band
254
255 ST0.setBlockParam('PIDF Controller h_theta Kp 2D Lookup Table', Kp);
256 ST0.setBlockParam('PIDF Controller h_theta Ki 2D Lookup Table', Ki);
257 ST0.setBlockParam('PIDF Controller h_theta Kd 2D Lookup Table', Kd);
258 ST0.setBlockParam('PIDF Controller h_theta N 2D Lookup Table', N);
259
260 STAlt = systune(ST0, [R1, R2, R5], [R3, R4, R6], sysTuneOpts);
261
262 % Get values from ST and write them to our desired variables
263 % Note that TF here stands for transfer function and simply captures...
       the
264 % values with the correct structure so that later they can be used ...
      for
265 % block subs.
266
267 tableValues = getBlockValue(STAlt);
268
269 SkywalkerX8.Control.Longitudinal.altController.KpAltTF = ...
      tableValues.PIDF_Controller_h_theta_Kp_2D_Lookup_Table;
270 SkywalkerX8.Control.Longitudinal.altController.KiAltTF = ...
      tableValues.PIDF_Controller_h_theta_Ki_2D_Lookup_Table;
271 SkywalkerX8.Control.Longitudinal.altController.KdAltTF = ...
      tableValues.PIDF_Controller_h_theta_Kd_2D_Lookup_Table;
272 SkywalkerX8.Control.Longitudinal.altController.NAltTF = ...
      tableValues.PIDF_Controller_h_theta_N_2D_Lookup_Table;
273
274 SkywalkerX8.Control.Longitudinal.altController.KpAlt = squeeze(...
      tableValues.PIDF_Controller_h_theta_Kp_2D_Lookup_Table);
275 SkywalkerX8.Control.Longitudinal.altController.KiAlt = squeeze(...
      tableValues.PIDF_Controller_h_theta_Ki_2D_Lookup_Table);
276 SkywalkerX8.Control.Longitudinal.altController.KdAlt = squeeze(...
      tableValues.PIDF_Controller_h_theta_Kd_2D_Lookup_Table);
277 SkywalkerX8.Control.Longitudinal.altController.NAlt = squeeze(...
      tableValues.PIDF_Controller_h_theta_N_2D_Lookup_Table);
278
279 % Get scheduling parameters and write them to our desired variables
280 % Note thase these are based on our basis function.
281
282 TGS = getBlockParam(STAlt);
283 SkywalkerX8.Control.Longitudinal.altController.BasisFunction = ...
      TGS.PIDF_Controller_h_theta_Kp_2D_Lookup_Table.BasisFunctions;
284 SkywalkerX8.Control.Longitudinal.altController.KpAltParameterizedGains...
       = getData(TGS.PIDF_Controller_h_theta_Kp_2D_Lookup_Table);
285 SkywalkerX8.Control.Longitudinal.altController.KiAltParameterizedGains...
       = getData(TGS.PIDF_Controller_h_theta_Ki_2D_Lookup_Table);
286 SkywalkerX8.Control.Longitudinal.altController.KdAltParameterizedGains...
       = getData(TGS.PIDF_Controller_h_theta_Kd_2D_Lookup_Table);
```

```
287 SkywalkerX8.Control.Longitudinal.altController.NAltParameterizedGains...
        = getData(TGS.PIDF_Controller_h_theta_N_2D_Lookup_Table);
288
289 % Next we define our tracking constant (not used for controller ...
      design, but
290 % provided by the designer to assist with bumpless transfer in
291 % implementation).
292
293 % This Is defined relative to our Ki gain via a scalar. As such, it ...
      should
294 % be x times faster than this integrator loop. 10 seems like a ...
      reasonable
295 % value.
296
297 SkywalkerX8.Control.Longitudinal.altController.KtScalar = 10;
298
299 disp('Alt Control Loop Design Complete')
```

B.1.3 Airspeed Controller - Propeller Thrust

Listing B.3: Airspeed controller, using propeller speed, tuning goals, SkywalkerX8_Longitudinal_ControlDesign_Va.m

```
1  disp('Starting Va_dt Control Loop Design...')
2
3  KpInit = 0.1818;
4  KiInit = 0.0921;
5  KdInit = -0.0313;
6  NInit = 4.6266;
7
8  Kp = tunableSurface('Kp', KpInit, VaTuningGrid, VaShapeFcn);
9  Ki = tunableSurface('Ki', KiInit, VaTuningGrid, VaShapeFcn);
10 Kd = tunableSurface('Kd', KdInit, VaTuningGrid, VaShapeFcn);
11 N = tunableSurface('N', NInit, VaTuningGrid, VaShapeFcn);
12
13 tunedBlocks = {'PIDF Controller Va_dt Kp Lookup Table',...
14                 'PIDF Controller Va_dt Ki Lookup Table',...
15                 'PIDF Controller Va_dt Kd Lookup Table',...
16                 'PIDF Controller Va_dt N Lookup Table'};
17
18 minRealTolerance = 1E-05;
19
20 VaDtPlantSubValue = [SkywalkerX8.Control.Longitudinal...
21                      .LinearizedPlantModels.Dt2VaLinearizedModels tf...
                            (0);...
22                      tf(0) tf(0);...
23                      tf(0) tf(0);...
24                      tf(0) tf(0);...
25                      tf(0) tf(0)];
26
```

```
27 VaDtPlantSubName = plantSubName;
28
29 VaDtPlantSub = struct('Name', VaDtPlantSubName, 'Value', ...
       VaDtPlantSubValue);
30
31 controllerThetaSubValue = [tf(0),...
32                            tf(0),...
33                            tf(0),...
34                            tf(0),...
35                            tf(0)];
36
37 controllerThetaSubName = [sys '/Gain Scheduled Controller Theta'];
38
39 controllerThetaSub = struct('Name', controllerThetaSubName, 'Value',...
       controllerThetaSubValue);
40
41 controllerSub = [tf(0) tf(0) tf(0)];
42 switchSub = [tf(0) tf(0) tf(0)]; %This changes from previous because...
       we now want our theta_alt controller to go through
43 algBreakSub = tf(1);
44
45 controllerVaThetaName = [sys '/Gain Scheduled Controller Va_theta'];
46 controllerVaThetaValue = controllerSub;
47
48 controllerVaThetaSub = struct('Name', controllerVaThetaName, 'Value'...
       , controllerVaThetaValue);
49
50 controllerSwitchName = [sys '/Switch'];
51 controllerSwitchValue = switchSub;
52
53 controllerSwitchSub = struct('Name', controllerSwitchName, 'Value', ...
       controllerSwitchValue);
54
55 VaDtClampSwitchName = [sys '/Gain Scheduled Controller Va_dt/PIDF ...
       Controller Va_dt/Clamp Switch'];
56 VaDtClampSwitchValue = [tf(0) tf(0) tf(1)];
57
58 VaDtClampSwitchSub = struct('Name', VaDtClampSwitchName, 'Value', ...
       VaDtClampSwitchValue);
59
60 algebraicLoopBreakValue = algBreakSub;
61
62 algebraicLoopBreak1Name = [sys '/Alg Break 1'];
63 algebraicLoopBreak2Name = [sys '/Alg Break 2'];
64 algebraicLoopBreak3Name = [sys '/Alg Break 3'];
65 algebraicLoopBreak4Name = [sys '/Alg Break 4'];
66 algebraicLoopBreak5Name = [sys '/Alg Break 5'];
67
68 algebraicLoopBreak1Sub = struct('Name', algebraicLoopBreak1Name, '...
       Value', algebraicLoopBreakValue);
69 algebraicLoopBreak2Sub = struct('Name', algebraicLoopBreak2Name, '...
       Value', algebraicLoopBreakValue);
```

```
algebraicLoopBreak3Sub = struct('Name', algebraicLoopBreak3Name, '...
    Value', algebraicLoopBreakValue);
algebraicLoopBreak4Sub = struct('Name', algebraicLoopBreak4Name, '...
    Value', algebraicLoopBreakValue);
algebraicLoopBreak5Sub = struct('Name', algebraicLoopBreak5Name, '...
    Value', algebraicLoopBreakValue);

BlockSubs = [VaDtPlantSub;...
             controllerThetaSub;...
             controllerVaThetaSub;...
             controllerSwitchSub;...
             VaDtClampSwitchSub;...
             algebraicLoopBreak1Sub;...
             algebraicLoopBreak2Sub;...
             algebraicLoopBreak3Sub;...
             algebraicLoopBreak4Sub;...
             algebraicLoopBreak5Sub];

% Create slTuner interface for tuning of tunedBlocks within the ...
    context of
% sys, after substituting out the blocks defined in BlockSubs, with ...
    the
% options given in LinOpt.

ST0 = slTuner(sys, tunedBlocks, BlockSubs, LinOpt);

ST0.Ts = 0; % We want continuous-time linearizations

% Add all relevant points of interest (POI) that we will examine in ...
    the
% model.

POI = {'SkywalkerX8_Longitudinal_Control/Va_c'
       'SkywalkerX8_Longitudinal_Control/Va_e'
       'SkywalkerX8_Longitudinal_Control/dt'
       'SkywalkerX8_Longitudinal_Control/SkywalkerX8 Aircraft + ...
          Aerodynamics Longitudinal/Va'
       'SkywalkerX8_Longitudinal_Control/SkywalkerX8 Aircraft + ...
          Aerodynamics Longitudinal/h'
       'SkywalkerX8_Longitudinal_Control/SkywalkerX8 Aircraft + ...
          Aerodynamics Longitudinal/theta'
       'SkywalkerX8_Longitudinal_Control/SkywalkerX8 Aircraft + ...
          Aerodynamics Longitudinal/alpha'
       'SkywalkerX8_Longitudinal_Control/SkywalkerX8 Aircraft + ...
          Aerodynamics Longitudinal/q'
       'SkywalkerX8_Longitudinal_Control/Gain Scheduled Controller ...
          Va_dt/PIDF Controller Va_dt/Sum/De'
       'SkywalkerX8_Longitudinal_Control/Gain Scheduled Controller ...
          Va_dt/PIDF Controller Va_dt/Product1/NDe'};

ST0.addPoint(POI);

```

```
109 ST0.addOpening('SkywalkerX8_Longitudinal_Control/SkywalkerX8 ...
       Aircraft + Aerodynamics Longitudinal/h');
110 ST0.addOpening('SkywalkerX8_Longitudinal_Control/SkywalkerX8 ...
       Aircraft + Aerodynamics Longitudinal/theta');
111 ST0.addOpening('SkywalkerX8_Longitudinal_Control/SkywalkerX8 ...
       Aircraft + Aerodynamics Longitudinal/alpha');
112 ST0.addOpening('SkywalkerX8_Longitudinal_Control/SkywalkerX8 ...
       Aircraft + Aerodynamics Longitudinal/q');
113
114 % We know we need to constrain what the maximum value can be for Kp ...
       due to
115 % actuator saturation.
116
117 % We know that dt cannot exceed 1. We also know what our expected ...
       maximum
118 % and minimum airspeeds are, so we use these to define our gain ...
       limit as being
119 % approximately 50% (small signal) of the absolute max difference in...
        Va.
120 % We also take this chance to set saturation limits.
121
122 dtMax = 1;
123 VaChangeMax = (SkywalkerX8.Control.Longitudinal...
124                .SchedulingVariables.VaArray(end)...
125               - SkywalkerX8.Control.Longitudinal...
126                 .SchedulingVariables.VaArray(1))*1/2;
127
128 SkywalkerX8.Control.Longitudinal.VaDtController.upperSaturationDt = ...
       dtMax;
129 SkywalkerX8.Control.Longitudinal.VaDtController.lowerSaturationDt = ...
       0;
130
131 % so we can generate a
132 % limit on gain from these 2 values:
133
134 VaDtGainLimit = dtMax/VaChangeMax;
135
136 % and we assume this will only really be commanded if we are in trim
137
138 VaDtPGainLimitInput = 'SkywalkerX8_Longitudinal_Control/Sum3/Va_e';
139 VaDtPGainLimitOutput = 'SkywalkerX8_Longitudinal_Control/Gain ...
       Scheduled Controller Va_dt/dt';
140
141 R1 = TuningGoal.Gain(VaDtPGainLimitInput, VaDtPGainLimitOutput, ...
       VaDtGainLimit);
142 R1.Focus = [1E-02 1E02];
143
144 % Next we know that our general response should have the standard ...
       gain
145 % margin of 6 dB, and a 60 deg phase margin forces a response that ...
       does not
146 % have a resonance peak, while still ensuring as rapid a response as
```

```
147 % possible (and around a 5% OS for a 2nd order system). So we assume...
        this
148
149 gainMargin = 6; % >6 dB gain margin ideally
150 phaseMargin = 60; % 60 deg phase margin should provide adequate ...
        damping
151 R2 = TuningGoal.Margins('SkywalkerX8_Longitudinal_Control/...
        SkywalkerX8 Aircraft + Aerodynamics Longitudinal/Va', gainMargin,...
         phaseMargin);
152 R2.Focus = [1E-02 1E02];
153
154 % Next we know that we want our crossover frequency to be at ...
        approximately
155 % 3 rad/s so that it's much faster than the theta loop. We try and ...
        bump this
156 % up slightly so that it's as fast as it can be.
157
158 LS = frd([100 1 0.0001],[0.03 3 300]);
159 R3 = TuningGoal.LoopShape('Va', LS, 0.5);
160
161 % Next we simply add a goal for our response in the time domain. We'...
        d like
162 % this to be as fast as possible, given the previous constraints on
163 % crossover frequency and margins.
164
165 wn = 0.1/2.2; %We want a rise time of approximately 0.1 s so we ...
        divide by 2.2 to get
166                 %wn.
167 OS = 4; %4% overshoot is fine on Va and corresponds to the 70 degree...
         phase
168         %we specified previously
169
170 R4 = TuningGoal.StepTracking('Va_c', 'Va', wn, OS);
171
172 % Filter Frequency is kept low because the actuator can only respond...
         in 0.2
173 % s, so our lag is 5 rad/s - meaning we limit this to half that.
174
175 NGainLimit = 2.5;
176
177 NGainLimitInput = 'SkywalkerX8_Longitudinal_Control/Gain Scheduled ...
        Controller Va_dt/PIDF Controller Va_dt/Sum/De';
178 NGainLimitOutput = 'SkywalkerX8_Longitudinal_Control/Gain Scheduled ...
        Controller Va_dt/PIDF Controller Va_dt/Product1/NDe';
179
180 NGainLimitLoopOpening = {NGainLimitInput,...
181                          NGainLimitOutput};
182
183 R5 = TuningGoal.Gain(NGainLimitInput, NGainLimitOutput, NGainLimit);
184 R5.Openings = NGainLimitLoopOpening;
185 R5.Focus = [1E-02 1E02]; %Only focus on this frequency band
186
```

```
187 ST0.setBlockParam('PIDF Controller Va_dt Kp Lookup Table', Kp);
188 ST0.setBlockParam('PIDF Controller Va_dt Ki Lookup Table', Ki);
189 ST0.setBlockParam('PIDF Controller Va_dt Kd Lookup Table', Kd);
190 ST0.setBlockParam('PIDF Controller Va_dt N Lookup Table', N);
191
192 STVaDt = systune(ST0, [R1, R2], [R3, R4, R5], sysTuneOpts);
193
194 tableValues = getBlockValue(STVaDt);
195
196 SkywalkerX8.Control.Longitudinal.VaDtController.KpVaDtTF = ...
       tableValues.PIDF_Controller_Va_dt_Kp_Lookup_Table;
197 SkywalkerX8.Control.Longitudinal.VaDtController.KiVaDtTF = ...
       tableValues.PIDF_Controller_Va_dt_Ki_Lookup_Table;
198 SkywalkerX8.Control.Longitudinal.VaDtController.KdVaDtTF = ...
       tableValues.PIDF_Controller_Va_dt_Kd_Lookup_Table;
199 SkywalkerX8.Control.Longitudinal.VaDtController.NVaDtTF = ...
       tableValues.PIDF_Controller_Va_dt_N_Lookup_Table;
200
201 SkywalkerX8.Control.Longitudinal.VaDtController.KpVaDt = squeeze(...
       tableValues.PIDF_Controller_Va_dt_Kp_Lookup_Table);
202 SkywalkerX8.Control.Longitudinal.VaDtController.KiVaDt = squeeze(...
       tableValues.PIDF_Controller_Va_dt_Ki_Lookup_Table);
203 SkywalkerX8.Control.Longitudinal.VaDtController.KdVaDt = squeeze(...
       tableValues.PIDF_Controller_Va_dt_Kd_Lookup_Table);
204 SkywalkerX8.Control.Longitudinal.VaDtController.NVaDt = squeeze(...
       tableValues.PIDF_Controller_Va_dt_N_Lookup_Table);
205
206 % Get scheduling parameters and write them to our desired variables
207 % Note thase these are based on our basis function.
208
209 TGS = getBlockParam(STVaDt);
210 SkywalkerX8.Control.Longitudinal...
211     .VaDtController.BasisFunction = ...
          TGS.PIDF_Controller_Va_dt_Kp_Lookup_Table.BasisFunctions;
212 SkywalkerX8.Control.Longitudinal...
213     .VaDtController.KpVaDtParameterizedGains = getData(...
          TGS.PIDF_Controller_Va_dt_Kp_Lookup_Table);
214 SkywalkerX8.Control.Longitudinal...
215     .VaDtController.KiVaDtParameterizedGains = getData(...
          TGS.PIDF_Controller_Va_dt_Ki_Lookup_Table);
216 SkywalkerX8.Control.Longitudinal...
217     .VaDtController.KdVaDtParameterizedGains = getData(...
          TGS.PIDF_Controller_Va_dt_Kd_Lookup_Table);
218 SkywalkerX8.Control.Longitudinal...
219     .VaDtController.NVaDtParameterizedGains = getData(...
          TGS.PIDF_Controller_Va_dt_N_Lookup_Table);
220
221 % Next we define our tracking constant (not used for controller ...
       design, but
222 % provided by the designer to assist with bumpless transfer in
223 % implementation).
224
```

```
225 % This Is defined relative to our Ki gain via a scalar. As such, it ...
      should
226 % be x times faster than this integrator loop. 10 seems like a ...
      reasonable
227 % value.
228
229 SkywalkerX8.Control.Longitudinal.VaDtController.KtScalar = 10;
230
231 disp('Va_dt Control Loop Design Complete')
```

B.1.4 Airspeed Controller - Pitch Angle

Listing B.4: Airspeed controller, using pitch angle, tuning goals, SkywalkerX8_Longitudinal_ControlDesign_VaAlpha.m

```
1  %% Va Control Design %%
2
3  % Va is controlled either by using propeller speed (dt) or theta. ...
       This
4  % script deals with the latter case, because the scheduling variable...
        is
5  % the same as the theta/altitude controller (both Va and alpha). ...
       This is
6  % because if the system has a strong dependency on theta, it will ...
       also
7  % depend on angle of attack and as such we will need to incorporate ...
       these
8  % effects into the gain scheduling. This is clear from the bode ...
       plots of
9  % the system and how the dynamics vary substantially with varying ...
       alpha.
10
11 % Va Theta Control Design %
12
13 disp('Va Theta Control Loop Design Started')
14
15 %We get our initial values from siso tool with a midpoint controller...
       (Va,
16 %alpha) = (Va(5), alpha(3)).
17
18 KpInit = 0.0106;
19 KiInit = 0.0247;
20 KdInit = 0.0031;
21 NInit = 1.5205;
22
23 Kp = tunableSurface('Kp', KpInit, TuningGrid, ShapeFcn);
24 Ki = tunableSurface('Ki', KiInit, TuningGrid, ShapeFcn);
25 Kd = tunableSurface('Kd', KdInit, TuningGrid, ShapeFcn);
26 N = tunableSurface('N', NInit, TuningGrid, ShapeFcn);
27
```

```
tunedBlocks = {'PIDF Controller Va_theta Kp 2D Lookup Table',...
               'PIDF Controller Va_theta Ki 2D Lookup Table',...
               'PIDF Controller Va_theta Kd 2D Lookup Table',...
               'PIDF Controller Va_theta N 2D Lookup Table'};

% We can get theta_c to Va as the sub model and then we can simply ...
    replace
% the theta controller with feedthrough and open up q and theta.

minRealTolerance = 1E-05;

thetaCToVa = minreal(tf(getIOTransfer(STTheta, 'theta_c', 'Va')), ...
    minRealTolerance, false);

VaThetaPlantSubValue = [tf(0) thetaCToVa;...
                        tf(0) tf(0);...
                        tf(0) tf(0);...
                        tf(0) tf(0);...
                        tf(0) tf(0)];

VaThetaPlantSubName = plantSubName;

VaThetaPlantSub = struct('Name', VaThetaPlantSubName, 'Value', ...
    VaThetaPlantSubValue);

controllerThetaSubValue = [tf(1),...
                           tf(0),...
                           tf(0),...
                           tf(0),...
                           tf(0)];

controllerThetaSubName = [sys '/Gain Scheduled Controller Theta'];

controllerThetaSub = struct('Name', controllerThetaSubName, 'Value',...
     controllerThetaSubValue);

switchSub = [tf(1) tf(0) tf(0)]; %This changes from previous because...
     we now want our Va_theta controller to go through
algBreakSub = tf(1);

controllerVaDtName = [sys '/Gain Scheduled Controller Va_dt'];
controllerVaDtValue = [tf(0) tf(0)];

controllerVaDtSub = struct('Name', controllerVaDtName, 'Value', ...
    controllerVaDtValue);

controllerAltName = [sys '/Gain Scheduled Controller h_theta'];
controllerAltValue = [tf(0) tf(0) tf(0)];

controllerAltSub = struct('Name', controllerAltName, 'Value', ...
    controllerAltValue);

```

```
73 controllerSwitchName = [sys '/Switch'];
74 controllerSwitchValue = switchSub;
75
76 controllerSwitchSub = struct('Name', controllerSwitchName, 'Value', ...
       controllerSwitchValue);
77
78 VaThetaClampSwitchName = [sys '/Gain Scheduled Controller Va_theta/...
       PIDF Controller Va_theta/Clamp Switch'];
79 VaThetaClampSwitchValue = [tf(0) tf(0) tf(1)];
80
81 VaThetaClampSwitchSub = struct('Name', VaThetaClampSwitchName, '...
       Value', VaThetaClampSwitchValue);
82
83 algebraicLoopBreakValue = algBreakSub;
84
85 algebraicLoopBreak1Name = [sys '/Alg Break 1'];
86 algebraicLoopBreak2Name = [sys '/Alg Break 2'];
87 algebraicLoopBreak3Name = [sys '/Alg Break 3'];
88 algebraicLoopBreak4Name = [sys '/Alg Break 4'];
89 algebraicLoopBreak5Name = [sys '/Alg Break 5'];
90
91 algebraicLoopBreak1Sub = struct('Name', algebraicLoopBreak1Name, '...
       Value', algebraicLoopBreakValue);
92 algebraicLoopBreak2Sub = struct('Name', algebraicLoopBreak2Name, '...
       Value', algebraicLoopBreakValue);
93 algebraicLoopBreak3Sub = struct('Name', algebraicLoopBreak3Name, '...
       Value', algebraicLoopBreakValue);
94 algebraicLoopBreak4Sub = struct('Name', algebraicLoopBreak4Name, '...
       Value', algebraicLoopBreakValue);
95 algebraicLoopBreak5Sub = struct('Name', algebraicLoopBreak5Name, '...
       Value', algebraicLoopBreakValue);
96
97 BlockSubs = [VaThetaPlantSub;...
98              controllerThetaSub;...
99              controllerVaDtSub;...
100             controllerAltSub;...
101             controllerSwitchSub;...
102             VaThetaClampSwitchSub;...
103             algebraicLoopBreak1Sub;...
104             algebraicLoopBreak2Sub;...
105             algebraicLoopBreak3Sub;...
106             algebraicLoopBreak4Sub;...
107             algebraicLoopBreak5Sub];
108
109 % Create slTuner interface for tuning of tunedBlocks within the ...
       context of
110 % sys, after substituting out the blocks defined in BlockSubs, with ...
       the
111 % options given in LinOpt.
112
113 ST0 = slTuner(sys, tunedBlocks, BlockSubs, LinOpt);
114
```

```
115 ST0.Ts = 0; % We want continuous-time linearizations
116
117 % Add all relevant points of interest (POI) that we will examine in ...
      the
118 % model.
119
120 POI = {'SkywalkerX8_Longitudinal_Control/Va_c'
121        'SkywalkerX8_Longitudinal_Control/Va_e'
122        'SkywalkerX8_Longitudinal_Control/theta_c_Va'
123        'SkywalkerX8_Longitudinal_Control/de'
124        'SkywalkerX8_Longitudinal_Control/SkywalkerX8 Aircraft + ...
             Aerodynamics Longitudinal/Va'
125        'SkywalkerX8_Longitudinal_Control/SkywalkerX8 Aircraft + ...
             Aerodynamics Longitudinal/h'
126        'SkywalkerX8_Longitudinal_Control/SkywalkerX8 Aircraft + ...
             Aerodynamics Longitudinal/theta'
127        'SkywalkerX8_Longitudinal_Control/SkywalkerX8 Aircraft + ...
             Aerodynamics Longitudinal/alpha'
128        'SkywalkerX8_Longitudinal_Control/SkywalkerX8 Aircraft + ...
             Aerodynamics Longitudinal/q'};
129
130 ST0.addPoint(POI);
131
132 ST0.addOpening('SkywalkerX8_Longitudinal_Control/SkywalkerX8 ...
      Aircraft + Aerodynamics Longitudinal/h');
133 ST0.addOpening('SkywalkerX8_Longitudinal_Control/SkywalkerX8 ...
      Aircraft + Aerodynamics Longitudinal/theta');
134 ST0.addOpening('SkywalkerX8_Longitudinal_Control/SkywalkerX8 ...
      Aircraft + Aerodynamics Longitudinal/alpha');
135 ST0.addOpening('SkywalkerX8_Longitudinal_Control/SkywalkerX8 ...
      Aircraft + Aerodynamics Longitudinal/q');
136
137 % We know we need to constrain what the maximum value can be for Kp ...
      due to
138 % actuator saturation.
139
140 % We'll make the assumption that theta will not exceed 60 degrees, ...
      and we
141 % know our maximum velocity change - so let's assume 1/2 of that
142 % will be the largest seen error signal. We also take this ...
      opportunity to
143 % set our saturation limits.
144
145 thetaMax = 60*pi/180;
146 VaChangeMax = (SkywalkerX8.Control.Longitudinal...
147                .SchedulingVariables.VaArray(end)...
148               - SkywalkerX8.Control.Longitudinal...
149                 .SchedulingVariables.VaArray(1))*2/3;
150
151 SkywalkerX8.Control.Longitudinal...
152     .VaThetaController.upperSaturationThetaC = thetaMax;
153 SkywalkerX8.Control.Longitudinal...
```

```
154      .VaThetaController.lowerSaturationThetaC = -thetaMax;
155
156  % so we can generate a
157  % limit on gain from these 2 values:
158
159  VaThetaGainLimit = thetaMax/VaChangeMax;
160
161  % and we assume this will only really be commanded if we are in trim
162
163  VaThetaPGainLimitInput = 'SkywalkerX8_Longitudinal_Control/Sum3/Va_e...
       ';
164  VaThetaPGainLimitOutput = 'SkywalkerX8_Longitudinal_Control/Gain ...
       Scheduled Controller Va_theta/theta_c_Va';
165
166  R1 = TuningGoal.Gain(VaThetaPGainLimitInput, VaThetaPGainLimitOutput...
       , VaThetaGainLimit);
167  R1.Focus = [1E-02 1E02];
168
169  % Next we know that our general response should have the standard ...
       gain
170  % margin of 6 dB, and a 70 deg phase margin forces a response that ...
       does not
171  % have a resonance peak, while still ensuring as rapid a response as
172  % possible (and around a 5% OS for a 2nd order system). So we assume...
        this
173
174  gainMargin = 6; % >6 dB gain margin ideally
175  phaseMargin = 40; % 40 degree phase margin is fine for this ...
       controller given that it's only going to be conditionally active
176  R2 = TuningGoal.Margins('SkywalkerX8_Longitudinal_Control/...
       SkywalkerX8 Aircraft + Aerodynamics Longitudinal/Va', gainMargin,...
        phaseMargin);
177  R2.Focus = [1E-02 1E02];
178
179  % Next we know that we want our crossover frequency to be at ...
       approximately 0.75 rad/s, as
180  % we know our second order approximations are only true up until ...
       around 3
181  % rad/s and there is a large resonance peak at approximately 1 rad/s...
       , So we
182  % try and band limit this so that our models are correct and we ...
       avoid the
183  % resonance peak.
184
185  LS = frd([100 1 0.0001],[0.005 0.5 50]);
186  R3 = TuningGoal.LoopShape('Va', LS, 0.2);
187
188  % Next we know that we want our Va loop to have a rise time that
189  % corresponds to the crossover frequency above, but also that ...
       overshoot
190  % is allowable. As such, we specify this as a requirement.
191
```

```
192 tau = 4/2.2;
193 OS = 4;
194
195 R4 = TuningGoal.StepTracking('Va_c', 'Va', tau, OS);
196
197 ST0.setBlockParam('PIDF Controller Va_theta Kp 2D Lookup Table', Kp)...
      ;
198 ST0.setBlockParam('PIDF Controller Va_theta Ki 2D Lookup Table', Ki)...
      ;
199 ST0.setBlockParam('PIDF Controller Va_theta Kd 2D Lookup Table', Kd)...
      ;
200 ST0.setBlockParam('PIDF Controller Va_theta N 2D Lookup Table', N);
201
202 STVaTheta = systune(ST0, [R1, R2], [R3, R4], sysTuneOpts);
203
204 % Get values from ST and write them to our desired variables
205 % Note that TF here stands for transfer function and simply captures...
       the
206 % values with the correct structure so that later they can be used ...
      for
207 % block subs.
208
209 tableValues = getBlockValue(STVaTheta);
210
211 SkywalkerX8.Control.Longitudinal.VaThetaController.KpVaThetaTF = ...
      tableValues.PIDF_Controller_Va_theta_Kp_2D_Lookup_Table;
212 SkywalkerX8.Control.Longitudinal.VaThetaController.KiVaThetaTF = ...
      tableValues.PIDF_Controller_Va_theta_Ki_2D_Lookup_Table;
213 SkywalkerX8.Control.Longitudinal.VaThetaController.KdVaThetaTF = ...
      tableValues.PIDF_Controller_Va_theta_Kd_2D_Lookup_Table;
214 SkywalkerX8.Control.Longitudinal.VaThetaController.NVaThetaTF = ...
      tableValues.PIDF_Controller_Va_theta_N_2D_Lookup_Table;
215
216 SkywalkerX8.Control.Longitudinal.VaThetaController.KpVaTheta = ...
      squeeze(tableValues.PIDF_Controller_Va_theta_Kp_2D_Lookup_Table);
217 SkywalkerX8.Control.Longitudinal.VaThetaController.KiVaTheta = ...
      squeeze(tableValues.PIDF_Controller_Va_theta_Ki_2D_Lookup_Table);
218 SkywalkerX8.Control.Longitudinal.VaThetaController.KdVaTheta = ...
      squeeze(tableValues.PIDF_Controller_Va_theta_Kd_2D_Lookup_Table);
219 SkywalkerX8.Control.Longitudinal.VaThetaController.NVaTheta = ...
      squeeze(tableValues.PIDF_Controller_Va_theta_N_2D_Lookup_Table);
220
221 % Get scheduling parameters and write them to our desired variables
222 % Note thase these are based on our basis function.
223
224 TGS = getBlockParam(STVaTheta);
225 SkywalkerX8.Control.Longitudinal...
226     .VaThetaController.BasisFunction = ...
          TGS.PIDF_Controller_Va_theta_Kp_2D_Lookup_Table.BasisFunctions...
          ;
227 SkywalkerX8.Control.Longitudinal...
```

```
228     .VaThetaController.KpVaThetaParameterizedGains = getData(...
            TGS.PIDF_Controller_Va_theta_Kp_2D_Lookup_Table);
229 SkywalkerX8.Control.Longitudinal...
230     .VaThetaController.KiVaThetaParameterizedGains = getData(...
            TGS.PIDF_Controller_Va_theta_Ki_2D_Lookup_Table);
231 SkywalkerX8.Control.Longitudinal...
232     .VaThetaController.KdVaThetaParameterizedGains = getData(...
            TGS.PIDF_Controller_Va_theta_Kd_2D_Lookup_Table);
233 SkywalkerX8.Control.Longitudinal...
234     .VaThetaController.NVaThetaParameterizedGains = getData(...
            TGS.PIDF_Controller_Va_theta_N_2D_Lookup_Table);
235
236 % Next we define our tracking constant (not used for controller ...
      design, but
237 % provided by the designer to assist with bumpless transfer in
238 % implementation).
239
240 % This Is defined relative to our Ki gain via a scalar. As such, it ...
      should
241 % be x times faster than this integrator loop. 10 seems like a ...
      reasonable
242 % value.
243
244 SkywalkerX8.Control.Longitudinal.VaThetaController.KtScalar = 10;
245
246 disp('Va Theta Control Loop Design Complete')
```

B.2 Lateral Controller Tuning Goals

B.2.1 Roll Angle Controller

Listing B.5: Roll angle controller tuning goals, SkywalkerX8_Lateral_ControlDesign_Va.m

```
disp('Starting Phi Control Loop Design...')

KpInit = 0.2861;
KiInit = 3.5357;
KdInit = -0.6717;
NInit = 2.822;

Kp2DOFInit = -1.0545;
Kd2DOFInit = 1.4099;
N2DOFInit = 0.9608;

Kp = tunableSurface('Kp', KpInit, VaTuningGrid, VaShapeFcn);
Ki = tunableSurface('Ki', KiInit, VaTuningGrid, VaShapeFcn);
Kd = tunableSurface('Kd', KdInit, VaTuningGrid, VaShapeFcn);
N = tunableSurface('N', NInit, VaTuningGrid, VaShapeFcn);

Kp2DOF = tunableSurface('Kp2DOF', Kp2DOFInit, VaTuningGrid, ...
    VaShapeFcn);
Kd2DOF = tunableSurface('Kd2DOF', Kd2DOFInit, VaTuningGrid, ...
    VaShapeFcn);
N2DOF = tunableSurface('N2DOF', N2DOFInit, VaTuningGrid, VaShapeFcn)...
    ;

tunedBlocks = {'PIDF Controller Phi Kp 2D Lookup Table',...
               'PIDF Controller Phi Ki 2D Lookup Table',...
               'PIDF Controller Phi Kd 2D Lookup Table',...
               'PIDF Controller Phi N 2D Lookup Table',...
               'PDF Controller Phi 2DOF Kp 2D Lookup Table',...
               'PDF Controller Phi 2DOF Kd 2D Lookup Table',...
               'PDF Controller Phi 2DOF N 2D Lookup Table'};

minRealTolerance = 1E-05;

controllerChiPhiSubValue = [tf(0),...
                            tf(0)];

controllerChiPhiSubName = [sys '/Gain Scheduled Controller Chi'];

controllerChiPhiSub = struct('Name', controllerChiPhiSubName, 'Value...
    ', controllerChiPhiSubValue);

algBreakSub = tf(1);

algebraicLoopBreakValue = algBreakSub;
```

```
41
42 algebraicLoopBreak1Name = [sys '/Alg Break 1'];
43 algebraicLoopBreak2Name = [sys '/Alg Break 2'];
44 algebraicLoopBreak3Name = [sys '/Alg Break 3'];
45 algebraicLoopBreak4Name = [sys '/Alg Break 4'];
46
47 algebraicLoopBreak1Sub = struct('Name', algebraicLoopBreak1Name, '...
       Value', algebraicLoopBreakValue);
48 algebraicLoopBreak2Sub = struct('Name', algebraicLoopBreak2Name, '...
       Value', algebraicLoopBreakValue);
49 algebraicLoopBreak3Sub = struct('Name', algebraicLoopBreak3Name, '...
       Value', algebraicLoopBreakValue);
50 algebraicLoopBreak4Sub = struct('Name', algebraicLoopBreak4Name, '...
       Value', algebraicLoopBreakValue);
51
52 BlockSubs = [plantSub;...
53              controllerChiPhiSub;...
54              algebraicLoopBreak1Sub;...
55              algebraicLoopBreak2Sub;...
56              algebraicLoopBreak3Sub;...
57              algebraicLoopBreak4Sub];
58
59 % Create slTuner interface for tuning of tunedBlocks within the ...
       context of
60 % sys, after substituting out the blocks defined in BlockSubs, with ...
       the
61 % options given in LinOpt.
62
63 ST0 = slTuner(sys, tunedBlocks, BlockSubs, LinOpt);
64
65 ST0.Ts = 0; % We want continuous-time linearizations
66
67 % Add all relevant points of interest (POI) that we will examine in ...
       the
68 % model.
69
70 POI = {'SkywalkerX8_Lateral_Control/Phi_c'
71        'SkywalkerX8_Lateral_Control/Phi_e'
72        'SkywalkerX8_Lateral_Control/da'
73        'SkywalkerX8_Lateral_Control/Gain Scheduled Controller Phi/DF...
              Controller Phi/Sum/da*_e'
74        'SkywalkerX8_Lateral_Control/Gain Scheduled Controller Phi/DF...
              Controller Phi/Product1/Nda*_e'
75        'SkywalkerX8_Lateral_Control/Gain Scheduled Controller Phi/DF...
              Controller Phi/Integrator/da*'
76        'SkywalkerX8_Lateral_Control/SkywalkerX8 Aircraft + ...
             Aerodynamics Lateral/Va'
77        'SkywalkerX8_Lateral_Control/SkywalkerX8 Aircraft + ...
             Aerodynamics Lateral/p'
78        'SkywalkerX8_Lateral_Control/SkywalkerX8 Aircraft + ...
             Aerodynamics Lateral/Phi'
```

```
79          'SkywalkerX8_Lateral_Control/SkywalkerX8 Aircraft + ...
               Aerodynamics Lateral/Chi'};
80
81  ST0.addPoint(POI);
82
83  ST0.addOpening('SkywalkerX8_Lateral_Control/SkywalkerX8 Aircraft + ...
       Aerodynamics Lateral/Phi_c');
84  ST0.addOpening('SkywalkerX8_Lateral_Control/SkywalkerX8 Aircraft + ...
       Aerodynamics Lateral/Va');
85  ST0.addOpening('SkywalkerX8_Lateral_Control/SkywalkerX8 Aircraft + ...
       Aerodynamics Lateral/Chi');
86
87  % We know we need to constrain what the maximum value can be for Kp ...
       due to
88  % actuator saturation.
89
90  % We know that da cannot exceed 30 degrees. We also know what our ...
       expected maximum
91  % and minimum phi are, so we use these to define our gain limit as ...
       being
92  % approximately 1/2 of the absolute max difference in phi. We also ...
       take this
93  % chance to set saturation limits.
94
95  daMax = 2*15*pi/180; %x2 because split over 2 elevons
96  phiMax = 30*pi/180*0.5;
97
98  SkywalkerX8.Control.Lateral.PhiController.upperSaturationDa = daMax;
99  SkywalkerX8.Control.Lateral.PhiController.lowerSaturationDa = -daMax...
       ;
100
101 % so we can generate a
102 % limit on gain from these 2 values:
103
104 PhiDaGainLimit = daMax/phiMax;
105
106 % and we assume this will only really be commanded if we are in trim
107
108 PhiDaPGainLimitInput = 'SkywalkerX8_Lateral_Control/Sum/Phi_e';
109 PhiDaPGainLimitOutput = 'SkywalkerX8_Lateral_Control/Gain Scheduled ...
       Controller Phi/da';
110
111 R1 = TuningGoal.Gain(PhiDaPGainLimitInput, PhiDaPGainLimitOutput, ...
       PhiDaGainLimit);
112 R1.Focus = [1E-02 1E02];
113
114 % Next we know that we need to limit our derivative in terms of ...
       bandwidth.
115 % We do this with the associated filter (ie. we constrain the gain ...
       of "N",
116 % the F in the PIDF controller). This needs to be limited to ...
       approximately
```

```
117 % 13 rad/s to not cause issues with the actuator.
118
119 NGainLimit = 13;
120
121 NGainLimitInput = 'SkywalkerX8_Lateral_Control/Gain Scheduled ...
       Controller Phi/DF Controller Phi/Sum/da*_e';
122 NGainLimitOutput = 'SkywalkerX8_Lateral_Control/Gain Scheduled ...
       Controller Phi/DF Controller Phi/Product1/Nda*_e';
123
124 NGainLimitLoopOpening = {'SkywalkerX8_Lateral_Control/Gain Scheduled...
        Controller Phi/DF Controller Phi/Sum/da*_e',...
125                          'SkywalkerX8_Lateral_Control/Gain Scheduled...
                                 Controller Phi/DF Controller Phi/...
                               Product1/Nda*_e'};
126
127 R2 = TuningGoal.Gain(NGainLimitInput, NGainLimitOutput, NGainLimit);
128 R2.Openings = NGainLimitLoopOpening;
129 R2.Focus = [1E-02 1E02]; %Only focus on this frequency band
130
131 % Next we know that our general response should have the standard ...
       gain
132 % margin of 6 dB, and a 70 deg phase margin forces a response that ...
       does not
133 % have a resonance peak, while still ensuring as rapid a response as
134 % possible (and around a 5% OS for a 2nd order system). However, we ...
       will
135 % accept a lower phase margin in the small signal design so as to ...
       allow for
136 % faster transients to the saturation limits on roll angle.
137
138 gainMargin = 6; % >6 dB gain margin ideally
139 phaseMargin = 30; % 30 deg phase margin should provide adequate ...
       damping
140 R3 = TuningGoal.Margins('SkywalkerX8_Lateral_Control/SkywalkerX8 ...
       Aircraft + Aerodynamics Lateral/Phi', gainMargin, phaseMargin);
141 R3.Focus = [1E-02 1E02];
142
143 % Next we know that our theta controller was capable of having a ...
       crossover
144 % frequency at approximately 8 rad/s. We also know that da -> l
145 % aerodynamically has about half the gain of de -> m. This, as well ...
       as the
146 % fact that Jyy is only 13.85% of Jxx means we expect our roll to be
147 % substantially slower than our pitch. We will aim for approximately...
        25-30%
148 % the speed but this might need to be adjusted.
149
150 LS = frd([1000 1 0.001],[0.08 8 800]);
151 R4 = TuningGoal.LoopShape('Phi', LS, 0.5);
152
153 % Next we simply add a goal for our response in the time domain. We'...
       d like
```

```
154 % this to be as fast as possible, given the previous constraints on
155 % crossover frequency and margins. We know that for a coordinated ...
      turn,
156 % dPsi/dt = g/Va*tan(phi). As such, we can estimate an optimal ...
      psi_dot
157 % assuming a constant bank angle (let's assume half the maximum) and...
      use
158 % this to determine an expected rise time (making use of the ...
      previous
159 % expected maximum Chi command as the reference command). We slow ...
      this
160 % requirement down slightly to account for having to actually get to...
      the
161 % bank angle from 0 roll - and assume this to mean we'll be 20% ...
      slower than
162 % optimal.
163
164 % We use the above, and take into account needing our inner phi loop...
      to be
165 % 2-3x faster than the outer loop to reduce the requirement for this...
      inner
166 % loop. We then increase this so as to account for the small signal
167 % analysis.
168
169 chiMax = 90*pi/180;
170
171 g = 9.81;
172 tauScalar = 1;
173 riseTimeToTau = 1/2.2;
174 bandwidthSeparation = 50;
175 tauValues = zeros(nVa, 1);
176 OSValues = zeros(nVa, 1);
177
178 for i = 1:nVa
179
180     psiDotMax = (g/...
          SkywalkerX8.Control.Lateral.SchedulingVariables.VaArray(i))*...
          tan(phiMax);
181     optimalRiseTime = chiMax/psiDotMax;
182     tauValues(i) = optimalRiseTime*tauScalar*riseTimeToTau/...
          bandwidthSeparation;
183     OSValues(i) = 5; %5% overshoot is fine given that this is an ...
          inner loop
184
185 end
186
187 FH = @(tau, OS) TuningGoal.StepTracking('Phi_c', 'Phi', tau, OS);
188
189 R5 = varyingGoal(FH, tauValues, OSValues);
190
191 ST0.setBlockParam('PIDF Controller Phi Kp 2D Lookup Table', Kp);
192 ST0.setBlockParam('PIDF Controller Phi Ki 2D Lookup Table', Ki);
```

```
193 ST0.setBlockParam('PIDF Controller Phi Kd 2D Lookup Table', Kd);
194 ST0.setBlockParam('PIDF Controller Phi N 2D Lookup Table', N);
195
196 ST0.setBlockParam('PDF Controller Phi 2DOF Kp 2D Lookup Table', ...
        Kp2DOF);
197 ST0.setBlockParam('PDF Controller Phi 2DOF Kd 2D Lookup Table', ...
        Kd2DOF);
198 ST0.setBlockParam('PDF Controller Phi 2DOF N 2D Lookup Table', N2DOF...
        );
199
200 STPhi = systune(ST0, [R1, R3], [R2, R4, R5], sysTuneOpts);
201
202 tableValues = getBlockValue(STPhi);
203
204 SkywalkerX8.Control.Lateral.PhiController.KpPhiTF = ...
        tableValues.PIDF_Controller_Phi_Kp_2D_Lookup_Table;
205 SkywalkerX8.Control.Lateral.PhiController.KiPhiTF = ...
        tableValues.PIDF_Controller_Phi_Ki_2D_Lookup_Table;
206 SkywalkerX8.Control.Lateral.PhiController.KdPhiTF = ...
        tableValues.PIDF_Controller_Phi_Kd_2D_Lookup_Table;
207 SkywalkerX8.Control.Lateral.PhiController.NPhiTF = ...
        tableValues.PIDF_Controller_Phi_N_2D_Lookup_Table;
208
209 SkywalkerX8.Control.Lateral.PhiController.KpPhi2DOFTF = ...
        tableValues.PDF_Controller_Phi_2DOF_Kp_2D_Lookup_Table;
210 SkywalkerX8.Control.Lateral.PhiController.KdPhi2DOFTF = ...
        tableValues.PDF_Controller_Phi_2DOF_Kd_2D_Lookup_Table;
211 SkywalkerX8.Control.Lateral.PhiController.NPhi2DOFTF = ...
        tableValues.PDF_Controller_Phi_2DOF_N_2D_Lookup_Table;
212
213 SkywalkerX8.Control.Lateral.PhiController.KpPhi = squeeze(...
        tableValues.PIDF_Controller_Phi_Kp_2D_Lookup_Table);
214 SkywalkerX8.Control.Lateral.PhiController.KiPhi = squeeze(...
        tableValues.PIDF_Controller_Phi_Ki_2D_Lookup_Table);
215 SkywalkerX8.Control.Lateral.PhiController.KdPhi = squeeze(...
        tableValues.PIDF_Controller_Phi_Kd_2D_Lookup_Table);
216 SkywalkerX8.Control.Lateral.PhiController.NPhi = squeeze(...
        tableValues.PIDF_Controller_Phi_N_2D_Lookup_Table);
217
218 SkywalkerX8.Control.Lateral.PhiController.KpPhi2DOF = squeeze(...
        tableValues.PDF_Controller_Phi_2DOF_Kp_2D_Lookup_Table);
219 SkywalkerX8.Control.Lateral.PhiController.KdPhi2DOF = squeeze(...
        tableValues.PDF_Controller_Phi_2DOF_Kd_2D_Lookup_Table);
220 SkywalkerX8.Control.Lateral.PhiController.NPhi2DOF = squeeze(...
        tableValues.PDF_Controller_Phi_2DOF_N_2D_Lookup_Table);
221
222 % Get scheduling parameters and write them to our desired variables
223 % Note thase these are based on our basis function.
224
225 TGS = getBlockParam(STPhi);
226 SkywalkerX8.Control.Lateral.PhiController.BasisFunction = ...
        TGS.PIDF_Controller_Phi_Kp_2D_Lookup_Table.BasisFunctions;
```

```
227 SkywalkerX8.Control.Lateral.PhiController.KpPhiParameterizedGains = ...
       getData(TGS.PIDF_Controller_Phi_Kp_2D_Lookup_Table);
228 SkywalkerX8.Control.Lateral.PhiController.KiPhiParameterizedGains = ...
       getData(TGS.PIDF_Controller_Phi_Ki_2D_Lookup_Table);
229 SkywalkerX8.Control.Lateral.PhiController.KdPhiParameterizedGains = ...
       getData(TGS.PIDF_Controller_Phi_Kd_2D_Lookup_Table);
230 SkywalkerX8.Control.Lateral.PhiController.NPhiParameterizedGains = ...
       getData(TGS.PIDF_Controller_Phi_N_2D_Lookup_Table);
231
232 SkywalkerX8.Control.Lateral.PhiController.KpPhi2DOFParameterizedGains...
        = getData(TGS.PDF_Controller_Phi_2DOF_Kp_2D_Lookup_Table);
233 SkywalkerX8.Control.Lateral.PhiController.KdPhi2DOFParameterizedGains...
        = getData(TGS.PDF_Controller_Phi_2DOF_Kd_2D_Lookup_Table);
```

B.2.2 Course Angle Controller

Listing B.6: Course angle controller tuning goals, SkywalkerX8_Lateral_ControlDesign_Va.m

```
1
2  %% Chi Control Design %%
3
4  disp('Starting Chi Control Loop Design...')
5
6  KpInit = 0.1;
7  KiInit = 1;
8  KdInit = 0.145;
9  NInit = 1/0.445;
10
11 Kp = tunableSurface('Kp', KpInit, VaTuningGrid, VaShapeFcn);
12 Ki = tunableSurface('Ki', KiInit, VaTuningGrid, VaShapeFcn);
13 Kd = tunableSurface('Kd', KdInit, VaTuningGrid, VaShapeFcn);
14 N = tunableSurface('N', NInit, VaTuningGrid, VaShapeFcn);
15
16 tunedBlocks = {'PIDF Controller Chi Kp 2D Lookup Table',...
17                 'PIDF Controller Chi Ki 2D Lookup Table',...
18                 'PIDF Controller Chi Kd 2D Lookup Table',...
19                 'PIDF Controller Chi N 2D Lookup Table'};
20
21 minRealTolerance = 1E-05;
22
23 % We now reduce these models to make the tuning easier. We know our
24 % bandwidth for this control loop is approximately 0.2 rad/s, as ...
       such, we
25 % can ignore any dynamics higher than around 0.8 rad/s when tuning. ...
       As
26 % such, we can use a reducton for r to a 3rd order model, and a ...
       reduction
27 % for Chi as a first order model. This should drastically simplify ...
       our
```

```
% tuning efforts.

phiMax = 30*pi/180;

for i = 1:nVa

    phiCToChiEstimate(:, :, i) =  tf((g/...
        SkywalkerX8.Control.Lateral.SchedulingVariables.VaArray(i))*...
        tan(phiMax*0.5), [1 0]);

end

plantSubValue = [tf(0) tf(0) tf(0);...
                 tf(0) tf(0) phiCToChiEstimate;...
                 tf(0) tf(0) tf(0);...
                 tf(0) tf(0) tf(0)];

plantSubName = [sys '/SkywalkerX8 Aircraft + Aerodynamics Lateral'];

plantSub = struct('Name', plantSubName, 'Value', plantSubValue);

controllerPhiSubValue = [tf(1),...
                         tf(0),...
                         tf(0),...
                         tf(0)];

controllerPhiSubName = [sys '/Gain Scheduled Controller Phi'];

controllerPhiSub = struct('Name', controllerPhiSubName, 'Value', ...
    controllerPhiSubValue);

algBreakSub = tf(1);

algebraicLoopBreakValue = algBreakSub;

algebraicLoopBreak1Name = [sys '/Alg Break 1'];
algebraicLoopBreak2Name = [sys '/Alg Break 2'];
algebraicLoopBreak3Name = [sys '/Alg Break 3'];
algebraicLoopBreak4Name = [sys '/Alg Break 4'];

algebraicLoopBreak1Sub = struct('Name', algebraicLoopBreak1Name, '...
    Value', algebraicLoopBreakValue);
algebraicLoopBreak2Sub = struct('Name', algebraicLoopBreak2Name, '...
    Value', algebraicLoopBreakValue);
algebraicLoopBreak3Sub = struct('Name', algebraicLoopBreak3Name, '...
    Value', algebraicLoopBreakValue);
algebraicLoopBreak4Sub = struct('Name', algebraicLoopBreak4Name, '...
    Value', algebraicLoopBreakValue);

BlockSubs = [plantSub;...
             controllerPhiSub;...
             algebraicLoopBreak1Sub;...
```

```
73              algebraicLoopBreak2Sub;...
74              algebraicLoopBreak3Sub;...
75              algebraicLoopBreak4Sub];
76
77  % Create slTuner interface for tuning of tunedBlocks within the ...
       context of
78  % sys, after substituting out the blocks defined in BlockSubs, with ...
       the
79  % options given in LinOpt.
80
81  ST0 = slTuner(sys, tunedBlocks, BlockSubs, LinOpt);
82
83  ST0.Ts = 0; % We want continuous-time linearizations
84
85  % Add all relevant points of interest (POI) that we will examine in ...
       the
86  % model.
87
88  POI = {'SkywalkerX8_Lateral_Control/Chi_c/1'
89         'SkywalkerX8_Lateral_Control/Chi_e'
90         'SkywalkerX8_Lateral_Control/Phi_c'
91         'SkywalkerX8_Lateral_Control/Sum2/Chi_e'
92         'SkywalkerX8_Lateral_Control/Gain Scheduled Controller Chi/...
              PIDF Controller Va_theta/POut'
93         'SkywalkerX8_Lateral_Control/Gain Scheduled Controller Chi/...
              PIDF Controller Va_theta/De*'
94         'SkywalkerX8_Lateral_Control/Gain Scheduled Controller Chi/...
              PIDF Controller Va_theta/NDe*'
95         'SkywalkerX8_Lateral_Control/SkywalkerX8 Aircraft + ...
              Aerodynamics Lateral/Va'
96         'SkywalkerX8_Lateral_Control/SkywalkerX8 Aircraft + ...
              Aerodynamics Lateral/p'
97         'SkywalkerX8_Lateral_Control/SkywalkerX8 Aircraft + ...
              Aerodynamics Lateral/Phi'
98         'SkywalkerX8_Lateral_Control/SkywalkerX8 Aircraft + ...
              Aerodynamics Lateral/Chi'};
99
100 ST0.addPoint(POI);
101
102 ST0.addOpening('SkywalkerX8_Lateral_Control/SkywalkerX8 Aircraft + ...
       Aerodynamics Lateral/Va');
103 ST0.addOpening('SkywalkerX8_Lateral_Control/SkywalkerX8 Aircraft + ...
       Aerodynamics Lateral/p');
104 ST0.addOpening('SkywalkerX8_Lateral_Control/SkywalkerX8 Aircraft + ...
       Aerodynamics Lateral/Phi');
105
106 % We know we need to constrain what the maximum value can be for Kp ...
       due to
107 % actuator saturation.
108
109 % We assume that our maximum expected Chi change wil be 90 degrees. ...
       We also know
```

```
110 % that we expect our maximum phi to be 60 degrees. We assume our ...
      maximum
111 % step in Chi will be approximately half the absolute maximum.
112
113 chiMax = 90*pi/180;
114
115 SkywalkerX8.Control.Lateral.ChiController.upperSaturationPhiC = ...
      phiMax;
116 SkywalkerX8.Control.Lateral.ChiController.lowerSaturationPhiC = -...
      phiMax;
117
118 % so we can generate a
119 % limit on gain from these 2 values:
120
121 ChiPhiGainLimit = phiMax/chiMax;
122
123 % and we assume this will only really be commanded if we are in trim
124
125 ChiPhiPGainLimitInput = 'SkywalkerX8_Lateral_Control/Sum2/Chi_e';
126 ChiPhiPGainLimitOutput = 'SkywalkerX8_Lateral_Control/Gain Scheduled...
      Controller Chi/Phi_c';
127
128 R1 = TuningGoal.Gain(ChiPhiPGainLimitInput, ChiPhiPGainLimitOutput, ...
      ChiPhiGainLimit);
129 R1.Focus = [1E-02 1E02];
130
131 % Next we know that our general response should have the standard ...
      gain
132 % margin of 6 dB, and a 70 deg phase margin forces a response that ...
      does not
133 % have a resonance peak, while still ensuring as rapid a response as
134 % possible (and around a 5% OS for a 2nd order system). But we allow...
       for a
135 % lower phase margin in the interests of increasing the system ...
      bandwidth
136
137 gainMargin = 6; % >6 dB gain margin ideally
138 phaseMargin = 40; % 40 deg phase margin should provide adequate ...
      damping
139 R2 = TuningGoal.Margins('SkywalkerX8_Lateral_Control/SkywalkerX8 ...
      Aircraft + Aerodynamics Lateral/Chi', gainMargin, phaseMargin);
140 R2.Focus = [1E-02 1E02];
141
142 % Next we know that our phi controller has a crossover frequency at
143 % approximately 10 rad/s. In order to have this be uncoupled for the
144 % inner loop dynamics, we specify a 50x bandwidth seperation.
145
146 LS = frd([100000 1 0.001],[0.002 0.2 2]);
147 R3 = TuningGoal.LoopShape('Chi', LS, 0.5);
148
149 % Next we simply add a goal for our response in the time domain. We'...
      d like
```

```
150 % this to be as fast as possible, given the previous constraints on
151 % crossover frequency and margins. We know that for a coordinated ...
      turn,
152 % dPsi/dt = g/Va*tan(phi). As such, we can estimate an optimal ...
      psi_dot
153 % assuming a constant bank angle (let's assume half the maximum) and...
      use
154 % this to determine an expected rise time (making use of the ...
      previous
155 % expected maximum Chi command as the reference command). We slow ...
      this
156 % requirement down slightly to account for having to actually get to...
      the
157 % bank angle from 0 roll - and assume this to mean we'll be 10% ...
      slower than
158 % optimal
159
160 g = 9.81;
161 tauScalar = 1;
162 bandwidthSeparation = 3;
163 riseTimeToTau = 1/2.2;
164 tauValues = zeros(nVa, 1);
165 OSValues = zeros(nVa, 1);
166
167 for i = 1:nVa
168
169     psiDotMax = (g/...
          SkywalkerX8.Control.Lateral.SchedulingVariables.VaArray(i))*...
          tan(phiMax);
170     optimalRiseTime = chiMax/psiDotMax;
171     tauValues(i) = optimalRiseTime*tauScalar*riseTimeToTau/...
          bandwidthSeparation;
172     OSValues(i) = 5; %5% overshoot is acceptable if possible
173
174 end
175
176 FH = @(tau, OS) TuningGoal.StepTracking('Chi_c', 'Chi', tau, OS);
177
178 R4 = varyingGoal(FH, tauValues, OSValues);
179
180 ChiPhiPGainLimitInput = 'SkywalkerX8_Lateral_Control/Sum2/Chi_e';
181 ChiPhiPGainLimitOutput = 'SkywalkerX8_Lateral_Control/Gain Scheduled...
      Controller Chi/PIDF Controller Va_theta/POut';
182
183 R5 = TuningGoal.Gain(ChiPhiPGainLimitInput, ChiPhiPGainLimitOutput, ...
      2*ChiPhiGainLimit);
184 R5.Focus = [1E-02 1E02];
185 R5.Openings = {ChiPhiPGainLimitInput, ChiPhiPGainLimitOutput};
186
187 % Filter Gain Limit
188
189 % We constrain our filter to be as aggressive as possible
```

```
190
191 NGainLimit = 1;
192
193 NGainLimitInput = 'SkywalkerX8_Lateral_Control/Gain Scheduled ...
       Controller Chi/PIDF Controller Va_theta/Sum/De*';
194 NGainLimitOutput = 'SkywalkerX8_Lateral_Control/Gain Scheduled ...
       Controller Chi/PIDF Controller Va_theta/Product1/NDe*';
195
196 NGainLimitLoopOpening = {NGainLimitInput,...
197                          NGainLimitOutput};
198
199 R6 = TuningGoal.Gain(NGainLimitInput, NGainLimitOutput, NGainLimit);
200 R6.Openings = NGainLimitLoopOpening;
201 R6.Focus = [1E-02 1E02]; %Only focus on this frequency band
202
203 ST0.setBlockParam('PIDF Controller Chi Kp 2D Lookup Table', Kp);
204 ST0.setBlockParam('PIDF Controller Chi Ki 2D Lookup Table', Ki);
205 ST0.setBlockParam('PIDF Controller Chi Kd 2D Lookup Table', Kd);
206 ST0.setBlockParam('PIDF Controller Chi N 2D Lookup Table', N);
207
208 STChi = systune(ST0, [R1, R2, R6], [R3, R4, R5], sysTuneOpts);
209
210 tableValues = getBlockValue(STChi);
211
212 SkywalkerX8.Control.Lateral.ChiController.KpChiTF = ...
       tableValues.PIDF_Controller_Chi_Kp_2D_Lookup_Table;
213 SkywalkerX8.Control.Lateral.ChiController.KiChiTF = ...
       tableValues.PIDF_Controller_Chi_Ki_2D_Lookup_Table;
214 SkywalkerX8.Control.Lateral.ChiController.KdChiTF = ...
       tableValues.PIDF_Controller_Chi_Kd_2D_Lookup_Table;
215 SkywalkerX8.Control.Lateral.ChiController.NChiTF = ...
       tableValues.PIDF_Controller_Chi_N_2D_Lookup_Table;
216
217 SkywalkerX8.Control.Lateral.ChiController.KpChi = squeeze(...
       tableValues.PIDF_Controller_Chi_Kp_2D_Lookup_Table);
218 SkywalkerX8.Control.Lateral.ChiController.KiChi = squeeze(...
       tableValues.PIDF_Controller_Chi_Ki_2D_Lookup_Table);
219 SkywalkerX8.Control.Lateral.ChiController.KdChi = squeeze(...
       tableValues.PIDF_Controller_Chi_Kd_2D_Lookup_Table);
220 SkywalkerX8.Control.Lateral.ChiController.NChi = squeeze(...
       tableValues.PIDF_Controller_Chi_N_2D_Lookup_Table);
221
222 % Get scheduling parameters and write them to our desired variables
223 % Note thase these are based on our basis function.
224
225 TGS = getBlockParam(STChi);
226 SkywalkerX8.Control.Lateral.ChiController.BasisFunction = ...
       TGS.PIDF_Controller_Chi_Kp_2D_Lookup_Table.BasisFunctions;
227 SkywalkerX8.Control.Lateral.ChiController.KpChiParameterizedGains = ...
       getData(TGS.PIDF_Controller_Chi_Kp_2D_Lookup_Table);
228 SkywalkerX8.Control.Lateral.ChiController.KiChiParameterizedGains = ...
       getData(TGS.PIDF_Controller_Chi_Ki_2D_Lookup_Table);
```

B.3 Guidance and Navigation Implementation

B.3.1 Straight-Line Following Algorithm

Listing B.7: Straight line following algorithm, straightLineFollowing.m

```
function [h_c, chi_c] = straightLineFollowing(r, q, p, chi, chi_inf,...
    kpath)

%% Setup

% Inputs are all in inertial co-ordinates

% Variables

% r = origin of the path [rn, re, rd] (m)
% q = desired course path (unit vector) [qn, qe, qd] (m)
% p = current aircraft location [pn, pe, pd] (m)
% chi = current course angle (rad)

% Constants

% chi_inf = maximum course angle command magnitude when infinitely ...
    far from the path we want to follow (rad)
% kpath = relative path error scaling factor for course angle ...
    command
% transitions from chi_inf to chi_q (unitless 0->1)

% Splitting up inputs into constituent elements (in inertial NED ...
    frame)

rn = r(1);
re = r(2);
rd = r(3);

pn = p(1);
pe = p(2);
pd = p(3);

qn = q(1);
qe = q(2);
qd = q(3);

%% Altitude Command (h_c)

% Relative path error in the inertial frame

ei = p - r;

% Unit vector normal to q-k plane (k = unit vector in the inertial D
```

```
% direction; ie k = (0, 0, 1))

k = [0, 0, 1];

n = cross(q, k)'/norm(cross(q, k));

% Projection of relative path error in the path plane onto the ...
    vertical
% plane containing the path directon vector (ie. q-k plane)

si = ei - dot(ei, n)*n;

sn = si(1);
se = si(2);

if sqrt(qn^2 + qe^2) ≥ 1E-06

    h_c = -rd - sqrt(sn^2 + se^2)*(qd/(sqrt(qn^2 + qe^2)));

else

    h_c = -rd;

end

%% Course Angle Command (Chi_c)

% Course angle of straight line defined by unit vector q. We adjust ...
    this to
% ensure that -pi ≤ chi_q - chi ≤ pi

chi_q = atan2(qe, qn);

while (chi_q - chi) < -pi
   chi_q = chi_q + 2*pi;
end

while (chi_q - chi) > pi
   chi_q = chi_q - 2*pi;
end

% Inertial to path frame rotation matrix

Rip = [cos(chi_q) , sin(chi_q), 0;
       -sin(chi_q), cos(chi_q), 0;
       0          , 0         , 1];

% Relative path error expressed in the path frame

ep = Rip*ei;

```

```
90 % Calculating Chi_c by using a vector field that asymptotically ...
       decays to
91 % the desired course angle
92
93 epy = ep(2);
94
95 chi_c = chi_q - chi_inf*(2/pi)*atan(kpath*epy);
96
97 end
```

B.3.2 Orbit Following Algorithm

Listing B.8: Orbit following algorithm, orbitFollowing.m

```
1  function [h_c, chi_c] = orbitFollowing(c, rho, lambda, p, chi, ...
       korbit)
2
3  %% Setup
4
5  % Inputs are all in inertial co-ordinates
6
7  % Variables
8
9  % c = center of the orbit we want to follow [cn, ce, cd] (m)
10 % rho = radius of the orbit we want to follow (m)
11 % lambda = direction of the orbin we want to follow (+1 clockwise, ...
       -1 counter-clockwise)
12 % p = current aircraft location [pn, pe, pd] (m)
13 % chi = current course angle (rad)
14
15 % Constants
16
17 % korbit = scalar for quickly we transition from moving toward the ...
       center
18 % of an orbit (the path we take when infinitely far from c) and to ...
       the
19 % ideal course angle command based on the orbit (ie. the course ...
       angle
20 % needed to maintain the orbit radius from the center)
21
22 % Splitting up inputs into constituent elements (in inertial NED ...
       frame)
23
24 cn = c(1);
25 ce = c(2);
26 cd = c(3);
27
28 pn = p(1);
29 pe = p(2);
30 pd = p(3);
```

```
31
32 %% Altitude Command (h_c)
33
34 % Our altitude command is simply the altitude of the orbit center
35
36 h_c = -cd;
37
38 %% Course Angle Command (chi_c)
39
40 d = sqrt((pn - cn)^2 + (pe - ce)^2);
41 phi = atan2( (pe - ce), (pn - cn) );
42
43 while (phi - chi) < -pi
44    phi = phi + 2*pi;
45 end
46
47 while (phi - chi) > pi
48    phi = phi - 2*pi;
49 end
50
51 chi_c = phi + lambda*(pi/2 + atan(korbit*(d - rho)/rho));
52
53 end
```

B.3.3 Path Management Algorithm

Listing B.9: Path management fillets algorithm, pathManager_followWaypointsFillet.m

```
1  function [r, qOut, c, rho, lambda, flag, i] = ...
       pathManager_followWaypointsFillet(W, WChangeCheck, p, R, ...
       prev_state, prev_i)
2
3  %% Setup
4
5  % Inputs are all in inertial co-ordinates
6
7  % Variables
8
9  % W = Waypoint path (array of Waypoints arrays) [W1, W2, W3... WN] (...
       Note 1
10 % indexed)
11 % Where:
12 % WN = [WNn, WNe, WNd] (m)
13 % Wprev = Previous Waypoint path (to check if the Waypoint path has ...
       been
14 % changed sinced the last iteration)
15 % p = current aircraft location [pn, pe, pd] (m)
16 % R = radius of the orbit we want to follow (m)
17
```

```
% Outputs

% flag = straight line or orbit followWing (1 = straight line, 2 = ...
    orbit
% followWing)
% r = origin of the path [rn, re, rd] (m)
% q = desired course path (unit vector) [qn, qe, qd] (m)
% c = center of the orbit We Want to followW [cn, ce, cd] (m)
% rho = radius of the orbit We Want to followW (m)
% lambda = direction of the orbin We Want to followW (+1 clockWise, ...
    -1 counter-clockWise)

state = prev_state;
i = prev_i;

r = zeros(3, 1);
qOut = zeros(3, 1);
c = zeros(3, 1);
travPlaneNormal = zeros(3, 1);
travPlanePoint = zeros(3, 1);
x = 0;
y = 0;
h_c = 0;
rho = 0;
lambda = 0;
flag = 0;

q = zeros(3, 1, size(W, 3));
lengths = zeros(size(W, 3), 1);
N = size(W, 3);

if (i + 1) > N
   iPlusOne = (i + 1) - N + 1;
else
   iPlusOne = i + 1;
end

if not(WChangeCheck)
   for k = 1:length(size(W, 3))
    lengths(k) = norm(W(:, :, k) - p);
   end
   [¬, ibuff] = min(lengths);
   i = min(ibuff) - 1;
   state = 1;
else

if norm( W(:, :, i) - W(:, :, i - 1) ) ≥ 1E-06
    q(:, :, i - 1) = ( W(:, :, i) - W(:, :, i - 1) )/norm( W(:, :, i...
        ) - W(:, :, i - 1) );
else
    q(:, :, i - 1) = zeros(3, 1);
end
```

```
67
68  if norm( W(:, :, iPlusOne) - W(:, :, i) ) ≥ 1E-06
69      q(:, :, i) = ( W(:, :, iPlusOne) - W(:, :, i) )/norm( W(:, :, ...
            iPlusOne) - W(:, :, i) );
70  else
71      q(:, :, i) = zeros(3, 1);
72  end
73
74  fillAngle = acos(-q(:, :, i - 1)'*q(:, :, i));
75
76  if state == 1
77
78      flag = 1;
79      r = W(:, :, i - 1);
80      qOut = q(:, :, i - 1);
81      z = W(:, :, i) - (R/tan(fillAngle/2))*q(:, :, i - 1);
82
83      if halfPlaneCheck(p, z, q(:, :, i - 1))
84
85          flag = 2;
86
87      end
88
89  else %if state == 2
90
91      flag = 2;
92
93      if norm( q(:, :, i - 1) - q(:, :, i) ) ≥ 1E-06
94          c = W(:, :, i) - (R/sin(fillAngle/2))*( q(:, :, i - 1) - q...
                (:, :, i) )/norm( q(:, :, i - 1) - q(:, :, i) );
95      else
96          c = W(:, :, i);
97      end
98
99      % In order to make our transition between line segments more ...
            graceful,
100     % we generate a plane between line segments and use our current
101     % position to determine what our ideal position should be to ...
            remain on
102     % this plane. We do this by making use of the standard equation:
103
104     % m(x - x0) + n(y - y0) + o(z - z0) = 0
105
106     % Where a, b and c make up our normal vector and x0, y0 and z0 ...
            makes up
107     % a point on the plane (which we know is c.
108
109     travPlaneNormal = cross(q(:, :, i), q(:, :, i - 1));
110     travPlanePoint = W(:, :, i);
111
112     m = travPlaneNormal(1);
113     n = travPlaneNormal(2);
```

```
114     o = travPlaneNormal(3);
115
116     x0 = travPlanePoint(1);
117     y0 = travPlanePoint(2);
118     z0 = travPlanePoint(3);
119
120     x = p(1);
121     y = p(2);
122
123     Neg_h_c = ((m*(x0 - x) + n*(y0 - y))/o + z0); %negative because ...
            conversion to alt takes place in orbit block
124
125     c(3, 1) = Neg_h_c;
126
127     if q(3, :, i - 1) > 0 %We're ascending
128
129         if ( c(3,1) ≥ W(3, :, i) ) && ( q(3, :, i) == 0 ) %If we're ...
                above the next required altitude and the next line ...
                segment is an altitude hold segment
130
131             c(3, 1) = W(3, :, i);
132
133         end
134
135     end
136
137     if q(3, :, i - 1) < 0 %We're descending
138
139         if ( c(3,1) ≤ W(3, :, i) ) && ( q(3, :, i) == 0 ) %If we're ...
                below the next required altitude and the next line ...
                segment is an altitude hold segment
140
141             c(3, 1) = W(3, :, i);
142
143         end
144
145     end
146
147     rho = R;
148     lambda = sign(q(1, 1, i - 1)*q(2, 1, i) - q(2, 1, i - 1)*q(1, 1,...
            i));
149     z = W(:, :, i) + R/tan(fillAngle/2)*q(:, :, i);
150
151     if halfPlaneCheck(p, z, q(:, :, i))
152
153         i = i + 1;
154
155         if i > N
156             i = 2;
157         end
158
159         state = 1;
```

```
160           flag = 1;
161
162       end
163
164   end
165
166   end
```