AAE4006 Flight Mechanics and Control Systems
Design Project Report

Tam Ka Hei (20045757D)

1 Introduction

Aircraft have over 120 years of history. With technological advancement, planes have different technologies such as auto-trim and autopilot. These technologies require computers that use formulae to calculate correct aircraft manoeuvres. It cannot perform visual flights like humans without using programmes and formulae. The systems need to undergo multiple simulations before implementing new technologies to ensure aviation safety. MATLAB and SimuLink are used in the design project to perform necessary simulations. The project aims to:

- Implement various aircraft control models
- Design PID controllers for the autopilot
- Build up a simulated safe flight control system

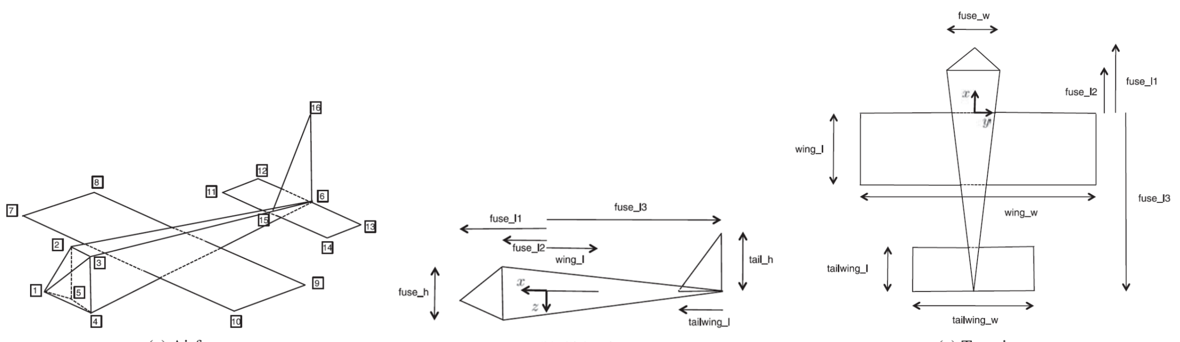
In this project, a simplified fixed-wing aircraft is created for the simulation. The 6-degree-of-freedom, 12-state model, force and moment model are implemented on the simplified aircraft. Two control models, manual and automatic, are designed for controlling the simplified aircraft. The manual control model represents pilots' manual control of aircraft, while the automatic control model represents the autopilot control system. The project creates an approximated automatic control model for fixed-wing aircraft applications. This model can be used by other researchers for further flight mechanics and control systems development, especially in the autopilot system.

2 Methodologies and results

The design project has 6 tasks. The design of the simplified aircraft and its coordinate system is carried out in tasks 1 and 2, while the designs of forces, torques and control surfaces of the aircraft are carried out in tasks 3 and 4. After building up the foundation of the simplified aircraft, the design models are computed in task 5, and the autopilot system is designed in task 6.

2.1 Task 1

This task consists of two goals, drawing the aircraft on the designed frames and performing colouring of the aircraft.



(a) Airframe (b) Side view (c) Top view

Figure 1: Aircraft structures

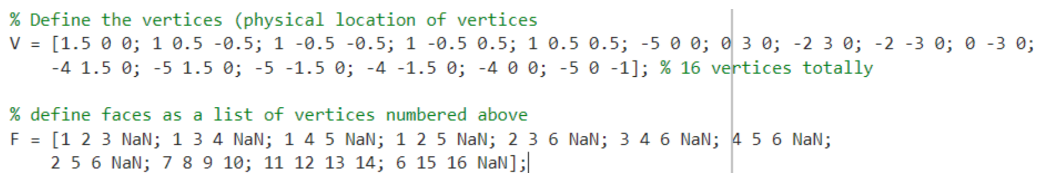
Points	X	Y	Z
1	1.5	0	0
2	1	0.5	-0.5
3	1	-0.5	-0.5
4	1	-0.5	0.5
5	1	0.5	0.5
6	-5	0	0
7	0	3	0
8	-2	3	0
9	-2	-3	0
10	0	-3	0
11	-4	1.5	0
12	-5	1.5	0
13	-5	-1.5	0
14	-4	-1.5	0
15	-4	0	0
16	-5	0	-1

Table 1: Aircraft coordinates

Parameters	Length (units)
fuse_l1	1.5
fuse_l2	1
fuse_l3	5
fuse_h	1
fuse_w	1
wing_l	2
wing_w	6
tail_h	1
tailwing_l	1
tailwing_w	3

Table 2: Aircraft parameters

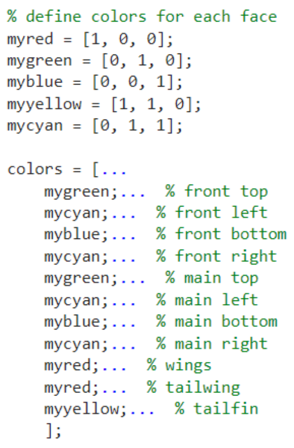
Figure 1 shows the basic structures of the aircraft. The coordinates are entered into the script, starting from point 1, referencing the x, y and z frames. Vertex lists are created after obtaining the vertices. The code (7 8 9 10) helps draw the face of 7-8-9-10, while the code (1 2 3 NaN) helps draw the face of 1-2-3. The codes used are as follows.



```
% Define the vertices (physical location of vertices
V = [1.5 0 0; 1 0.5 -0.5; 1 -0.5 -0.5; 1 -0.5 0.5; 1 0.5 0.5; -5 0 0; 0 3 0; -2 3 0; -2 -3 0; 0 -3 0;
    -4 1.5 0; -5 1.5 0; -5 -1.5 0; -4 -1.5 0; -4 0 0; -5 0 -1]; % 16 vertices totally

% define faces as a list of vertices numbered above
F = [1 2 3 NaN; 1 3 4 NaN; 1 4 5 NaN; 1 2 5 NaN; 2 3 6 NaN; 3 4 6 NaN; 4 5 6 NaN;
    2 5 6 NaN; 7 8 9 10; 11 12 13 14; 6 15 16 NaN];
```

Figure 2: Vertices and faces of aircraft



```
% define colors for each face
myred = [1, 0, 0];
mygreen = [0, 1, 0];
myblue = [0, 0, 1];
myyellow = [1, 1, 0];
mycyan = [0, 1, 1];

colors = [...
    mygreen;...  % front top
    mycyan;...  % front left
    myblue;...  % front bottom
    mycyan;...  % front right
    mygreen;...  % main top
    mycyan;...  % main left
    myblue;...  % main bottom
    mycyan;...  % main right
    myred;...  % wings
    myred;...  % tailwing
    myyellow;...  % tailfin
    ];
```

Figure 3: Colouring of aircraft

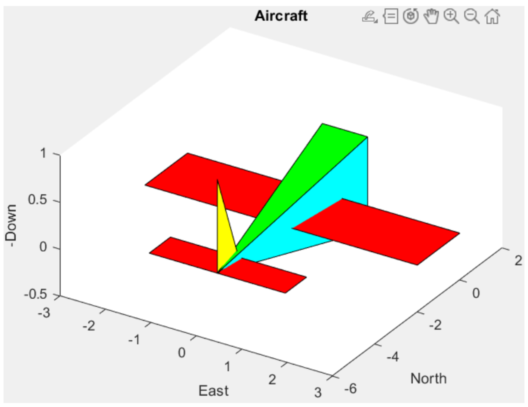


Figure 4: Result of task 1

After obtaining the aircraft vertex lists, a coloured aircraft can be drawn using the patch function shown in figure 3 nad 4. The aircraft has red wing surfaces and a yellow tail. The fuselage faces are coloured cyan, blue and green, respectively. Apart from colouring, the X, Y and Z axes are labelled as“East”, “North” and “-Down” respectively, forming the North-East-Down (NED) frame. Task 1’s result is as above.

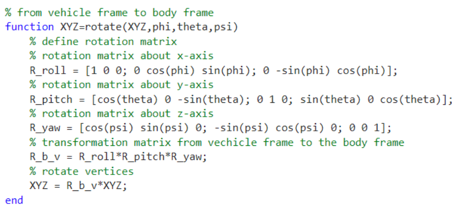
2.2 Task 2

The goal of this task is to perform coordinate transformation. The rotational matrices are input into Matlab for the aircraft to perform roll, pitch and yaw.

$$R\_roll = \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos\phi & sin\phi \\ 0 & --sin\phi & cos\phi \end{pmatrix} \qquad (1) \qquad\qquad R\_pitch = \begin{pmatrix} cos\theta & 0 & -sin\theta \\ 0 & 1 & 0 \\ sin\theta & 0 & cos\theta \end{pmatrix} \qquad (2)$$

$$R\_yaw = \begin{pmatrix} cos\psi & sin\psi & 0 \\ -sin\psi & cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad (3)$$

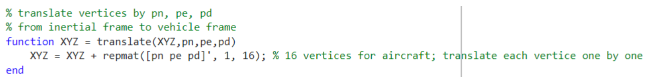
These rotation matrices are multiplied together with the aircraft coordinates to rotate the aircraft. The equations help change the aircraft attitude, which is the angular orientation of the fixed-wing aircraft concerning the NED frame. However, the aircraft rotation is in the inverse direction. Therefore, the input arguments of the function rotate() are changed to “(V’, -phi, -theta, -psi)’ ”. The related codes are below.



```
% from vehicle frame to body frame
function XYZ=rotate(XYZ,phi,theta,psi)
    % define rotation matrix
    % rotation matrix about x-axis
    R_roll = [1 0 0; 0 cos(phi) sin(phi); 0 -sin(phi) cos(phi)];
    % rotation matrix about y-axis
    R_pitch = [cos(theta) 0 -sin(theta); 0 1 0; sin(theta) 0 cos(theta)];
    % rotation matrix about z-axis
    R_yaw = [cos(psi) sin(psi) 0; -sin(psi) cos(psi) 0; 0 0 1];
    % transformation matrix from vechicle frame to the body frame
    R_b_v = R_roll*R_pitch*R_yaw;
    % rotate vertices
    XYZ = R_b_v*XYZ;
end
```

Figure 5: Rotation of fixed-wing aircraft

Apart from rotating the aircraft, the aircraft will travel to different positions. It is necessary to create the translate function. The translation is done by translating each vertex one by one. Therefore, the function repmat() helps create a 1×16 matrix with repeated elements of $\begin{pmatrix} p_n \\ p_e \\ p_d \end{pmatrix}$. An addition is done to the original XYZ matrix.



```
% translate vertices by pn, pe, pd
% from inertial frame to vehicle frame
function XYZ = translate(XYZ,pn,pe,pd)
    XYZ = XYZ + repmat([pn pe pd]', 1, 16); % 16 vertices for aircraft; translate each vertice one by one
end
```

Figure 6: Translation of fixed-wing aircraft

After the implementation of the rotation matrix and translation nation, the fixed-wing aircraft can perform different manoeuvres and move to different locations. Various results are below.

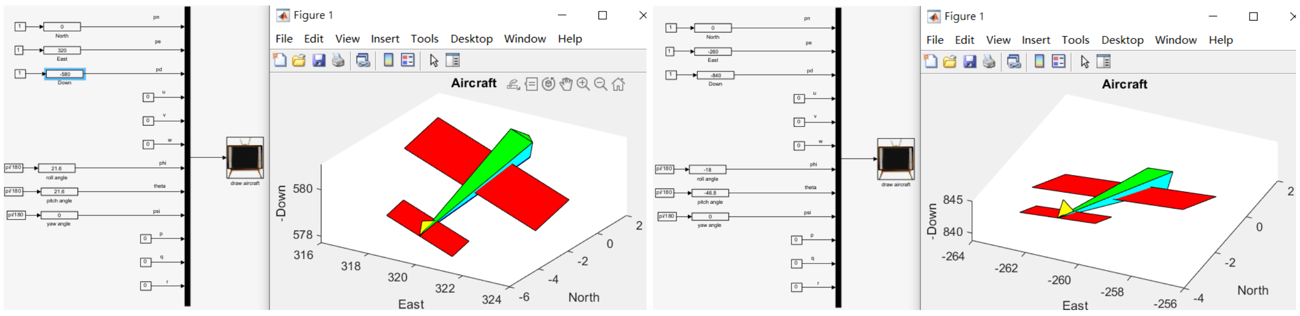


Figure 7: Results of task 2

2.3 Task 3

The aircraft cannot perform any movements with the coding in task 2. It is necessary to implement the kinematics and dynamics models, a 6-degree-of-freedom, 12-state model, to build a foundation for the aerodynamic forces and moments. Task 3 is important in the development of other tasks. Various time derivatives are input.

$$\begin{pmatrix} \dot{p}_n \\ \dot{p}_e \\ \dot{p}_d \end{pmatrix} = \begin{pmatrix} cos\theta cos\psi & sin\phi sin\theta cos\psi - cos\phi sin\psi & cos\phi sin\theta cos\psi + sin\phi sin\psi \\ cos\theta sin\psi & sin\phi si\theta sin\psi + cos\phi cos\psi & cos\phi sin\theta sin\psi - sin\phi cos\psi \\ -sin\theta & sin\phi cos\theta & cos\phi cos\theta \end{pmatrix} \begin{pmatrix} u \\ v \\ w \end{pmatrix} \tag{4}$$

$$\begin{pmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{pmatrix} = \begin{pmatrix} rv - qw \\ pw - ru \\ qu - pv \end{pmatrix} + \frac{1}{m} \begin{pmatrix} f_x \\ f_y \\ f_Z \end{pmatrix} \quad (5) \qquad \begin{pmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{pmatrix} = \begin{pmatrix} 1 & sin\phi tan\theta & cos\phi tan\theta \\ 0 & cos\phi & -sin\phi \\ 0 & \frac{sin\phi}{cos\theta} & \frac{cos\phi}{cos\theta} \end{pmatrix} \begin{pmatrix} p \\ q \\ r \end{pmatrix} \quad (6)$$

$$\begin{pmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{pmatrix} = \begin{pmatrix} \Gamma_1 pq - \Gamma_2 qr \\ \Gamma_5 pr - \Gamma_6 (p^2 - r^2) \\ \Gamma_7 pq - \Gamma_1 qr \end{pmatrix} + \begin{pmatrix} \Gamma_3 \ell + \Gamma_4 n \\ \frac{m}{J_y} \\ \Gamma_4 \ell + \Gamma_8 n \end{pmatrix} \tag{7}$$

After inputting time derivatives, changing the input parameters, forces, and torques result in the following output. The plane constantly moves to the North without performing any turns and pitch. The plane can also perform a 360° turn.

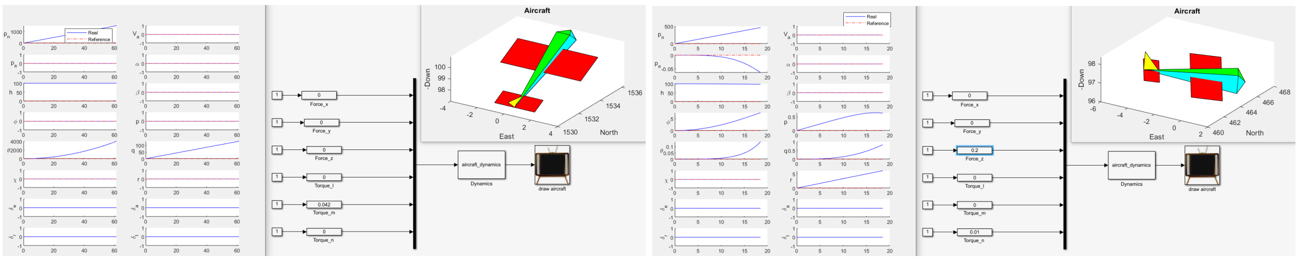
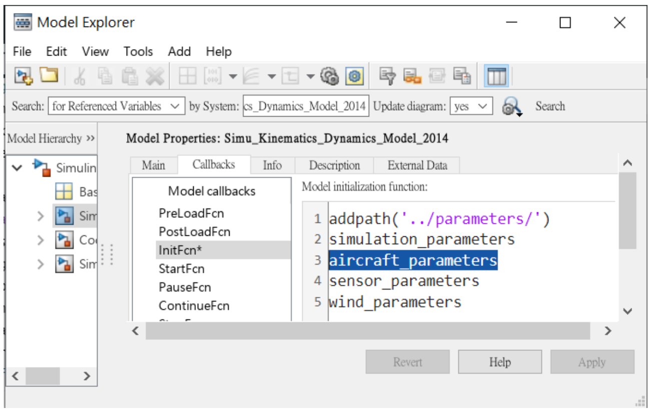


Figure 8: Results of task 3

I encountered some errors in this task. They occurred when the SimuLink model ran. It said the programme needed more input parameters, meaning the plane required more data. However, I input all the necessary parameters already. At last, I discovered that the cause was a typo in the angle term. I typed $\cos(\theta)$ as $\cos\ (\theta)$. There was a space before the bracket. Matlab could not recognise the cosine term and raised the error. After solving it, the programme still could not run due to the "aerosonde_parameter". It was not in the parameter file of task 3. After investigating, I found that the related parameters were in "aircraft_parameter". I changed the file name in the model explorer, allowing the programme to run smoothly.



Model Properties: Simu_Kinematics_Dynamics_Model_2014

Model initialization function:

```
addpath('../parameters/')
simulation_parameters
aircraft_parameters
sensor_parameters
wind_parameters
```

Figure 9: Model explorer in SimuLink

2.4 Task 4

This task aims to implement the forces and moments models so that the aircraft can be manually operated. In task 3, we can change the applied forces and moments of the aircraft, but this is not simulating the real situation. In reality, the aircraft can be affected by gusts and wind. Pilots can also change the control surfaces to control the aircraft. Therefore, several models are used to simulate these situations.

$$\begin{pmatrix} f_x \\ f_y \\ f_z \end{pmatrix} = \begin{pmatrix} -mgsin\theta \\ mgcos\theta sin\phi \\ mgcos\theta cos\phi \end{pmatrix} + \frac{1}{2}\rho V_a^2 S \begin{pmatrix} C_X(\alpha) + C_{X_q}(\alpha)\frac{c}{2V_a}q + C_{X_{\delta_e}}(\alpha)\delta_e \\ C_{Y_0} + C_{Y_\beta}\beta + C_{Y_p}\frac{b}{2V_a}p + C_{Y_r}\frac{b}{2V_a}r + C_{Y_{\delta_a}}\delta_a + C_{Y_{\delta_r}}\delta_r \\ C_Z(\alpha) + C_{Z_q}(\alpha)\frac{c}{2V_a}q + C_{Z_{\delta_e}}(\alpha)\delta_e \end{pmatrix} + \frac{1}{2}\rho S_{prop}C_{prop}\begin{pmatrix} (k_{motor}\delta_t)^2 - V_a^2 \\ 0 \\ 0 \end{pmatrix} \tag{8}$$

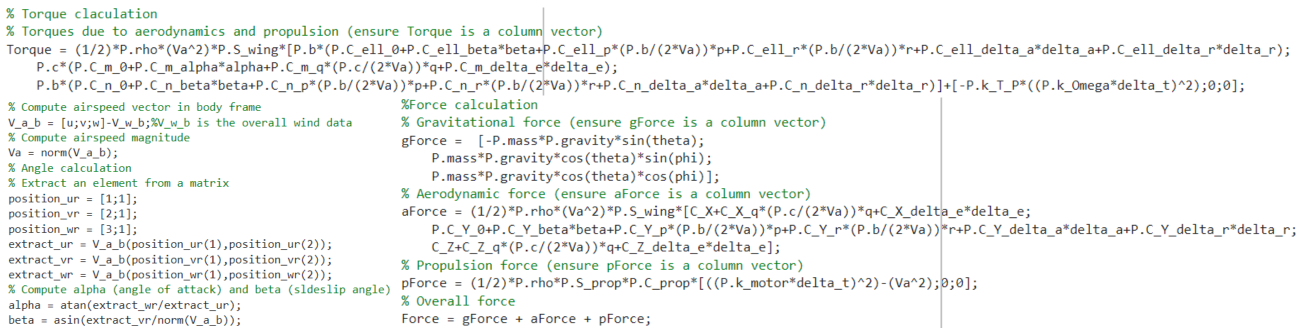
$$\begin{pmatrix} \ell \\ m \\ n \end{pmatrix} = \frac{1}{2}\rho V_a^2 S \begin{pmatrix} b(C_{\ell_0} + C_{\ell_\beta}\beta + C_{\ell_p}\frac{b}{2V_a}p + C_{\ell_r}\frac{b}{2V_a}r + C_{\ell_{\delta_a}}\delta_a + C_{\ell_{\delta_r}}\delta_r) \\ c(C_{m_0} + C_{m_\alpha}\alpha + C_{m_q}\frac{c}{2V_a}q + C_{m_{\delta_e}}\delta_e) \\ b(C_{n_0} + C_{n_\beta}\beta + C_{n_p}\frac{b}{2V_a}p + C_{n_r}\frac{b}{2V_a}r + C_{n_{\delta_a}}\delta_a + C_{n_{\delta_r}}\delta_r) \end{pmatrix} + \begin{pmatrix} -k_{T_p}(k_\Omega\delta_t)^2 \\ 0 \\ 0 \end{pmatrix} \tag{9}$$

$$V_a = \sqrt{u_r^2 + v_r^2 + w_r^2} \tag{10}$$

$$\alpha = tan^{-1}(\frac{w_r}{u_r}) \tag{11}$$

$$\beta = sin^{-1}(\frac{v_r}{V_a}) \tag{12}$$

The models are input into the Matlab script for running in SimuLink with adjustable parameters, δ_e, δ_a, δ_r, δ_t. Some inputs, meanings of parameters and results are below.



```
% Torque claculation
% Torques due to aerodynamics and propulsion (ensure Torque is a column vector)
Torque = (1/2)*P.rho*(Va^2)*P.S_wing*[P.b*(P.C_ell_0+P.C_ell_beta*beta+P.C_ell_p*(P.b/(2*Va))*p+P.C_ell_r*(P.b/(2*Va))*r+P.C_ell_delta_a*delta_a+P.C_ell_delta_r*delta_r);
    P.c*(P.C_m_0+P.C_m_alpha*alpha+P.C_m_q*(P.c/(2*Va))*q+P.C_m_delta_e*delta_e);
    P.b*(P.C_n_0+P.C_n_beta*beta+P.C_n_p*(P.b/(2*Va))*p+P.C_n_r*(P.b/(2*Va))*r+P.C_n_delta_a*delta_a+P.C_n_delta_r*delta_r)]+[-P.k_T_P*((P.k_Omega*delta_t)^2);0;0];
```

```
% Compute airspeed vector in body frame
V_a_b = [u;v;w]-V_w_b;%V_w_b is the overall wind data
% Compute airspeed magnitude
Va = norm(V_a_b);
% Angle calculation
% Extract an element from a matrix
position_ur = [1;1];
position_vr = [2;1];
position_wr = [3;1];
extract_ur = V_a_b(position_ur(1),position_ur(2));
extract_vr = V_a_b(position_vr(1),position_vr(2));
extract_wr = V_a_b(position_wr(1),position_wr(2));
% Compute alpha (angle of attack) and beta (sldeslip angle)
alpha = atan(extract_wr/extract_ur);
beta = asin(extract_vr/norm(V_a_b));
```

```
%Force calculation
% Gravitational force (ensure gForce is a column vector)
gForce =  [-P.mass*P.gravity*sin(theta);
    P.mass*P.gravity*cos(theta)*sin(phi);
    P.mass*P.gravity*cos(theta)*cos(phi)];
% Aerodynamic force (ensure aForce is a column vector)
aForce = (1/2)*P.rho*(Va^2)*P.S_wing*[C_X+C_X_q*(P.c/(2*Va))*q+C_X_delta_e*delta_e;
    P.C_Y_0+P.C_Y_beta*beta+P.C_Y_p*(P.b/(2*Va))*p+P.C_Y_r*(P.b/(2*Va))*r+P.C_Y_delta_a*delta_a+P.C_Y_delta_r*delta_r;
    C_Z+C_Z_q*(P.c/(2*Va))*q+C_Z_delta_e*delta_e];
% Propulsion force (ensure pForce is a column vector)
pForce = (1/2)*P.rho*P.S_prop*P.C_prop*[((P.k_motor*delta_t)^2)-(Va^2);0;0];
% Overall force
Force = gForce + aForce + pForce;
```

Figure 10: Forces, moments, airspeed, angle calculation

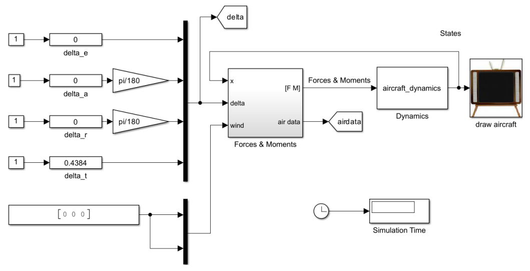
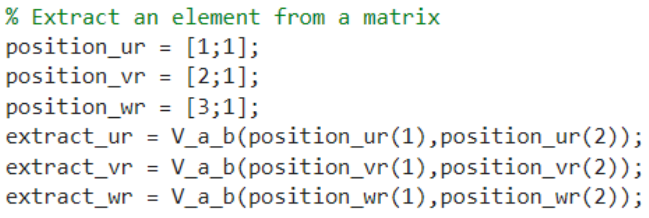


Figure 11: SimuLink model of Task 4

Parameters	Definition
delta_e (δ_e)	Elevator deflection
delta_a (δ_a)	Aileron deflection
delta_r (δ_r)	Rudder deflection
delta_t (δ_t)	Pulse-width-modulation command

Table 3: Parameters definition



```
% Extract an element from a matrix
position_ur = [1;1];
position_vr = [2;1];
position_wr = [3;1];
extract_ur = V_a_b(position_ur(1),position_ur(2));
extract_vr = V_a_b(position_vr(1),position_vr(2));
extract_wr = V_a_b(position_wr(1),position_wr(2));
```

Figure 12: Result of Task 4

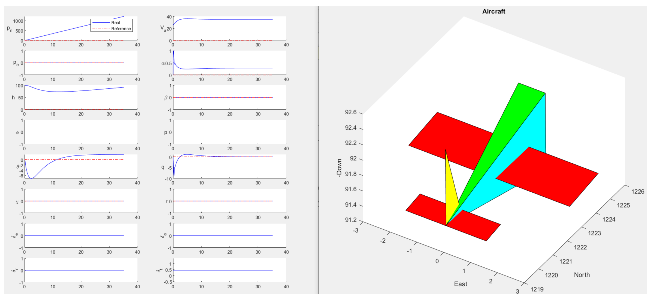


Figure 13: Result of Task 4

There were several typos during the aircraft's construction. The casing is important when calling the aircraft parameters from aircraft_parameters. "P" stands for the aircraft object name. If a "p" is typed, the aircraft object cannot be called successfully. Moreover, α and β require some elements from V_a^b, the airspeed vector. These elements can be obtained either by expanding vectors into the original form or by extracting elements. The extraction method is used to improve the accuracy and simplify the process. A small function is written to perform the extraction, as shown in figure 12.

2.5 Task 5

After installing different aircraft dynamics models, it needs a design model to help the pilots. When a pilot is flying the aircraft, he will feel tired if he needs to hold the control rod for the whole trip, especially in the climbing stage. He needs to pull the handle continuously until the aircraft reaches cruising altitude. Therefore, trimming is important for captains to relieve the workload. Trimming of this fixed-wing aircraft can be done automatically by the trim function. A SimuLink model is created to integrate the original model with the trim function.

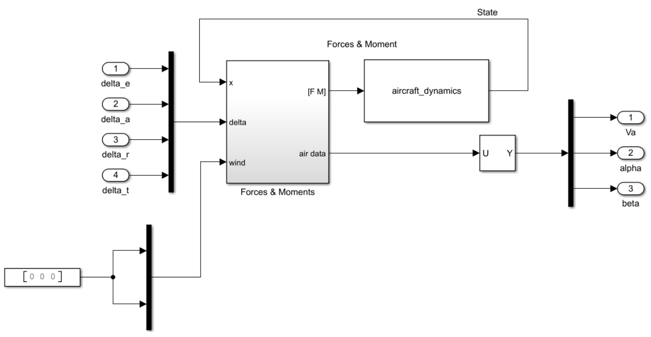
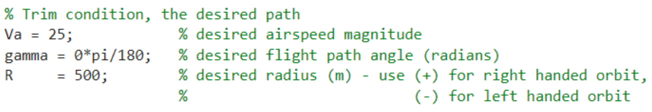


Figure 14: SimuLink model for integration of Task 5

After performing trimming, the aircraft can fly on the desired path with minor deviations without doing manual control continuously. The parameters used and the result are as follows.



```
% Trim condition, the desired path
Va = 25;            % desired airspeed magnitude
gamma = 0*pi/180;   % desired flight path angle (radians)
R     = 500;        % desired radius (m) - use (+) for right handed orbit,
                    %                          (-) for left handed orbit
```

Figure 15: Trim condition

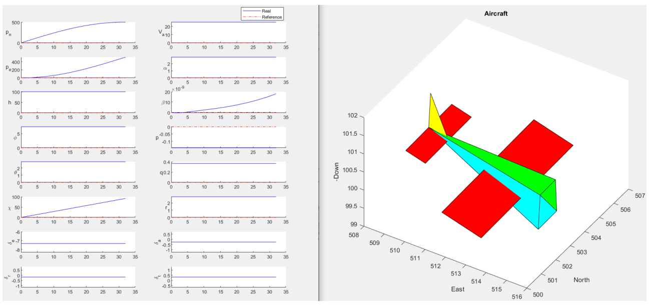
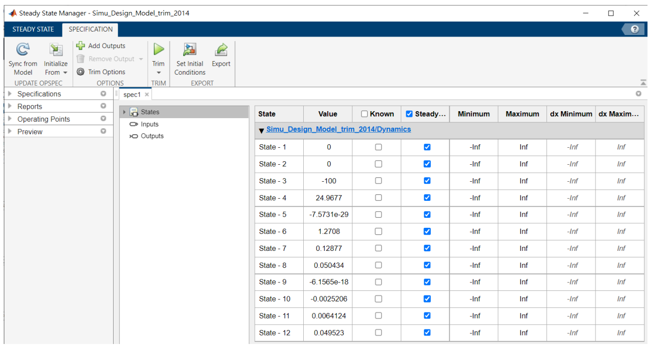


Figure 16: Result of Task 5

The trim() function is useful in performing aircraft trimming. In control systems, zero steady-state error should be obtained to improve the accuracy of the systems. A non-undamped system should be used to run the control systems smoothly. These requirements can be met by the trim() function. This function helps find the steady state. Signals can reach steady states easily with the use of trim(). In the simulation, 500 unit radius is used. The aircraft can perform turning in 500 units radius. However, as shown in SimuLink, the result is not 0. It is a very small number, $2.8584e^{-08}$, a value close to 0. This result can be interpreted as the final steady state of the control systems. Although computer functions are used to simulate the trimming, the aircraft still cannot perform a perfect trim. It is difficult to trim a system to a perfect one. Further investigations are needed to improve the trimming of the aircraft.

This improvement can be made by using the steady state manager. This function provides a platform to analyse the steady state of a system. With the input of 12 parameters, the order of parameters trimming can be chosen, allowing accurate trimming of each parameter to increase system accuracy.



Simu_Design_Model_trim_2014/Dynamics

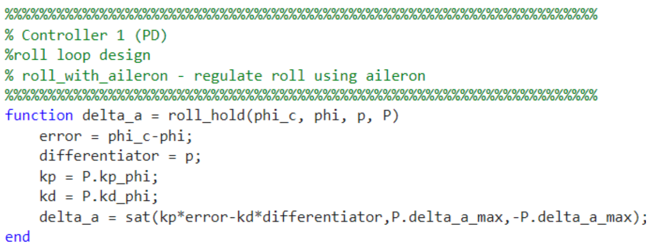
State	Value	☐ Known	✓ Steady…	Minimum	Maximum	dx Minimum	dx Maxim…
State - 1	0	☐	✓	-Inf	Inf	-Inf	Inf
State - 2	0	☐	✓	-Inf	Inf	-Inf	Inf
State - 3	-100	☐	✓	-Inf	Inf	-Inf	Inf
State - 4	24.9677	☐	✓	-Inf	Inf	-Inf	Inf
State - 5	-7.5731e-29	☐	✓	-Inf	Inf	-Inf	Inf
State - 6	1.2708	☐	✓	-Inf	Inf	-Inf	Inf
State - 7	0.12877	☐	✓	-Inf	Inf	-Inf	Inf
State - 8	0.050434	☐	✓	-Inf	Inf	-Inf	Inf
State - 9	-6.1565e-18	☐	✓	-Inf	Inf	-Inf	Inf
State - 10	-0.0025206	☐	✓	-Inf	Inf	-Inf	Inf
State - 11	0.0064124	☐	✓	-Inf	Inf	-Inf	Inf
State - 12	0.049523	☐	✓	-Inf	Inf	-Inf	Inf

Figure 17: Steady state manager

2.6 Task 6

Trimming, as completed in task 5, may not be a good approach to controlling the aircraft in the long run. The trimming requires inputs, airspeed magnitude, flight path angle and radius. However, these inputs could be better and more user-friendly. Pilots will receive air traffic commands to fly the aircraft, consisting of airspeed, flying altitude, heading and waypoints. It is difficult for pilots to calculate the related inputs in trim(). Therefore, a user-friendly system is needed, which is the autopilot design.

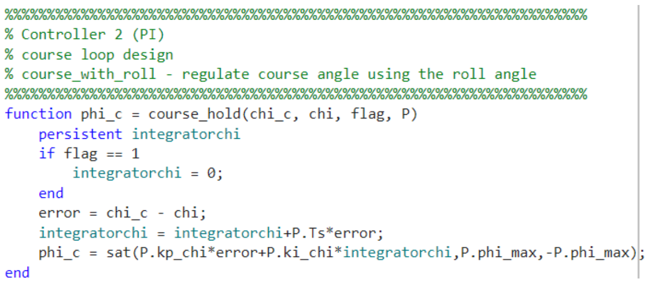
This task will create and implement the autopilot design into the fixed-wing aircraft. The autopilot requires 7 controllers, namely roll loop design, course loop design, sideslip loop design, pitch hold loop, altitude hold loop using pitch, airspeed hold loop using pitch and airspeed hold loop using throttle. The details of the controllers are below.



```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Controller 1 (PD)
%roll loop design
% roll_with_aileron - regulate roll using aileron
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function delta_a = roll_hold(phi_c, phi, p, P)
    error = phi_c-phi;
    differentiator = p;
    kp = P.kp_phi;
    kd = P.kd_phi;
    delta_a = sat(kp*error-kd*differentiator,P.delta_a_max,-P.delta_a_max);
end
```

$$\delta_a(t) = k_{p\phi}(\phi^c(t) - \phi(t)) - k_{d_\phi}p(t) \tag{13}$$

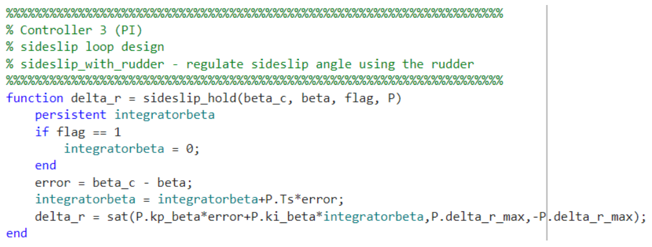
Figure 18: Controller 1



```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Controller 2 (PI)
% course loop design
% course_with_roll - regulate course angle using the roll angle
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function phi_c = course_hold(chi_c, chi, flag, P)
    persistent integratorchi
    if flag == 1
        integratorchi = 0;
    end
    error = chi_c - chi;
    integratorchi = integratorchi+P.Ts*error;
    phi_c = sat(P.kp_chi*error+P.ki_chi*integratorchi,P.phi_max,-P.phi_max);
end
```

$$\phi^c(t) = k_{p_\chi}(\chi^c(t)-\chi(t))+\int_0^t k_{i_\chi}(\chi^c(\tau)-\chi(\tau))d\tau \tag{14}$$

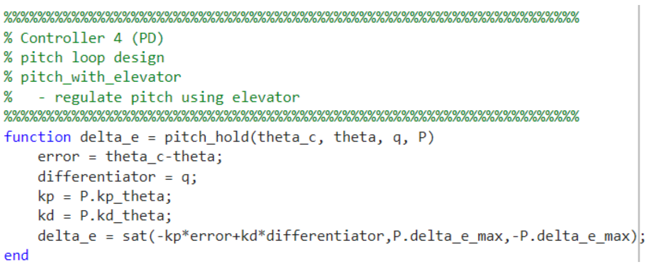
Figure 19: Controller 2



```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Controller 3 (PI)
% sideslip loop design
% sideslip_with_rudder - regulate sideslip angle using the rudder
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function delta_r = sideslip_hold(beta_c, beta, flag, P)
    persistent integratorbeta
    if flag == 1
        integratorbeta = 0;
    end
    error = beta_c - beta;
    integratorbeta = integratorbeta+P.Ts*error;
    delta_r = sat(P.kp_beta*error+P.ki_beta*integratorbeta,P.delta_r_max,-P.delta_r_max);
end
```

$$\delta_r(t) = -k_\beta\beta(t) - \int_0^t k_{i_\beta}\beta(\tau)d\tau \tag{15}$$

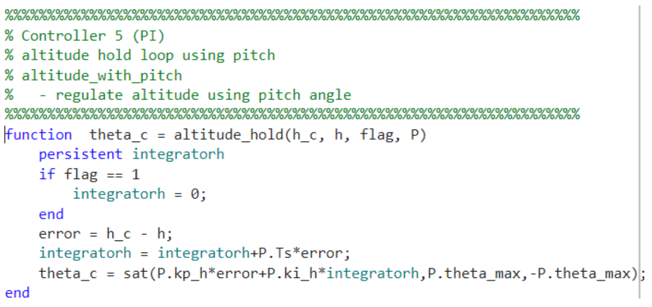
Figure 20: Controller 3



```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Controller 4 (PD)
% pitch loop design
% pitch_with_elevator
%   - regulate pitch using elevator
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function delta_e = pitch_hold(theta_c, theta, q, P)
    error = theta_c-theta;
    differentiator = q;
    kp = P.kp_theta;
    kd = P.kd_theta;
    delta_e = sat(-kp*error+kd*differentiator,P.delta_e_max,-P.delta_e_max);
end
```

$$\delta_e(t) = k_{p_\theta}(\theta^c(t) - \theta(t)) - k_{d_\theta}q(t) \tag{16}$$

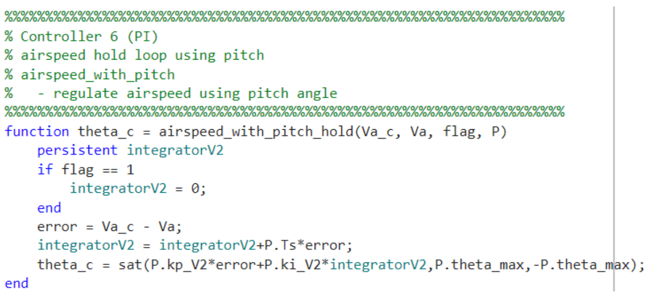
Figure 21: Controller 4



```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Controller 5 (PI)
% altitude hold loop using pitch
% altitude_with_pitch
%   - regulate altitude using pitch angle
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function  theta_c = altitude_hold(h_c, h, flag, P)
    persistent integratorh
    if flag == 1
        integratorh = 0;
    end
    error = h_c - h;
    integratorh = integratorh+P.Ts*error;
    theta_c = sat(P.kp_h*error+P.ki_h*integratorh,P.theta_max,-P.theta_max);
end
```

$$\theta^c(t) = k_{p_h}(h^c(t) - h(t)) + \int_0^t k_{i_h}(h^c(\tau) - h(\tau))d\tau \tag{17}$$

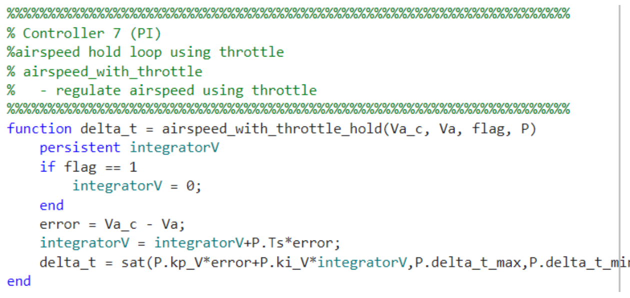
Figure 22: Controller 5



```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Controller 6 (PI)
% airspeed hold loop using pitch
% airspeed_with_pitch
%   - regulate airspeed using pitch angle
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function theta_c = airspeed_with_pitch_hold(Va_c, Va, flag, P)
    persistent integratorV2
    if flag == 1
        integratorV2 = 0;
    end
    error = Va_c - Va;
    integratorV2 = integratorV2+P.Ts*error;
    theta_c = sat(P.kp_V2*error+P.ki_V2*integratorV2,P.theta_max,-P.theta_max);
end
```

$$\theta^c(t) = k_{p_{V_2}}(V_a^c(t) - V_a(t)) + \int_0^t k_{i_{V_2}}(V_a^c(\tau) - Va(\tau))d\tau \tag{18}$$

Figure 23: Controller 6



```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Controller 7 (PI)
%airspeed hold loop using throttle
% airspeed_with_throttle
%   - regulate airspeed using throttle
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function delta_t = airspeed_with_throttle_hold(Va_c, Va, flag, P)
    persistent integratorV
    if flag == 1
        integratorV = 0;
    end
    error = Va_c - Va;
    integratorV = integratorV+P.Ts*error;
    delta_t = sat(P.kp_V*error+P.ki_V*integratorV,P.delta_t_max,P.delta_t_min);
end
```

$$\theta^c(t) = k_{p_V}(V_a^c(t) - V_a(t)) + \int_0^t k_{i_{V_2}}(V_a^c(\tau) - Va(\tau))d\tau \tag{19}$$

Figure 24: Controller 7

A SimuLink model is used to integrate all the functions into the aircraft. After several trials and errors in tuning the parameters, e_ϕ^{max}, ζ_ϕ, ω_{n_χ}, ζ_χ, e_β^{max}, ζ_β, e_θ^{max}, ζ_θ, ζ_h, ζ_{V_2}, the aircraft works well.

Parameters	Value
e_ϕ^{max}	15°
ζ_ϕ	3
ω_{n_χ}	$\frac{\omega_{n_\phi}}{W_\chi}$
ζ_χ	1.2
e_β^{max}	20°
ζ_β	10
e_θ^{max}	10°
ζ_h	1.5
ζ_{V_2}	2.83
ω_{n_v}	$\omega_{n_{V_2}}$
ζ_V	0.7

Table 4: Design parameters of Task 6

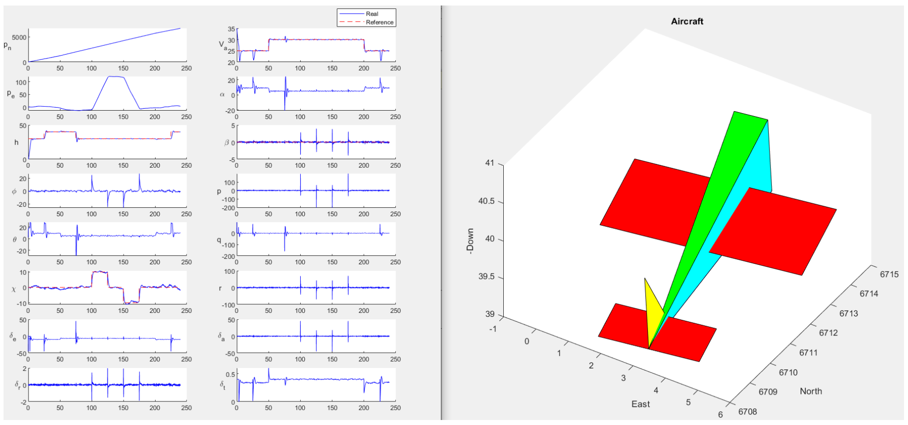


Figure 25: Result of Task 6

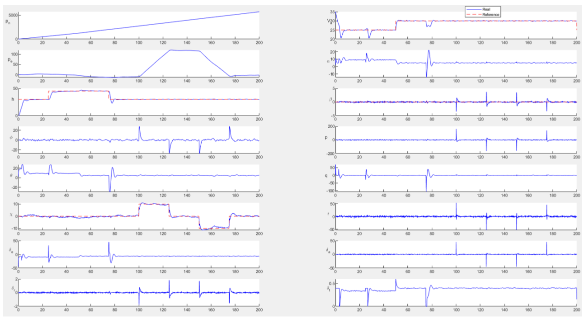


Figure 26: Expected result

The result obtained is slightly different from the expected result. However, this difference is so small that it would not greatly affect performance. Therefore, this result can be taken as one of the expected results. It can help build a safe and accurate autopilot system.

A failed trial resulted in the creation of the autopilot system. The aircraft kept spinning and descending, leading to stalling due to the reverse elevator direction. The elevator should be in the negative direction for a positive pitch-up motion. The correct code should be "$delta\_e = sat(-kp*error+kd*differentiator, P.delta\_e\_max, -P.delta\_e\_max)$", rather than "$delta\_e = sat(kp*error - kd*differentiator, P.delta\_e\_max, -P.delta\_e\_max)$".

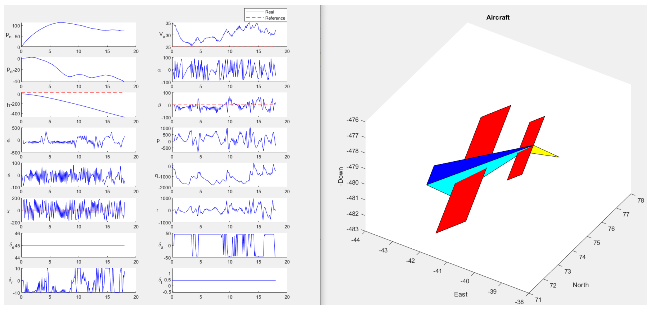


Figure 27: Failed trial in task 6

2.7 Developments and Improvements

Apart from the design of the control systems, the minimum airspeed can also be found. With an aircraft mass of 20.0 kg, the minimum airspeed is 21 m/s. The climbing speed is slightly higher than the cruising speed as some thrust is used to compensate for the aircraft's weight. Having an airspeed lower than the minimum airspeed will cause stalling. However, if the airspeed is slightly lower than the minimum speed, it can still be recovered. The airspeed is recommended to be above 30 m/s to ensure safety, with a safety factor of 1.42.

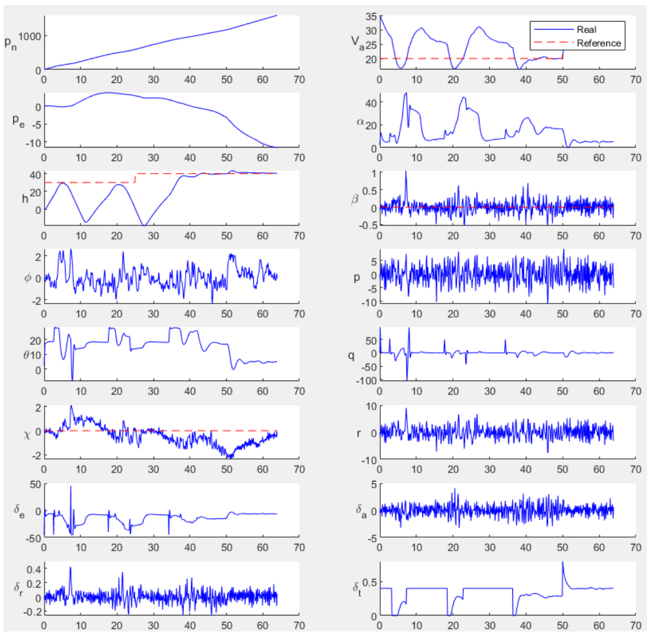


Figure 28: Stalling

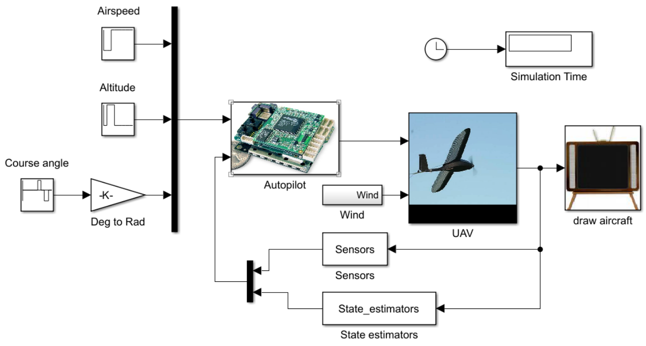


Figure 29: Sensors and state estimation

However, the accuracy of task 6 is not very high. There are still some deviations from the ideal case. Some sensor parameters were not fed back to the system for state estimation. If state estimation is installed in the system (as shown in figure 29), the aircraft can fly with a more precise flight path. The sensors and state estimation are installed in the system, and the accuracy has been improved.

3 Concluding remarks

In sum, this project creates a foundation model for aircraft systems optimisation. The simplified aircraft can be controlled manually through trimming and autopilot. Different control modes have their application and are widely used during flying procedures. Pilots need to control the aircraft manually to take off and land, while they may need to use the auto-trim function during the climbing stage. These actions represent the manual and trim modes of the aircraft model. Moreover, pilots will use autopilot during cruising. Therefore, developing the autopilot system is also crucial for this design project.

Apart from developing the manual, trim and control functions for the simplified fixed-wing aircraft model, various models are implemented into the aircraft to simulate the aircraft flying environment. However, some models may only partially simulate the environment with the ISA model. The stratosphere environment is different from the troposphere environment. Using a different simulation model for designing an aircraft flying in the stratosphere is suggested.

After designing this aircraft, I learnt a lot about designing different control systems. In AAE3004, I was taught the control system design through the root locus method. Using the root locus helps find the suitable PI, PD and PID controllers to adjust the system to a critically damped or underdamped system. However, I needed to design the controllers for a third-order system. Finding the correct controllers using the root locus method in Matlab is difficult. I used some approximations to create suitable PI and PD controllers. This project allowed me to develop PI and PD controller design methods further. I used the successive loop closure method, which helped me simplify part of the models and find the remaining controllers. I finally found suitable design parameters for various PI and PD controllers by trial and error. There are many methods to design the PID controllers, such as the root locus method and the trial and error method. However, the trial and error method was used as calculating some higher-order systems was difficult. Using the trial and error method could be much faster.

Apart from designing the PID controllers, I earned lots of new skills in using Matlab. The SimuLink in Matlab was very useful in doing simulations. By building the system model in SimuLink, I could simulate the model without testing it with a real system. I could simplify the design procedures. The system could be tuned well before the application. For example, the autopilot system designed in this project could be tested many times to ensure its accuracy. After performing multiple testing, the system could be installed on the aircraft for further testing in the real environment. Moreover, SimuLink provided many useful tools, such as the steady state manager and parameter estimator. These tools helped predict and adjust the system's performance. It assisted me in designing the control systems efficiently.

At the same time, Matlab provided many useful simulation and calculation functions. The trim() function could be used for stable output. In the design project, the trim function tuned the systems to obtain results close to 0. Repmat() function was also helpful in creating a repeated matrix. It helped save time in doing the repetitive input and reduced typos. Matlab is also similar to other programming platforms like C++ and Python. However, it is more useful when doing simulations, which is SimuLink. I can write various functions using the knowledge I learned in the C++ programming course. I wrote a function to extract the matrix elements for further calculation. Matlab was useful for doing various simulations. It could be used in multiple situations, such as the control system and aerodynamic simulations, allowing me to visualise different objects.

This lab report was written in Latex/Overleaf, which was much more useful than Word. I did not need to spend much time doing the formatting. Most of the formatting tasks were done by Latex itself. I could spend less effort when writing passages. Moreover, Latex is a good software for typing equations. It helped save time in searching the symbols. For example, if I needed to type δ in Word, I had to spend time searching the symbols in Word or online. However, if I used Latex, I only needed to know the pronunciation of the symbols. I could type "\delta" to get the correct symbols. This method is similar to what I did when using calculators. It saved me a lot of time. Latex could help me in future tasks, such as my final year project and other company projects.

This project gave me a lot of real-life experience in designing control systems. It broadened my horizons on using different design methods to adjust the control systems. I learnt a lot about using Matlab and Latex. They are some useful tools for me to carry out various projects. Although Matlab provides a good simulation platform for the control systems, there are still some limitations to the simulation in this project. The simulation was based on a simplified aircraft which may only partially demonstrates some of the aerodynamics of a real aircraft. Civil aircraft, like A380 and B787, have curved fuselage and perfectly shaped airfoils such as NACA 2412 and NACA 2415. These kinds of aerodynamics are not simulated in this design project. The use of more accurate data obtained from DATCOM is needed to implement this simulation onto civil aircraft. Moreover, this simulation does not perform any path trajectory. Further design is required for this aspect to help visualise and record the aircraft's flight path. Moreover, multiple simulations should be done before implementing related systems in Matlab. The system should be highly accurate to avoid deviating from the desired flight path. As aviation safety is the most important, the simulated system should be tested on a real aircraft many times to ensure its integrity.

4 Appendix

All materials have been uploaded to Github and OneDrive.
Github: https://github.com/Hobart-tam/Fixed-wing-aircraft-design-project
OneDrive: https://connectpolyu-my.sharepoint.com/:f:/g/personal/20045757d_connect_polyu_hk/EkSuFYSlrNhKvUATr97ndAcBuw981ENqAz6WFpH3M2_18Q?e=O01kJb