Dynamic Modeling and LQR Control of an Aerodynamic System

Thomas W. C. Carlson
EEE246 - Discrete Control Systems, Prof. Tatro

5-12-2021

Contents

List of Figures

List of Tables

1 Introduction

This paper represents a semester-long effort to demonstrate competency in the modeling and control of a dynamic system using a controller designed via the Linear Quadratic Regulator (LQR) method. At the onset of the semester the declared intent of the project was to implement a Thrust Vector Control (TVC) system on a simplified model of a rocket subject to standard 6 degree of freedom (DoF) dynamics to reject a disturbance. However, as the semester and my understanding of forces in aerodynamic systems progressed, it would prove nearly impossible to spin up a workable model of a simple rocket without relying on external data regarding flight control derivatives, lift and drag coefficients, and the dynamic pressure experienced by a generic object in flight.

As a result, the direction shifted to rely on available data for more well-understood systems. Literature suggests that airplanes are more complex to model, but the wealth of public knowledge offsets this difficulty. One model in particular, the Research Civil Aircraft Model (RCAM), represents a plane similar in design and specification to the Boeing 720 with fully public specifications for the express purpose of control system design. The parameters provided in a paid, private copy of [1] were the ones selected for modeling, trimming, linearization, and subsequent LQR controller design of the project. They differ slightly, but not meaningfully, from the ones presented in the public-access version.

This paper is presented in 3 sections. Section 2 covers the modeling theory and relevant steps toward the development of the state space model of the system, followed by a discussion of LQR controller development techniques. Section 3 discusses the implementation of the discussed theory in MATLAB and Simulink to both create the system to be controlled and develop the control law. Section 4 summarizes results and presents my conclusions as well as discussing future work and methods to improve the behavior of the model.

2 Theory

The vast majority of the effort expended by a controls engineer in the development of a control system for any given set of dynamics is focused on the development of an accurate and usable dynamic model of the system to be controlled. This remains true in the case of this project, where the system in question is that of an airplane model with 6 DoF.

For reference in future sections, Table 1 contains a list of the constant values provided in [1] which describe the RCAM aircraft. Table 2 contains a list of intermediate calculated variables useful for shorthand description of the equations of motion.

Variable	Description	Value	Units
Engine Parameters			
X_{APT1}	x position of engine 1 thrust application point	0.0	m
Y_{APT1}	y position of engine 1 thrust application point	7.94	m
Z_{APT1}	z position of engine 1 thrust application point	1.9	m
X_{APT2}	x position of engine 2 thrust application point	0.0	m
Y_{APT2}	y position of engine 2 thrust application point	-7.94	m
Z_{APT2}	z position of engine 2 thrust application point	1.9	m
Aerodynamic Parameters			
l	Airfoil generalised length	6.6	m
l_t	Tail distance from Center of Gravity	24.8	m
S	Wing planform area	260.0	m^2
S_t	Tail planform area	64.0	m^2

Table 1: RCAM Model Constants and Description

Variable	Description	Units
V_A	Total Airspeed	m/s
α	Angle of Attack	rad
β	Side Slip Angle	rad
Q	Dynamic Pressure	Pa

Table 2: RCAM Intermediate Values and Description

2.1 System Dynamics

As discussed in [1], the RCAM model of an airplane is fully characterized by 9 state elements and controlled by 5 inputs. A summary of these states and inputs is given in Table 3. Exclusion of the XYZ position in earth-fixed coordinates is intentional as the control system will not attempt to drive the system to a specific position.

State-Space Label	Free-body Diagram Label	Description	Units
x_1	u	Body-fixed forward velocity	m/s
x_2	v	Body-fixed lateral velocity	m/s
x_3	w	Body-fixed vertical velocity	m/s
x_4	p	Body-fixed roll rate	rad/s
x_5	q	Body-fixed pitch rate	rad/s
x_6	r	Body-fixed roll rate	rad/s
x_7	ϕ	Referential roll angle	rad
x_8	θ	Referential pitch angle	rad
x_9	ψ	Referential yaw angle	rad
u_1	δ_a	Aileron deflection angle	rad
u_2	δ_e	Elevator deflection angle	rad
u_3	δ_r	Rudder deflection angle	rad
u_4	δ_{TH_1}	Engine 1 Throttle	-
u_5	δ_{TH_2}	Engine 2 Throttle	-

Table 3: RCAM Model States and Physical Meaning

A visualization of the system states and the control inputs is given in Figures 1 and 2. These figures represent the body-fixed coordinate system. Also present on Figure 1 are the body-fixed axis-aligned forces and moments the aircraft is subject to in flight. Their physical meanings are summarized in Table 4.

Label	Description	Typical Physical effect
X	Sum of forces in body-fixed x direction	Surge
Y	Sum of forces in body-fixed y direction	Sway
Z	Sum of forces in body-fixed z direction	Heave
L	Sum of moments about body-fixed x-axis	Roll
M	Sum of moments about body-fixed y-axis	Pitch
N	Sum of moments about body-fixed z-axis	Yaw

Table 4: Summary of forces and moments

The equations of motion which define the system dynamics and therefore the states of aerodynamic flight are well defined in literature and are given by equations (1)-(3). They are developed under the assumption that the body in motion is rigid and symmetric. The constants m and I represent the mass of the rigid body and the inertia tensor of the body respectively. In general, the product moment of inertias involving the y-axis can be treated as zero due to symmetry, leading to the expression given in (4). The derivation of these equations from first principles is somewhat arduous and is left for the appendices.

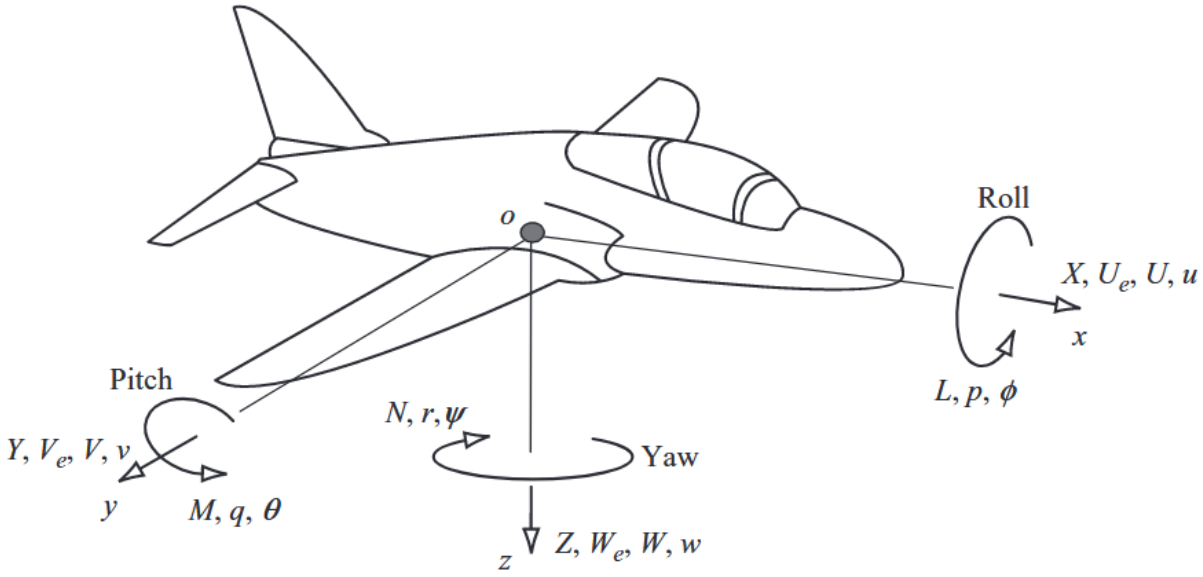


Figure 1: Body-fixed axis states, forces, and moments.

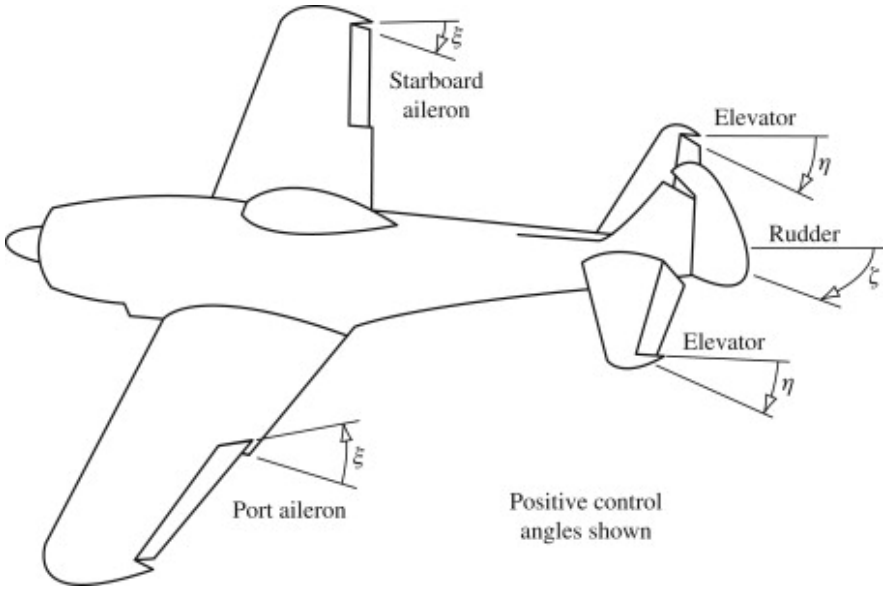


Figure 2: Control surfaces and deflection angles.

$$\bar{A}_b = \begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix} = \frac{1}{m} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} - \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times \begin{bmatrix} u \\ v \\ w \end{bmatrix} \tag{1}$$

$$\dot{\bar{\omega}}_b = \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \frac{1}{I} \left(\begin{bmatrix} L \\ M \\ N \end{bmatrix} + \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times \left(I \begin{bmatrix} p \\ q \\ r \end{bmatrix} \right) \right) \tag{2}$$

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi\sec\theta & \cos\phi\sec\theta \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{3}$$

$$I = \begin{bmatrix} I_x & 0 & -I_{xz} \\ 0 & I_y & 0 \\ -I_{xz} & 0 & I_z \end{bmatrix} \tag{4}$$

Integration of the results of these equations returns the states of the system that are desired.

Characterization of the sum of the disturbance forces X, Y, Z and moments L, M, N is all that remains in order to fully define the system. For the rigid body model of airplane, we consider disturbance forces which arise from **a**erodynamic forces, **g**ravitational forces, and **p**ropulsion forces and their resulting moments.

2.1.1 Aerodynamic Forces

The effect of aerodynamic forces can be separated into three primary behaviors: lift (L), drag (D), and sideforce (Y). These forces lie along the axes of the stability frame and must be transformed to find their components along the body-fixed axes (represented by the b superscript) of the RCAM plane. These forces are nontrivial to calculate and rely on expressions provided by [1] found to be generally accurate for the specific characteristics of the wing and body of the RCAM plane for given angles of attack, fluid density, and relative airspeed. Expression of the coefficients C_D, C_Y, and C_L is left to the appendix, but the resulting aerodynamic forces are given by (5), where α represents the angle of attack of the plane, ρ the air density, and V_A the airspeed magnitude. The constant S is the wing planform area, a property of the airfoil design.

$$\bar{F}_a^b = \begin{bmatrix} F_{A_x}^b \\ F_{A_y}^b \\ F_{A_z}^b \end{bmatrix} = \begin{bmatrix} \cos\alpha & 0 & -\sin\alpha \\ 0 & 1 & 0 \\ \sin\alpha & 0 & \cos\alpha \end{bmatrix} \begin{bmatrix} -\frac{1}{2}C_D\rho V_A^2 S \\ \frac{1}{2}C_Y\rho V_A^2 S \\ -\frac{1}{2}C_L\rho V_A^2 S \end{bmatrix} \tag{5}$$

2.1.2 Aerodynamic Moments

Similar formulations are provided for moments due to the aerodynamic forces, reliant on wingspan b and the coefficients C_l, C_m, and C_n, as shown in (6). Moment is transferred from the aerodynamic center (X_{AC},Y_{AC},Z_{AC}) to the body-fixed axis origin (X_{CG},Y_{CG},Z_{CG}) using the parallel axis theorem, as shown in the appendix. $\bar{C}^b_{M_{ac}}$ is a complex vector of coefficients defined in the appendix.

$$\bar{M}^b_{a_{cg}} = \begin{bmatrix} L_A \\ M_A \\ N_A \end{bmatrix} = \bar{C}^b_{M_{ac}} QS\bar{b} + \bar{F}^b_A \times \left(\begin{bmatrix} X_{CG} \\ Y_{CG} \\ Z_{CG} \end{bmatrix} - \begin{bmatrix} X_{AC} \\ Y_{AC} \\ Z_{AC} \end{bmatrix} \right) \tag{6}$$

2.1.3 Propulsion Forces

The twin engines of the RCAM plane are identical and each exert a thrust force characterized by [1] as a fractional expression of the gravitational force acting on the aircraft, tuned so that the maximum setting results in a thrust over weight ratio of approximately 0.35, which is common in aviation. The engines are aligned with the body-axis forward direction, and so exert a force only along the x-axis.

$$\bar{F}^b_p = \begin{bmatrix} F^b_{p_x} \\ F^b_{p_y} \\ F^b_{p_z} \end{bmatrix} = \bar{F}^b_{p_1} + \bar{F}^b_{p_2} = \begin{bmatrix} \delta_{TH_1} mg \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} \delta_{TH_2} mg \\ 0 \\ 0 \end{bmatrix} \tag{7}$$

2.1.4 Propulsion Moments

Because each engine is symmetrically mounted a known distance away from the body-fixed axis origin, the engines exert a moment upon the aircraft in the usual fashion.

$$\bar{M}^b_p = \begin{bmatrix} L_p \\ M_p \\ N_p \end{bmatrix} = r \times \bar{F}^b_{p_1} + r \times \bar{F}^b_{p_2} = \begin{bmatrix} X_{APT1} \\ Y_{APT1} \\ Z_{APT1} \end{bmatrix} \times \begin{bmatrix} \delta_{TH_1} mg \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} X_{APT2} \\ Y_{APT2} \\ Z_{APT2} \end{bmatrix} \times \begin{bmatrix} \delta_{TH_2} mg \\ 0 \\ 0 \end{bmatrix} \tag{8}$$

2.1.5 Gravitation Forces

Gravity is considered to be constant in the model. Because it acts through the center of gravity of the RCAM plane it exerts no moment. The force of gravity, however, is found in the earth reference frame and must be rotated to the RCAM body-fixed axes in order to be properly applied to the model. The direction cosine matrix of euler angles is used to perform this rotation. It has been derived in the appendix.

$$\bar{F}_g^b = C_{b/e}(\phi, \theta, \psi)\bar{F}_g^e = m \begin{bmatrix} -g\sin\theta \\ g\cos\theta\sin\phi \\ g\cos\theta\cos\phi \end{bmatrix} \tag{9}$$

2.1.6 Wind Forces

Wind forces were considered negligible for this project.

2.1.7 The Full Equations of Motion

With the system fully characterized, it is now possible to establish the complete equations of motion to be used in modeling the system.

$$\dot{u} = \frac{F_{A_x}^b + F_{P_x}^b + F_{g_x}^b}{m} - (qw - rv) \tag{10}$$

$$\dot{v} = \frac{F_{A_y}^b + F_{P_y}^b + F_{g_y}^b}{m} - (ru - pw) \tag{11}$$

$$\dot{w} = \frac{F_{A_z}^b + F_{P_z}^b + F_{g_z}^b}{m} - (pv - qu) \tag{12}$$

$$\dot{p} = \frac{L_A + L_p - qr(I_z - I_y)}{I_x} \tag{13}$$

$$\dot{q} = \frac{M_A + M_p - rp(I_x - I_z)}{I_y} \tag{14}$$

$$\dot{r} = \frac{N_A + N_p - pq(I_y - I_x)}{I_z} \tag{15}$$

$$\dot{\phi} = p + (q\sin\phi + r\cos\phi)\tan\theta \tag{16}$$

$$\dot{\theta} = q\cos\phi - r\sin\phi \tag{17}$$

$$\dot{\psi} = \frac{q\sin\phi + r\cos\phi}{\cos\theta} \tag{18}$$

From these expressions it is immediately clear that the system is highly nonlinear. States such as q and w are multiplied together in (10), and the Euler angle equations contain trigonometric terms. This system cannot be directly converted to state space as a result, prohibiting the use of control law design techniques such as LQR. As such, the system must first be linearized before further analysis can proceed.

2.1.8 Control Saturation

Most real-world applications of controls have physical limits—actuators may only extend so far, and a jet engine may only provide so much thrust. Similarly, deflection surfaces on the RCAM model as well as the thrust sources propelling the plane have physical limits. [1] provides these saturation values:

$$\begin{aligned}
\text{Thrust Limits:} &\quad +0.5 \leq \delta_{TH} \leq 1 \\
\text{Saturation of Aileron deflection:} &\quad -25 \leq \delta_A \leq 25 \text{ deg} \\
\text{Saturation of Elevator deflection:} &\quad -25 \leq \delta_E \leq 10 \text{ deg} \\
\text{Saturation of Rudder deflection:} &\quad -25 \leq \delta_R \leq 30 \text{ deg}
\end{aligned}$$

2.2 Linearization of the Equations of Motion

There are several methods available in literature to linearize a nonlinear dynamic system. All of them require the selection of a "trim", or known controlled behavior point, around which to examine the behavior of the system. The basis of this theory is that by "zooming in" enough on a selected point, the dynamics of the system appear linear and a comparable linear system may be developed which is resilient to small perturbations about the trim point.

2.2.1 Seeking a Trim Point

The first step in identifying a trim point around which to linearize a system is to identify a physical condition of the system in which the dynamics are stable but not necessarily unchanging. For the case of an airplane, a common trim point is steady rectilinear flight. This is characterized by a constant forward velocity (u) and zero sideslip or downward velocity as well as zero roll, pitch, and yaw rates. That is, the state vector at a trim point of steady rectilinear flight may look like (19).

$$\dot{\bar{x}} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \\ x_9 \end{bmatrix} = \begin{bmatrix} u \\ v \\ w \\ p \\ q \\ r \\ \phi \\ \theta \\ \psi \end{bmatrix} = \begin{bmatrix} 85m/s \\ 0m/s \\ 0m/s \\ 0rad/s \\ 0rad/s \\ 0rad/s \\ 0rad \\ 0rad \\ 0rad \end{bmatrix} \tag{19}$$

It should be noted that this is distinct from an equilibrium point, which refers to a state in which the system is experiencing zero change, and in the case of the RCAM model every state would be zero. There may also be some additional constraints or solutions to intermediate variables. For instance, in straight and level rectilinear flight we would expect the sideslip angle rate γ to be zero, and the overall airspeed should be equal to the forward velocity so as to ensure no lateral or vertical motion.

$$V_A = 85 m/s \tag{20}$$
$$\gamma = 0 rad/s \tag{21}$$

It is now the job of the engineer to determine what control inputs can minimize the difference between the simulated operating point closest to the trim point and the trim point itself. This is a numeric optimization process which can be performed algorithmically using a penalty or cost function. Much like a control system, the algorithm seeks to minimize differences between a reference and output set of variables. A function in terms of the related penalty functions is constructed and evaluated by substituting in the complex expressions for the state variables derived in Section 2.1.7.

$$\hat{f}_o = \alpha_1 \dot{x}_1^2 + \alpha_2 \dot{x}_2^2 + \alpha_3 \dot{x}_3^2 + \alpha_4 \dot{x}_4^2 + \dots \tag{22}$$

The function is then re-expressed in terms of a set of inputs $\bar{z}$, which is a much more complex expression than the original quadratic form of (22).

$$\bar{z} = \begin{bmatrix} \bar{x} \\ \bar{u} \end{bmatrix} \tag{23}$$

It is known that the resulting function is always positive, and indeed only obtains a value of zero when all constraints are satisfied. As a result this function can be searched for its absolute minimum value for given constraints. The absolute minimum is the set of states which is closest to the desired trim state. The task becomes to evaluate the function across a large range of inputs and obtain the inputs which result in the best trim values.

For each evaluated point, a difference matrix Q is calculated from the states and the estimated control parameters. It is then multiplied by its transpose as well as a penalty matrix H to form the quadratic expression penalized based on the operator's preference. The value of this quadratic expression is the value to optimize, as the objective is to minimize the difference matrix Q, thus minimizing the value of the expression as a whole.

2.2.2 Linearization of the System

With a trim point obtained, it is now possible to obtain the linear dynamics of the system about the trim point. This is most readily done via a Taylor Series expansion of the functions about the trim point. The Taylor Series expansion is a numerical approximation of the value of a given function a point. For a 1-dimensional Taylor Series expansion, successively higher order partial derivatives are used to narrow the approximation as shown.

$$F(x) = F(x_o) + \frac{\partial F(x_o)}{\partial x}(x - x_o) + \frac{\partial^2 F(x_o)}{\partial x^2}\frac{(x - x_o)^2}{2!} + \dots \tag{24}$$

Typically, the linearization process does not consider the terms of higher order than 1. This approach generalizes to n-dimensions so long as the trim point is expressed as a vector $\bar{\eta_o} = \begin{bmatrix}\dot{\bar{x}} & \bar{x} & \bar{u}\end{bmatrix}^T$.

$$F(\bar{\eta}) = F(\eta_o) + \frac{\partial F(\bar{\eta_o})}{\partial \bar{\eta}}(\bar{\eta} - \bar{\eta_o}) + H.O.T \tag{25}$$

Equivalently,

$$F(\bar{\eta}) = F(\dot{\bar{x}}, \bar{x}, \bar{u}) + \frac{\partial F(\dot{\bar{x}}, \bar{x}, \bar{u})}{\partial \dot{\bar{x}}}(\dot{\bar{x}} - \dot{\bar{x}}_o) + \frac{\partial F(\dot{\bar{x}}, \bar{x}, \bar{u})}{\partial \bar{x}}(\bar{x} - \bar{x}_o) + \frac{\partial F(\dot{\bar{x}}, \bar{x}, \bar{u})}{\partial \bar{u}}(\bar{u} - \bar{u}_o) \tag{26}$$

Where each partial derivative of the vector-valued function $F(\dot{\bar{x}}, \bar{x}, \bar{u})$ is a Jacobian of partial derivatives of dimension $n \times n$ or $n \times m$ in the case of the controls Jacobian as shown

$$E = \frac{\partial F(\dot{\bar{x}}, \bar{x}, \bar{u})}{\partial \dot{\bar{x}}} = \left| \begin{bmatrix} \frac{\partial F_1}{\partial \dot{x}_1} & \frac{\partial F_1}{\partial \dot{x}_2} & \cdots & \frac{\partial F_1}{\partial \dot{x}_n} \\ \frac{\partial F_2}{\partial \dot{x}_1} & \frac{\partial F_2}{\partial \dot{x}_2} & \cdots & \frac{\partial F_2}{\partial \dot{x}_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial F_n}{\partial \dot{x}_1} & \frac{\partial F_n}{\partial \dot{x}_2} & \cdots & \frac{\partial F_n}{\partial \dot{x}_n} \end{bmatrix} \right|_{\dot{\bar{x}}=\dot{\bar{x}}_o, \bar{x}=\bar{x}_o, \bar{u}=\bar{u}_o} \tag{27}$$

$$A' = \frac{\partial F(\dot{\bar{x}}, \bar{x}, \bar{u})}{\partial \bar{x}} = \left| \begin{bmatrix} \frac{\partial F_1}{\partial x_1} & \frac{\partial F_1}{\partial x_2} & \cdots & \frac{\partial F_1}{\partial x_n} \\ \frac{\partial F_2}{\partial x_1} & \frac{\partial F_2}{\partial x_2} & \cdots & \frac{\partial F_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial F_n}{\partial x_1} & \frac{\partial F_n}{\partial x_2} & \cdots & \frac{\partial F_n}{\partial x_n} \end{bmatrix} \right|_{\dot{\bar{x}}=\dot{\bar{x}}_o, \bar{x}=\bar{x}_o, \bar{u}=\bar{u}_o} \tag{28}$$

$$B' = \frac{\partial F(\dot{\bar{x}}, \bar{x}, \bar{u})}{\partial \bar{u}} = \left| \begin{bmatrix} \frac{\partial F_1}{\partial u_1} & \frac{\partial F_1}{\partial u_2} & \cdots & \frac{\partial F_1}{\partial u_m} \\ \frac{\partial F_2}{\partial u_1} & \frac{\partial F_2}{\partial u_2} & \cdots & \frac{\partial F_2}{\partial u_m} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial F_n}{\partial u_1} & \frac{\partial F_n}{\partial u_2} & \cdots & \frac{\partial F_n}{\partial u_m} \end{bmatrix} \right|_{\dot{\bar{x}}=\dot{\bar{x}}_o, \bar{x}=\bar{x}_o, \bar{u}=\bar{u}_o} \tag{29}$$

We regard the subtracted terms in the Taylor Series expansion as an expression of the distance from the trim point at which the evaluation of the system dynamics is taking place, and so conveniently express them in Δ notation. Further, $F(\eta_o)$ being an expression of the value of the vector-valued function at the trim point should be equal to zero. This leads to an expression which looks nearly familiar to the state space representation being sought.

$$0 \approx E\Delta\dot{\bar{x}} + A'\Delta\bar{x} + B'\Delta\bar{u} \tag{30}$$

Rearranging terms leads to

$$\Delta\dot{\bar{x}} \approx -E^{-1}A'\Delta\bar{x} - E^{-1}B'\Delta\bar{u} = A\Delta\bar{x} + B\Delta\bar{u} \tag{31}$$

Which is an expression of a linear state space model of the system valid for small perturbations about the trim point.

The evaluation of the partial derivatives listed in equations (27-29) is nontrivial to perform analytically due to the complexity of the full expression of the functions. It is generally best to approach their evaluation numerically, although this clouds the effect of certain variables on the system.

2.2.3 The Symmetric Difference Quotient

Again relying on the idea that the function is treated as linear for perturbations about the trim point, the derivative of a function corresponds to the slope of the function at the point of evaluation. This can be expressed via the slope formula for a line using two perturbations in opposite directions along the function. For example, for the (i,j)th element of A', a point in the negative direction, $\bar{x}_{ij-}$, and in the positive direction, $\bar{x}_{ij+}$, are used—each a distance Δx_{ij} from the point of interest.

$$\bar{x}_{ij+} = \begin{bmatrix} x_{1_o} & x_{2_o} & \dots & x_{j_o} + \Delta x_{ij} & \dots & x_{n_o} \end{bmatrix} \tag{32}$$

$$\bar{x}_{ij-} = \begin{bmatrix} x_{1_o} & x_{2_o} & \dots & x_{j_o} - \Delta x_{ij} & \dots & x_{n_o} \end{bmatrix} \tag{33}$$

$$A'(i,j) = \frac{\partial F_i(\bar{\eta}_o)}{\partial x_j} \approx \frac{F_i(\dot{\bar{x}}, \bar{x}_{ij+}, \bar{u}) - F_i(\dot{\bar{x}}, \bar{x}_{ij-}, \bar{u})}{2\Delta x_{ij}} \tag{34}$$

This will need to be repeated for each element of each Jacobian matrix in order to obtain the final state-space representation of the nonlinear dynamic system about the trim point.

2.2.4 The State Space System

In summary, using a trim point and by linearizing the equations of motion about the trim point a state-space representation has been developed. For simplicity of notation, the Δ notation will be

dropped with the understanding that the model if only valid for sufficiently small perturbations about the trim point. The output matrix C is the identity matrix of size $n \times n$ to express every state as an output of the system.

$$\dot{\bar{x}} = A\bar{x} + B\bar{u} \tag{35}$$
$$\bar{y} = C\bar{x} \tag{36}$$

2.3 Linear Quadratic Regulator Theory

Linear Quadratic Regulator control theory rests upon the objective of finding an optimal gain matrix K which drives the system to a commanded set point in a fashion that minimizes a cost function J. This cost function is defined in terms of state and control weighting matrices that define the "expense" of manipulating variables in the system. The following sections summarize the design process and theory behind the development of an LQR controller for a linear state-space system described by (37).

$$\dot{x} = Ax + Bu \tag{37}$$

2.3.1 Weighting Matrices

In any dynamic system model the default assumption is that all control inputs are equally expensive and that all states are equally important to control. However, many applications of control systems are not well-represented by this assumption. For instance, if a control input is a motor with a weak torque it may be considered more expensive to drive the motor to rated torque. In the case of flight, a relative cost of fuel could be established and related to the cost of driving the thrust control of an engine. These costs are expressed in a diagonal matrix R of dimension $m \times m$:

$$R = diag[W_{u_1}, W_{u_2}, W_{u_3}, W_{u_4}, W_{u_5}] \tag{38}$$

Similarly, the relative importance of a state being driven to the commanded value can be expressed in an $n \times n$ diagonal matrix Q of weighting values corresponding to each state. As an example, in a commercial aircraft with loose baggage and persons aboard, it may be considered more important to maintain steady roll and pitch angles as well as zero downward velocity, while allowing some sideslip.

$$Q = diag[W_{x_1}, W_{x_2}, W_{x_3}, W_{x_4}, W_{x_5}, W_{x_6}, W_{x_7}, W_{x_8}, W_{x_9}] \tag{39}$$

The matrices are then used in cost function to distort the calculated control law in a desired manner. It is not possible to empirically determine the "best" weighting matrices. The process

of refining control and state weighting matrices is an iterative one in which the designer needs to attempt to develop an understanding of how modifying certain weighting elements affects the system control inputs and state outputs.

2.3.2 The LQR Cost Function

The typical cost function for a continuous-time LQR function is expressed as in (40). It produces a quadratic expression which can be searched for a minimum value yielding the lowest cost to drive the system to the set point.

$$J = \int_0^\infty (x^T Q x + u^T R u) dt \tag{40}$$

The critical state can be found when the differential of $J = 0$. Using calculus of variation the problem can be brought into a form with a known solution, the Matrix Riccati Equation. Solution of the equation returns a matrix P which is then used to find the optimal controller gains K.

2.3.3 Optimal Control Law

Using the above, the optimal controller gains at a specific point in time and state of the system can be expressed as in (41).

$$K = R^{-1}(B_T P(t)) \tag{41}$$

These gains are then implemented in the control system using expression (42) which results in an expression of the error signal as in (43).

$$u = -Kx \tag{42}$$

$$e_{err} = K(x_{cmd} - x_{state}) \tag{43}$$

2.3.4 Simulink Implementation of LQR

Implementation of a state space model with LQR control in MATLAB's Simulink environment is straightforward. Using the known state space matrices A, B, C, as well as equations (37) and (43) the block diagram shown in Figure 3 is readily obtained.

The input x_{cmd} is the commanded state vector, while $delta_x$ represents the difference in commanded and current state to be multiplied by the LQR optimal gain matrix K. The initial condition is fed to the integrator block which converts the $\dot{\bar{x}}$ state derivative vector to the state vector x. $\dot{\bar{x}}$ and y are obtained by implementation of equations (35) and (36).

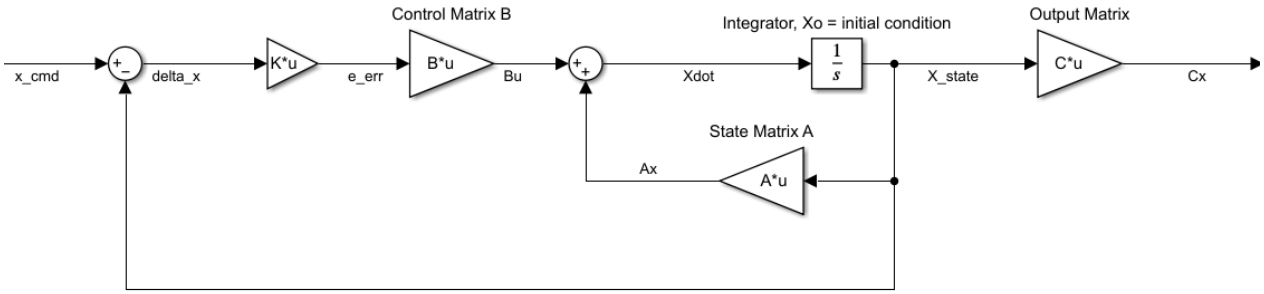


Figure 3: LQR Simulink Implementation.

With the model full established it is now possible to simulate the effects of varying the state and control weighting matrices to identify a proper LQR controller weighting scheme.

3 Simulation

Implementation of the system dynamics, trimming of the system, linearization of the system, and simulation of the system were all performed in MATLAB, using both the primary programming environment and the Simulink environment. Many of the steps were performed in several different ways to prove redundancy in the methodology. This section covers each function and script utilized in the development of the project's final state, with reference to the theory discussed in section 2.

3.1 RCAM Model

The dynamics of the system as well as calculation of aerodynamic coefficients discussed in the appendix are implemented in this file. The purpose of this function is to accept an input of the current state vector X, and the current control vector U, calculate the system response vector $\dot{X}$ and return it. This will be used to simulate system dynamics for a fixed command signal, but more importantly be used as a basis for trimming and linearization of the model.

Of note, there are two version of this function included in the submission. RCAM_model.m outputs a vector $\dot{X}$ of length 12. This is because it includes the earth-fixed axis coordinates of the plane for use in the FlightGear visualization software. This implementation is not important to the project, but was helpful in understanding the motion of the plane. RCAM_model_9.m outputs the standard state vector discussed in the previous section.

```
function [xdot] = RCAM_model(x,u)
%% Extract the inputs
```

```
% State vector
x1 = x(1);              % u, body XR velocity
x2 = x(2);              % v, body YR velocity
x3 = x(3);              % w, body ZR velocity
x4 = x(4);              % p, roll rate
x5 = x(5);              % q, pitch rate
x6 = x(6);              % r, yaw rate
x7 = x(7);              % phi, euler roll angle
x8 = x(8);              % theta, euler pitch angle
x9 = x(9);              % psi, euler yaw angle

% Control vector
u1 = u(1);              % d_A aileron deflection
u2 = u(2);              % d_T tail deflection
u3 = u(3);              % d_R rudder deflection
u4 = u(4);              % d_th1 engine 1 throttle
u5 = u(5);              % d_th2 engine 2 throttle
```

Input state and control vectors are extracted to variables used in the script. Their descriptions are provided in comments.

```
%% 1. Constants
% Based on the RCAM data

% Geometric Constants
m = 120000;                     % [kg] RCA total mass

% [kg m^2] Inertia tensor matrix
Ib = m*[40.07  0  -2.0923;
        0      64  0;
       -2.0923 0   99.92];

% [kg m^2] Inverse of the Inertia tensor matrix
invIb = (1/m)*[0.0249836   0        0.000523151;
               0           0.015625 0;
               0.000523151 0        0.010019];

cbar = 6.6;                     % [m] mean aerodynamic chord
lt = 24.8;                      % [m]
```

```
S = 260;                        % [m^2] Wing planform surface area
St = 64;                        % [m^2] Tail planform surface area

Xcg = 0.23*cbar;                % [m] Center of Gravity X position
Ycg = 0;                        % [m] Center of Gravity Y position
Zcg = 0.10*cbar;                % [m] Center of Gravity Z position

Xac = 0.12*cbar;                % [m] Aerodynamic Center X position
Yac = 0;                        % [m] Aerodynamic Center Y position
Zac = 0;                        % [m] Aerodynamic Center Z position

% Engine Constants
Xapt1 = 0;          % [m] Engine 1 X position, body axis
Yapt1 = -7.94;      % [m] Engine 1 Y position, body axis
Zapt1 = -1.9;       % [m] Engine 1 Z position, body axis

Xapt2 = 0;          % [m] Engine 2 X position, body axis
Yapt2 = 7.94;       % [m] Engine 2 Y position, body axis
Zapt2 = -1.9;       % [m] Engine 2 Z position, body axis

% Other Constants
rho = 1.225;                % [kg/m^3] Standard air density assumption
g = 9.81;                   % [m/s^2] Gravitation Acceleration
depsda = 0.25;              % [rad/rad] Change in downwash from angle of attack
```

Many constants are enumerated pertaining to the RCAM for use in calculating coefficients later in the program. Of particular note are the lines describing the nonlinear lift curve:

```
alpha_L0 = -11.5*pi/180;    % [rad] Negative stall angle of attack
n = 5.5;                    % [rad] Slope of the linear lift curve
a3 = -768.5;                % [rad] Coefficient of the nonlinear lift curve
a2 = 609.2;                 % [rad] Coefficient of the nonlinear lift curve
a1 = -155.2;                % [rad] Coefficient of the nonlinear lift curve
a0 = 15.212;                % [rad] Coefficient of the nonlinear lift curve
alpha_switch = 14.5*pi/180; % [rad] Switch point between linear/nonlinear drag
```

These functions describe a lift curve provided by [1] described by a piecewise function containing a linear portion and a nonlinear portion. Critical points on the curve are the stall-inducing angles of attack of the airplane.

$$C_{L_{wb}} = \begin{cases} n(\alpha - \alpha_{L=0}) & \alpha \leq 14.5\frac{\pi}{180} \\ a_3\alpha^3 + a_2\alpha^2 + a_1\alpha + a_0 & otherwise \end{cases} \tag{44}$$

```
%% 2. Control Limits
% Limit values
u1_min = -25*pi/180;    % [rad] Minimum aileron deflection angle
u1_max = 25*pi/180;     % [rad] Maximum aileron deflection angle
u2_min = -25*pi/180;    % [rad] Minimum elevator deflection angle
u2_max = 10*pi/180;     % [rad] Maximum elevator deflection angle
u3_min = -30*pi/180;    % [rad] Minimum rudder deflection angle
u3_max = 30*pi/180;     % [rad] Maxmimum rudder deflection angle
u4_min = 0.5*pi/180;    % [rad] Throttle 1 setting, to be passed through a TF
u4_max = 10*pi/180;     % [rad]
u5_min = 0.5*pi/180;    % [rad] Throttle 2 setting, to be passed through a TF
u5_max = 10*pi/180;     % [rad]

% Enforcing the limits
if u1 > u1_max
u1 = u1_max;
elseif u1 < u1_min
u1 = u1_min;
end

if u2 > u2_max
u2 = u2_max;
elseif u2 < u2_min
u2 = u2_min;
end

if u3 > u3_max
u3 = u3_max;
elseif u3 < u3_min
u3 = u3_min;
end

if u4 > u4_max
u4 = u4_max;
```

```
elseif u4 < u4_min
u4 = u4_min;
end

if u5 > u5_max
u5 = u5_max;
elseif u5 < u5_min
u5 = u5_min;
end
```

This section implements the control saturation limits discussed in section 2.1.8, ensuring that no physically impossible deflection angles or thrust forces are introduced by the simulation.

```
%% 3. Calculate frequently used variables
Va = sqrt(x1^2 + x2^2 + x3^2);  % [m/s] Magnitude of air speed vector
alpha = atan2(x3,x1);           % [rad] Angle of attack
beta = asin(x2/Va);             % [rad] Sideslip angle
Q = 0.5 * rho * Va^2;           % [Pa] Dynamic pressure
V_b = [x1;x2;x3];       % Linear velocity vector, in the body frame (u,v,w)
wbe_b = [x4;x5;x6];     % Angular velocity vector, in the body frame (p,q,r)
```

Intermediate variables discussed in Table 2 are calculated in this section.

```
%% 4. Aerodynamic Force Calculations
% Coefficients are derived in the stability axes, that is, the directions
% of the roll, pitch, and yaw axes during steady flight, will need to be
% rotated to the wind, or velocity-determined, axes
% Lift curve expression, generates coefficient of lift for the wing+body
% combination
if alpha <= alpha_switch
CL_wb = n*(alpha-alpha_L0);
else
CL_wb = a3*alpha^3 + a2*alpha^2 + a1*alpha + a0;
end

% Calculate the tail coefficient of lift
epsilon = depsda*(alpha - alpha_L0);
alpha_t = alpha - epsilon + u2 + 1.3 * x5 * lt / Va;
CL_t = 3.1 * St / S * alpha_t;
```

```
% Total lift coefficient
CL = CL_wb + CL_t;

% Drag coefficient
CD = 0.13 + 0.07*(5.5*alpha + 0.654)^2;

% Sideforce coefficient
CY = -1.6*beta + 0.24 * u3;

% Dimensionalized forces
FA_s = [ -CD*Q*S;
CY*Q*S;
-CL*Q*S];
C_bs = [ cos(alpha) 0 -sin(alpha);
0 1 0;
sin(alpha) 0 cos(alpha)];      % Rotation matrix from stability to body axes
FA_b = C_bs*FA_s;
```

The procedure for calculating the body-axis aligned aerodynamic forces from the calculated lift coefficients described in Section 2.1.1. CL_wb, CL_t, CL, CD, and CY represent the various aerodynamic force coefficients discussed in the appendix whose expressions are found in [1]. The dimensionalized body-axis forces are then determined according to equation (5).

```
%% 5. Aerodynamic Moment calculations
 % First part of the moment coefficient expression
eta = [ -1.4*beta;
-0.59 - (3.1*(St*lt)/(S*cbar)*(alpha-epsilon));
(1-alpha*180/(15*pi))*beta];

% Direction cosine matrix for velocities
dCMdx = (cbar/Va)*[-11 0 5;
0 (-4.03*(St*lt^2)/(S*cbar^2)) 0;
1.7 0 -11.5];

% Direction cosine matrix for control surfaces
dCMdu = [-0.6 0 0.22;
0 (-3.1*(St*lt)/(S*cbar)) 0;
0 0 -0.63];
```

```
% Moment coefficient expression 2.33
CMac_b = eta + dCMdx*wbe_b + dCMdu*[u1;u2;u3];

% Aerodynamic moments about aerodynamic center
MAac_b = CMac_b*Q*S*cbar;

rcg_b = [Xcg;Ycg;Zcg];              % Move the moments to be about the CG
rac_b = [Xac;Yac;Zac];
MAcg_b = MAac_b + cross(FA_b, rcg_b - rac_b);
```

These calculations implement equation (6), again relying on the expressions developed in [1] to express the coordinate transformation as well as the complex moment coefficients which are then used to calculate the moment forces in the final line.

```
%% 6. Engine Forces and Moments
F1 = u4*m*g;                            % [N] Engine 1 thrust force
F2 = u5*m*g;                            % [N] Engine 2 thrust force
% Body aligned engine forces are
FE1_b = [F1;0;0];
FE2_b = [F2;0;0];
FE_b = FE1_b + FE2_b;                   % [N] Engine combined body-axis forces

% Engine moments due to offset distance from center of gravity
mew1 = [Xcg - Xapt1;
        Yapt1 - Ycg;
        Zcg - Zapt1];                   % [m] Moment arms for engine 1
mew2 = [Xcg - Xapt2;
        Yapt2 - Ycg;
        Zcg - Zapt2];                   % [m] Moment arms for engine 2
MEcg1_b = cross(mew1,FE1_b);            % [N-m] Moment from engine 1
MEcg2_b = cross(mew2,FE2_b);            % [N-m] Moment from engine 1
MEcg_b = MEcg1_b + MEcg2_b;             % [N-m] Net moment from engines
```

As covered in Sections 2.1.3 and 2.1.4, the engines exert a thrust force aligned with the body forward axis. The resulting forces and moments are calculated consistent with the relevant equations in those sections. They are left in vector format in which the first entry represents force in the X-direction, the second entry represents force in the Y-direction, and the third represents force in the Z-direction, with a similar construction for the moments.

```
%% 7. Gravity Forces
% [m/s^2] Gravity acceleration components in body axes
g_b = [-g*sin(x8);
        g*cos(x8)*sin(x7);
        g*cos(x8)*cos(x7)];
Fg_b = m*g_b;                % [N] Gravity force components in body axes
```

The composition of the gravitational force is simple, as discussed in Section 2.1.5.

```
F_b = Fg_b + FE_b + FA_b;              % [N] Sum of all forces in body axes
x1to3dot = (1/m)*F_b-cross(wbe_b,V_b); % [m/s^2] Linear accelerations
```

All the force vectors are summed to find the net force along each body-aligned axis. Then, consistent with equation (1), the resulting linear acclerations are calculated and stored to be added to the state derivative vector.

```
% [N-m] Sum of all moments about CG in body axes
Mcg_b = MAcg_b + MEcg_b;
% [rad/s^2] Angular accelerations
x4to6dot = invIb*(Mcg_b-cross(wbe_b,Ib*wbe_b));
```

Similarly, all the moment vectors are summed to find the net moments acting about each body-align axis. Then, consistent with equation (1), the resulting angular accelerations are calculated and stored to be added to the state derivative vector.

```
J = [1 sin(x7)*tan(x8) cos(x7)*tan(x8);
     0 cos(x7)         -sin(x7);
     0 sin(x7)/cos(x8) cos(x7)/cos(x8)];
x7to9dot = J*wbe_b;
```

Finally, the coordinate transformation from body angular rates to Euler angular rates is calculated using the angular heading of the plane and, consistent with equation (3), used to find the Euler angular rate vector and stored to be added to the state derivative vector. In the case of the RCAM_model.m file, where the output includes the inertial axis position of the plane, an additional step is taken to implement navigation equations.

```
%% 8.5 Implement navigation equations to find position states
C1v = [cos(x9) sin(x9) 0;
      -sin(x9) cos(x9) 0;
       0       0       1];
```

```
C21 = [cos(x8) 0 -sin(x8);
       0       1  0;
       sin(x8) 0  cos(x8)];
Cb2 = [1 0        0;
       0 cos(x7)  sin(x7);
       0 -sin(x7) cos(x7)];
Cbv = Cb2*C21*C1v;          % Find the rotation matrix from NED to body axes
Cvb = Cbv';                 % Orthogonal, so the transpose is the inverse
x10to12dot = Cvb*V_b;       % Calculate position in the NED frame as output
```

These consist of rotations using the Euler angles about the Euler axes in the standard format. The results are then added to the output vector.

```
%% 9. Form the output vector xdot
% The complete output state derivative vector
xdot = [x1to3dot; x4to6dot; x7to9dot];
end
```

The final line of the function constructs the state derivative vector to be used as an output. This marks the completion of function-based modeling of the nonlinear dynamics of the RCAM system. This function can now be accessed using the Simulink interpreted function block to simulate the airplane behavior over a period of time.

3.1.1 Simulink RCAM Model

RCAM_sim.slx is a Simulink environment implementation of the RCAM_model.m. It has two modes of operation controlled by a manual switch. The lower branch uses m-file driver-defined control settings which are fixed and exert themselves continuously throughout the simulation time. The upper branch allows the operator to use a saved time history of command inputs to drive the model, which is useful in verifying the behavior of the nonlinear system under controls developed for the linearized system. These inputs are fed into a subsystem which muxes the state vector output of the RCAM_model.m file and the control input to create the inputs to the RCAM_model.m function. The control timehistory is recorded for analysis and plotting.

The interpreted MATLAB function block accepts a custom string of inputs which must be sorted and portioned according to the function's acceptable inputs. The output dimensions are inherited from the function output. The strings for calling each of the RCAM_model files are given as:

```
RCAM_model(u(1:12),u(13:17))
or
RCAM_model_9(u(1:9),u(10:14))
```

After the function provides its output, an integrator block initialized to the initial conditions of the RCAM model integrates the derivative state vector to obtain the state vector. The output states are then logged to the workspace. They are then de-muxed to be used in the calculation of V_A and γ. If the model is using the RCAM_model.m file which provides navigation data, then the operator may enable the FlightGear simulation to visualize the motion of the airplane. This requires installation of FlightGear and the generation of an executable file.

3.1.2 Nonlinear Driver Script

In order to run the Simulink model, initial states must be defined, and proper visualization of the data established to enable further analysis. This job is performed by the RCAM_driver.m script. There are a few ways to define the initial conditions. Initially it makes sense to use an operator guess of what a trim point may look like:

```
%% INITIALIZES THE RCAM_MODEL.slx SIMULATION
clc, clear
%% Define Initial Conditions
x0 = [85;               % x1 - [m/s] Initial forward speed
0;                      % x2 - [m/s] Initial sideways speed
0;                      % x3 - [m/s] Initial downward speed
0;                      % x4 - [rad/s] Initial roll rate
0;                      % x5 - [rad/s] Initial pitch rate
0;                      % x6 - [rad/s] Initial yaw rate
0;                      % x7 - [rad] Initial euler roll angle
0.1;                    % x8 - [rad] Initial euler pitch angle
0;                      % x9 - [rad] Initial euler yaw angle
0;                      % x10 - [m] Initial Earth N Position
0;                      % x11 - [m] Initial Earth E Position
500];                   % x12 - [m] Initial Earth D Position

u = [0;                 % u1 - [rad] Initial aileron angle
-0.1;                   % u2 - [rad] Initial elevator angle
0;                      % u3 - [rad] Initial rudder angle
0.08;                   % u4 - [rad] Initial engine 1 throttle
0.08];                  % u5 - [rad] Initial engine 2 throttle
```

These are hardcoded values. But once a trim point is found using the methods discussed in future sections, it becomes more expedient to load the saved trim point into the workspace and decompose the values into the relevant vectors. MATLAB saves data in .mat files which contain

fields. These fields must be identified using the fieldnames() function, and then navigating to the relevant fields to extract the data. This is repeated for the $x0$ initial condition vector field and the u1-5 control fields.

```
% If using a trim point, uncomment
op_trim = load('OP_trim_01_straight_level.mat');
op_trim1 = op_trim.op_trim17;
FieldNames = fieldnames(op_trim1);

x0dx = getfield(op_trim1,FieldNames{4});
x0dxFieldNames = fieldnames(x0dx);
x0 = getfield(x0dx,x0dxFieldNames{3});
x0 = x0(1:9)

udat = getfield(op_trim1,FieldNames{2});
u1dat = udat(1);
u1datFieldNames = fieldnames(u1dat);
u1 = getfield(u1dat,u1datFieldNames{4});
u2dat = udat(2);
u2datFieldNames = fieldnames(u2dat);
u2 = getfield(u2dat,u2datFieldNames{4});
u3dat = udat(3);
u3datFieldNames = fieldnames(u3dat);
u3 = getfield(u3dat,u3datFieldNames{4});
u4dat = udat(4);
u4datFieldNames = fieldnames(u4dat);
u4 = getfield(u4dat,u4datFieldNames{4});
u5dat = udat(5);
u5datFieldNames = fieldnames(u5dat);
u5 = getfield(u5dat,u5datFieldNames{4});
u = [u1;u2;u3;u4;u5];
```

If the operator instead wishes to use a control timehistory, such a timehistory must exist in the primary directory named 'Control_timeseries.mat' to be loaded into the workspace as a struct. Again the data is extracted and placed into the relevant vectors for use in the Simulink model. In this case, the manual switch should be toggled to use the upper path.

```
u = load('Control_timeseries.mat')
tout = u.StructOut.time;
```

```
u1 = u.StructOut.signals.values.u1;
u2 = u.StructOut.signals.values.u2;
u3 = u.StructOut.signals.values.u3;
u4 = u.StructOut.signals.values.u4;
u5 = u.StructOut.signals.values.u5;
u1Struct.signals.values = u1;
u1Struct.time = tout;
u2Struct.signals.values = u2;
u2Struct.time = tout;
u3Struct.signals.values = u3;
u3Struct.time = tout;
u4Struct.signals.values = u4;
u4Struct.time = tout;
u5Struct.signals.values = u5;
u5Struct.time = tout;
```

Now that all variables have been declared and the mode of operation has been decided, the Simulink model can be called. This is achieved by setting a runtime through the SimTime variable, and then calling the 'sim()' function. Data to workspace is stored in the 'out' variable which will contain cells of data named according to the To Workspace blocks in the Simulink file.

```
%% Call the model
SimTime = 60;         % [s] Duration of flight simulation
out = sim('RCAM_SIM.slx',SimTime);
```

After the simulation, the user may want to review the data, so the simulation data stored in the 'out' must be extracted to be plottable. The time series data become important, and so it too is extracted and stored.

```
%% Plot the results
t = out.simX.Time;
u1 = out.simU.Data(:,1);
u2 = out.simU.Data(:,2);
u3 = out.simU.Data(:,3);
u4 = out.simU.Data(:,4);
u5 = out.simU.Data(:,5);

x1 = out.simX.Data(:,1);
x2 = out.simX.Data(:,2);
```

```
x3 = out.simX.Data(:,3);
x4 = out.simX.Data(:,4);
x5 = out.simX.Data(:,5);
x6 = out.simX.Data(:,6);
x7 = out.simX.Data(:,7);
x8 = out.simX.Data(:,8);
x9 = out.simX.Data(:,9);
% x10 = out.simX.Data(:,10);
% x11 = out.simX.Data(:,11);
% x12 = out.simX.Data(:,12);
```

Plotting the data is done using a number of very similar sets of instructions. For brevity, only one is listed here. The rest share a similar form but pertain to different variables.

```
figure(1)
subplot(5,1,1)
plot(t, u1)
title('u_1')
grid on
```

Simulation results using a fixed operating point described by the following states and controls are shown in Figure 4 and 5.

```
x0 = [85;               % x1 - [m/s] Initial forward speed
5;                      % x2 - [m/s] Initial sideways speed
0;                      % x3 - [m/s] Initial downward speed
0;                      % x4 - [rad/s] Initial roll rate
-0.1;                   % x5 - [rad/s] Initial pitch rate
0;                      % x6 - [rad/s] Initial yaw rate
0;                      % x7 - [rad] Initial euler roll angle
0.1;                    % x8 - [rad] Initial euler pitch angle
0.1;                    % x9 - [rad] Initial euler yaw angle
0;                      % x10 - [m] Initial Earth N Position
0;                      % x11 - [m] Initial Earth E Position
500];                   % x12 - [m] Initial Earth D Position

u = [0;                 % u1 - [rad] Initial aileron angle
-0.1;                   % u2 - [rad] Initial elevator angle
0;                      % u3 - [rad] Initial rudder angle
```

```
0.08;                   % u4 - [rad] Initial engine 1 throttle
0.08];                  % u5 - [rad] Initial engine 2 throttle
```

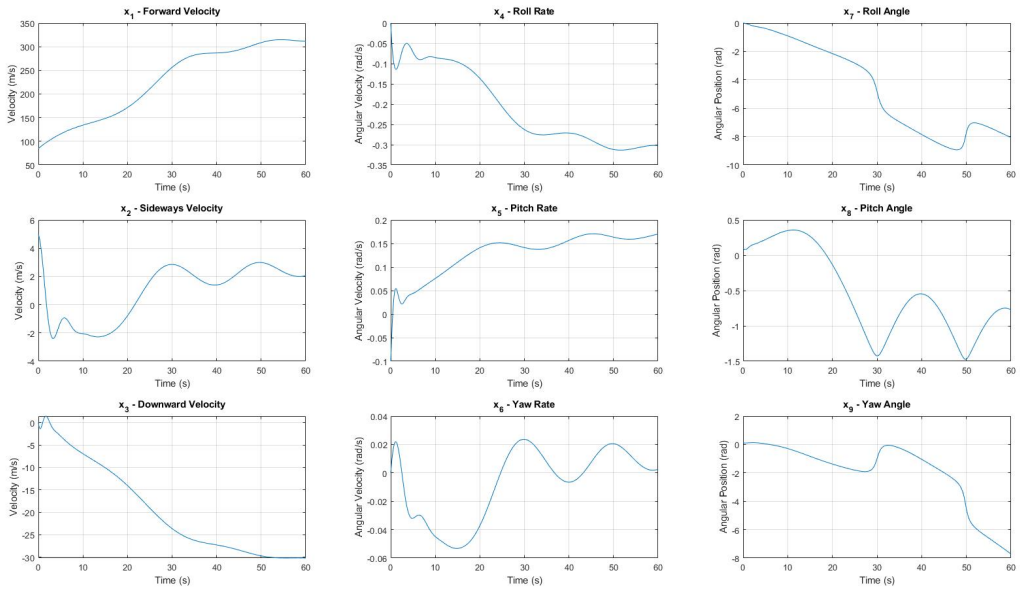


Figure 4: States for a fixed control input.

It can be seen that static controls of those values cause the RCAM plane to exhibit wildly uncontrolled motion, likely tumbling through space. Because it is difficult to immediately understand what is going on based on only the state traces, the FlightGear implementation proves useful in analyzing the system.

3.1.3 Trimming the Model

Now that the system's behavior can be simulated, it is possible to implement the optimization function which seeks a trim point in the behavior of a model for a given set of states. This is done by implementing the cost function (22) discussed in Section 2.2.1.

As discussed, the function takes in a vector $\bar{z}$ which is a concatenation of the state and control vectors. It then simulates the model given the operator guess at a trim point and calculates the remaining trim conditions of overall airspeed V_A, angle of attack α, and sideslip angle β.

These values are then concatenated into the total trim restriction matrix Q and multiplied by the penalty matrix H which can be simply be an identity matrix of compatible dimensions to

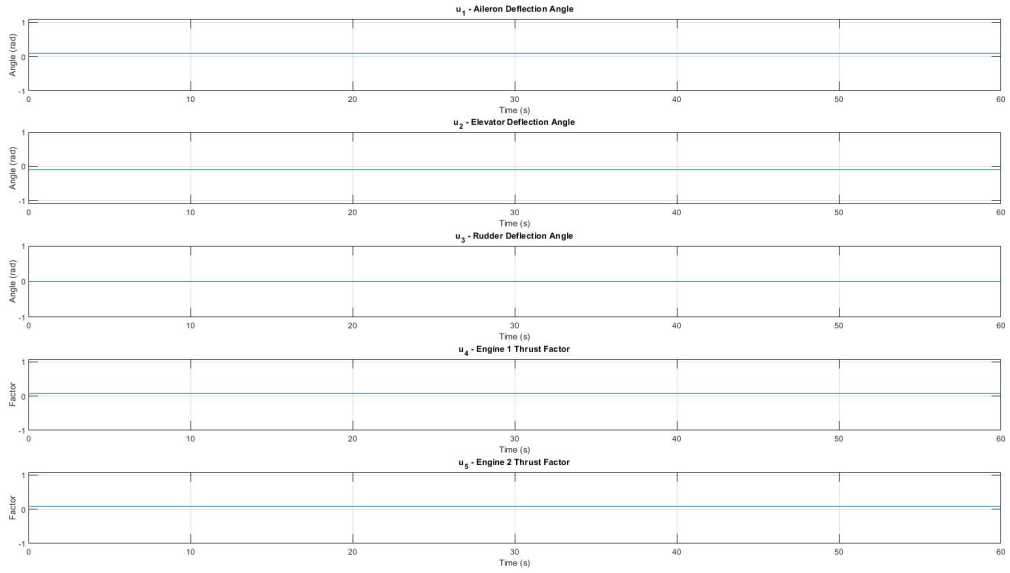


Figure 5: Fixed controls.

Q. Finally, the matrix product is multiplied by Q^T to obtain a singular value representing the deviation from the operator-defined trim point at those model evaluation conditions.

```
function [F0] = cost_straight_level(Z)
X = Z(1:9);
U = Z(10:14);

xdot = RCAM_model_9(X,U);
theta = X(8);
Va = sqrt(X(1)^2+X(2)^2+X(3)^2);
alpha = atan2(X(3),X(1));
gam = theta-alpha;

Q = [xdot;
Va-85;
gam;
X(2);
X(7);
X(9)];
```

```
H = diag(ones(1,14));
F0 = Q'*H*Q;
end
```

This function is called repeatedly to search for a minimum value $F0$ across a span of values with small deviations using a driver script which sets the initial guess Z_guess to be equivalent to the trim condition (19) suggested in Section 2.2.1. To do this, the MATLAB function 'fminsearch()' is used, with a set number of iterations and function evaluations as well as the search step distance. The starting point is the operator-defined trim point 'Z_guess'.

```
%% TRIMMING THE RCAM MODEL
% For a given set of operating conditions, locate a trim point of the model
% which best approximates those conditions while experiencing zero change
% in desired variables
clc, clear
%% DEFINE THE DESIRED STEADY STATE VALUES
Z_guess = zeros(14,1);
Z_guess(1) = 85;
%% SEARCH THE COST FUNCTION
[ZStar,f0] = fminsearch('cost_straight_level',Z_guess,...
optimset('TolX',1e-10,'MaxFunEvals',10000,'MaxIter',10000))
% Extract the input which provided the minimum f0 value
XStar = ZStar(1:9);
UStar = ZStar(10:14);
% Simulate the model to reobtain relevant values
XdotStar = RCAM_model_9(XStar,UStar);
VaStar = sqrt(XStar(1)^2 + XStar(2)^2 + XStar(3)^2);
gammaStar = XStar(8) - atan2(XStar(3),XStar(1));
vStar = XStar(2);
phiStar = XStar(7);
psiStar = XStar(9);

save trim_values_straight_level XStar UStar
```

The driver may need to be called multiple times depending on the quality of the value returned by 'fminsearch'. The value should be effectively zero if a trim point satisfying the conditions. The values of each constraint are calculated one more time by using the control and initial state vector found by the search function with the 'RCAM_model.m' function for operator evaluation. Three

iterations were used in the search for an adequate trim point in this project before a desirable result was obtained:

```
Exiting: Maximum number of function evaluations has been exceeded
- increase MaxFunEvals option.
Current function value: 0.000000


ZStar =

84.9905
0.0000
1.2712
0.0000
-0.0000
0.0000
-0.0000
0.0150
0.0000
0.0000
-0.1780
0.0000
0.0821
0.0821


f0 =

1.1442e-12
```

The forward velocity u is very close to the desired velocity of 85 m/s, driven by the two engines providing equal (thus not inducing a moment) thrust fraction of 0.0821. The elevator is angled slightly downward, maintaining the pitch angle of the RCAM model to ensure that it does not drop out of the sky. The trim point is saved to workspace and used to linearize the model.

An alternative to this method is to use the Simulink Linear Analysis Toolbox to seek a trim point. To do this a model with input and output ports of the desired controls, states, and intermediate variables to be constrained is created called 'RCAM_linearization.slx'. All the variables that should not change should remain in steady state. This implies that the state vector should

remain steady, but inputs should be unconstrained. The intermediate variables in the output tab should be marked for steady state.

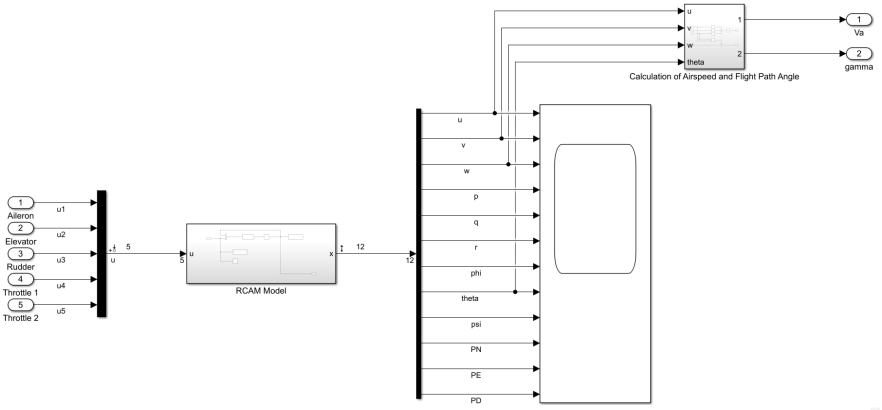
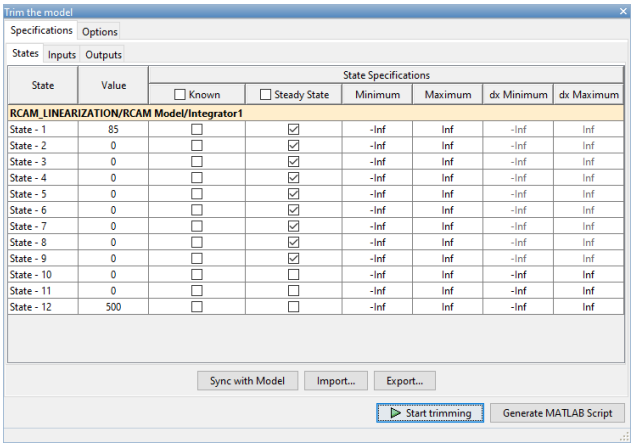


Figure 6: Simulink model for linearization.



State	Value	Known	Steady State	Minimum	Maximum	dx Minimum	dx Maximum
RCAM_LINEARIZATION/RCAM Model/Integrator1							
State - 1	85	☐	☑	-Inf	Inf	-Inf	Inf
State - 2	0	☐	☑	-Inf	Inf	-Inf	Inf
State - 3	0	☐	☑	-Inf	Inf	-Inf	Inf
State - 4	0	☐	☑	-Inf	Inf	-Inf	Inf
State - 5	0	☐	☑	-Inf	Inf	-Inf	Inf
State - 6	0	☐	☑	-Inf	Inf	-Inf	Inf
State - 7	0	☐	☑	-Inf	Inf	-Inf	Inf
State - 8	0	☐	☑	-Inf	Inf	-Inf	Inf
State - 9	0	☐	☑	-Inf	Inf	-Inf	Inf
State - 10	0	☐	☐	-Inf	Inf	-Inf	Inf
State - 11	0	☐	☐	-Inf	Inf	-Inf	Inf
State - 12	500	☐	☐	-Inf	Inf	-Inf	Inf

Figure 7: Trimming Settings.

The resulting trim point is saved to the workspace manually, and confirms the results found using the 'fminsearch' function methodology.

3.1.4 Linearizing the Model

There are two techniques for linearizing the system function: the symmetric difference approach discussed in Section 2.2.2 and Simulink's Linear Analysis Toolbox. Use of the symmetric difference

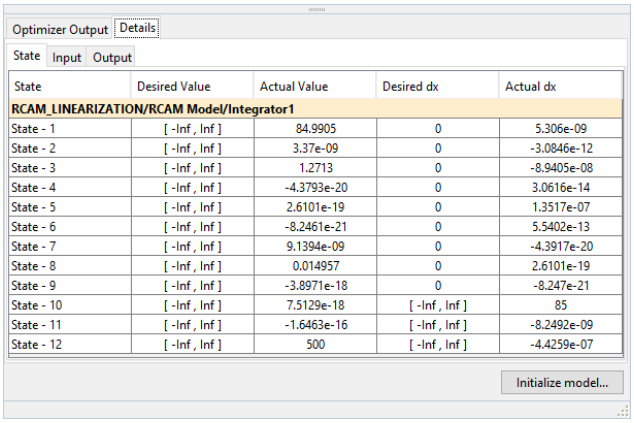


Figure 8: Trim point states.

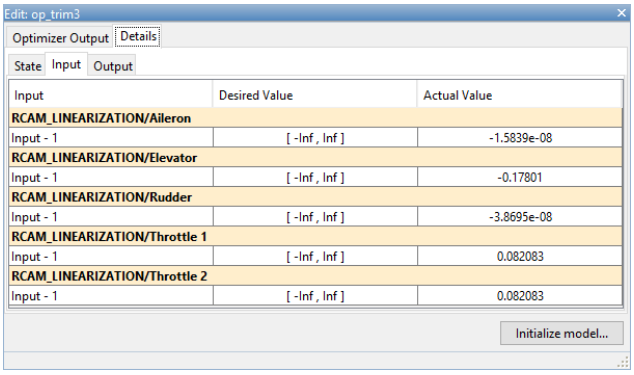


Figure 9: Trim point controls.

method requires that the system equations be given in implicit form rather than the explicit form described in Section 2.1 by equations (1 - 3). To do this, simply subtract the differential state vector from the 'RCAM_model.m' function as was done in 'RCAM_model_implicit.m'.

```
function [fval] = RCAM_model_implicit(xdot, x, u)
% Calculates the implicit form of the system at a given state derivative,
% state, and input condition
fval = RCAM_model_9(x,u)-xdot;
end
```

This function will act as an input to the 'Implicit_linmod.m' function which calculates the matrices E, A', and B'. It employs the same procedure discussed in Section 2.2.2 by iterating through each element of each Jacobian matrix and determining the slope of the system function via two perturbations:

```
function [E,A_prime,B_prime] = Implicit_linmod(SYS_function,XDOT_TRIM,X_TRIM,U_TRIM,DX
% Uses the symmetric difference method to calculate the jacobian matrices
% for E, A', B' of a system
% XDOT_TRIM, X_TRIM, U_TRIM is the expansion/trim point
% DXDOT, DX, DU are the perturbation distances

% Number of states
n = length(XDOT_TRIM);
% Number of controls
m = length(U_TRIM);

%% Calculate E
% Initialize the matrix
E = zeros(n,n);

% 2 for loops used to fill the matrix
for i = 1:n
 for j = 1:n
  % Perturbation magnitude
  dxdot = DXDOT(i,j);

  % Perturbation vector, only the jth variable is perturbed to fill
  % the Jacobian
  xdot_plus = XDOT_TRIM;
  xdot_minus = XDOT_TRIM;
  xdot_plus(j) = xdot_plus(j) + dxdot;
  xdot_minus(j) = xdot_minus(j) - dxdot;

  % Calculate the resulting change by calling the implicit system at
  % the trim point with the symmetric differences added
  F = feval(SYS_function,xdot_plus,X_TRIM,U_TRIM);
  F_plus_keep = F(i);
  F = feval(SYS_function,xdot_minus,X_TRIM,U_TRIM);
  F_minus_keep = F(i);

  % The element is found by the symmetric difference formula and
  % saved in the matrix E:
  E(i,j) = (F_plus_keep - F_minus_keep)/(2*dxdot);
```

```
 end
end

%% Calculate A_prime
A_prime = zeros(n,n);

% 2 for loops used to fill the matrix
for i = 1:n
 for j = 1:n
  % Perturbation magnitude
  dx = DX(i,j);

  % Perturbation vector, only the jth variable is perturbed to fill
  % the Jacobian
  x_plus = X_TRIM;
  x_minus = X_TRIM;
  x_plus(j) = x_plus(j) + dx;
  x_minus(j) = x_minus(j) - dx;

  % Calculate the resulting change by calling the implicit system at
  % the trim point with the symmetric differences added
  F = feval(SYS_function,XDOT_TRIM,x_plus,U_TRIM);
  F_plus_keep = F(i);
  F = feval(SYS_function,XDOT_TRIM,x_minus,U_TRIM);
  F_minus_keep = F(i);

  % The element is found by the symmetric difference formula and
  % saved in the matrix A_prime:
  A_prime(i,j) = (F_plus_keep - F_minus_keep)/(2*dx);
 end
end

%% Calculate B_prime
B_prime = zeros(n,m);

% 2 for loops used to fill the matrix
for i = 1:n
 for j = 1:m
```

```
    % Perturbation magnitude
    du = DU(i,j);

    % Perturbation vector, only the jth variable is perturbed to fill
    % the Jacobian
    u_plus = U_TRIM;
    u_minus = U_TRIM;
    u_plus(j) = u_plus(j) + du;
    u_minus(j) = u_minus(j) - du;

    % Calculate the resulting change by calling the implicit system at
    % the trim point with the symmetric differences added
    F = feval(SYS_function,XDOT_TRIM,X_TRIM,u_plus);
    F_plus_keep = F(i);
    F = feval(SYS_function,XDOT_TRIM,X_TRIM,u_minus);
    F_minus_keep = F(i);

    % The element is found by the symmetric difference formula and
    % saved in the matrix A_prime:
    B_prime(i,j) = (F_plus_keep - F_minus_keep)/(2*du);
   end
  end
```

Calculation of the final state space matrices is done by a driver script titled 'Linearize_RCAM.m' which loads in the trim point developed in Section 3.1.3 and extracts the relevant trim state variables and controls. Suitably small perturbations about the trim point are defined of proper dimension to enable matrix multiplication with the corresponding E A' and B' matrices in the function call.

```
%% LINEARIZE THE RCAM MODEL
% Uses the symmetric difference method
% clc, clear
%% Define the trim values
% Straight and level conditions
Xdot_trim = zeros(9,1);

op_trim = load('OP_trim_01_straight_level.mat');
op_trim1 = op_trim.op_trim17;
FieldNames = fieldnames(op_trim1);
```

```
x0dx = getfield(op_trim1,FieldNames{4});
x0dxFieldNames = fieldnames(x0dx);
X_trim = getfield(x0dx,x0dxFieldNames{3});

udat = getfield(op_trim1,FieldNames{2});
u1dat = udat(1);
u1datFieldNames = fieldnames(u1dat);
u1 = getfield(u1dat,u1datFieldNames{4});
u2dat = udat(2);
u2datFieldNames = fieldnames(u2dat);
u2 = getfield(u2dat,u2datFieldNames{4});
u3dat = udat(3);
u3datFieldNames = fieldnames(u3dat);
u3 = getfield(u3dat,u3datFieldNames{4});
u4dat = udat(4);
u4datFieldNames = fieldnames(u4dat);
u4 = getfield(u4dat,u4datFieldNames{4});
u5dat = udat(5);
u5datFieldNames = fieldnames(u5dat);
u5 = getfield(u5dat,u5datFieldNames{4});
U_trim = [u1;u2;u3;u4;u5];

%% Define perturbation matrices
dxdot_matrix = 10e-12*ones(9,9);
dx_matrix = 10e-12*ones(9,9);
du_matrix = 10e-12*ones(9,5);
```

The implicit linearization function is then called, which returns the matrices.

```
%% Calculate E, A prime, B prime
% Using the implicit model function handle
[E,Ap,Bp] = Implicit_linmod(@RCAM_model_implicit,Xdot_trim,X_trim,U_trim,dxdot_matrix,
```

Finally, the state space matrices A and B are obtained via equation (31) and displayed.

```
%% Calculate A, B matrices
A = -inv(E)*Ap;
B = -inv(E)*Bp;
```

```
disp(A)
disp(B)
```

$$A = \begin{bmatrix} -0.0354 & 0 & 0.0612 & 0 & -1.2298 & 0.0000 & 0 & -9.8089 & 0 \\ -0.0000 & -0.1805 & -0.0000 & 1.2713 & 0 & -84.9905 & 9.8089 & -0.0000 & 0 \\ -0.2203 & 0 & -0.7063 & -0.0000 & 82.2157 & 0 & 0 & -0.1467 & 0 \\ 0.0000 & -0.0286 & 0.0000 & -1.3460 & 0 & 0.5842 & 0 & 0 & 0 \\ -0.0010 & 0 & -0.0336 & 0 & -1.1073 & 0 & 0 & 0 & 0 \\ 0.0000 & 0.0077 & 0.0000 & 0.0554 & 0 & -0.5533 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1.0000 & 0.0000 & 0.0150 & 0.0000 & -0.0000 & 0 \\ 0 & 0 & 0 & 0 & 1.0000 & -0.0000 & 0.0000 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.0000 & 1.0001 & 0.0000 & -0.0000 & 0 \end{bmatrix} \tag{45}$$

$$B = \begin{bmatrix} 0 & 0.1094 & 0 & 9.8100 & 9.8100 \\ 0 & 0 & 2.3012 & 0 & 0 \\ 0 & -7.3156 & 0 & 0 & 0 \\ -0.9486 & 0 & 0.3640 & 0.0407 & -0.0407 \\ 0 & -2.9193 & 0 & 0.3924 & 0.3924 \\ -0.0199 & 0 & -0.4081 & 0.7804 & -0.7804 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{46}$$

Of note are the last 3 rows of B which indicate that the control surfaces have no interaction with the Euler roll, pitch, and yaw angles. This makes sense as the Euler angles are indirectly calculated from the change in body-fixed angular velocities and are not related to the control surfaces in the equations of motion.

The Simulink Linear Analysis toolbox can again be used to obtain a state space representation. By marking the input and output signals as inputs for perturbation and outputs for measurement by the toolbox, Simulink can automatically perform perturbations to seek a linear expression of the system. No further configuration is required, and response plots can be automatically provided for controller design.

The resulting state space system can be expressed as is standard and can be shown to be suitable for LQR control via a controllability test using 'rank(ctrb(A,B))'. If the matrix resulting from 'ctrb(A,B)' is full rank, then the system is controllable.

```
size(ctrb(A,B),1) == rank(ctrb(A,B))
```

```
A =

   -0.0354         0    0.0612         0   -1.2298    0.0000         0   -9.8089         0
   -0.0000   -0.1805   -0.0000    1.2713         0  -84.9905    9.8089   -0.0000         0
   -0.2203         0   -0.7064   -0.0000   82.2157         0   -0.0000   -0.1467         0
    0.0000   -0.0286    0.0000   -1.3460    0.0000    0.5842         0         0         0
   -0.0010         0   -0.0336         0   -1.1073         0         0         0         0
    0.0000    0.0077    0.0000    0.0554    0.0000   -0.5533         0         0         0
         0         0         0    1.0000    0.0000    0.0150    0.0000   -0.0000         0
         0         0         0         0    1.0000   -0.0000    0.0000         0         0
         0         0         0         0    0.0000    1.0001    0.0000   -0.0000         0


B =

         0    0.1094         0    9.8100    9.8100
         0         0    2.3012         0         0
         0   -7.3157         0         0         0
   -0.9486         0    0.3640    0.0407   -0.0407
         0   -2.9193         0    0.3924    0.3924
   -0.0199         0   -0.4081    0.7804   -0.7804
         0         0         0         0         0
         0         0         0         0         0
         0         0         0         0         0


C =

     1     0     0     0     0     0     0     0     0
     0     1     0     0     0     0     0     0     0
     0     0     1     0     0     0     0     0     0
     0     0     0     1     0     0     0     0     0
     0     0     0     0     1     0     0     0     0
     0     0     0     0     0     1     0     0     0
     0     0     0     0     0     0     1     0     0
     0     0     0     0     0     0     0     1     0
     0     0     0     0     0     0     0     0     1


D =

     0     0     0     0     0
     0     0     0     0     0
     0     0     0     0     0
     0     0     0     0     0
     0     0     0     0     0
     0     0     0     0     0
     0     0     0     0     0
     0     0     0     0     0
     0     0     0     0     0
```

Figure 10: Simulink State Space Results.

```
ans =
 1
```

So the system is suitable for LQR control.

3.2 LQR Implementation in Simulink

Development of the LQR controller is handled by the script 'LQR_development.m'. It first extracts the calculated state space system as well as the trim states and controls for use as the initial state of the system.

```
%% LQR CONTROL SCHEME SCRIPT
% Uses the saved linear system to define and simulate the effect of a
% control system
clc, clear
%% ESTABLISH THE STATE SPACE SYSTEM
% Load in the linear system for trimmed steady state rectilinear flight
LinSys = load('Linsys_straight_and_level.mat');
A = LinSys.linsys1.A;
B = LinSys.linsys1.B;
C = LinSys.linsys1.C;
D = LinSys.linsys1.D;

%% DEFINE THE INITIAL STATE VECTOR
% Plane is banked, drifting sideways in the body axis, and rotating about
% all 3 axes
trim = load('trim_values_straight_level.mat');
x_trim = trim.XStar;
uo = x_trim(1);                 % [m/s] Initial forward velocity
vo = x_trim(2);                 % [m/s] Initial lateral velocity
wo = x_trim(3);                 % [m/s] Initial downward velocity
po = x_trim(4);                 % [rad/s] Initial roll rate
qo = x_trim(5);                 % [rad/s] Initial pitch rate
ro = x_trim(6);                 % [rad/s] Initial yaw rate
phio = x_trim(7);               % [rad] Initial Euler roll angle
thetao = x_trim(8);             % [rad] Initial Euler pitch angle
psio = x_trim(9);               % [rad] Initial Euler yaw angle
Xo = [uo; vo; wo; po; qo; ro; phio; thetao; psio]; % Initial State Vector
```

Many hours of simulation trial and error were expended in trying to adjust the 9 diagonal gains of the Q weighting matrix and the 5 diagonal gains of the control weighting matrix. For such a complex system with many coupled equations, it would be impossible to briefly summarize the complete sequence of tests performed with any sort of completeness. Instead, a set of summarizing observations is presented in the conclusion of the paper. For the final version of the controller the following weights were selected for use in calculating the optimal control gains.

```
%% DEFINE THE LQR CONTROLLER GAINS
% Make a choice for the Q/R weights
Q = diag([3e20 3e21 2.5e17 4.7e6 9e5 5e8 2.5e17 2.5e16 2.5e16])
R = diag([4e20 4e20 4e20 6.5e20 6.5e20])
% Calculate the optimal gain vector
[K S E] = lqr(A,B,Q,R);
```

At this point the simulation may be run. This implementation utilizes the Simulink model 'RCAM_LQR.slx' which is a standard implementation of the state space Simulink model augmented with the optimal controller gain law as obtained using equation (43). The initial condition of the integrator is again set to the values of the trim point, and the commanded states act as constant reference inputs on the left hand side of the model. Control actuation and state data is logged to the workspace to be plotted, as shown in Figure 11.

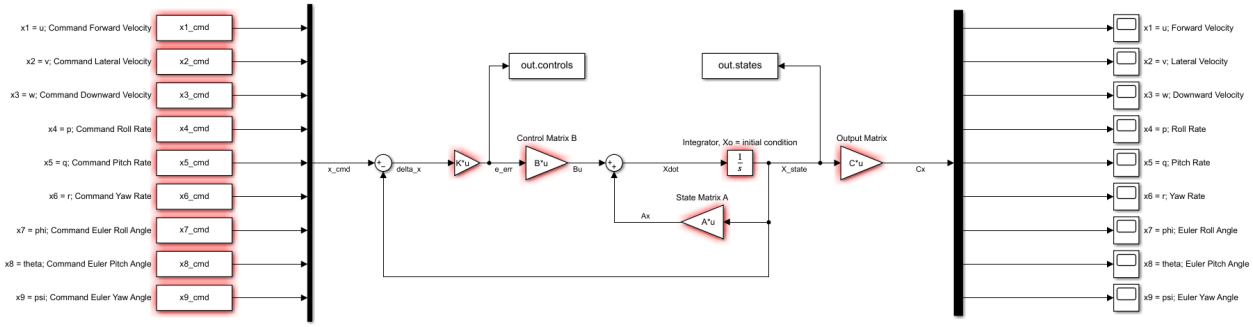


Figure 11: Simulink LQR Implementation.

```
%% EXECUTE THE SIMULATION USING THE DESIGNED CONTROLLER
simTime = 60;           % [s] Duration of simulation
out = sim('RCAM_LQR.slx',simTime);
```

Similarly to the RCAM model for static control inputs, the data is extracted and placed into variables for plotting. The graphing code is repetitive, and so only the one variable's worth of plotting code is shown here.

```
%% MANAGE AND PLOT THE DATA
% Extract performance data
t = out.tout;
u1 = out.controls.Data(:,1);
u2 = out.controls.Data(:,2);
u3 = out.controls.Data(:,3);
u4 = out.controls.Data(:,4);
u5 = out.controls.Data(:,5);

x1 = out.states.Data(:,1);
x2 = out.states.Data(:,2);
x3 = out.states.Data(:,3);
x4 = out.states.Data(:,4);
x5 = out.states.Data(:,5);
x6 = out.states.Data(:,6);
x7 = out.states.Data(:,7);
x8 = out.states.Data(:,8);
x9 = out.states.Data(:,9);

% Plot
% Control inputs
figure(1)
subplot(5,1,1)
plot(t, u1)
title('u_1 = Aileron Deflection')
xlabel('Time (s)'),ylabel('Rad')
grid on
```

To bring the control sequence created by the LQR controller back to the nonlinear system representation for validation, it is necessary to export the control sequences as a timeseries of data. To do this, it is necessary to construct the data in a struct with cells for the control sequence as well as the timeseries. For completeness, the code necessary to load the control sequence into the nonlinear model is included here.

```
%% EXPORT THE CONTROL SEQUENCES
StructOut.signals.values.u1 = u1;
StructOut.signals.values.u2 = u2;
StructOut.signals.values.u3 = u3;
StructOut.signals.values.u4 = u4;
```

```
StructOut.signals.values.u5 = u5;
StructOut.time = t;
save('Control_timeseries','StructOut');

%% APPLY THE CONTROL TO NONLINEAR SYSTEM
% Obtain the control sequence
% u = load('Control_timeseries.mat')
% tout = u.StructOut.time;
% u1 = u.StructOut.signals.values.u1;
% u2 = u.StructOut.signals.values.u2;
% u3 = u.StructOut.signals.values.u3;
% u4 = u.StructOut.signals.values.u4;
% u5 = u.StructOut.signals.values.u5;
% u1Struct.signals.values = u1;
% u1Struct.time = tout;
% u2Struct.signals.values = u2;
% u2Struct.time = tout;
% u3Struct.signals.values = u3;
% u3Struct.time = tout;
% u4Struct.signals.values = u4;
% u4Struct.time = tout;
% u5Struct.signals.values = u5;
% u5Struct.time = tout;
```

3.2.1 Simulation Results

Because the system was linearized about the trim point (19) described in Section 2.2.1, it follows that it is only a valid linearization for small perturbations from that trim point. As a result, the trim point is used as the initial state vector of the RCAM plane. A few cases of stepped commands were tested and evaluated:

- Forward velocity commanded to 95 m/s from 85 m/s: x1_cmd = 95;
- Pitch angle commanded to 90 deg from 0 deg: x8_cmd = 90/180*pi;
- Roll angle commanded to 15 deg from 0 deg: x7_cmd = 15/180*pi;
- Yaw angle commanded to 5 deg from 0 deg: x9_cmd = 5/180*pi;
- Lateral velocity commanded to 5 m/s from 0 m/s: x2_cmd = 5;

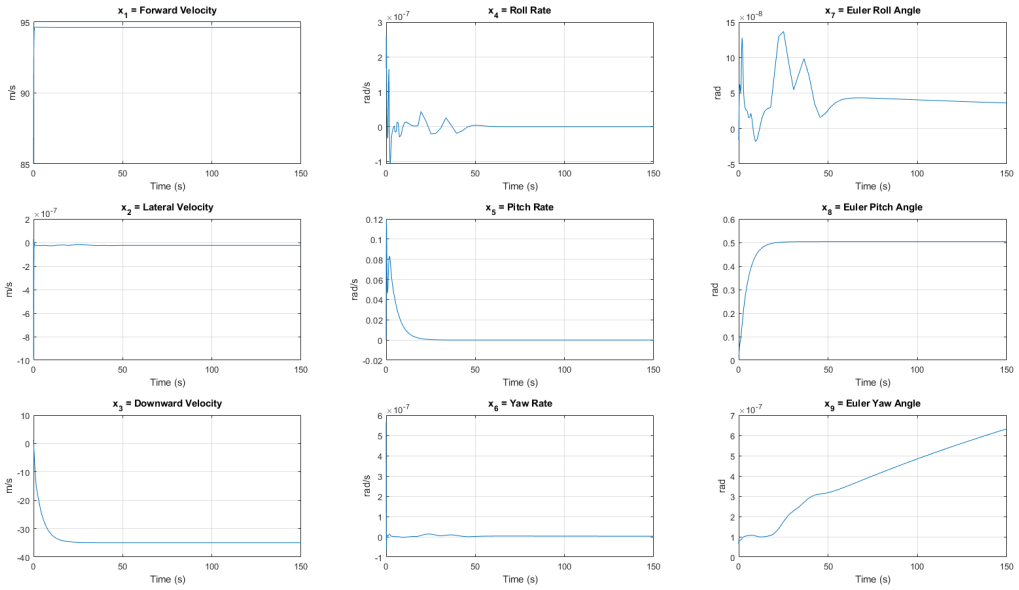


Figure 12: Forward velocity step state results.

For the case of the 10 m/s step in forward velocity, the simulation obtained the results of Figure 12. As can be seen, the system instantly reaches the commanded velocity but seems to experience a large oscillation in pitch rate at the onset of the simulation. This is likely due to a lack of control saturation consideration in the LQR version of the RCAM aircraft. As will be explained in the conclusion of the report, LQR cannot appropriately account for such considerations and will gain the signal as necessary to obtain control. Also of note is the relatively large steady state value of the Pitch angle x_8 of 28.6 degrees. This likely accounts for the large negative (and therefore upward) velocity steady state value as well. The scales on all other state variables are quite small, representing their lack of involvement in the forward rectilinear motion, as expected. This suggests that perhaps the systems could be decoupled.

When the pitch angle was commanded to 90 degrees, the steady state pitch angle was found to be roughly 125 degrees. This is once again a large steady state error which suggests that the pitch angle is difficult to control. As a function of maintaining trim conditions, the velocity is still referenced to the trim forward velocity of 85 m/s, and stays near this value with a steady state error of only 2 m/s. Predictably, the upward (negative downward, but body-axis aligned) is still quite large, but even larger than in the previous simulation. This is likely due to the dynamics of traveling in a vertical loop—forward velocity is preserved despite the rotation of the rigid body. This arrives at a steady state value more slowly as a result. States not involved in the motion

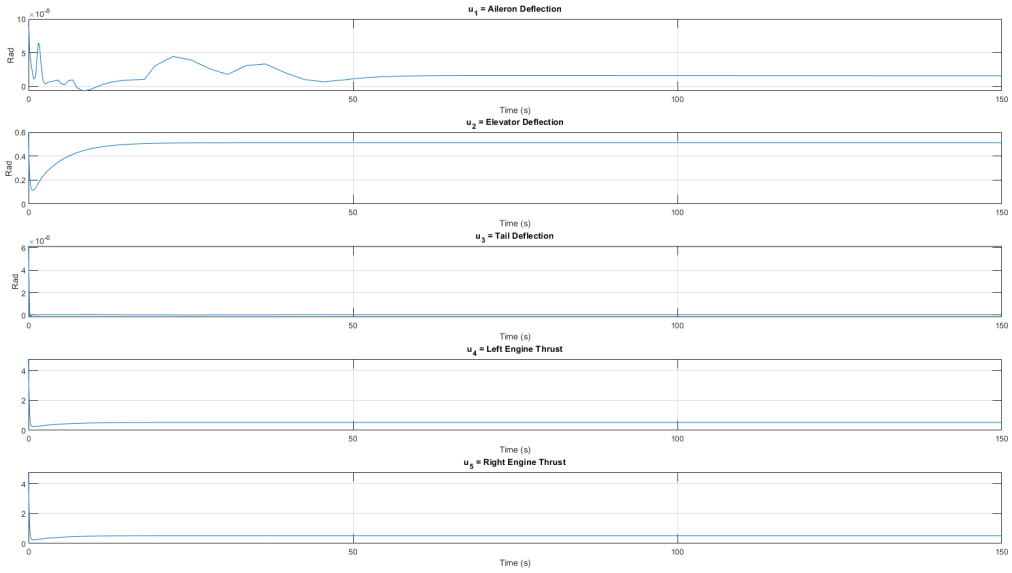


Figure 13: Forward velocity step control results

remain negligibly changed. Contrary to the previous simulation, the transition from initial pitch angle to final pitch angle is far from instantaneous, taking approximately 20 seconds, further reinforcing the idea that angular positions are difficult to control in this model. The results for this simulation are displayed in Figures 14 and 15.

When the roll angle is commanded to 15 degrees the simulation seems to have difficulty handling the dynamics of the situation such that a greater change in yaw is actually observed. This suggests that there is a problem with the implementation of the Aileron deflection, as its commanded value is also quite small. In addition, the lack of difference between the thrusts of the two engines suggests that the selection for the control weighting matrix makes the two engines quite likely to behave similarly. This is in fact a necessity, being that they are the same engine, but this coupling is exposed by the aileron control difficulty. It is possible that control saturation could alleviate the issue by restraining the ability of the rudder to effect the controlled states. As expected, the bank angle produces a small lateral velocity and a yaw angle throughout the turn. The rise time of the roll angle is quite high, taking almost 3 minutes to achieve steady state. The results for this simulation are displayed in Figures 16 and 17.

In contrast to the other angles, the yaw angle is comparatively extremely well tuned. For a commanded step of 5 degrees, and a long rise time of approximately 150 seconds, the system achieves a yaw angle of 4.96 degrees. As expected by the dynamics of the system, a change in

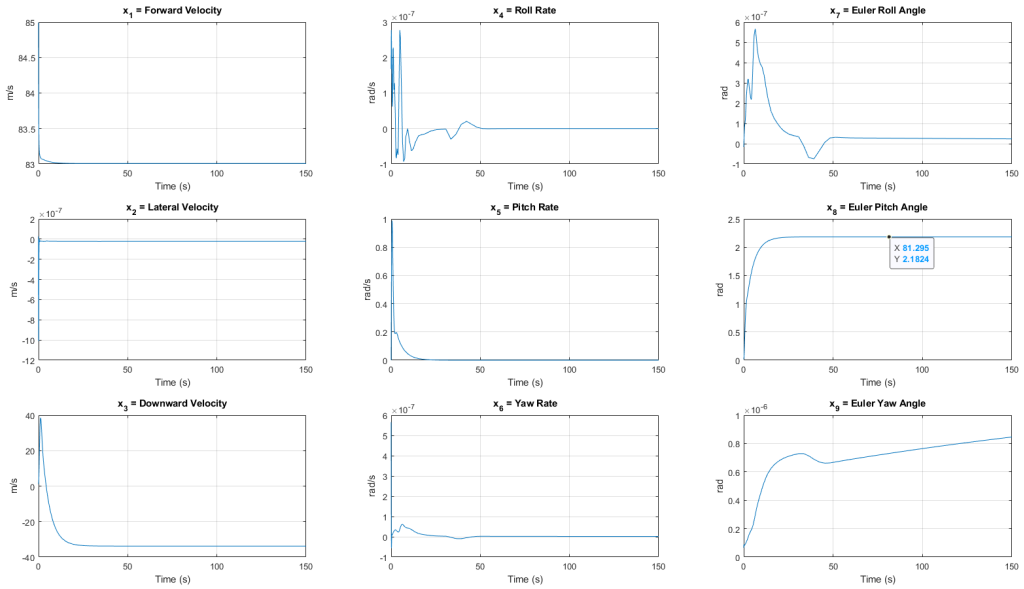


Figure 14: Pitch angle step state results.

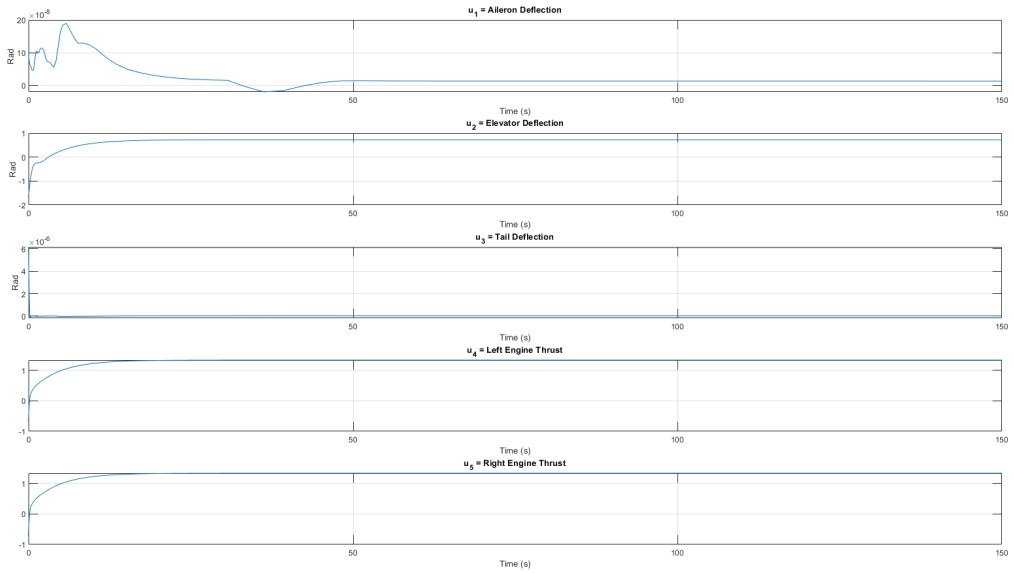


Figure 15: Pitch angle step control results

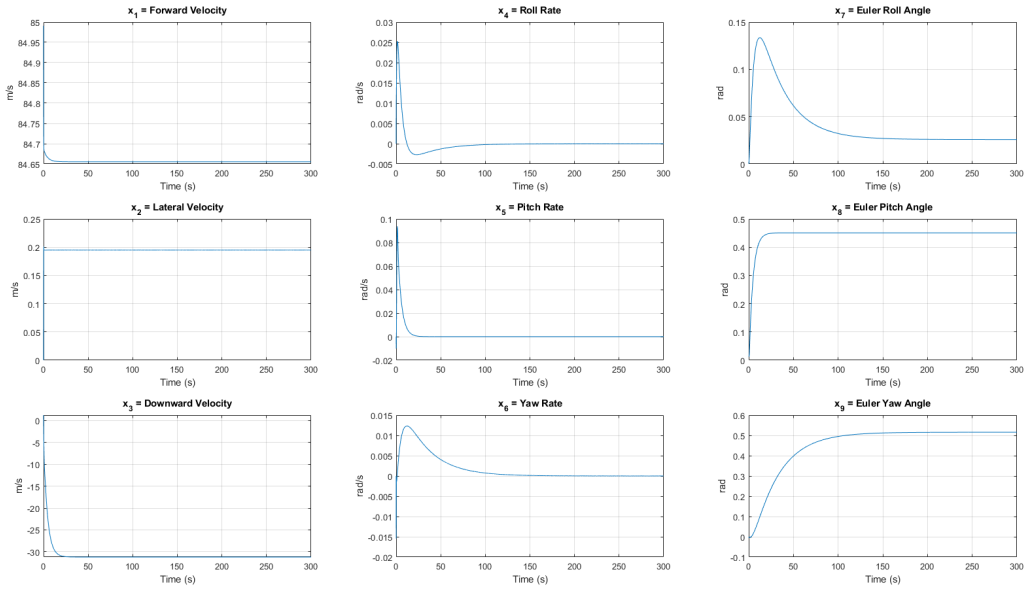


Figure 16: Roll angle step state results.

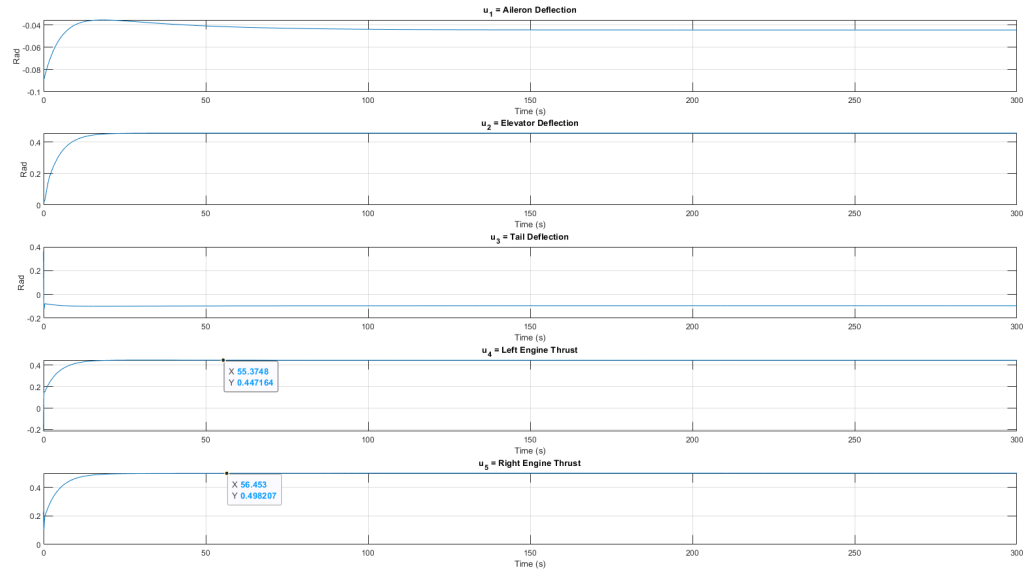


Figure 17: Roll angle step control results

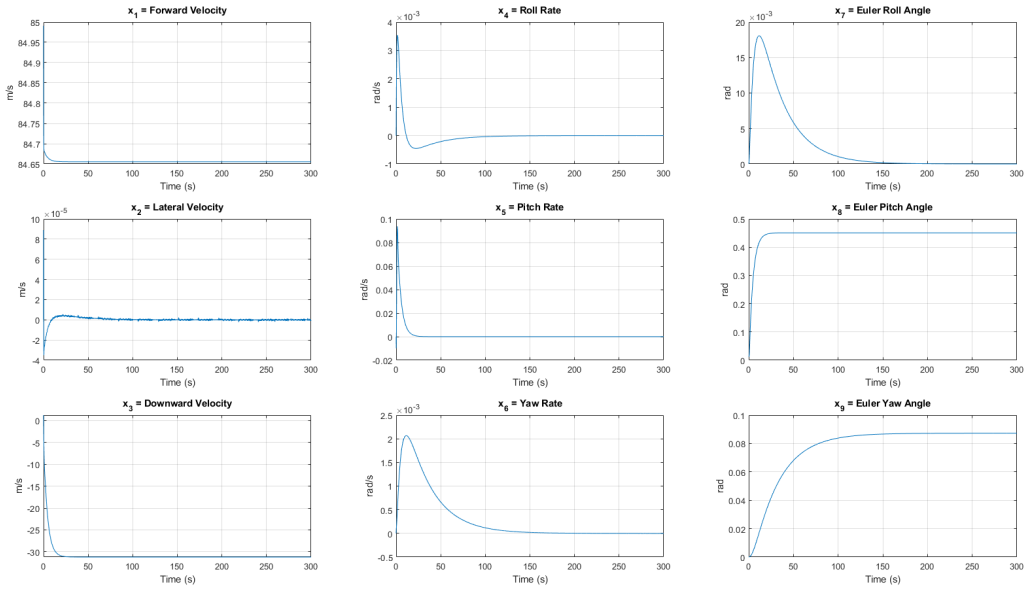


Figure 18: Yaw angle step state results.

yaw is handled in equal parts by the Aileron and Tail deflections, while the other control inputs handle the reference forward speed of 85 m/s. This suggests that the systems for longitudinal and latitudinal motion are easily decoupled. Lessening the weighting of the tail has only a negligible effect on the transient response time, which again suggests a control saturation problem as the command increases significantly. The results for this simulation are displayed in Figures 18 and 19.

A step command in lateral velocity has similarly accurate results, although much more quickly owing to the freeness of the thrust controls in the weighting matrices. The difference in thrust seems to give rise to a roll and yaw angle, alongside the now-familiar pitch angle responsible for steady forward velocity. This is yet more evidence that the equations of motion can be decoupled and further simplified for aircraft in steady rectilinear flight. Steady state for the angular positions is once again slow to reach, but ultimately the system is brought into steady state. The results for this simulation are displayed in Figures 20 and 21.

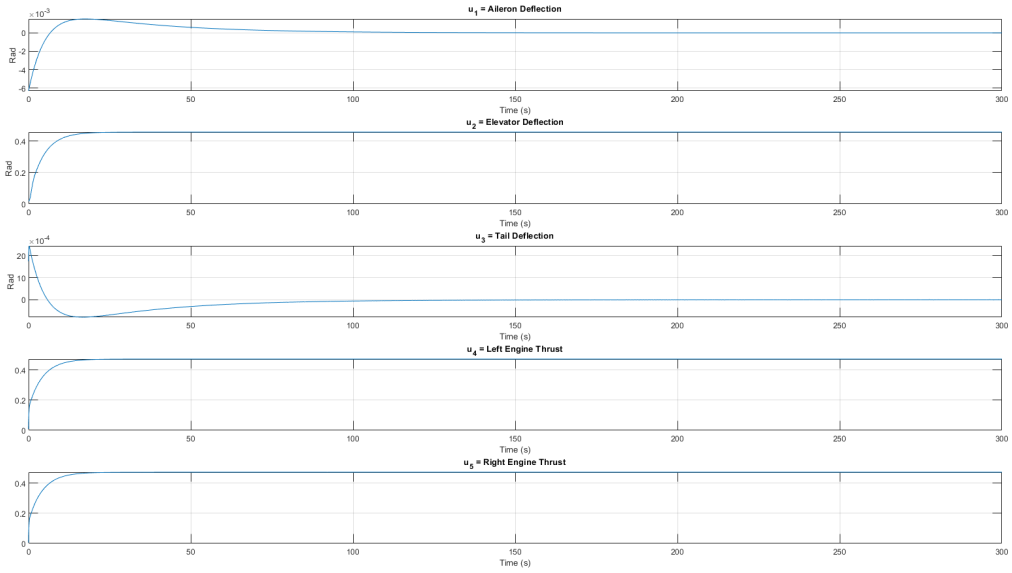


Figure 19: Yaw angle step control results

4 Conclusion

The original stated objective of this project was to simply implement LQR control on a dynamic system, tune the controller, and simulate the results using MATLAB. The system under consideration could have been much simpler than what I selected for this project, and my interest in aerospace led me to embark on a modeling effort for which I was underprepared. Over the course of the semester I watched many tutorials and read several chapters of aircraft dynamic textbooks, listed in the references. The modeling effort of the dynamics of an aircraft is not straightforward, but has been done before and the results are able to be used off the shelf if enough time and care is taken to understand their meaning.

However, the resulting dynamic model of the system proves to be difficult to control. In tuning the LQR weighting matrices Q and R, many things became quickly apparent:

- In order to bring the values within the realm of possibility, the weighting values for control inputs would need to be quite expensive, as LQR control has no provision for control saturation. This in turn forces the state weighting matrix to be quite expensive, else the system would never be driven to a set point. This distribution of values may not be realistic, and also makes tuning the system difficult, as the act of balancing the relative weights tends to have drastic results when disturbed.

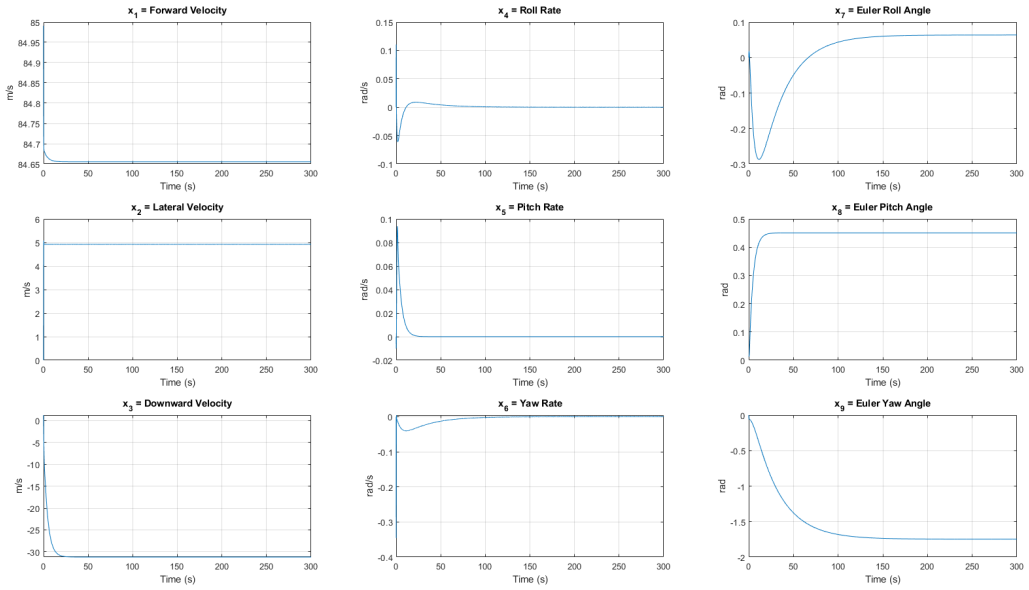


Figure 20: Lateral Velocity step state results.

- The control inputs representing the two identical engines would need to be given identical weights, else a yawing or rolling moment would inevitably be introduced which could not be accounted for in the controller.
- This version of the model will always have a large steady state error in pitch angle. In practice this is quite a significant problem as the RCAM aircraft is meant to represent a large freight airliner, and the pitch angle is quite important to a smooth flight path.
- The controller lacking a saturation provision usually means that there is a significant and instantaneous kick in the behavior of the system as the control surfaces instantly reach their commanded values rather than being constrained by slew rates.
- The two most difficult states to control are the pitch angle and, as a result, the vertical velocity.
- In contrast, forward and lateral velocities as well as the yaw angle are very easy to control.
- Angular positions respond slowly to step inputs on the command variables.

Many of these concerns can be attributed to modeling difficulties, but one which cannot be overcome by a pure implementation of LQR is the lack of control saturation consideration. In an

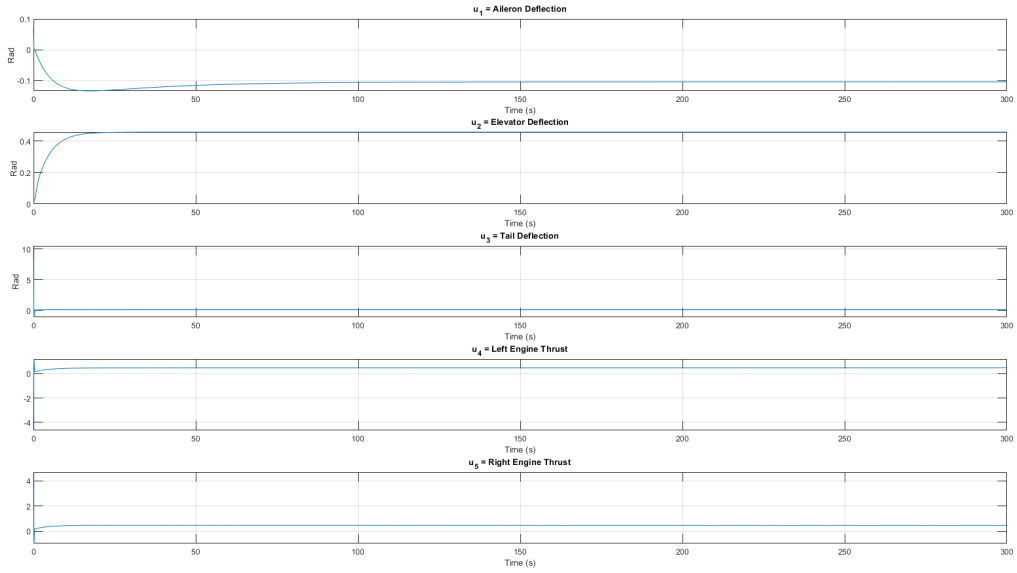


Figure 21: Lateral Velocity step control results

LQR system the optimal gains are calculated agnostically, without reference to any limit values and therefore the commanded signal can easily exceed the physical capabilities of the system. One way to account for the control saturations and their associated slew rates is through Model-Predictive Control (MPC), which is outside the scope of this project. Future improvements on this model should include MPC provisions.

Overall, the system is not consistent enough to draw firm conclusions on the overall system responses. There are too many moving parts and too few concretely well-behaved states and control inputs to make predictions regarding the behavior of the system subjected to a combination of control inputs, and the system often outright fails to reach certain values without massive steady-state errors as a result.

References

[1] GARTEUR Open Publication: Robust Flight Control Design Challenge Problem Formulation and Manual: the REsearch Civil Aircraft Model (RCAM)
`https://garteur.org/wp-content/reports/FM/FM_AG-08_TP-088-3.pdf`

[2] J. Aranda et al., Design of a Linear Quadratic Optimal Control for Aircraft Flight Control by Genetic Algorithm
https://www.researchgate.net/publication/228954549_Design_of_a_linear_quadratic_optimal_control_for_aircraft_flight_control_by_genetic_algorithm

[3] Kumar, Vinodh and Jerome, Jovitha, Robust LQR Controller Design for Stabilizing and Trajectory Tracking of Inverted Pendulum
https://core.ac.uk/download/pdf/82723202.pdf

[4] Cook, Michael V., Flight Dynamics Principles, Second Edition. Free online:
https://aerocastle.files.wordpress.com/2012/04/flightdynamicsprinciples.pdf

[5] Aliyu Bhar Kisabo and Aliyu Funmilayo Adebimpe (June 5th 2019). State-Space Modeling of a Rocket for Optimal Control System Design, Ballistics, Charles Osheku, IntechOpen, DOI: 10.5772/intechopen.82292. Available from:
https://www.intechopen.com/books/ballistics/state-space-modeling-of-a-rocket-for-optimal-control-system-design

[6] Professor Christopher Lum's Lecture Series on Flight Mechanics and Modeling
https://www.youtube.com/playlist?list=PLxdnSsBqCrrEx3A6W94sQGClk6Q4YCg-h

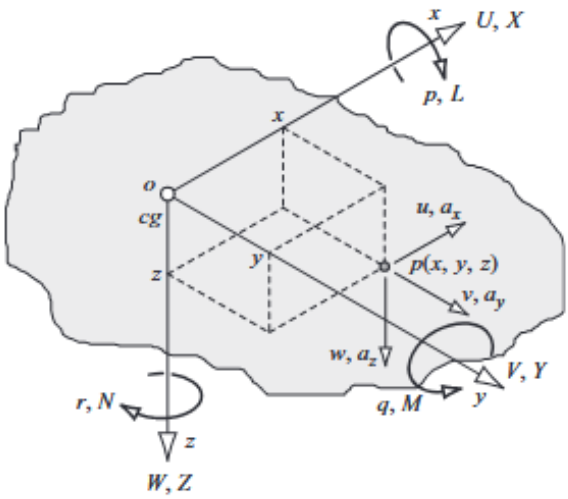
A Glossary of Terms

LQR, Linear Quadratic Regulator
TVC, Thrust Vector Control
DoF, Degrees of Freedom
Flight control Derivatives, Terms used in the expression of differential equations of motion as laid out in [4]
Lift, Aerodynamic force acting perpendicular to the flat wing surface
Drag, Aerodynamic force acting against the motion of the aircraft
Dynamic pressure, Kinetic energy per unit volume of a fluid
RCAM, Research Civil Aircraft Model, a public dataset for a control challenge
trimming, The act of locating a point at which the system states are steady
Linearization, reducing a system of nonlinear functions to a system of linear functions
MATLAB, a programming environment
Simulink, a visual programming environment offered with MATLAB
Planform, projected area of an aircraft wing onto a horizontal plane
Sideforce, forces acting laterally against the body of an aircraft
Angle of Attack, the nose angle above or below the horizontal axis of an aircraft
Body Axis, the set of axes placed at the center of gravity of an aircraft such that the x axis always points through the nose of the plane, and the z axis always points through the floor
Direction Cosine Matrix, a coordinate transform matrix
Control Saturation, the physical maximum or minimum output of a control variable
Slew Rate, the physical maximum or minimum rate of change of output of a control variable

B Derivation of Equations of Motion

The derivation of the nonlinear equations of motion is covered in great detail in [4]. The process is reproduced here for completeness.

Consider an orthogonal axis set (x, y, z) with its origin coincident with the center of gravity of a rigid body. Consider also an orthogonal axis set (X, Y, Z) representing the inertial frame. That is, (x, y, z) moves with respect to (X, Y, Z), which does not move at all. The components of velocity with respect to the rigid body frame are expressed as (U, V, W). Angular velocities with respect to the rigid body frame are given by (p, q, r) while moments are expressed as (L, M, N). Consider a point $\bar{p}$ which exists within the rigid body some distance from the rigid body axis origin. It has velocity relative to the body axes expressed as (u, v, w) and acceleration expressed as (a_x, a_y, a_z). Its velocity components relative to the body axis origin are expressed as



$$
\begin{aligned}
u =& \dot{x} - ry + qz \\
v =& \dot{y} - pq + rz \\
w =& \dot{z} - qx + py
\end{aligned}
$$

The rotary terms represent tangential velocity which arises due to the resolution of moments to forces with a moment arm. For a rigid body, the motion due to change in the body structure $(\dot{x}, \dot{y}, \dot{z})$ is zero, and the equations reduce to

$$
\begin{aligned}
u =& qz - ry \\
v =& rz - pq \\
w =& py - qx
\end{aligned}
$$

The corresponding expression of linear acceleration at point $\bar{p}$ is given by

$$
\begin{aligned}
a_x =& \dot{u} - rv + qw \\
a_y =& \dot{v} - pw + ru \\
a_z =& \dot{w} - qu + pv
\end{aligned}
$$

where again the rotary terms arise from the interaction of angular and linear velocities at a distance away from the origin of the axes. By superimposing the overall velocity of the body the absolute velocity of the point $\bar{p}$ can be obtained

$$
\begin{aligned}
u' =& U + u = U - ry + qz \\
v' =& V + v = V - pz + rx \\
w' =& W + w = W - qx + py
\end{aligned}
$$

Differentiating with respect to time,

$$
\begin{aligned}
\dot{u}' =& \dot{U} - \dot{r}y + \dot{q}z \\
\dot{v}' =& \dot{V} - \dot{p}z + \dot{r}x \\
\dot{w}' =& \dot{W} - \dot{q}x + \dot{p}y
\end{aligned}
$$

Inertial acceleration of point $\bar{p}$, (a'_x, a'_y, a'_z) can be expressed via substitution of the above into the general expression for acceleration in the rigid body

$$
\begin{aligned}
a'_x =& \dot{u}' - rv' + qw' \\
a'_y =& \dot{v}' - pw' + ru' \\
a'_z =& \dot{w}' - qu' + pv'
\end{aligned}
$$

Substituting all the known expressions yields the full formulation of the accelerations

$$
\begin{aligned}
a'_x =& \dot{U} - rV + qW - x(q^2 + r^2) + y(pq - \dot{r}) + z(pr + \dot{q}) \\
a'_y =& \dot{V} - pW + rU + x(pq + \dot{r}) - y(p^2 + r^2) + z(qr - \dot{p}) \\
a'_z =& \dot{W} - qU + pV + x(pr - \dot{q}) + y(qr + \dot{p}) - z(p^2 + q^2)
\end{aligned}
$$

If we then consider a differential mass m located in the rigid body, Newton's Second Law gives rise to the generalised force equations where X, Y, and Z represent the sums of forces in x-, y-, and z-directions respectively, while the sums of a differential mass multiplied by distances reduces to zero:

$$
\begin{aligned}
m(\dot{U} - rV + qW) =& X \\
m(\dot{V} - pW + rU) =& Y \\
m(\dot{W} - qU + pV) =& Z
\end{aligned}
$$

Which are the generalised force equations of a rigid body.

Turning next to the generalised moment equations of a rigid boy, we consider the forces and their moment arms:

$$\begin{aligned}\sum \delta m(ya'_z - za'_y) =& L\\ \sum \delta m(za'_x - xa'_z) =& M\\ \sum \delta m(xa'_y - ya'_x) =& L\end{aligned}$$

Which, after substitution and rearrangement of the expressions for accelerations as well as noting that terms under the $\sum$ symbol have units of the moment of inertia, gives

$$\begin{aligned}I_x\dot{p} - (I_y - I_z)qr + I_{xy}(pr - \dot{q}) - I_{xz}(pq + \dot{r}) + I_{yz}(r^2 - q^2) =& L\\ I_y\dot{q} + (I_x - I_z)pr + I_{yz}(pq - \dot{r}) + I_{xz}(p^2 - r^2) - I_{xy}(qr + \dot{p}) =& M\\ I_z\dot{r} - (I_x - I_y)pq - I_{yz}(pr + \dot{q}) + I_{xz}(qr - \dot{p}) + I_{xy}(q^2 - p^2) =& N\end{aligned}$$

Because most aircraft are symmetric about the vertical plane, the product terms of inertia $I_{xy} = I_{yz} = 0$, which simplifies the expression of the moment equations

$$\begin{aligned}I_x\dot{p} - (I_y - I_z)qr - I_{xz}(pq + \dot{r}) &= L\\ I_y\dot{q} + (I_x - I_z)pr + I_{xz}(p^2 - r^2) &= M\\ I_z\dot{r} - (I_x - I_y)pq + I_{xz}(qr - \dot{p}) &= N\end{aligned}$$

The forces and moments can be expressed in terms of components which may contribute to the system. [4] notes that there are 5 main sources of these effects: **a**erodynamic effects, **g**ravitational effects, **c**ontrol effects, **p**ower effects, and **d**isturbance effects.

$$\begin{aligned}X =& X_a + X_g + X_c + X_p + X_d\\ Y =& Y_a + Y_g + Y_c + Y_p + Y_d\\ Z =& Z_a + Z_g + Z_c + Z_p + Z_d\\ L =& L_a + L_g + L_c + L_p + L_d\\ N =& N_a + N_g + N_c + N_p + N_d\\ M =& M_a + M_g + M_c + M_p + M_d\end{aligned}$$

C Expression of Aerodynamic Coefficients

As provided in [1], there are many aerodynamic coefficients which must be calculate in order to develop a correct model of the aircraft. This appendix will collect the equations provided in the RCAM documentation.

The aerodynamic force coefficients C_D, C_Y, C_L are derived as follows, where β is the sideslip angle and δ_R is the rudder deflection.

$$\begin{aligned} C_D =& 0.13 + 0.07(C_{L_{wb}} - 0.45)^2 \\ C_Y =& -1.6\beta + 0.24\delta_R \\ C_L =& C_{L_{wb}} + C_{L_t} \end{aligned}$$

The coefficient of lift of the wing-body alone is expressed as

$$C_{L_{wb}} = \begin{cases} n(\alpha - \alpha_{L=0}) & \alpha \leq 14.5\frac{\pi}{180} \\ a_3\alpha^3 + a_2\alpha^2 + a_1\alpha + a_0 & otherwise \end{cases}$$

and the lift coefficient of the tail unit is

$$C_{L_t} = \frac{S_t}{S} 3.1\alpha_t$$

where α_t is the angle of attack of the tail unit given by

$$\begin{aligned} \frac{d\epsilon}{d\alpha} &= 0.25 \\ \epsilon &= \frac{d\epsilon}{d\alpha}(\alpha - \alpha_0) \end{aligned}$$

The moment coefficients of lift are expressed as follows

$$\begin{bmatrix} C_l \\ C_m \\ C_n \end{bmatrix} = \begin{bmatrix} -1.4\beta \\ -0.59 - 3.1\frac{S_t l_t}{Sl}(\alpha - \epsilon) \\ (1 - \alpha\frac{180}{15\pi})\beta \end{bmatrix} + \frac{l}{V} \begin{bmatrix} -11 & 0 & 5 \\ 0 & -4.03\frac{S_t l_t^2}{Sl^2} & 0 \\ 1.7 & 0 & -11.5 \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} + \begin{bmatrix} -0.6 & 0 & 0.22 \\ 0 & -3.1\frac{S_t l_t}{Sl} & 0 \\ 0 & 0 & -0.63 \end{bmatrix} \begin{bmatrix} \delta_A \\ \delta_B \\ \delta_C \end{bmatrix}$$

These coefficients can now be used in the expression of aerodynamic forces.

D The Direction Cosine Matrix

The Direction Cosine Matrix is an expression of the sequential rotations to transform from one coordinate axes to another. Consider the linear components of a quantity in an axis system $(ox_0y_0z_0)$ to be (ox_0, oy_0, oz_0). The expression of these components in another axis system $(ox_3y_3z_3)$ will be (ox_3, oy_3, oz_3). The transformation matrix is composed of rotations about specific Euler axes. By first yawing, then pitching, and then rolling, the transformation becomes

$$\begin{bmatrix} ox_3 \\ oy_3 \\ oz_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix} \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix} \begin{bmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} ox_0 \\ oy_0 \\ oz_0 \end{bmatrix}$$

The direction cosine matrix is the matrix product of the three rotation operations

$$D = \begin{bmatrix} \cos\theta\cos\psi & \cos\theta\sin\psi & -\sin\theta \\ \sin\phi\sin\theta\cos\psi - \cos\phi\sin\psi & \sin\phi\sin\theta\sin\psi + \cos\phi\cos\psi & \sin\phi\cos\theta \\ \cos\phi\sin\theta\cos\psi + \sin\phi\sin\psi & \cos\phi\sin\theta\sin\psi - \sin\phi\cos\psi & \cos\phi\cos\theta \end{bmatrix}$$

This matrix can be inverted to transform coordinates in the other direction.