

Engineering Flight Simulator Using MATLAB, Python and FLIGHTGEAR

Istas F. Nusyirwan

Department of Aeronautical Engineering, Faculty of Mechanical Engineering
Universiti Teknologi Malaysia
istaz@utm.my

Abstract. The paper outlines the development of a low-cost six degree-of-freedom flight simulator using FlightGear, Python and MATLAB for teaching and learning. The development of the aircraft mathematical model is discussed. It involves the formulation of twelve nonlinear state equations and the utilization of lookup tables in MATLAB. The interface with human pilot is accomplished by interfacing MATLAB with a Python code that talks to a human input device. This follows by the development of the stability augmentation system of the aircraft model. The fourth-order Runge-Kutta method calculates the states with variable time step to achieve a real time simulation. The states are transmitted to several FlightGear image generator servers via UDP. The instrument panels are developed by using Open Glass Cockpit and Python with OpenGL. With these tools, a complete flight simulator is developed with a working navigation system.

1. INTRODUCTION

The simulation of aircraft in flight is undoubtedly one of the most exciting and yet complicated fields in the engineering world today. As commercial off-the-shelf computers have become cheaper but more powerful, the opportunity to simulate the complex behaviour of aircraft motion in real time has become possible. Many flight simulation software have emerged such Microsoft Flight Simulator, X-Plane and FlightGear. The software provides close to real the world scenery. The implementation of 6DOF aircraft equations of motion, the quaternion transformation method, experimentally derived aerodynamic and stability coefficient implementation could simulate an aircraft with higher level of realism. Coupled with the actual propulsion model, atmospheric model and other system models, it gives many unique simulation features.

Developing an aircraft simulator is an evolving process. It requires a continuous improvement on the model such as the equations of motion, the scenery and the level of the fidelity of the model. Unlike commercial flight simulators where the simulation engine is hard coded, an opensource flight simulator provides the opportunity to customise the simulation engine to run various operating conditions. This will enable the study of the response and stability characteristics various aircraft parameters.

A flight simulator can be initially static based. It does not have any moving part to give the feel of acceleration to the pilot. The only realism felt by the pilot is the out-of-window visual cues. In that respect, the paper deals with such simulator. The desire to fully understand on the development of a 6DOF aircraft engineering simulator has lead to the study of employing opensource flight simulation software, FLIGHTGEAR [3] and a widely used technical computing language, MATLAB. FLIGHTGEAR provides the much needed tools required to visualise the instrument panels and the out-of-window scenery. The use of this software has minimised the time needed to build a fully functional

instrument panels and out-of-window scenery. The out-of-window scenery in FLIGHTGEAR is adequate for the flight simulator. It has a complete terrain database for every part in the world¹.

However, the flight dynamics model (FDM) in FLIGHTGEAR is difficult to customise as it requires a complete software compilation every time there are major changes to the equations of motion. Due to this limitation, the FDM are calculated by MATLAB. MATLAB has all the tools required to calculate the FDM in real time.

2. VEHICLE MATHEMATICAL MODEL

The simulator uses 6-degree-of-freedom equations of motion [1] to model the vehicle. In [1], the aircraft aerodynamic and propulsion data are provided in the form of tables.

2.1 The equations of motion

A complete list of the equations of motion is given in Eq. 1-13. They represent the states of the aircraft at any time instant during the simulation.

Force Equation

$$\dot{U} = RV - QW - g_D \sin \theta + (X_A + X_T)/m \quad (1)$$

$$\dot{V} = -RU + PW + g_D \sin \phi \cos \theta + (Y_A + Y_T)/m \quad (2)$$

$$\dot{W} = QU - PV + g_D \cos \phi \cos \theta + (Z_A + Z_T)/m \quad (3)$$

Kinematic Equations

$$\dot{\phi} = P + \tan \theta (Q \sin \phi + R \cos \phi) \quad (4)$$

$$\dot{\theta} = Q \cos \phi - R \sin \phi \quad (5)$$

$$\dot{\psi} = (Q \sin \phi + R \cos \phi) / \cos \theta \quad (6)$$

Moment Equations

$$\Gamma \dot{P} = J_{xz} [J_x - J_y + J_z] PQ - [J_z (J_z - J_y) + J_{xz}^2] QR + J_z \ell + J_{xz} n \quad (7)$$

$$J_y \dot{Q} = (J_z - J_x) PR - J_{xz} (P^2 - R^2) + m \quad (8)$$

$$\Gamma \dot{R} = [(J_x - J_y) J_x + J_{xz}^2] PQ - J_{xz} [J_x - J_y + J_z] QR + J_{xz} \ell + J_x n \quad (9)$$

where

$$\Gamma = J_x J_z - J_{xz}^2 \quad (10)$$

Navigation Equations

$$\dot{p}_N = U \cos \theta \cos \psi + V(-\cos \phi \sin \psi + \sin \phi \sin \theta \cos \psi) \dots \quad (11)$$

$$+ W(\sin \phi \sin \psi + \cos \phi \sin \theta \cos \psi) \quad (12)$$

$$\dot{p}_E = U \cos \theta \sin \psi + V(\cos \phi \cos \psi + \sin \phi \sin \theta \sin \psi) \dots \quad (13)$$

$$\dot{h} = U \sin \theta - V \sin \phi \cos \theta - W \cos \phi \cos \theta$$

The equations of motion are integrated using the Runge Kutta 4th order method [4]. The integration time step is carefully calculated at every cycle to achieve real time simulation. This is performed by measuring the time taken for each integration cycle and using Eq. 14.

$$\Delta t = |t_i - t_{i-1}| \times k \quad (14)$$

where k is a constant.

The value of k depends on the speed of the processor and is determined through a trial-and-error method. The outcomes are the state of the aircraft in body-axes as in Eq. 15.

$$X = [p_N \ p_E \ h \ \phi \ \theta \ \psi \ U \ V \ W \ P \ Q \ R]^T \quad (15)$$

where p_N is a north position, p_E is an east position, h is a height, ϕ is a roll angle, θ is a pitch angle, ψ is a yaw angle, U is a velocity at x-axis, V is a velocity at y-axis, Z is velocity at z-axis, P is roll rate, Q is pitch rate and R is yaw rate.

Next is control vectors, i.e. the aileron, elevator, rudder and throttle input are given in Eq. 16.

$$U = [\delta_a \ \delta_e \ \delta_r \ \delta_t]^T \quad (16)$$

State derivatives alpha-dot and beta-dot are approximated from previous simulation time-step. Control surface deflections are read from the joystick, pedals and throttle.

2.2 Aerodynamic coefficients

The stability and control characteristics of the aircraft flight model used in the simulation are defined by non-dimensional aerodynamic coefficients. They are used in conjunction with the aircraft geometry, mass and dynamic pressure.

The aerodynamic and thrust forces and moment coefficients are calculated through look-up tables with an interpolator that also will extrapolate beyond the limits of the tables. Therefore, the simulation could recover without loss of data even though temporarily exceeding the lookup-table limits.

Although, currently, the only aerodynamic data is from [1], the equations of motion can be customized to simulator other types of aircraft with minimal effort.

2.3 The propulsion

The engine model uses data provided from NASA [2]. The engine is an afterburning turbofan engine. The thrust response is modelled with a first-order lag. The lag time constant is a function of the actual engine power level and the commanded power. The engine thrust is basically a function of power level, altitude and Mach number.

2.4 The limits of the control surfaces

The control surface limits are given in Table 1. These values form the limit of the maximum deflection of control surfaces.

Table 1 : Control Surface Actuators and Throttle

| Controls | Deflection Limit | Rate Limit | Time Constant |
|----------|------------------|----------------------|----------------|
| Elevator | $\pm 25.0^\circ$ | $60^\circ/\text{s}$ | 0.0495 sec lag |
| Aileron | $\pm 21.5^\circ$ | $80^\circ/\text{s}$ | 0.0495 sec lag |
| Rudder | $\pm 30.0^\circ$ | $120^\circ/\text{s}$ | 0.0495 sec lag |
| Throttle | 2.5 – 1.0 | | |

2.6 Earth Coordinate System

The coordinate calculate by the simulation engine is based on Euclidean geometry, i.e. in x , y and z at every time step. In order to be useful in FlightGear, the coordinate has to be converted into Earth coordinate system widely known as World Geodetic System (WGS). The latest revision is WGS 84 which is valid until 2010. In WGS 84, the primary ellipsoid parameters are listed in Table 2.

Table 2 : The primary WGS 84 ellipsoid parameters

| Name | Values |
|--------------------------|---------------------|
| Semi-major axis, a | 6,378,137.0 m |
| Semi-minor axis, b | 6,356,752.314 245 m |
| Inverse flattening (1/f) | 298.257 223 563 |

The initial location of the flight is also important. In this case, the Kuala Lumpur International Airport (KLIA) is selected. The longitude and latitude of the taking off position at the airport are 101.7210° E and 2.7496° N, respectively. The heading angle is 326° . With this information, the aircraft will be positioned on end of runway 14L/32R.

2.7 Stability Augmentation System

The aircraft model is based on an existing fighter aircraft [2]. Without the stability augmentation system (SAS), it is a near impossible to fly the aircraft because of the sensitivity of the response of the aircraft with the given control surface deflections.

To design the stability augmentation system, the information required are the states of the aircraft at any time instance. These include the angular positions (roll, pitch and yaw) and also the rate of change of the angular position at any time instance. The information will be fed back to the SAS module.

The inputs given by the joystick are the demanded roll rate and the demanded pitch rate. The demanded roll rate and the demanded pitch rate are fed into the

MATLAB function that calculates the demand pitch and demand roll at every time step before calling another function that uses classical control method to determine the necessary deflection of the control surfaces.

2.8 Autopilot System

A simple autopilot system is also developed. The autopilot system is able to perform Altitude Hold and Heading Hold. A user will only need to switch on the Altitude Hold button and select the desired altitude. This is performed by using PID controller such as in Eq. 17.

$$\theta_d(t) = 0.055e(t) + \frac{8}{1e8} \int_0^t e(\tau) d\tau + 1.85 \frac{de}{dt} \quad (17)$$

where $\theta_d(t)$ = desired pitch angle, $e(t)$ = altitude error.

2.9 Navigation System

Without navigation, it is very difficult to fly from point A to point B. A simple navigation function is developed on MATLAB to allow user to select his departing airport and destination. A panel is developed to show to the user about his current position and the relative heading between his aircraft and the destination. The panel also shows the estimated time of arrival and the current distance.

2.10 Instrument Landing System

A simple instrument landing system is developed as well. The system is solely depending on the position of the aircraft in latitude, longitude and the heading. The instrument landing system works by measuring the perpendicular distance of the aircraft with the extended centerline of the runway as depicted in Fig. 1.

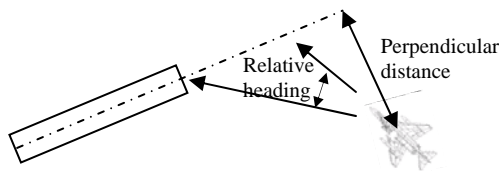


Figure 1: Definition of relative heading and perpendicular distance for the ILS system.

In order to land, the aircraft has to achieve zero relative heading and zero perpendicular distance with respect to the runway. Zero relative heading and perpendicular distance mean that the aircraft is properly aligned to the runway for landing. The relative bearing is calculated using Eq. 18.

$$\psi = \text{atan} \left(\frac{s(l_2 - l_1) c(t_2)}{c(t_1) s(t_2) - s(t_1) c(t_2) c(l_2 - l_1)} \right) - hdg_1 \quad (18)$$

where $s=\sin, c=\cos, l=\text{longitude}, t=\text{latitude}, hdg=\text{aircraft's heading}$, subscript 1=aircraft, subscript 2= destination airport.

The knowledge of the location of the runway and the position of the aircraft are used to calculate the 3-degree glide slope.

3. IMAGE GENERATORS

There are many image generators (IGs) available in the market. One of the producers of these IGs is Rockwell-Collins visual systems. Their products are used in many military and commercial flight simulators. The quality of the image is very good but the cost is prohibitive for a low-cost flight simulator. The opensource and free alternative image generator is FLIGHTGEAR (FG). FG is jointly developed by many flying enthusiast around the world and it is given free of charge. The source code is also available for download and licensed under GNU General Public License v2. FG allows anyone to study and modify the source code so that it can be used with other third party applications.

FG provides complete world scenery despite the quality of the image is not as good as the one provided by commercial IG. On top of that, FG can be customized to show an out-of-window view for a preset view angle. This means by using multiple FG computers, multiple views provides a 180-degree out-of-cockpit view.

Another advantage of using FG is that it accepts external input via TCP/IP. The external input in this case is a UDP broadcast from MATLAB. With multiple FG computers, the display of the 180-degree out-of-cockpit view can be done easily with a continuous broadcast from the MATLAB computer. The frequency of the broadcast is set 60 Hz. This frequency is found to be adequate to avoid jitter in the display. A jitter happens when MATLAB experiences a sudden hiccup caused by other applications in the computer which disturbs the transmission of the data. To avoid this, multiple computers are needed to perform other important tasks such as displaying the instrument panels and computing the states of the aircraft. Inputs required by the equations of motion can be fed into the computer via TCP/IP packets from other computers. This approach will equally distribute the work load hence reducing jitter.

4. MODELLING IN MATLAB

The development of the simulator starts by preparing the equations of motion as a function in MATLAB. The function is called from a main program that will perform other functions such as auto-pilot, navigation, reading input from the pilot, and communicating with image generators.

4.1 The main program

The main program in MATLAB (Figure 3) starts the simulation. It reads the aircraft initial data and initializes the communication with the input system and the image generators.

The simulation starts by invoking the infinite *while-loop*. All of the important functions are inside the loop.

4.2 Communication with input devices

The input devices are connected to a separate program. Three special programs are written in Python to read input from the joystick, pedals and throttle, respectively.

Each program communicates with MATLAB through the internet connection via TCP/IP. The reason behind this idea is to reduce the workload of MATLAB and to maintain the industry standard of 33 ms of simulation cycle.

A USB-based joystick is used to control the aileron and elevator. It has 13 buttons that can be configured to perform various tasks such as a panic button. A panic button is a button that will automatically bring the aircraft back to straight and level flight if the flying condition is uncontrollable. A Python script is used to read input from the joystick and send the input to MATLAB via UDP port 5400.

The rudder is controlled by pedals. Input from the pedals is read by using a different Python script and send to MATLAB via UDP port 5401. A different input device is used as the throttle. The throttle input is read by a separate Python script and the data is send to MATLAB via UDP port 5402.

With this approach, the processing load can be distributed among several computers in order to achieve real time simulation.

4.3 Communication with image generators

At every cycle, MATLAB and image generator (FlightGear) will exchange data. MATLAB sends the states of the aircraft to FlightGear. FlightGear will use the information to display the out-of-window display.

However, MATLAB has no information about the terrain and ground elevation. The information is important for landing purposes and terrain avoidance. FlightGear terrain database is used to provide the information to MATLAB. The information is sent to MATLAB at every cycle through TCP/IP.

This configuration allows the use of multiple image generators hosted on separate computers. Each image generator displays a particular view angle and when combined, a wide view angle display is achieved.

In this setting, MATLAB broadcasts the states using UDP packets via port 5500. The quality of the display is found to be satisfactory.

5. IMPLEMENTATION

The hardware of the computer is from off-the-shelf components. This allows the cost to be kept as low as possible. The only expensive hardware is the video controller. The video controller has to support 1080p resolution to increase the resolution. The video controller can be connected to an LCD project via a standard VGA cable or via an HDMI cable to an HD LCD television. A simple tweak on the configuration is needed on the FlightGear side to change the view angle displayed on the display system. Hence, when combined, it allows the display system to show a 180 degree view angle. In fact, if the cost of hardware is not an issue, the system could be configured to show a wider angle view.



Figure 2: The complete simulator system.

The simulator requires several computers to function properly. A complete simulator system is shown in Fig. 2.

The amount of processing load required by MATLAB is relatively high and in order to maintain the length of the time step not to exceed 33 ms, the reading of input devices is performed on a separate computer. The computer runs several Python scripts. Each script reads input from a joystick, pedals and a throttle, respectively. This is found to be the best approach. Since all input devices are USB-based, they are very easy to install and operate. Those scripts will continuously scan for changes from the input devices and send the information to MATLAB.

The interesting aspect of this simulator is the use of TCP/IP for communication between computers. This allows tasks to be distributed onto several computers and keep the processing load low.

5.1 Instrument Panel



Figure 3: The instrument panel.

The instrument panel consists of a glass cockpit, horizontal situation indication, autopilot panel, ILS panel, navigation panel and a map. Google Earth is used to show the position of the aircraft on the map. Fig. 3 shows the whole panel display of the simulator. It consists of several instrument panels. Some panels are permanently displayed on the screen. Those are the artificial horizon, the horizontal situation indicator and the map. Some other panels can be switched on and off by the user.

The glass cockpit and horizontal situation indicator display are shown in Figure 4. The left side of Figure 4 shows the artificial horizon indicator, the speed, the vertical speed, the altitude and heading. The indicator is from an opensource project named Open Glass Cockpit (OpenGC). MATLAB communicates with OpenGC via UDP port 5800. A special MATLAB function is developed to talk with OpenGC.

A horizontal situation indicator (HSI) is developed using Python and OpenGL. The data is transmitted from MATLAB via UDP port 5900. The information send are heading, the normalized values of the localizer and glide slope.



Figure 4: Glass cockpit and HIS

The autopilot panel, shown in Figure 5, interacts with an autopilot function in MATLAB. The panel reads the user's input for Altitude-hold/Value, Heading Hold/Value and Vertical Speed/Value. In Figure 5, if the "Alt" button is pressed and the slider is set to a specific value that represents the targeted altitude, the simulator will compute the required change of the pitch rate and begin climbing until the desired altitude is reached.



Figure 5: The autopilot panel.

The panel for landing is shown in Figure 6a. In order to align to the runway, the user has to fly the aircraft in a way that the values of "Bearing To" and "Distance to CL" are almost zero.

The main control panel is shown in Figure 6b. This panel starts the simulator by starting up Python-based functions and OpenGC; toggling the Gear button and enabling the SAS button. Other functions include stop buttons and a button to show the time history of the states. The time history of the states will appear when the simulation stops and can be used to evaluate the performance and response of the aircraft.

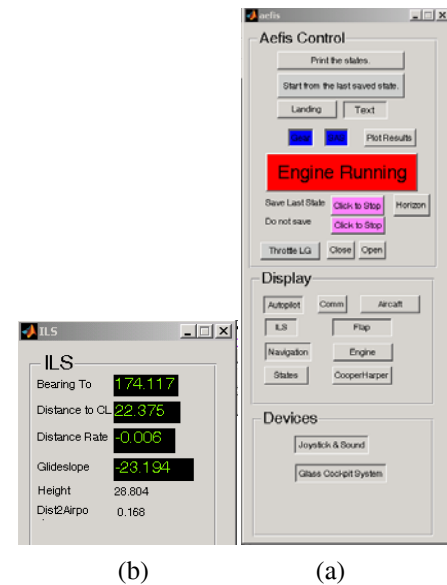


Figure 6: Panel to guide the aircraft for landing and main control panel.

6. CONCLUSION

A complete simulator was developed successfully using low-cost alternatives. All equations and functions are written in-house which give an additional advantage for customization for further development. The development cost is kept low by using opensource software and tools developed in-house thus very suitable as a teaching tool in aircraft control and design courses.

REFERENCES

1. Stevens, Brian L. (2003) *Aircraft Dynamics and Simulation*, John Wiley and Sons: New York.
2. Gilbert, W.P. et al (1976) "Simulator Study of the Effectiveness of an Automatic Control System Designed to Improve the High Angle-of-Attack Characteristics of a Fighter Plane" NASA TN D-8176
3. FLIGHTGEAR <http://www.flightgear.org> 30 January 2011.
4. Butcher, JC (1987) *The Numerical Analysis of Ordinary Differential Equations: Runge-Kutta and General Linear Methods*, John Wiley & Sons, New York.