Design & Development of Fixed Wing & Multirotor UAV Flight Models

Rutvik Solanki
1008247531

Prakriti Saini
1007495195

Siva Subramanian Vadakkanthara Sivaramakrishnan
1006892757

AER1216 - FUNDAMENTALS OF UAVS

UNIVERSITY OF TORONTO, INSTITUTE OF AEROSPACE STUDIES

Contents

List of Figures

OVERVIEW

An unmanned aerial vehicle (UAV), commonly known as a drone, is an aircraft without any human pilot, crew, or passengers on board. UAVs are a component of an unmanned aircraft system (UAS), which has three basic components:

1. An autonomous or human-operated control system which is usually on the ground or a ship but may be on another airborne platform.
2. An Unmanned Aerial Vehicle (UAV)
3. A command and control (C2) system - sometimes referred to as a communication, command, and control (C3) system - to link the two.

These systems include but are not limited to remotely piloted air systems (RPAS) in which the UAV is controlled by a 'pilot' using a radio data link from a remote location. UAS can also include an autonomously controlled UAV or, more likely, a semi-autonomous UAV.

NOTE: In recent years, the tendency to refer to any UAV as a "drone" has developed but the term is not universally considered appropriate. UAVs can vary in size from those which can be hand launched to purpose-built or adapted vehicles the size of conventional fixed or rotary wing aircraft.

In this given task, we have designed and developed two individual UAS systems (Fixed-Wing sUAS and multi-rotor drone) with respect to the given configuration. The chosen models for the fixed-wing and the multirotor models are the Aerosonde UAV and a quadrotor respectively. The design and analysis of the UAS system are done with respect to the selection of parameters that suit the given configuration. A linearized dynamics model was developed and subjected to multiple simulations and test conditions in order to estimate the stability of the system. The control system is implemented by incorporating PID control over the control points of the aircraft and the test results are established. The software platform used for the analysis of the aircraft system is MATLAB and Simulink.

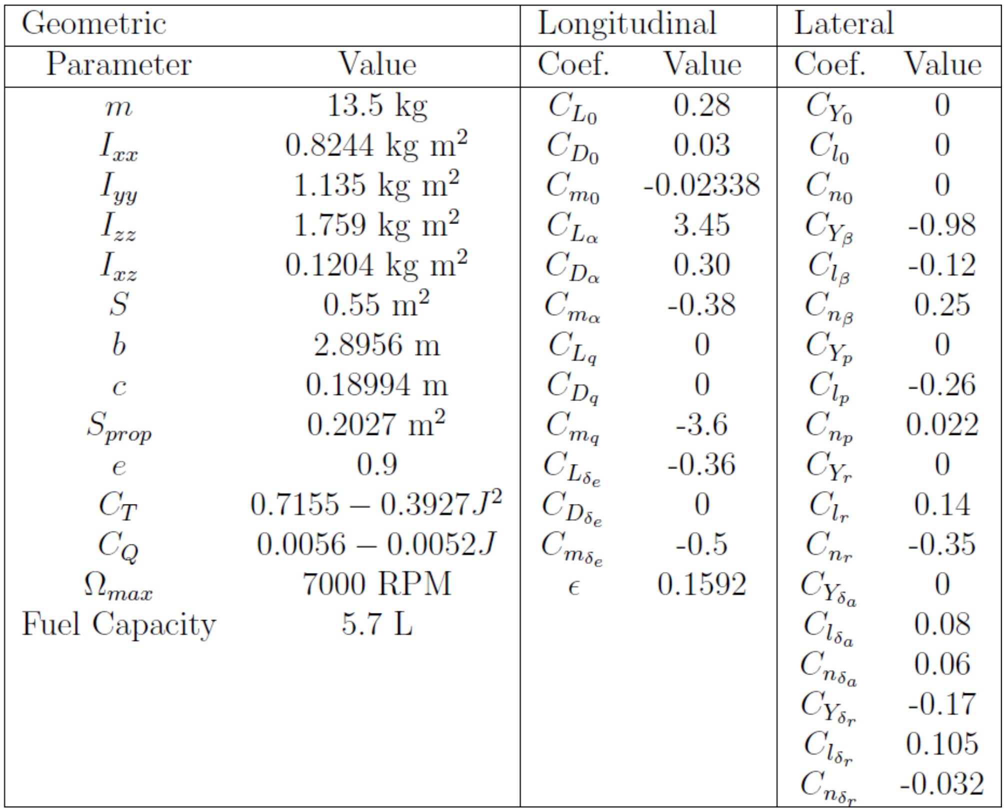
Keywords: sUAS, Multirotor, Control System, Fixed-wing

Chapter 1

Fixed Wind sUAS Development

1.1 Range & Endurance

The fixed-wing sUAS we used for this project was the Aerosonde UAV. The parameters as given for the Aerosonde UAV[5] are in Figure 1.1.



Geometric		Longitudinal		Lateral	
Parameter	Value	Coef.	Value	Coef.	Value
m	13.5 kg	C_{L_0}	0.28	C_{Y_0}	0
I_{xx}	0.8244 kg m^2	C_{D_0}	0.03	C_{l_0}	0
I_{yy}	1.135 kg m^2	C_{m_0}	-0.02338	C_{n_0}	0
I_{zz}	1.759 kg m^2	C_{L_α}	3.45	C_{Y_β}	-0.98
I_{xz}	0.1204 kg m^2	C_{D_α}	0.30	C_{l_β}	-0.12
S	0.55 m^2	C_{m_α}	-0.38	C_{n_β}	0.25
b	2.8956 m	C_{L_q}	0	C_{Y_p}	0
c	0.18994 m	C_{D_q}	0	C_{l_p}	-0.26
S_{prop}	0.2027 m^2	C_{m_q}	-3.6	C_{n_p}	0.022
e	0.9	$C_{L_{\delta_e}}$	-0.36	C_{Y_r}	0
C_T	$0.7155 - 0.3927J^2$	$C_{D_{\delta_e}}$	0	C_{l_r}	0.14
C_Q	$0.0056 - 0.0052J$	$C_{m_{\delta_e}}$	-0.5	C_{n_r}	-0.35
Ω_{max}	7000 RPM	ϵ	0.1592	$C_{Y_{\delta_a}}$	0
Fuel Capacity	5.7 L			$C_{l_{\delta_a}}$	0.08
				$C_{n_{\delta_a}}$	0.06
				$C_{Y_{\delta_r}}$	-0.17
				$C_{l_{\delta_r}}$	0.105
				$C_{n_{\delta_r}}$	-0.032

Figure 1.1: Given Parameters

1.1.1 Range

The range of an aircraft is technically defined as the total distance measured with respect to the ground traversed by the aircraft on one full tank of fuel. It is used to estimate engine[2] performance in terms of fuel consumption. The maximum total range is the maximum distance an aircraft can fly between takeoff and landing, as limited by fuel capacity in powered aircraft or cross-country speed and environmental conditions in unpowered aircraft. In this project, the maximum range of the given aircraft is calculated with respect to the propeller data obtained from the UIUC database for the given propeller (APC 16X8)[6].

To calculate the range of the aircraft with respect to the different powertrain systems, the following specifications are adopted:

1. **Specific Fuel Consumption (SFC) (for propeller)** : Specific fuel consumption is one of the critical metrics in determining the performance of a propeller-driven aircraft. SFC is defined as the weight of the fuel consumed by the reciprocating engine per unit power per unit time.
2. **Thrust specific fuel consumption (TSFC) (for jet engine)** : The thrust specific fuel consumption of a jet engine is defined as the fuel efficiency of an engine design with respect to thrust output. TSFC is the mass of fuel burned by an engine in one hour divided by the thrust that the engine produces.

The maximum range for the APC 16X8 propeller is calculated using the given formula:

$$R = \int_{W_0}^{W_1} \frac{\eta}{c_p}\frac{C_L}{C_D}\frac{dW}{W} = \frac{\eta}{c_p}\frac{C_L}{C_D} ln(\frac{W_0}{W_1}) \tag{1.1}$$

$$R_{max} = \frac{\eta}{c_p}\frac{C_L}{C_D} ln(\frac{W_0}{W_1}) = \frac{\eta/c_p}{2\sqrt{KC_{D_0}}} ln(\frac{W_0}{W_1}) \tag{1.2}$$

Note : To cover the longest distance, common sense says that we must use the minimum fuel consumption per unit distance (e.g., km or mile).

1.1.2 Endurance

The endurance of an aircraft is defined as the total time that an aircraft stays in the air on a tank of fuel. Like the range characteristics, to achieve maximum endurance, it is advised to use the minimum thrust per unit time. For a steady level flight, the lift is equal to the weight, and the thrust is equal to the drag (L = W, T = D).

To calculate the maximum endurance for a propeller-driven aircraft, the following parameters are required :

1. Maximum weight loss $(W_f = W_0 – W_1)$
2. Maximum $C_l^{\frac{3}{2}}/C_d$

The maximum endurance for a propeller-driven aircraft is calculated using the given formula :

$$E = \int_{W_0}^{W_1} \frac{\eta}{c_p V} \frac{C_L}{C_D} \frac{dW}{W} \tag{1.3}$$

$$= \int_{W_0}^{W_1} \frac{\eta}{c_p} \sqrt{\frac{\rho S}{2}} \frac{C_L^{1.5}}{C_D} \frac{dW}{W^{1.5}} \tag{1.4}$$

$$E = \frac{\eta}{c_p} \sqrt{2\rho S} (\frac{C_L^{1.5}}{C_D})(\frac{1}{\sqrt{W_1}} - \frac{1}{\sqrt{W_0}}) \tag{1.5}$$

Based on the above-mentioned procedures, the results of the fixed-wing UAV model are as follows:

Max. Range = 390.04 kms

Max. Endurance = 5.88 hours

1.2 Fixed Wing Aircraft Dynamics

We are using the linearized models in the [4] for developing the flight dynamics for the fixed-wing aircraft. The aircraft dynamics are primarily divided into two different parts, The longitudinal dynamics and the lateral dynamics. The linearized equations are demonstrated in the State Space Models as given in the further subsections. The longitudinal dynamics have the elevator and the thrust as the inputs. The elevator controls the pitching, and the thrust controls the Speed. The lateral dynamics have the aileron and the rudder as the inputs.

1.2.1 Longitudinal Flight Dynamics

The longitudinal dynamics has six states, u, w, q, θ, h. The longitudinal state space can be seen as below. The longitudinal dynamics equations for all these states are linearized using Taylor's sequence at the equilibrium locations. We calculate the equilibrium values using the non-linear Dynamic Equations.

Given a non-linear system described by the differential equations

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

where $f : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n, x$ is the state of the system, and u is the input, the system is said to be in equilibrium at the state x^* and input u^* if

$$f\left(x^*, u^*\right) = 0.$$

When an aircraft is in constant-altitude, wings-level steady flight, a subset of its states are in equilibrium. In particular, the altitude h; the body frame velocities u, v, w; the Euler angles ϕ, θ, ψ; and the angular rates p, q, and r are all constant, Zero in this case.

The final state-space equation is as given below

$$\begin{pmatrix} \dot{\bar{u}} \\ \dot{\bar{w}} \\ \dot{\bar{q}} \\ \dot{\bar{\theta}} \\ \dot{\bar{h}} \end{pmatrix} = \begin{pmatrix} X_u & X_w & X_q & -g\cos\theta^* & 0 \\ Z_u & Z_w & Z_q & -g\sin\theta^* & 0 \\ M_u & M_w & M_q & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ \sin\theta^* - \cos\theta^* & 0 & u^*\cos\theta^* + w^*\sin\theta^* 0 & & \end{pmatrix} \begin{pmatrix} \bar{u} \\ \bar{w} \\ \bar{q} \\ \bar{\theta} \\ \bar{h} \end{pmatrix}$$

$$+ \begin{pmatrix} X_{\delta_e} & X_{\delta_t} \\ Z_{\delta_e} & 0 \\ M_{\delta_e} & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \bar{\delta}_e \\ \bar{\delta}_t \end{pmatrix}$$

The coefficients of the matrix are as given below:

```
Cxo = -P.CDo*cos(alpha) + P.CLo*sin(alpha)
Cxa = -P.CDa*cos(alpha) + P.CLa*sin(alpha)
Cxq = -P.CDq*cos(alpha) + P.CLq*sin(alpha)
Cxdelta_e = -P.CDdelta_e*cos(alpha) + P.CLdelta_e*sin(alpha)

Xu  = (2*qs*Cxo - rho*Sprop*Ue*C_T)/m
Xw  = Cxa*qs/m
Xq  = Cxq*qs1*c_bar/(2*m)
Xdelta_e = Cxdelta_e*qs1_2/m
Xdelta_t = (rho*Sprop*C_T*k^2)/m

Czo = -P.CLo*cos(alpha) - P.CDo*sin(alpha)
Cza = -P.CLa*cos(alpha) - P.CDa*sin(alpha)
Czq = -P.CLq*cos(alpha) - P.CDq*sin(alpha)
Czdelta_e = -P.CLdelta_e*cos(alpha) - P.CDdelta_e*sin(alpha)
```

```
Zu    = 2*qs*Czo/m
Zw    = Cza*qs/m
Zq    = Ue + Czq*qs1*c_bar/(2*m)
Zdelta_e = Czdelta_e*qs1_2/m

Mu    = 2*qs*c_bar*Cmo/Iyy
Mw    = Cma * qs * c_bar/Iyy
Mq    = Cmq*qs1*c_bar*c_bar/(2*Iyy)
Mdelta_e = Cmdelta_e*qs1_2*c_bar/Iyy
```

1.2.2 Lateral Flight Dynamics

For the lateral dynamics, the variables of interest are the roll angle ϕ, the roll rate p, the heading angle ψ, and the yaw rate r. The control surfaces used to influence the lateral dynamics are the ailerons δ_a, and the rudder δ_r . The ailerons are primarily used to influence the roll rate p, while the rudder is primarily used to control the yaw ψ of the aircraft.

$$\begin{pmatrix} \dot{\bar{v}} \\ \dot{\bar{p}} \\ \dot{\bar{r}} \\ \dot{\bar{\phi}} \\ \dot{\bar{\psi}} \end{pmatrix} = \begin{pmatrix} Y_v & Y_p & Y_r & g\cos\theta^*\cos\phi^* & 0 \\ L_v & L_p & L_r & 0 & 0 \\ N_v & N_p & N_r & 0 & 0 \\ 0 & 1 & \cos\phi^*\tan\theta^* & q^*\cos\phi^*\tan\theta^* - r^*\sin\phi^*\tan\theta^* & 0 \\ 0 & 0 & \cos\phi^*\sec\theta^* & p^*\cos\phi^*\sec\theta^* - r^*\sin\phi^*\sec\theta^* & 0 \end{pmatrix}$$
$$\times \begin{pmatrix} \bar{v} \\ \bar{p} \\ \bar{r} \\ \bar{\phi} \\ \bar{\psi} \end{pmatrix} + \begin{pmatrix} Y_{\delta_a} & Y_{\delta_r} \\ L_{\delta_a} & L_{\delta_r} \\ N_{\delta_a} & N_{\delta_r} \\ 0 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \bar{\delta}_a \\ \bar{\delta}_r \end{pmatrix}$$

The coefficients of the matrix are as given below:

```
Yv = 2*Cyo*qs/m + qs*Cyb/m
Yp = Cyp*qs1*b/(2*m)
Yr = -Ue + (Cyr*qs1*b/(2*m))
Ydelta_a = Cydelta_a*qs1_2/m
Ydelta_r = Cydelta_r*qs1_2/m

Lv = Cpb*qs*b
Lp = Cpp*qs1*b^2/2
Lr = Cpr*qs*b^2/2
```

```
Ldelta_a = Cpdelta_a * qs1_2 * b
Ldelta_r = Cpdelta_r * qs1_2 * b

Nv = Crb * qs * b
Np = Crp * qs1 * b^2 / 2
Nr = Crr * qs1 * b^2 / 2
Ndelta_a = Crdelta_a * qs1_2 * b
Ndelta_r = Crdelta_r * qs1_2 * b
```

1.3 Simulink Model

The given state-space setup was implemented in MATLAB with a Level2 S-Function block. The block takes 4 inputs in the following order : δ_e, δ_a, δ_r, δ_t. The block further provides 12 outputs, giving the state for both the longitudinal and lateral Systems. The complete Simulink Model can be seen in the figure 1.2.

The Simulink model performs all the three maneuvers in the order provided in the equation. The input for the system is controlled from the reference position subsystem, which gives out five outputs, height h, ψ, ϕ, V, U. The errors are calculated using feedback loops and are further fed to PIDs. A radian to degree converter is used in ϕ, and ψ as the outputs are in radians. The reference position subsystem is time-based with a digital clock as input and a time-based if-else block. We calculated the time for each maneuver and used the digital clock to change the input for the positions.

PID tuning methods used for this project were in-built PID tuners provided by Simulink. For tuning PD controllers, the following steps were taken:

1. Proportional gain K_p was increased until steady oscillations were obtained

2. Derivate gain K_d was increased until the oscillations were critically damped

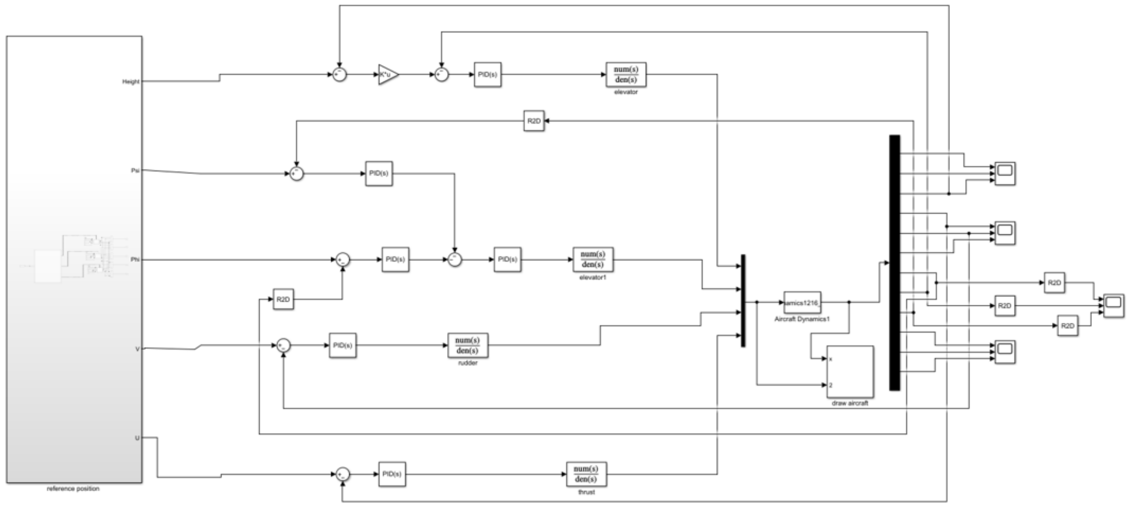


Figure 1.2: Simulink Model

Chapter 2

Multi-Rotor Development

2.1 Range & Endurance

For a multirotor, the range can be estimated by how far it can be controlled by the controller before losing its connection. The range of a multirotor is dependent on certain factors like weight, motor power, weather conditions, etc[3]. Based on the application requirement, they can be customized for longer ranges and better performances. In this project, the forward flight momentum theory was used to estimate the range and endurance of the aircraft for a given set of forward speeds. A velocity range of 0-20 m/s was chosen to perform this calculation, and the estimated values of range, endurance, and forward speeds of the multi-rotor drone were calculated using the following steps as follows:

2.1.1 Forward Flight Momentum Theory

The forward flight expression can be written as:

$$v = \frac{T}{2\rho A\sqrt{V^2 cos^2\alpha_D + (v + V sin\alpha_D)^2}} \tag{2.1}$$

At a given forward speed, V we can then solve for α_D, v, T, P_{ind}, P_{tot} as follows:

1. Calculate the quadrotor drag $D = \frac{\rho C_D V^2}{2}$ at $\alpha_D = 0$ Assume α_D does not affect the drag, otherwise need to iterate to find solution
2. Solve $\alpha_D = tan^{-1}(\frac{D}{W})$
3. Square both sides of 2.1 replace T^2 by W^2 + D^2 and re-arrange to get,

   $v^4 + (2V sin(\alpha_D))v^3 + v^2V^2 - (W^2 + D^2)/(2\rho A)^2 = 0$

4. Positive real root of equation gives v, then $P_{ind} = T_v; P_{tot} = T(v + Vsin(\alpha_D))$

5. Notice that this total power does not include profile drag, swirl, or additional losses due to non-uniform induced velocity.

6. Solve for a range of speeds and plot results versus V

The maximum range of the quadrotor is calculated using the formula:

Max. Range = $(\frac{E_b * m_e * esc_e}{MinimumTotalPower/Velocity})$

Where, E_b = Energy of the battery; M_e = Motor efficiency; Esc_e = ESC efficiency

The maximum endurance of the quadrotor is calculated using the formula:

Max. Endurance = $(E_b * m_e * esc_e)/(MinimumTotalPower)$
Where, E_b = Energy of the battery; M_e = Motor efficiency; Esc_e = ESC efficiency

Based on the above-mentioned procedures, the results of the multi-rotor drone (quadrotor) are as follows:

- Max. Range = 34 kms
- Max. Endurance = 68.1 minutes
- Forward Speed for range = 9.5 m/s
- Forward Speed for endurance = 7 m/s

2.2 Quadrotor Dynamics Model

The development of the dynamics model of the quadrotor was done by using the following state-space matrices given for each control channel.

Given State-Space Matrices :

- Roll

$$A = \begin{bmatrix} -4.2683 & -3.1716 \\ 4 & 0 \end{bmatrix} \qquad B = \begin{bmatrix} 2 \\ 0 \end{bmatrix}$$

$$C = \begin{bmatrix} 0.7417 & 0.4405 \end{bmatrix} \qquad D = \begin{bmatrix} 0 \end{bmatrix}$$

- Pitch

$$A = \begin{bmatrix} -3.9784 & -2.9796 \\ 4 & 0 \end{bmatrix} \qquad B = \begin{bmatrix} 2 \\ 0 \end{bmatrix}$$

$$C = \begin{bmatrix} 1.2569 & 0.6083 \end{bmatrix} \qquad D = \begin{bmatrix} 0 \end{bmatrix}$$

- Yaw

$$A = \begin{bmatrix} -0.0059 \end{bmatrix} \qquad B = \begin{bmatrix} 1 \end{bmatrix}$$

$$C = \begin{bmatrix} 1.2653 \end{bmatrix} \qquad D = \begin{bmatrix} 0 \end{bmatrix}$$

- Height

$$A = \begin{bmatrix} -5.8200 & -3.6046e^{-6} \\ 3.8147e^{-6} & 0 \end{bmatrix} \qquad B = \begin{bmatrix} 1024 \\ 0 \end{bmatrix}$$

$$C = \begin{bmatrix} 1.4907e^{-4} & 1.3191e^{3} \end{bmatrix} \qquad D = \begin{bmatrix} 0 \end{bmatrix}$$

- Pitch to u

$$A = \begin{bmatrix} -0.665 \end{bmatrix} \qquad B = \begin{bmatrix} 2 \end{bmatrix}$$

$$C = \begin{bmatrix} -3.0772 \end{bmatrix} \qquad D = \begin{bmatrix} 0 \end{bmatrix}$$

- Roll to v

$$A = \begin{bmatrix} -0.4596 \end{bmatrix} \qquad B = \begin{bmatrix} 2 \end{bmatrix}$$

$$C = \begin{bmatrix} 2.3868 \end{bmatrix} \qquad D = \begin{bmatrix} 0 \end{bmatrix}$$

Figure 2.1: State-Space Matrices

The output from the position controller was fed into the dynamics model. The dynamics model consists of each control channel mentioned above put in a separate state-space block in Simulink. For roll control, the input was the desired roll angle, and the output from the state-space block was the actual roll angle. This actual roll angle was then fed into the roll to v control channel as input to get the actual velocity v about the y-axis.

For pitch control, the input was desired pitch angle, and the output was the actual pitch angle. This actual pitch angle was then fed into the pitch to u control channel, which gives the actual velocity about the x-axis.

The yaw control channel used desired yaw rate, also called as desired heading rate, for input, while the output of this control channel is the actual yaw angle. The last control channel used for the dynamic model of this quadrotor is altitude control. For this, the input was desired vertical velocity w while the output was orientation along the z-axis, which is also called altitude.

The Simulink design of dynamic model is shown in the figure below:

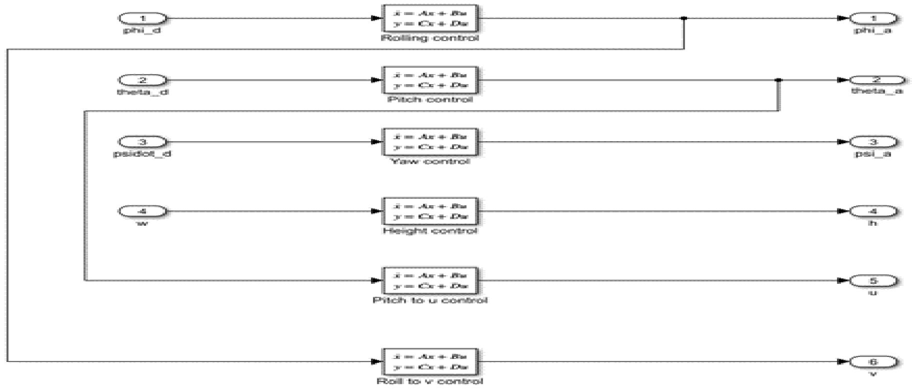


Figure 2.2: Simulink Design Dynamic Model

2.3 Development of Position/Orientation control system:

Position/Orientation control system contains two types of state input, one is the target x, y, and z coordinate input, and the other is actual x, y and z coordinates of the quadrotor. A reference yaw command is also given into the position controller to ensure that the quadrotor maintains the commanded yaw while operating for the target position.

The target x-coordinate is given into a summation block while the actual x orientation of the quadrotor is given to the summation block as feedback. This command is fed into the PD controller, which gives corresponding derivative $\dot{x}$ (i.e., speed along the x-axis). This derived velocity u is given to the summation block, whereas the actual velocity u_a is given as feedback in the same block. A PD controller is used again, which finally gives the derivative $\ddot{x}$.

Similarly, the target y position and actual y position of the quadrotor are also given into the summation block and then into the PD controller. The output from this PD controller is further given into a summation block where the actual velocity v is given as feedback. This output is then given into the PD controller to get acceleration along the y-axis.

Actual heading angle psi, along with $\ddot{x}$ and $\ddot{y}$ is used to calculate the desired pitch angle and desired roll angle[1]. The following equations are used for this:

$$\begin{bmatrix} \phi_d \\ \theta_d \end{bmatrix} = \begin{bmatrix} -\sin\psi & -\cos\psi \\ \cos\psi & -\sin\psi \end{bmatrix}^{-1} \frac{m}{U_1} \begin{bmatrix} \ddot{x}_d \\ \ddot{y}_d \end{bmatrix}$$

Since the derivation of these equations in (reference no) uses a small-angle approximation, we must ensure that the desired angles θ_d and ϕ_d are within the limit of –20° and 20°. Therefore, a saturation block is placed at the output of the MATLAB function block.

We also need the desired vertical velocity w and the desired heading rate

2.4 Development of Position Estimation

For estimating the position of the quadrotor, the output states from the dynamic model, i.e., psi, psi, theta, h, u and v, are used. The dynamic model provides information about the actual value of states of the quadrotor. To acquire data regarding the actual x and y coordinates of the quadrotor, we integrate u and v velocities, respectively. The Simulink model of the position estimation subsystem is shown in 2.3.

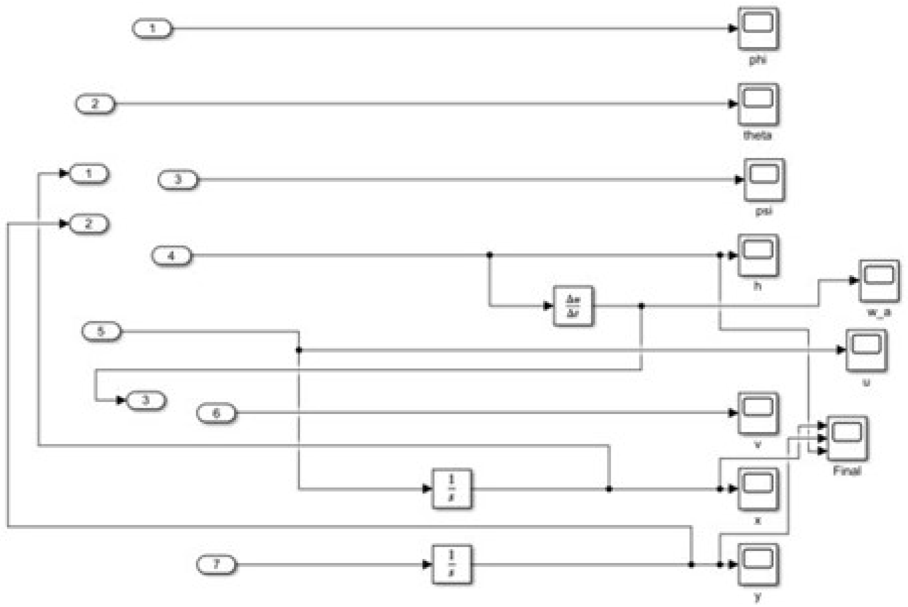


Figure 2.3: Position Estimation

2.5 Development of linear simulation model

For the development of the linear simulation model, a reference subsystem was created, which was used for giving the desired coordinate information to the position controller. The desired pitch and roll angles θ_d and ϕ_d along with the heading rate $\dot{psi}$ and desired vertical velocity wd goes into a summation block followed by a PID controller. To this summation, block feedback is given of the actual orientation of the respective states.

Further, these values are given into the dynamics model subsystem, which is also explained in the previous section. The actual values obtained from the dynamic model are used in the position estimation subsystem to get information about Euler angles, position, and velocities of the quadrotor.

The reference position subsystem developed in for this project is an If-Else condition block which is used with action block to activate a particular command for an If-Else condition. The reference command subsystem is shown in the Figure 2.4.

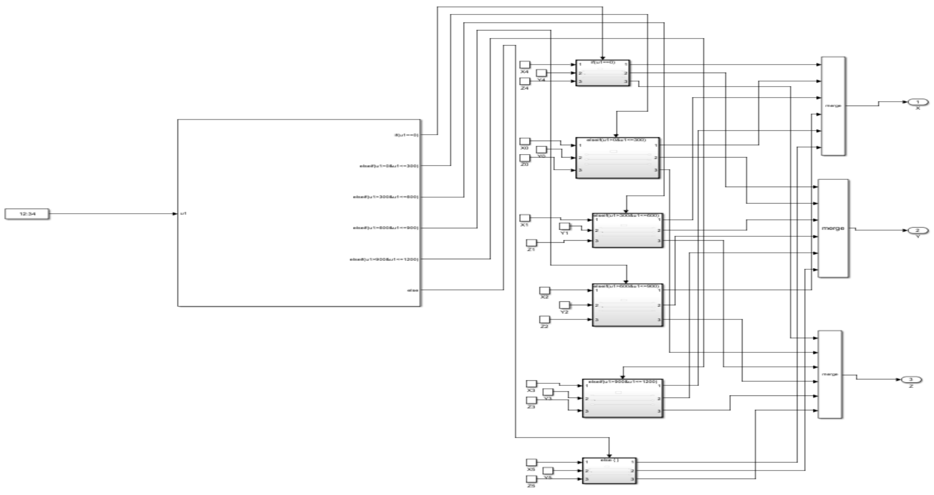


Figure 2.4: Reference Position

The complete linear simulation system that was developed in Simulink is shown in Figure 2.5

PID tuning methods used for this project were in-built PID tuners provided by Simulink. For tuning PD controllers, the following steps were taken:

1. Proportional gain K_p was increased until steady oscillations were obtained

2. Derivate gain K_d was increased until the oscillations were critically damped

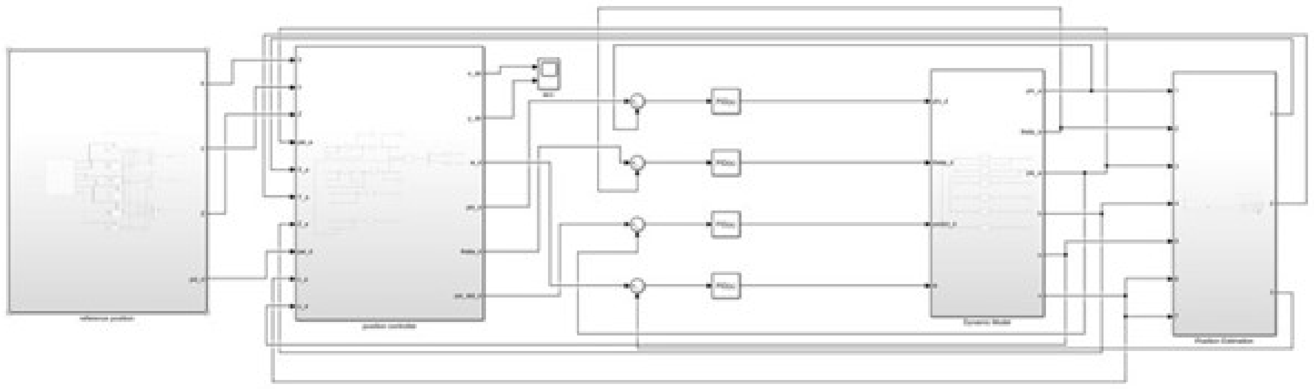


Figure 2.5: Complete Multirotor Dynamic System

Total simulation time was taken to be 1500 seconds for the following four maneuvers:

1. Take off and hover at 2 meters above origin
2. Fly to the first target (x = 5 m, y = 6 m, h = 4 m) and hover
3. Fly to the second target (x = -5 m, y = -6 m, h = 4 m) and hover
4. Return to 2 meters above origin and land

2.6 Result

The quadcopter was successfully able to complete all the given maneuvers for this project. The simulation result, which shows the trajectory of x, y and z coordinates of the quadcopter for 1500 seconds, is shown in.

In the figure, the quadcopter is initially at x=0, y=0 and z=0 position. In the first 300 seconds, it takes off and hovers at x=0 y=0 and z=2 m. Next, the quadcopter reaches x=5, y=6 and z=4. For the third maneuver, it goes from its current position to the target position, x=-5, y=-6 and z=4 and then for the last maneuver are goes from the current position to the target position of x=0, y=0 and z= two and then back to the origin.

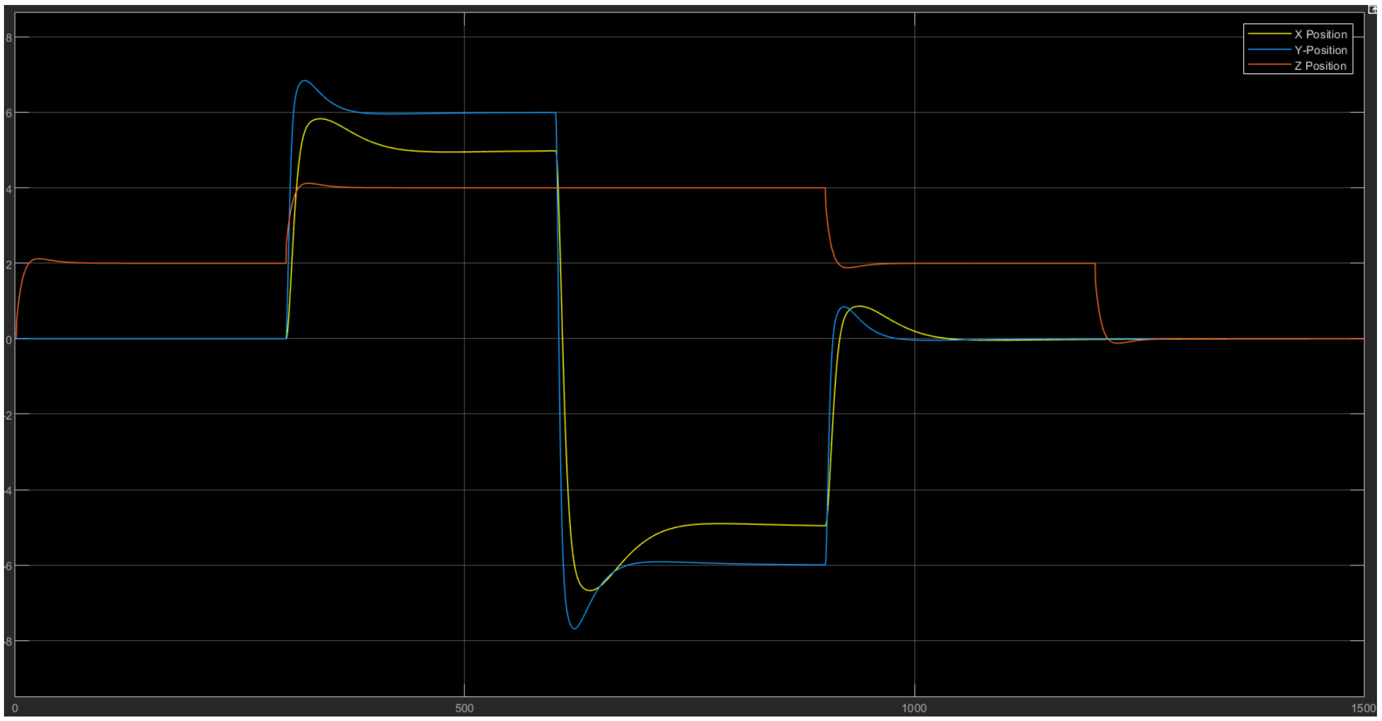


Figure 2.6: Quadrotor Simulation Solution

Chapter 3

Conclusions

The various Technical Conclusions we arrived upon during the implementation of the fixed-wing and multi-rotor UAS configurations are listed below as follows:

- The theoretical values obtained using various theories like the Momentum Theory, or Thin-Airfoil theory will vary largely from the obtained values of a flying system because of the assumptions and inconsistencies in data.
- Implementation of non-linear dynamics will provide more accurate simulation than implementing a linearized system. Linearization decouples the connections between the parameters, so it can be beneficial for learning about the properties of the system.
- The obtained values of range and endurance in the fixed-wing and multirotor aircraft are unrealistic, mainly because of the inconsistencies in the data.
- The same control action can be obtained with multiple control actions, like coordinated turn in Fixed-wing and the horizontal movements in multi-rotor

This was our first attempt at developing an Autopilot system, and we learned a lot of lessons from this project. Some of the most notable technical learning was working with Simulink, setting up control blocks and designing a control system in Simulink. This exercise gave us a firm foundation in control system design and development, which will benefit us in further courses.

The Work was divided into five main parts, with three individual components.

1. Range & Endurance Calculations(both Questions) : Siva Subramanian
2. Fixed Wing Control System Development: Rutvik Solanki
3. Multirotor Control System Development : Prakriti Saini
4. Setting up the System for Final Simulation & all Maneuvers: Everyone
5. Documentation & Presentation : Everyone

Though we needed to divide Work amongst ourselves, working together on the simulation and Documentation provided each one of us with thorough knowledge of both the Fixed-Wing and Multi-rotor control systems.

Bibliography

[1] Maki K. Habib Heba Elkholy. *Advancements in Robotics and Mechatronics*. IGI Global, 2015.

[2] Lycoming. *Lycoming Engines*. URL: `https://www.lycoming.com/engines/el-005`.

[3] NPTEL. *NPTEL Lecture*. URL: `https://nptel.ac.in/content/storage2/courses/101104007/Module2/Lec8.pdf`.

[4] Timothy W. McClain Randal W. Beard. *Small Unmanned Aircraft*. Princeton, 2012.

[5] Textron Systems. *Aerosonde UAV*. URL: `https://www.textronsystems.com/products/aerosonde`.

[6] UIUC. *Propeller Data*. URL: `https://m-selig.ae.illinois.edu/props/volume-3/propDB-volume-3.html`.

Chapter 4

Appendix

4.1 Fixed Wing Aircraft Dynamics

```
function aircraft_dynamics1216_PROJECT(block)
%MSFUNTMPL_BASIC A Template for a Level-2 MATLAB S-Function
%   The MATLAB S-function is written as a MATLAB function with the
%   same name as the S-function. Replace 'msfuntmpl_basic' with the
%   name of your S-function.
%
%   It should be noted that the MATLAB S-function is very similar
%   to Level-2 C-Mex S-functions. You should be able to get more
%   information for each of the block methods by referring to the
%   documentation for C-Mex S-functions.
%
%   Copyright 2003-2010 The MathWorks, Inc.

% AER1216 Fall 2021
% Fixed Wing Project Code
%
% aircraft_dynamics.m
%
% Fixed wing simulation model file, based on the Aerosonde UAV, with code
% structure adapted from Small Unmanned Aircraft: Theory and Practice by
% R.W. Beard and T. W. McLain.
%
% Inputs:
% delta_e            elevator deflection [deg]
% delta_a            aileron deflection [deg]
% delta_r            rudder deflection [deg]
% delta_t            normalized thrust []
%
% Outputs:
```

```
% pn              inertial frame x (north) position [m]
% pe              inertial frame y (east) position [m]
% pd              inertial frame z (down) position [m]
% u               body frame x velocity [m/s]
% v               body frame y velocity [m/s]
% w               body frame z velocity [m/s]
% phi             roll angle [rad]
% theta           pitch angle [rad]
% psi             yaw angle [rad]
% p               roll rate [rad/s]
% q               pitch rate [rad/s]
% r               yaw rate [rad/s]
%
% Last updated: Pravin Wedage 2021-11-09

%% TA NOTE
% The code segements you must modify are located in the derivatives
% function in this .m file. Modify other sections at your own risk.


%
% The setup method is used to set up the basic attributes of the
% S-function such as ports, parameters, etc. Do not add any other
% calls to the main body of the function.
%
setup(block);

end


%% Function: setup ===================================================
% Abstract:
%   Set up the basic characteristics of the S-function block such as:
%   - Input ports
%   - Output ports
%   - Dialog parameters
%   - Options
%
%   Required         : Yes
%   C-Mex counterpart: mdlInitializeSizes
%
function setup(block)

% Register number of ports
block.NumInputPorts  = 1;
block.NumOutputPorts = 1;

% Setup port properties to be inherited or dynamic
block.SetPreCompInpPortInfoToDynamic;
block.SetPreCompOutPortInfoToDynamic;

% Override input port properties
for i = 1:block.NumInputPorts
    block.InputPort(i).Dimensions        = 4;
    block.InputPort(i).DatatypeID  = 0;  % double
    block.InputPort(i).Complexity  = 'Real';
    block.InputPort(i).DirectFeedthrough = false; % important to be false
end

% Override output port properties
for i = 1:block.NumOutputPorts
    block.OutputPort(i).Dimensions       = 12;
    block.OutputPort(i).DatatypeID  = 0; % double
    block.OutputPort(i).Complexity  = 'Real';
%     block.OutputPort(i).SamplingMode = 'Sample';
end

% Register parameters
block.NumDialogPrms     = 1;
P = block.DialogPrm(1).Data; % must duplicate this line in each function

% Register sample times
%  [0 offset]            : Continuous sample time
%  [positive_num offset] : Discrete sample time
%
%  [-1, 0]               : Inherited sample time
%  [-2, 0]               : Variable sample time
block.SampleTimes = [0 0];

% Register multiple instances allowable
% block.SupportMultipleExecInstances = true;

% Register number of continuous states
block.NumContStates = 12;

% Specify the block simStateCompliance. The allowed values are:
%    'UnknownSimState', < The default setting; warn and assume DefaultSimSt
%    'DefaultSimState', < Same sim state as a built-in block
%    'HasNoSimState',   < No sim state
%    'CustomSimState',  < Has GetSimState and SetSimState methods
%    'DisallowSimState' < Error out when saving or restoring the model sim
block.SimStateCompliance = 'DefaultSimState';

% -----------------------------------------------------------------
% The MATLAB S-function uses an internal registry for all
% block methods. You should register all relevant methods
% (optional and required) as illustrated below. You may choose
% any suitable name for the methods and implement these methods
% as local functions within the same file. See comments
% provided for each function for more information.
% -----------------------------------------------------------------

% block.RegBlockMethod('PostPropagationSetup',    @DoPostPropSetup); % disc
block.RegBlockMethod('SetInputPortSamplingMode', @SetInpPortFrameData);
block.RegBlockMethod('InitializeConditions',    @InitializeConditions);
% block.RegBlockMethod('Start',                   @Start); % Initialize Con
block.RegBlockMethod('Outputs',                 @Outputs); % Required
% block.RegBlockMethod('Update',                  @Update); % only required
block.RegBlockMethod('Derivatives',             @Derivatives); % Required f
block.RegBlockMethod('Terminate',               @Terminate); % Required

end


%% PostPropagationSetup:
%   Functionality    : Setup work areas and state variables. Can
%                      also register run-time methods here
%   Required         : No
%   C-Mex counterpart: mdlSetWorkWidths
%
function DoPostPropSetup(block)
block.NumDworks = 1;

  block.Dwork(1).Name            = 'x1';
  block.Dwork(1).Dimensions      = 1;
  block.Dwork(1).DatatypeID      = 0;      % double
  block.Dwork(1).Complexity      = 'Real'; % real
  block.Dwork(1).UsedAsDiscState = true;

end


%% InitializeConditions:
%   Functionality    : Called at the start of simulation and if it is
%                      present in an enabled subsystem configured to reset
%                          states , it will be called when the enabled subsystem
%                          restarts execution to reset the states.
%   Required             : No
%   C-MEX counterpart:  mdlInitializeConditions
%
function InitializeConditions(block)

% Rename parameters
P = block.DialogPrm(1).Data; % must duplicate this line in each function

% Initialize continuous states
block.ContStates.Data(1) = P.pn0;
block.ContStates.Data(2) = P.pe0;
block.ContStates.Data(3) = P.pd0;
block.ContStates.Data(4) = P.u0;
block.ContStates.Data(5) = P.v0;
block.ContStates.Data(6) = P.w0;
block.ContStates.Data(7) = P.phi0;
block.ContStates.Data(8) = P.theta0;
block.ContStates.Data(9) = P.psi0;
block.ContStates.Data(10) = P.p0;
block.ContStates.Data(11) = P.q0;
block.ContStates.Data(12) = P.r0;

end

%% Start:
%   Functionality     : Called once at start of model execution. If you
%                          have states that should be initialized once, this
%                          is the place to do it.
%   Required             : No
%   C-MEX counterpart:  mdlStart
%
function Start(block)

block.Dwork(1).Data = 0;

end

%% Input Port Sampling Method:
function SetInpPortFrameData(block, idx, fd)

  block.InputPort(idx).SamplingMode = 'Sample';
  for i = 1:block.NumOutputPorts
    block.OutputPort(i).SamplingMode  = 'Sample';
```

```
   end
end

%% Outputs:
%   Functionality     : Called to generate block outputs in
%                       simulation step
%   Required          : Yes
%   C-MEX counterpart: mdlOutputs
%
function Outputs(block)

temp_mat = zeros(block.NumContStates,1); % thirteen states
for i = 1:block.NumContStates
     temp_mat(i) = block.ContStates.Data(i);
end

block.OutputPort(1).Data = temp_mat; % states

% for i = 1:block.NumOutputPorts
%     block.OutputPort(1).Data(i) = block.ContStates.Data(i);
% end

end


%% Update:
%   Functionality     : Called to update discrete states
%                       during simulation step
%   Required          : No
%   C-MEX counterpart: mdlUpdate
%
function Update(block)

block.Dwork(1).Data = block.InputPort(1).Data;

end


%% Derivatives:
%   Functionality     : Called to update derivatives of
%                       continuous states during simulation step
%   Required          : No
%   C-MEX counterpart: mdlDerivatives
%
function Derivatives(block)
```

```
% Rename parameters
P = block.DialogPrm(1).Data; % must duplicate this line in each function

% compute inertial constants
% K = ;
% k1 = ;
% k2 = ;
% k3 = ;
% k4 = ;
% k5 = ;
% k6 = ;
% k7 = ;
% k8 = ;

% map states and inputs
pn    = block.ContStates.Data(1);
pe    = block.ContStates.Data(2);
h    = block.ContStates.Data(3);
u     = block.ContStates.Data(4);
v     = block.ContStates.Data(5);
w     = block.ContStates.Data(6);
phi   = block.ContStates.Data(7);
theta = block.ContStates.Data(8);
psi   = block.ContStates.Data(9);
p     = block.ContStates.Data(10);
q     = block.ContStates.Data(11);
r     = block.ContStates.Data(12);
delta_e = block.InputPort(1).Data(1)*pi/180 ; % converted inputs to radians
delta_a = block.InputPort(1).Data(2)*pi/180 ; % converted inputs to radians
delta_r = block.InputPort(1).Data(3)*pi/180 ; % converted inputs to radians
delta_t = block.InputPort(1).Data(4);


%% Basic Inputs

g = 9.81;                    % gravity (m/s^2)
m = 13.5;             % mass (kg)
W = m*g;   % weight (N)
Ixx = 0.8244;   % moment of inertia (kg m^2)
Iyy = 1.135;   %
Izz = 1.759;   %
Ixz = 0.1204;
Izx = Ixz;
```

```
S = 0.55;                          % wing plan area (m^2)
c_bar = 0.18994;             % mean chord (m)
b = 2.8956;                        % wing span (m)
AR = (b^2)/S;
e = 0.9;      % oswald factor , CHECK
k = 1 /(pi*e*AR);

C_T=0.78; % We need to redefine


%  (2) flight conditions reference point ****

 Va_trim=P.Va_trim;
 Va = P.Va;
 Ue = 28.35          ;       % steady flight velocity (m/s):
 alpha = 0;
 theta_e = 0;
 Sprop = 0.2027;

 rho = 1.0582;    % density (kg/m^3) %% CHECK
% =========  dimensional derivatives ================
% q = 0.5*rho*Ue^2;
 qs = rho*Ue*S/2;
 qs1 = rho*Va*S/2;
 qs1_2 = rho*Va*Va*S/2;

%% Longitudinal terms

 Cxo = -P.CDo*cos(alpha) + P.CLo*sin(alpha);
 Cxa = -P.CDa*cos(alpha) + P.CLa*sin(alpha);
 Cxq = -P.CDq*cos(alpha) + P.CLq*sin(alpha);
 Cxdelta_e = -P.CDdelta_e*cos(alpha) + P.CLdelta_e*sin(alpha);

 Xu  = (2*qs*Cxo - rho*Sprop*Ue*C_T)/m;%
Xw  = Cxa*qs/m;
 Xq  = Cxq*qs1*c_bar/(2*m);
 Xdelta_e = Cxdelta_e*qs1_2/m;
 Xdelta_t = (rho*Sprop*C_T*k^2)/m;

 Czo = -P.CLo*cos(alpha) - P.CDo*sin(alpha);
 Cza = -P.CLa*cos(alpha) - P.CDa*sin(alpha);
 Czq = -P.CLq*cos(alpha) - P.CDq*sin(alpha);
 Czdelta_e = -P.CLdelta_e*cos(alpha) - P.CDdelta_e*sin(alpha);

 Zu  = 2*qs*Czo/m ;
```

```
Zw  = Cza*qs/m;
Zq  = Ue + Czq*qs1*c_bar/(2*m); %% no m term in ss matrix
Zdelta_e = Czdelta_e*qs1_2/m;

Cmo = P.Cmo;
Cma = P.Cma;
Cmq = P.Cmq;
Cmdelta_e = P.Cmdelta_e;

Mu  = 2*qs*c_bar*Cmo/Iyy;
Mw  = Cma * qs * c_bar/Iyy;
Mq  = Cmq*qs1*c_bar*c_bar/(2*Iyy);
Mdelta_e = Cmdelta_e*qs1_2*c_bar/Iyy;

%% Lateral terms

Cyo = P.Cyo;
Cyb = P.Cyb;
Cyp = P.Cyp;
Cyr = P.Cyr;
Cydelta_a = P.Cydelta_a;
Cydelta_r = P.Cydelta_r;

Yv = 2*Cyo*qs/m + qs*Cyb/m;
Yp = Cyp*qs1*b/(2*m);
Yr = -Ue + (Cyr*qs1*b/(2*m));
Ydelta_a = Cydelta_a*qs1_2/m;
Ydelta_r = Cydelta_r*qs1_2/m;

Cpb = P.Clb;
Cpp = P.Clp;
Cpr = P.Clr;
Cpdelta_a = P.Cldelta_a;
Cpdelta_r = P.Cldelta_r;

Lv = Cpb*qs*b;
Lp = Cpp*qs1*b^2/2;
Lr = Cpr*qs*b^2/2;
Ldelta_a = Cpdelta_a * qs1_2 * b;
Ldelta_r = Cpdelta_r * qs1_2 * b;

Crb = P.Cnb;
Crp = P.Cnp;
Crr = P.Cnr;
Crdelta_a = P.Cndelta_a;
```

```
Crdelta_r = P.Cndelta_r;

Nv = Crb * qs * b;
Np = Crp * qs1 * b^2 / 2;
Nr = Crr * qs1 * b^2 / 2;
Ndelta_a = Crdelta_a * qs1_2 * b;
Ndelta_r = Crdelta_r * qs1_2 * b;

%% Final Matrices

X_long=[u-Va w q theta h]';
U_long=[delta_e delta_t]';

Along=[Xu Xw Xq -g*cos(theta_e) 0;
       Zu Zw Zq -g*sin(theta_e) 0;
       Mu Mw Mq 0 0
       0 0 1 0 0;
       sin(theta_e) -cos(theta_e) 0 Ue*cos(theta_e) 0];

Blong=[Xdelta_e Xdelta_t;
       Zdelta_e 0;
       Mdelta_e 0;
       0 0;
       0 0];

X_long_dot=Along*X_long + Blong*U_long;

udot=X_long_dot(1);
wdot=X_long_dot(2);
qdot=X_long_dot(3);
thetadot=X_long_dot(4);
hdot = X_long_dot(5);

%Lateral
X_latr=[v p r phi psi]';
U_latr=[delta_a delta_r]';

Alatr=[Yv Yp Yr g*cos(theta_e) 0;
       Lv Lp Lr    0           0;
       Nv Np Nr    0           0;
       0 1 tan(theta_e) 0      0;
       0 0 sec(theta_e) 0      0];

Blatr=[Ydelta_a Ydelta_r;
       Ldelta_a Ldelta_r;
```

```
         Ndelta_a  Ndelta_r;
         0         0        ;
         0         0        ];

 Blatr(5,:)=[0  0];

 X_latr_dot=Alatr*X_latr + Blatr*U_latr;

 vdot=X_latr_dot(1);
 pdot=X_latr_dot(2);
 rdot=X_latr_dot(3);
 phidot=X_latr_dot(4);
 psidot=X_latr_dot(5);

 C_BE=[cos(theta)*cos(psi) sin(phi)*sin(theta)*cos(psi)-cos(phi)*sin(psi) co
     cos(theta)*sin(psi) sin(phi)*sin(theta)*sin(psi)+cos(phi)*cos(psi) cos(
     -sin(theta) sin(phi)*cos(theta) cos(phi)*cos(theta)]';

 X_e=C_BE*[u v w]';
 pndot = X_e(1); % X Value
 pedot =X_e(2) ; % Y Value

% rotation matrix

% Aerodynamic Coefficients
% compute the nondimensional aerodynamic coefficients here

% aerodynamic forces and moments
% compute the aerodynamic forces and moments here

% propulsion forces and moments
% compute the propulsion forces and moments here

% gravity
% compute the gravitational forces here

% total forces and moments (body frame)

% state derivatives
% the full aircraft dynamics model is computed here
% pdot = ;
% pndot = ;
% pedot = ;
% pddot = ;
% udot = ;
% vdot = ;
% wdot = ;
%
% phidot = ;
% thetadot = ;
% psidot = ;
%
% pdot = ;
% qdot = ;
% rdot = ;

% map derivatives
block.Derivatives.Data(1) = pndot;
block.Derivatives.Data(2) = pedot;
block.Derivatives.Data(3) = hdot;
block.Derivatives.Data(4) = udot;
block.Derivatives.Data(5) = vdot;
block.Derivatives.Data(6) = wdot;
block.Derivatives.Data(7) = phidot;
block.Derivatives.Data(8) = thetadot;
block.Derivatives.Data(9) = psidot;
block.Derivatives.Data(10)= pdot;
block.Derivatives.Data(11)= qdot;
block.Derivatives.Data(12)= rdot;

end


%% Terminate:
%   Functionality    : Called at the end of simulation for cleanup
%   Required         : Yes
%   C-MEX counterpart: mdlTerminate
%
function Terminate(block)

end
```

4.2 Multirotor State-Space Equations

```
%% AER1216 MULTI_ROTOR CODE
clc;
clear all;
```

```
A_roll=[4.2683  -3.1716;4  0];
B_roll=[2;0];
C_roll=[0.7417  0.4405];
D_roll=0;
%
A_pitch=[-3.9784  -2.9796;4  0];
B_pitch=[2;0];
C_pitch=[1.2569  0.6083];
D_pitch=0;
%
A_yaw=-0.0059;
B_yaw=1;
C_yaw=1.2653;
D_yaw=0;
%
A_h=[-5.8200  -3.6046e-6
3.8147e-6  0];
B_h=[1024;0];
C_h=[1.4907e-4  1.3191e3];
D_h=0;
%
A_p2u=-0.665;
B_p2u=2;
C_p2u=-3.0772;
D_p2u=0;
%
A_r2v=-0.4596;
B_r2v=2;
C_r2v=2.3868;
D_r2v=0;
%%
```