

| Course Number | AER 404 |
| --- | --- |
| Course Title | Intro to Aerospace Engineering |
| Semester/Year | W2025 |
| Instructor | Dr. Faieghi |
| Section Number | 02 |
| Group Number | 02 |

**Project 2: Flight Simulator Report**

| **Name** | **Email** | **Signature** |
| --- | --- | --- |
| Max Niclas Sommer | max.sommer@torontomu.ca | MS |
| Alec Alexander Tabachnik | alec.tabachnik@torontomu.ca | AT |
| Newt Klocokova | nklocokova@torontomu.ca | NK |
| Mithun Sivakumar | mithun.sivakumar@torontomu.ca | MS |
| Arshil Mansuri | arshilbhai.mansuri@torontomu.ca | AM |

5

# Table of Contents

[Table of Contents 1](#_gmx7gqakbnk2)

[List of Figures 1](#_yab0jr5gzkxu)

[Introduction 2](#_qxmxslsukphm)

[Problem Statement, Objectives and WSB 2](#_p9sy7w66ud11)

[Design Description](#_ouc1mwiqy9eq) 5

[Input Handling](#_xomz8w14s7ax)6

[Aerodynamic Forces and Moment](#_89f4kwyseklj) 7

[Engine Forces and Moments](#_tulxbveyfrhw)  9

[6-DoF Equations of Motions](#_uikzqn8ftmwz) 10

[Navigation and Geodetic Conversion](#_ehltjf9mgfso)  11

[Design Evaluation](#_vx3dhxqr9pe8) 13

[Conclusion](#_a4do4lfj0gzp) 14

[References](#_5loscou5kjp7) 14

[Appendix 16](#_v0i1m1pctut)

# List of Figures

Figure 1: Objective tree 4

Figure 2: Hierarchical Block-Diagram 5

Figure 3: Simulink/Matlab Implementation (Physics System) 6

Figure 4: Simulink/Matlab Implementation (Aircraft Subsystem) 7

Figure 5: Output Test Graphs 13

Figure 6: Changed Values in Saturation Block 14

# 

# Introduction

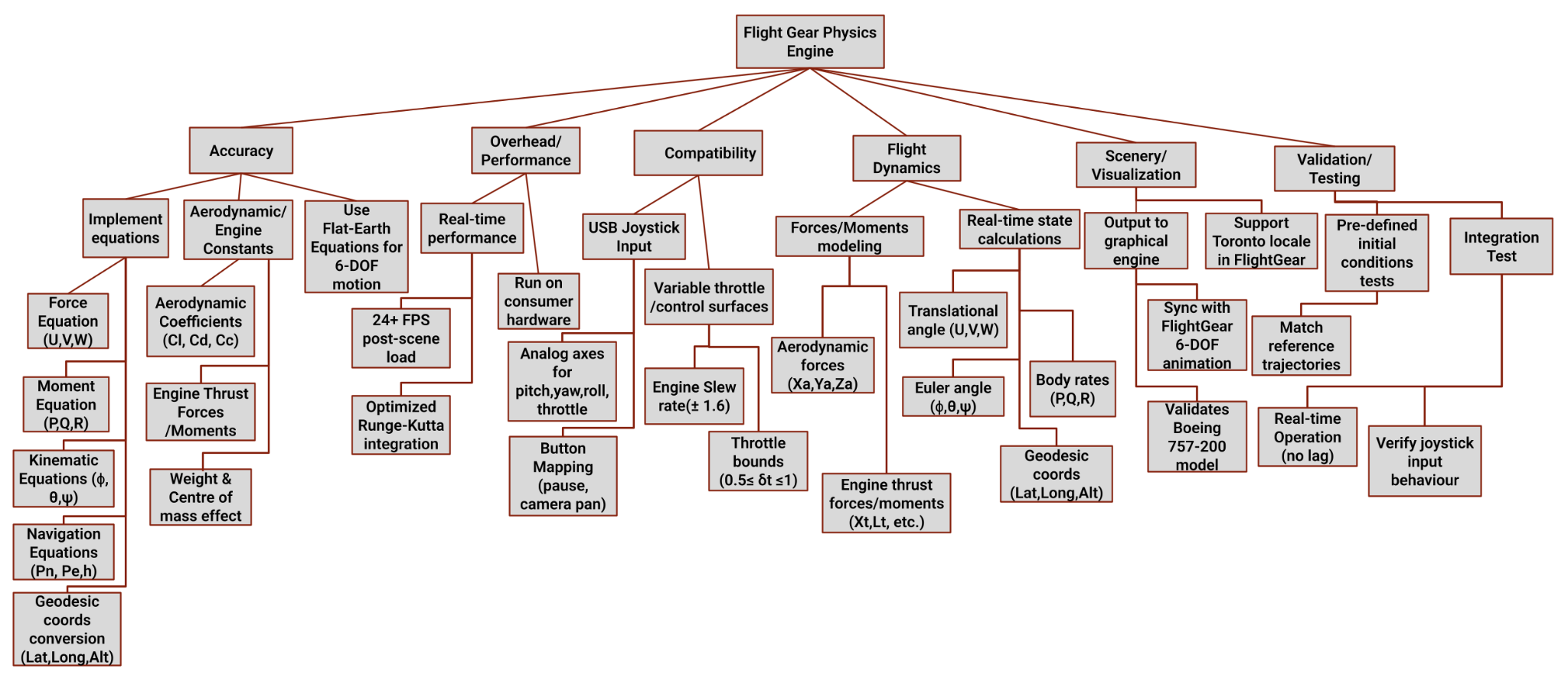
Within this report developments towards a flight matlab flight simulator system are documented. The design problem calls for creation of a flight simulator physics engine, computing various parameters including speed, direction and position of an aircraft [1]. Furthermore a flight stick is utilized as input into a matlab physics engine which then outputs into a graphical flight simulator framework which provides the user interface. The report covers detailed analysis of the problem statement as well as a final design description and evaluation.

# Problem Statement, Objectives & WSB

For our aircraft manufacturing company a physics engine for a narrow-body twinjet airliner is to be constructed [1]. Matlab and Simulink will be used for direct FlightGear joystick input, calculations and output into the graphical UI system. Various parameters regarding aircraft specification and performance are provided for the specified aircraft within the supplementary materials document [2].

**Objectives List:**

* Accuracy
  + Implement all of the equations to the dynamics of the aircraft with accuracy.
    - Force equations (translational velocities:
    - Moment equations (body rates:
    - Kinematic equations (Euler angles:
    - Navigation equations (NED coordinates:
    - Geodetic coordinate conversion (latitude, longitude, altitude)
  + Use the Flat-Earth equations to represent motion in six degrees of freedom.
  + Accurately incorporate aerodynamic coefficients and constants.
* Performance/Overhead
  + Verify that it works with consumer-grade hardware.
  + Continue to run simulations in real time.
    - Reach a post-scene load of 24 frames per second or more.
    - Optimize numerical integration using Runge-Kutta (4th order).
* Compatibility
  + Take standard USB joystick inputs.
    - Map the pitch, roll, yaw, and throttle analogue axes.
    - Allow button mapping for auxiliary controls, such as camera pan and pause.
  + Include control surface deflections and various throttle settings.
    - Evaluate engine reaction time constraints (slew rate: ±1.6) in simulation.
    - Limit the throttle angle to 0.5 ≤ ≤ 1.
* Flight Dynamics
  + Real-time aircraft state calculations
    - Euler angle ( and body rates (.
    - Translational velocity ().
    - Geodetic coordinates (latitude, longitude, altitude)
  + Model engine and aerodynamic forces and moments
    - Aerodynamic forces () and Moments ().
    - Engine Thrust Forces () and Moments ().
* Scenery and Visualization
  + FlightGear's pregenerated Toronto locale is supported.
  + Smoothly output aircraft states to the graphical engine
    - Make sure to align with the 6-DoF animation of FlightGear.
    - Use the Boeing 757-200 model to verify scene loading.
* Validation and Testing
  + Software should be validated against specified initial conditions
    - Compare the simulation paths to the reference charts.
  + Test the FlightGear and joystick combination
    - Verify lag-free real-time operation.
    - Check that the aircraft is operating correctly with the control inputs.



**Figure 1**: *Objective Tree*

The figure below provides a block diagram describing the process of input, calculation, output and validation of results. Starting with inputs and initial values, then utilized for calculation of various parameters, many of which are utilized to perform further calculation. Once calculation is complete results are outputted and utilizing scopes directly into the graphical interface which can be validated and debugged. Subsequently this process repeats with new inputs and initial values.

## 

## 

## **Figure 2**: Hierarchical Block-Diagram

# Design Description

The framework of the system describes how the equations, Simulink blocks, and data flow for each subsystem in the physics engine that has been implemented. There are primarily five subsystems within the system, namely:

* Handling Input
* Forces and Moments in Aerodynamics
* Engine Moments & Forces
* 6-DOF Motion Equations
* Geodetic Conversion and Navigation

Below is a description of each subsystem along with relevant equations, Simulink settings, and explanations of data flow.

## I - Input Handling

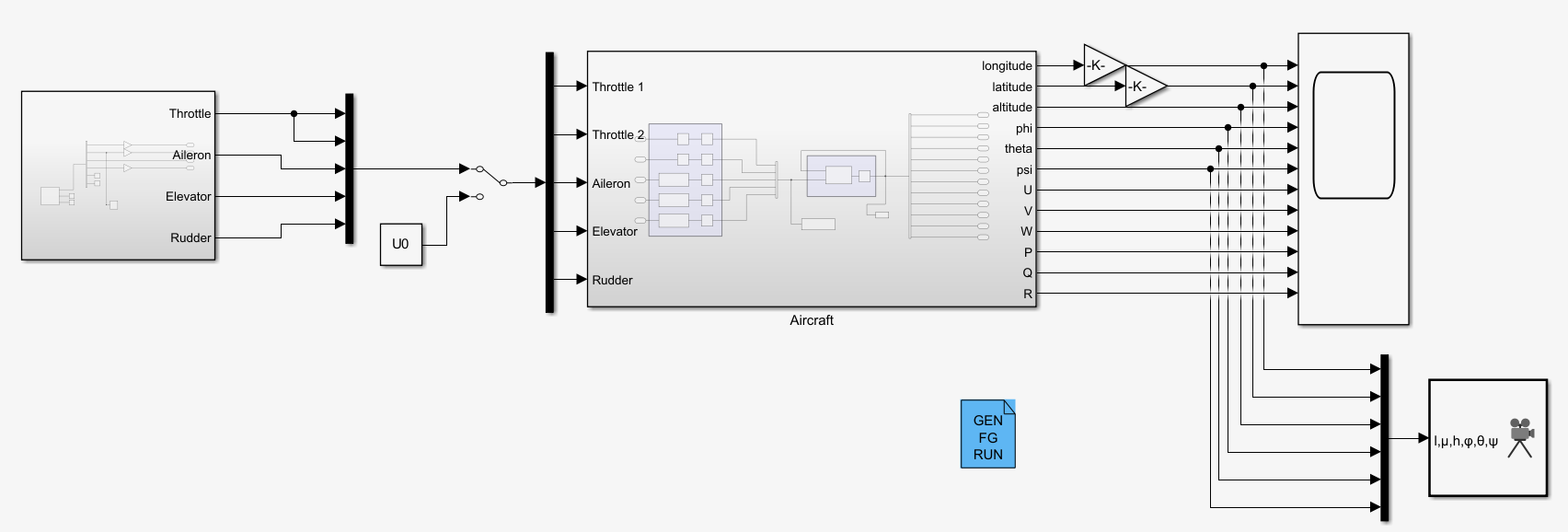
**Purpose:**

Transforms joystick inputs (rudder, elevator, aileron, and throttle) into signals that the physics engine may use.

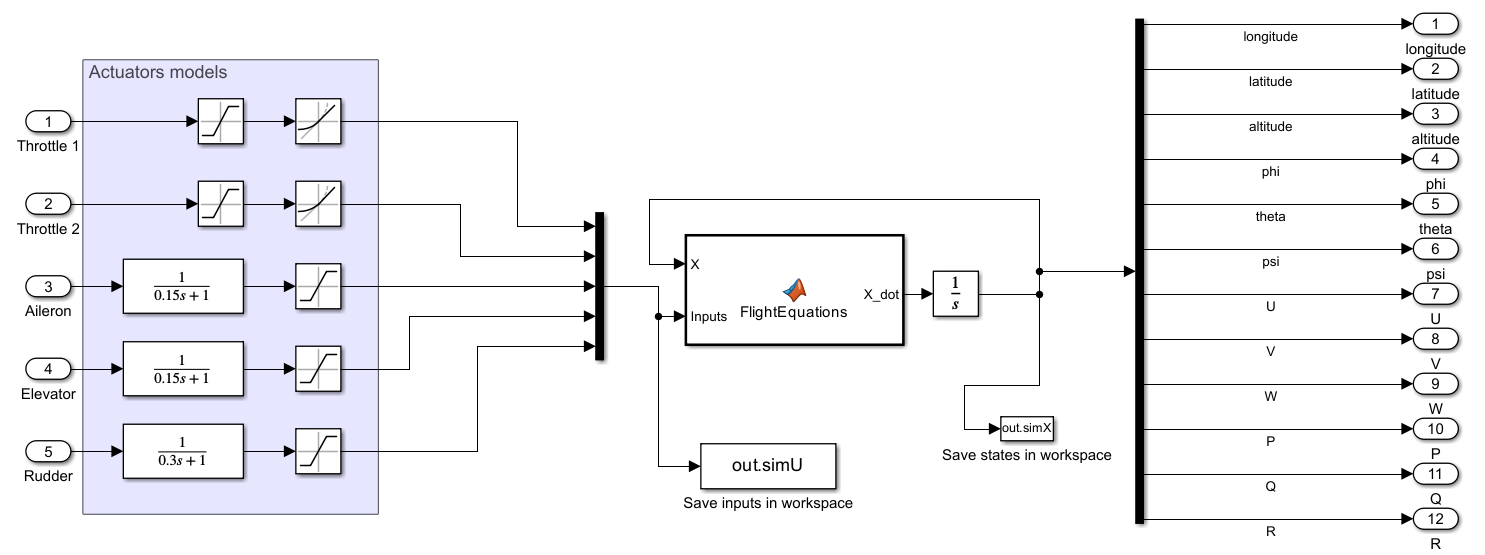
**Simulink/Matlab Implementation:**

The Input Handling subsystem uses a Joystick block to capture analog inputs for throttle, aileron, elevator, and rudder. These signals are separated using a Demux block and passed through Gain blocks to scale the inputs [2]. The control signals are then routed to a Saturation block and a Rate Limiter to ensure realistic physical constraints on the throttle and control surfaces. The processed signals are then fed into the aerodynamic and engine subsystems for further computation [2].

The majority of the calculations describing the state of the aircraft are done inside of a MATLAB Function block, which contains MATLAB code of all of the equations. The inputs to this Function block are the states as well as the inputs from the joystick, as can be seen in Figure 4 below. The function outputs a vector X\_dot, which contains the derivatives of the states. A Simulink Integrator block then integrates the vector. From there, the input vector is fed back into the function for the next state, and following a Demux block, the longitude, latitude, altitude, and Euler angles are passed into the FlightGear Preconfigured 6DOF Block, which sends the values to Flight Gear to simulate the aircraft.



**Figure 3**: *Simulink/Matlab Implementation (Physics System)*

**

**Figure 4**: *Simulink/Matlab Implementation (Aircraft subsystem)*

Data Flow

Joystick Inputs Saturation block(rate limiter) Aerodynamic and Engine subsystems.

* **Key equations [2]:**
  + Throttle saturation: 0.5
  + Rate limited throttle response: ± 1.6 units/sec
  + Equation to throttle thrust:
  + Aerodynamic computations directly receive control surface deflections (rudder, elevator, and aileron).

## 

## II - Aerodynamic Forces and Moments

**Purpose:**

Utilizes the aircraft's foundational conditions and control inputs to calculate the aerodynamic forces such as lift, drag, and moment.

**Foundational Values:**

The foundational aerodynamic variables are computed using the axis velocities and flight angles. The true airspeed VT is calculated from the magnitude of the velocity (U,V,W). From this, the angle of attack α and sideslip angle β are calculated using inverse trigonometric functions. The dynamic pressure q is found using the formula 1/2ρVT2. Additionally, downwash ϵ and tail angle of attack αt are included to adjust for aerodynamic center and elevator effects [2].

| %% Foundational Values  Vt = sqrt(U^2 + V^2 + W^2); %true air speed  q = 0.5 \* rho \* Vt^2; %dynamic pressure  alpha = atan(W/U); %angle of attack  beta = asin(V/Vt); %sideslip angle  epsilon = 0.25 \* (alpha - alpha0); %downwash angle of the wings  alphat = alpha - epsilon + deltaE + 1.3\* Q \* (Lt / Vt); %angle of attack of the tail |
| --- |

**Aerodynamic Coefficients:**

The aerodynamic coefficients are calculated using standard linear and quadratic approximations, commonly applied in simplified aircraft models. For example, the wing-body lift coefficient CLwb is modeled as a linear function of angle of attack α, while the tail lift coefficient CLt is calculated similarly using the tail angle of attack αt. The total lift coefficient CL is a combination of both, adjusted based on tail size and position. The drag coefficient CD is a function of angle of attack, which means it increases more as the angle gets steeper. This rise in drag takes place because the aircraft experiences more air resistance (parasitic drag) and also more drag from working harder to stay lifted in the air (induced drag). Rolling, pitching, and yawing moment coefficients are also calculated using simple approximations based on control surface deflections, body rotation rates, and aircraft geometry [2].

| %% Aerodynamic Coefficients  CL = CLwb + CLt; %lift coefficient  CLwb = 5.5 \* (alpha - alpha0); %lift coefficient wing  CLt = 3.1 \* (St / S) \* alphat; %lift coefficient tail  CD = 0.13 + 0.07 \* ((5.5\* alpha) - 0.654)^2; %drag coefficient  CC = (-1.6 \* beta) + (0.24 \* deltaR); %crosswind coefficient  Cl = (-1.4 \* beta) - (11 \* (l / Vt) \* P) + (5 \* (l / Vt) \* R) - (0.6 \* deltaA) + (0.22 \* deltaR); %rolling moment coefficient  Cm = -0.59 - ((3.1 \* ((St \* Lt) / (S \* L))) \* (alpha - epsilon)) - ((4.03 \* ((St \* Lt^2)/(S \* L^2))) \* (L / Vt) \* Q) - ((3.1 \* ((St \* Lt) / (S \* L)) \* deltaE)); %Pitching moment coefficient  Cn = ((1 - 3.8179 \* alpha) \* beta) + (1.7 \* (L / Vt) \*P) - (11.5 \* (L / Vt) \* R) - (0.63 \* deltaR); %yawing moment coefficient |
| --- |

**Forces in Wind axes:**

The aerodynamic forces — lift (L), drag (D), and crosswind (C) — are computed by multiplying their respective coefficients with dynamic pressure and wing reference area. These forces are then transformed from the wind axis to the body axis using trigonometric projections using α and β. The final body-axis force components XA, YA, and ZA​ are used in the translational motion equations [2].

| %% Aerodynamic Forces  D = q \* S \* CD; %drag  C = q \* S \* CC; %crosswind  L = q \* S \* CL; %lift  XA = (-D \* cos(alpha) \* cos(beta)) + (C \* cos(alpha) \* cos(beta)) + (L \* sin(alpha)); %real drag  YA = (-D \* sin(beta)) - (C \* cos(beta)); %real crosswind  ZA = (-D \* sin(alpha) \* cos(beta)) + (C \* sin(alpha) \* sin(beta)) - L \* cos(alpha); %real lift |
| --- |

**Moment calculations:**

Aerodynamic moments *l*A, *m*A, *n*A, are calculated by multiplying their corresponding moment coefficients with the dynamic pressure, wing area, and chord. This process converts the unit-less coefficients into actual rolling, pitching and yawing moments. The cross product of the aerodynamic force vector and the distance between the aerodynamic center and the center of mass is also added to account for the extra torque caused by the force being applied away from the aircraft center. These moments are then used in the rotational motion equations [2].

| %% Aerodynamic Moments  la = q\*c\*S\*Cl;  ma = q\*c\*S\*Cm;  na = q\*c\*S\*Cn;  [lA, mA, nA] = [la, ma, na] + cross([XA, YA, ZA],[0.11\*c, 0, 0.1\*c]); % Actual Moments |
| --- |

Data Flow

(U,V,W)

**Key equations [2]:**

* Basic Variables:
* Aerodynamic Coefficients:

## III - Engine Forces and Moments

**Purpose:**

Calculates moments and thrust forces based on engine inputs.

**Thrust Calculations:**

Thrust force is calculated from the throttle inputs δT1​ (*XT1*) and δT2​ (*XT2*), which represent the throttle percentage for engine 1 and engine 2, respectively. Each input is scaled by the aircraft’s weight mg to determine the thrust from each engine. The total thrust XT is the sum of both [2].

| %% Engine Forces  XT1 = deltaT1\*m\*g;  XT2 = deltaT2\*m\*g;  XT = XT1 + XT2; |
| --- |

**Engine moments:**

Engine moments are generated by the cross product between each thrust force vector and the position vector of the engine subject to the aircraft's center of mass. The coordinates of the thrust points are subtracted from the center of mass to compute this moment. Therefore, resulting in rolling, pitching and yawing moments *l*T, *m*T, *n*T [2].

| %% Engine Moments  engine1Moments = cross((cm-apt1), [XT1;0;0]); % [lT1, mT1, nT1];  engine2Moments = cross((cm-apt1), [XT1;0;0]); % [lT2, mT2, nT2]; |
| --- |

Data Flow:

Throttle inputs Thrust Force cross-product with engine position Engine Moment

**Key equations [2]:**

|  |  | ╳ |  |
| --- | --- | --- | --- |

## IV - 6-DOF Equations of motion

**Purpose:**

Solutions for the translational and rotational dynamics of airplanes.

**Translational Dynamics (Force equations):**

The translational accelerations UDot,VDot,WDot are calculated using Newton’s second law in the aircraft’s body. These accelerations account for gravitational forces acting on the body and its axes based on the aircraft’s pitch and roll angles, as well as the total aerodynamic and thrust forces acting on the aircraft. This describes how the aircraft moves forward, sideways, and vertically in response to those forces.

These values are integrated to update the velocity in each axis direction [2].

| %% Force Equations  UDot = R\*V - Q\*W - gsin(theta)+(XT+XA)/m;  VDot = -R\*U + P\*W + g\*sin(phi)\*cos(theta)+(YT+YA)/m;  WDot = Q\*U - P\*V + g\*cos(phi)\*sin(theta)+(ZT+ZA)/m; |
| --- |

**Rotational Dynamics (Moment equations):**

Angular accelerations PDot, QDot and RDot are calculated using Euler’s equations for rotational motion. These equations consider the full inertia matrix, cross coupling terms like J*xz*, and the total aerodynamic and engine moments acting about each axis. The results capture the aircraft's roll, pitch, and yaw. These angular accelerations are then integrated over time to update the rotational velocity [2].

| %% Moment Equations  J\_x = J(1,1);  J\_y = J(2,2);  J\_z = J(3,3);  J\_xz = J(1,3);  Gamma = J\_x\*J\_z - J\_xz^2;  PDot = ( J\_xz\*(J\_x - J\_y + J\_z)\*P\*Q - (J\_z\*(J\_x - J\_y) + J\_xz^2)\*Q\*R + J\_z\*(lA + lT) + J\_xz\*(nA + nT) )/Gamma;  QDot = ( (J\_z - J\_x)\*P\*R - J\_xz\*(P^2 - R^2) + mA + mT )/J\_y;  RDot = ( ((J\_x - J\_y)\*J\_x + J\_xz^2)\*P\*Q - J\_xz\*(J\_x - J\_y + J\_z)\*Q\*R + J\_xz\*(lA + lT) + J\_x\*(nA + nT) )/ Gamma; |
| --- |

**Kinematic equations (Euler angles):**

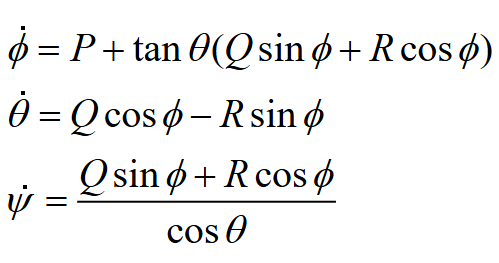
The Euler angles ϕ(roll), θ (pitch), and ψ (yaw) describe the aircraft’s orientation and are calculated from the angular velocity components P, Q, and R. Using trigonometric relations that relate body angular rates to changes in orientation and their time derivatives are calculated to update the aircraft’s altitude over time [2].

| %% Kinematic Equations  phiDot = P + tan(theta)\*(Q\*sin(phi) + Rcos(phi));  thetaDot = Qcos(phi) - Rsin(phi);  psiDot = (Q\*sin(phi) + R\*cos(phi))/cos(theta); |
| --- |

**Data Flow:**

Forces and moments Integrators ( For U,V,W,P,Q,R) Euler angles

**Key equations [2]:**



## V - Navigation and Geodetic conversion

**Purpose:**

Converts NED coordinates to GPS (Latitude, Longitude) coordinates.

**Navigation equations:**

The aircraft’s motion in the Earth’s reference frame is calculated using body-axis velocities U, V, W amd orientation angles ϕ, θ, and ψ. These values are used to compute the North (pNDot), East (pEDot), and vertical (hDot) rates of change, converting from the body frame to the local NED (North-East-Down) frame. This conversion allows the simulator to track the aircraft’s movement over the Earth’s surface [2].

| %% Navigation Equations  pNDot = U\*cos(theta)\*cos(psi) + V(-cos(phi)\*sin(psi) + sin(phi)\*sin(theta)\*cos(psi)) + W(sin(phi)\*sin(psi) + cos(phi)\*sin(theta)\*cos(psi));  pEDot = U\*cos(theta)\*sin(psi) + V(cos(phi)\*cos(psi) + sin(phi)\*sin(theta)\*sin(psi)) + W(-sin(phi)\*cos(psi) + cos(phi)\*sin(theta)\*sin(psi));  hDot = U\*sin(theta) - V\*sin(phi)\*cos(theta) - W\*cos(phi)\*cos(theta); |
| --- |

**Geodetic conversion (WGS 84 Standard):**

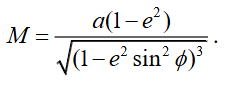
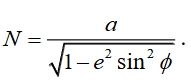
The computed NED velocities are then used to update geodetic coordinates(latitude and longitude) using the standard Earth model parameters (matching the WGS-84 reference model). The values for the semi-major axis and eccentricity are used to calculate the radius of curvature in the meridian (M) and the normal radius (N). These are then used to compute the rate of change of latitude (μDot) and longitude (λDot). Integrating these rates over time provides the aircraft’s GPS position, which is useful for flight tracking [2].

| %% Geodetic Coordinate Equations  M = a\*(1-e^2)\*(1-(e^2)\*(sin(phi))^2)^-3/2;  N = a\*(1-(e^2)\*(sin(phi))^2)^-1/2;  muDot = pNDot/(M+h);  mu = integral(muDot);  lambdaDot = pEDot/((N+h)\*cos(mu)); |
| --- |

Data Flow

(U,V,W) NED coordinates Geodetic conversions FlightGear Output

**Key equations [2]:**

# 

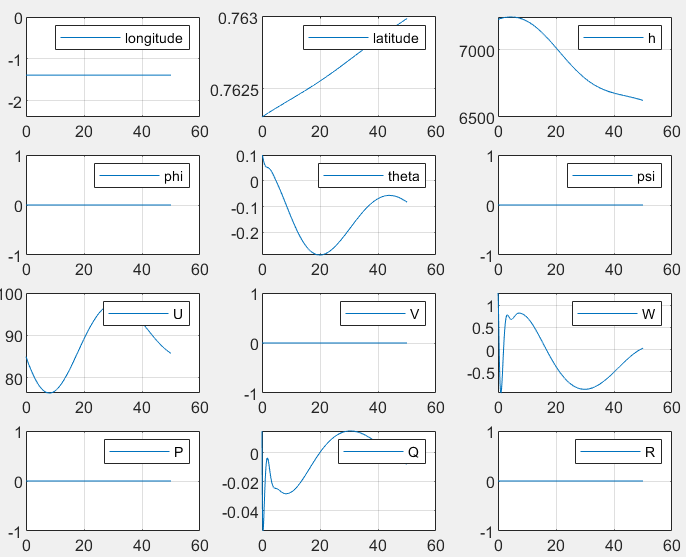
# 

# 

# 

# 

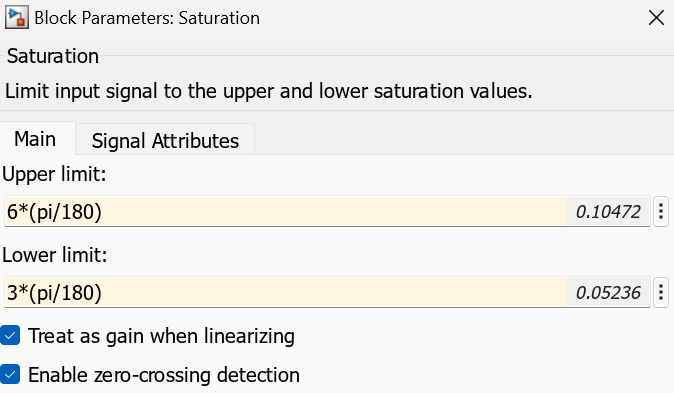
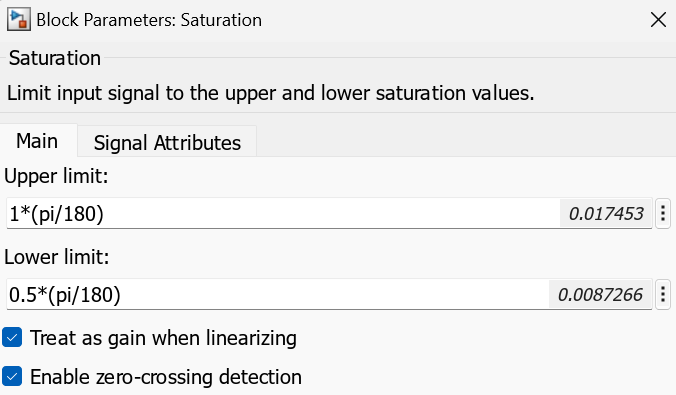
# Design Evaluations



**Figure 5**: *Output Test Graphs*

Once the model was complete, testing was done to ensure that it was functional. To do this, values specified in the supplementary material were set as the inputs from the joystick, which remained constant throughout a 50 second time interval, enabling testing of the simulator without use of the joystick. Once finished, the program output graphs of the values of the latitude, longitude, height, translational velocity components, body rate components, and Euler angles changing with respect to time. The output graphs of the simulator matched those provided in the supplementary material, meaning that the flight simulator worked and the equations were valid [2].

Testing with the joystick unveiled some problems. When using the joystick, it was observed that the plane would point nose-down and slowly lose altitude. Increasing the thrust using the joystick did not solve this problem, and it was impossible to gain altitude. To fix this, the values relating to thrust were altered (Note; the code submitted does not include the changed values). Most importantly, the limits for the thrust were changed. As shown in the figure below, the upper and lower limits for the Saturation functions were changed from *1\*(pi/180)* and *0.5\*(pi/180)* to *6\*(pi/180)* and *3\*(pi/180)* respectively. This allowed the thrust to be increased such that enough force was provided to lift the aircraft upwards.



**Figure 6**: *Changed Values in Saturation Block*

Another noteworthy comment about the behavior of the flight simulator is that it is relatively difficult to control. It is incredibly sensitive to inputs from the joystick, meaning that a smooth flight is virtually impossible. This is to be expected, however, since the flight simulator and equations used to model it are rather rudimentary. Onboard a real-world aircraft, there would be many other subsystems in play which would constantly monitor the state of the aircraft and limit or counteract jagged movements. Overall, the flight simulator functions as needed, and the input from the joystick translates into movement of the aircraft in the simulation.

# 

# Conclusion

Through this design assignment, a working flight simulator engine was successfully constructed and implemented using MATLAB and Simulink, with full integration of flight controls and graphical output; completing the clients problem statement request [1]. Key takeaways from this design include a deeper understanding of how aerodynamic forces, engine thrust, and control surface deflections influence an aircraft’s motion through the 6-DOF dynamics. Additionally, the project reinforced the importance of transforming coordinates from body axes coordinates to geodetic coordinates for realistic navigation and flight visualization. By simulating the real-time control inputs through the joystick, in this project we demonstrated how engineering topics can be translated into practical tools for flight analysis, training and testing.

Overall, this assignment provided us with valuable experience in applying theoretical aerospace engineering concepts to a complex systems-level design. It strengthened our understanding of control system integration, flight dynamics and simulation models.

# 

# References

[1] Reza Faieghi N.D. AER 404: Project 2 Flight Simulator Physics Engine Material. Toronto Metropolitan University.

[Project 2 Problem Statement - AER404 - Intro to Aerospace Engineering - W2025](https://courses.torontomu.ca/d2l/le/content/979815/viewContent/6305223/View)

[2] Reza Faieghi N.D. AER 404: Project 2 Supplementary Material. Toronto Metropolitan University.

[Project 2 Supplementary Material - AER404 - Intro to Aerospace Engineering - W2025](https://courses.torontomu.ca/d2l/le/content/979815/viewContent/6305226/View)

# 

# Appendix