AIAA Modeling and Simulation Technologies Conference and Exhibit
20 - 23 August 2007, Hilton Head, South Carolina

AIAA 2007-6372

The Development and Application of *SimpleFlight*, a Variable-Fidelity Flight Dynamics Model

Matthew M. Duquette[1]
Air Force Research Laboratory, Wright-Patterson AFB, OH, 45433

The design and application of a fidelity-independent flight model is presented. The principal aim of this model is to serve the needs of low to medium fidelity applications by providing a modular framework design in which users can add or remove components. The development of performance-based models are presented, showing simple methods for determining basic aerodynamic coefficients, implementing a stable autopilot, and performing automatic navigation. Finally, a description of the implementation of this model in C++ and Java is given.

Nomenclature

α	=	Angle of attack
β	=	Angle of sideslip
ε	=	Thrust angle relative to body-x axis
ρ	=	Air density
API	=	Application Programming Interface
[ψ, θ, ϕ]	=	Body Euler angles (yaw, pitch, roll)
B	=	Wing span
[C_L,C_D]	=	Aerodynamic force coefficients (lift, drag)
DOF	=	Degree of freedom
g	=	Earth gravitational acceleration
M_∞	=	Mach Number
n	=	Load factor
PID	=	Proportional, Integral, Derivative
[p, q, r]	=	Body angular rates (roll, pitch, yaw)
S	=	Wing area
[u, v, w]	=	body velocities
V_∞	=	Free stream air speed
W	=	Weight
WGS-84	=	An ellipsoidal Earth model
[x, y, z]	=	Body axes (positive out of nose, positive out of right wing, positive downward)
V_Z	=	Downward velocity in the inertial frame

I. Background

A cursory examination of the aerospace modeling and simulation field shows a myriad of flight models. Within industry, academia, and the government, models range from basic 3-DOF representations to complex and physically accurate 6-DOF models. When choosing a flight model to use in a simulation, there may be a tendency to assume that the highest fidelity model available must be the "best" and therefore should be used. However, advanced modeling requires a large set of data. Often, detailed information regarding the stability and performance of an aircraft is not available. Nelson[1] lists twenty-six stability and performance coefficients as well as six mass properties and three geometric properties for a typical linear stability model of an aircraft. If wind tunnel testing and analysis is necessary, obtaining data can cost hundreds of thousands of dollars. If non-linear dynamics are considered, the cost can be substantially more[2]. In recent years, a number of web-based projects and commercial games have implemented increasingly advanced aircraft models[3,4,5]. These models, though many intended for

[1] Aerospace Simulation Engineer, AFRL/VACD, 2180 8th Street, AIAA Member.

gaming, achieve impressive results compared to real-life data. However, even models created for games are made up of many aircraft parameters, and are often complex and difficult to change.

Consider the case of the concept vehicle; one which is loosely defined by basic performance parameters such as weight and size, but may not have an established geometry. This presents a very limited set of data from which to develop a flight model. This case requires an easily configurable flight dynamics model; one that would provide realistic flight dynamics with a minimum set of input data. Such a model would be useful for studies that involve parametric comparisons of several proposed aircraft configurations. Given a limited set of data, this is a difficult task. Aerodynamics, propulsion, navigation, and control would need to be abstracted from the end user, allowing analysts to create simple and stable flight models. In most cases, the flight model would also require a stable control system with autopilot control. Many high-fidelity flight models feature stable control systems, but are sensitive to changes in aircraft characteristics. While it is certainly possible to simplify a high-fidelity model by changing the myriad of aerodynamic coefficients, or removing some entirely, the result would be a very different airframe than the original model. This would require changes in the control system and autopilot to maintain acceptable stability and control. For detailed control systems, many man-hours can be spent tweaking control system parameters to derive a stable and controllable aircraft. If this process is repeated over several designs, the cost of the model can become significant.

For some studies, a 3-DOF model (one that only considers vehicle translation) may be sufficient. Performance characteristics based on statistical models or parametric variation can be used to create adequate representations. Spreadsheet analysis can be used to perform comparisons of designs performing a particular task. However, 3-DOF models consider only the position of an aircraft. In many research situations, the orientation and rotational rate of the aircraft are important parameters. Some examples of these situations include,

1) Human factors studies,
2) Analysis dependent on threat signatures,
3) Viewing aircraft motion,
4) Performing sensor tasks from an aircraft

In these situations, the aircraft needs to exhibit realistic rotational characteristics to be effective. It is clear that there is a need for a flight model that can fill gaps in current modeling capability between the low-fidelity and high fidelity models. *SimpleFlight* aims to fill the gap between the low fidelity 3-DOF flight model and the high fidelity, specialized 5-DOF or 6-DOF model. It does this by providing an aircraft model framework that is based on modern computer languages and standards, and supplies users with a set of models for aircraft components to provide easily configurable aircraft models.

II. *SimpleFlight* Approach

Four goals drove the development of *SimpleFlight*. First of all, the model should satisfy the need for an easily configurable, stable flight model that includes all of the necessary components to fly waypoint-controlled flight paths. Secondly, the model should be fidelity-independent, allowing users to build new components to model more advanced aircraft and aircraft systems. Thirdly, the design should allow for mixed levels of fidelity by using a modular architecture. This enables users to implement the desired level of detail in each aircraft component. Lastly, *SimpleFlight* should be written using modern programming languages, standards and techniques.

Several techniques were developed to allow *SimpleFlight* to provide realistic dynamics without the need for an extensive set of data. The combination of a fidelity-independent modern architecture with a set of modules aimed to satisfy an under-populated area of the flight model market, *SimpleFlight* provides one solution to a difficult problem; modeling aircraft to a level of fidelity that is appropriate for the situation while minimizing the effort to do so. This paper describes some of the modeling techniques used to create modules for *SimpleFlight*, as well as the implementation of the program as a whole.

III. Flight Model Dynamics

The essence of a flight model lies in its core vehicle dynamics. *SimpleFlight* was designed with the concept that many levels of fidelity may be mixed together in one model. However, the core equations of motion and basic environmental properties should be available for use in any model. Moreover, the calculation of the motion of the vehicle should tolerate varying levels of detail. For example, quaternion-based rotation[6] was implemented for its lack of singularity compared to the Euler-based direction cosine matrix. A WGS-84 model[7] was added to enable real-world geodetic motion. A 1976 standard atmosphere model was also developed. References 1, and 3 – 6 show various methods for developing a proper flight dynamics model using these techniques and models. Figure 1 shows

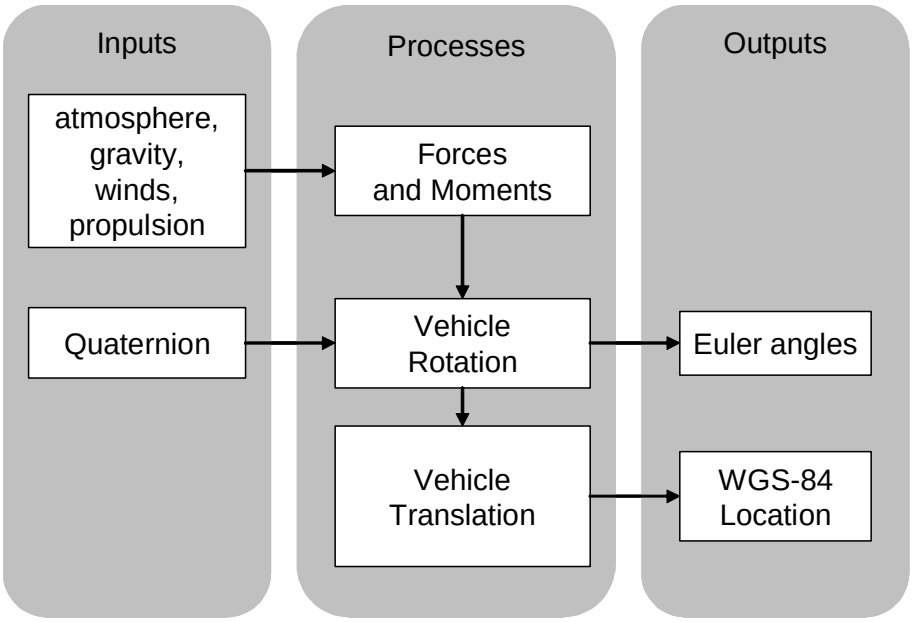


Figure 1. The *SimpleFlight* Dynamics Engine. *The vehicle dynamics in SimpleFlight follows the common formula for a flight model. Several built-in methods include an atmosphere model, support for winds, and WGS-84 geodetic motion.*

notionally how *SimpleFlight* uses inputs to compute vehicle motion as well as produce outputs that may be useful to other elements of the flight model.

SimpleFlight includes routines for the basic equations of motion for an aircraft. However, one additional characteristic of the *SimpleFlight* equations is the optional inclusion of a no-slip condition. In this mode, the aircraft flies in a manner that ensures zero sideslip. All lateral force is converted to rotation around the vertical axis. This is achieved by using the properties shown in Eq. 1. At each step in the calculation, β and v are set to zero, and all change in β is added to the yaw rate of the aircraft. While this method is not a realistic representation of aircraft dynamics in all situations, it does reduce the complexity in the control system, and guarantees coordinated flight. For the types of applications that are listed in previous sections, this added flight stability is an attractive attribute of *SimpleFlight*.

$$r = \dot{\beta}, \quad v = 0, \quad \beta = 0 \tag{1}$$

IV. Inverse Aerodynamic Calculation

Aerodynamic coefficients are usually determined through experimentation or by the use of analytical tools that use geometric data for the aircraft of interest. However, if one knows the desired performance of an aircraft, classical methods usually require iterations of estimation and testing in order to match known performance. A method is presented which allows the determination of the coefficient of lift (C_L) and the coefficient of drag (C_D) via the inversion of the equations of motion for steady-state flight. This method is useful in situations where one is interested in modeling the performance of an aircraft where one or more flight conditions are known, but information regarding aerodynamic derivatives is not available.

A. Force Determination

First, the relevant forces acting on the aircraft must be established. Consider an aircraft flying in steady-state flight conditions. The aircraft has zero angular velocity in all three axes, and there is zero side slip. Fig. 2 shows this situation. Equations 2 through 4 describe the forces in terms of the body axes of the airplane, where the x-axis extends out the nose of the aircraft, and the z-axis is positive downwards. Theta (θ) describes the pitch angle relative to the horizon. Figure 2 shows the freebody force diagram.

$$F_x = T\cos\varepsilon - D\cos\alpha + L\sin\alpha - W\sin\theta \tag{2}$$

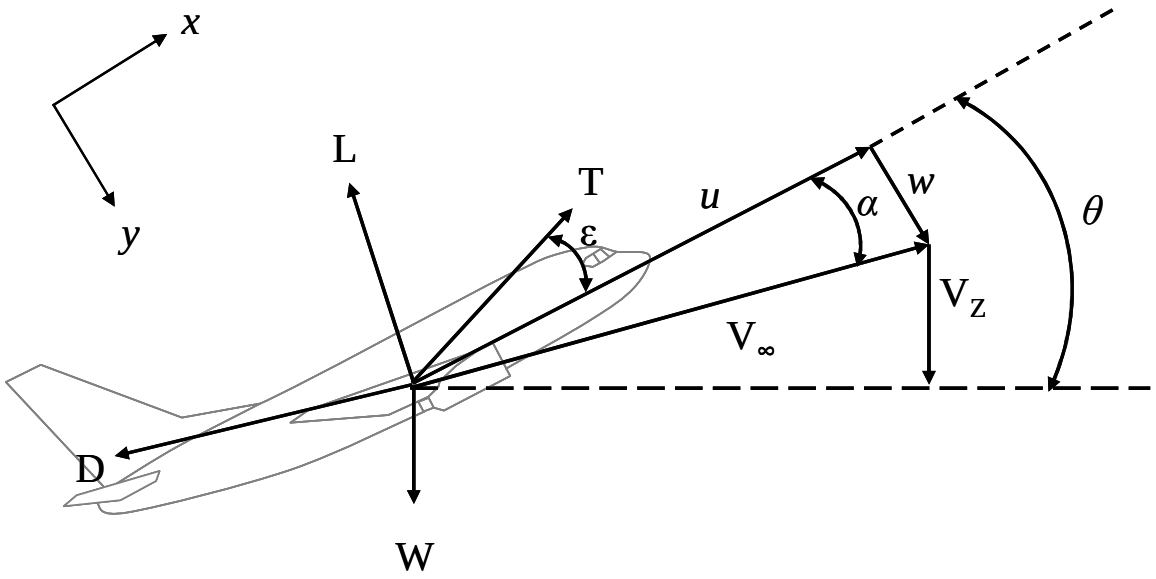


Figure 2. Force Diagram. *The forces acting against an aircraft in flight can be used to determine steady state C_L and C_D.*

$$F_y = 0 \tag{3}$$

$$F_z = -T\sin\varepsilon - D\sin\alpha - L\cos\alpha - T\sin\varepsilon + W\cos\theta \tag{4}$$

Equations 2 through 4 require thrust to be known. For air breathing engines, thrust is a function of the variables shown in Eq. 5.

$$T = f\left(Throttle, \rho, V_\infty, M_\infty\right) \tag{5}$$

The simplest assumption is that thrust is only dependent on throttle setting. However, a *SimpleFlight* module was developed to model a turbojet engine based on Anderson[8]. Anderson also shows basic relationships for other engine types as well as relationships for the thrust specific fuel consumption (TSFC).

The thrust angle and weight are assumed to be known. The aerodynamic forces, lift and drag, are unknown. For a given angle of attack and pitch angle, Eqs. 2-4 can be restated to solve for lift and drag. If the aircraft is in non-accelerated flight, the force in each dimension must sum to zero. Equations 6 and 7 show the body forces restated in terms of the aerodynamic forces. All other forces are summed on the right hand side of the equation and set equal to the aerodynamic forces. Solving for lift and drag, Eqs. 8 and 9 are found.

$$F_{X,aero} \equiv L\sin\alpha - D\cos\alpha = W\sin\theta - T\cos\varepsilon \tag{6}$$

$$F_{Z,aero} \equiv -L\cos\alpha - D\sin\alpha = T\sin\varepsilon - W\cos\theta \tag{7}$$

$$L = F_{X,aero}\sin\alpha - F_{Z,aero}\cos\alpha \tag{8}$$

$$D = -F_{X,aero}\cos\alpha - F_{Z,aero}\sin\alpha \tag{9}$$

The angle of attack (α) is found using Eq. 10. The vertical velocity of the aircraft in the inertial frame (V_Z), and the airspeed (u) can be used to find w and V_∞, as shown in Eqs. 11 and 12.

$$\alpha = \arctan\left(\frac{w}{u}\right) \tag{10}$$

$$w = \frac{(V_Z + u\sin\theta)}{\cos\theta} \tag{11}$$

$$V_\infty = \sqrt{w^2 + u^2} \tag{12}$$

The lift and drag can be found by solution of the linear system of Eqs. 8 and 9. The coefficients of lift and drag are easily found by using the commonly accepted definition of those coefficients, and are shown in Eqs. 13 and 14.

$$C_L = \frac{L}{\frac{1}{2}\rho V_\infty^2 S} \tag{13}$$

$$C_D = \frac{D}{\frac{1}{2}\rho V_\infty^2 S} \tag{14}$$

The above method can be used to find the C_L and C_D for an aircraft at one or more points. If two points are chosen, such as a climb or stall condition and a cruise condition, then this method can used to develop curves for C_L and C_D. While the method ensures deterministic performance for steady state flight at each point calculated, a curve fitting method must be used in order to determine the C_L and C_D curves for the aircraft over a range of angles of attack.

B. Curve Fitting Methods

Three curve fitting methods are presented. First, a method is shown that assumes a linear lift curve and a drag curve based on C_L. The second method is based on curve-fitting periodic functions to two performance points. The third method introduces stall modeling for more accurate high-angle-of-attack values.

1. Linear Method

Assuming that lift is a linear function of angle of attack, the lift and drag curves can be described by Eqs. 15 and 16, where C_{Lo} is the C_L at $\alpha = 0$ and C_{Do} is the C_D at $C_L = 0$.

$$C_L = C_{Lo} + A\alpha \tag{15}$$

$$C_D = C_{Do} + BC_L^2 \tag{16}$$

Two sets of aerodynamic conditions [α, C_L, C_D] can be determined by the method described in section A. Equations 15 and 16 can then be solved for [A, B, C_{Lo}, C_{Do}]. The result is a linear lift curve for all angles of attack. The drag curve is parabolic with C_L. If the aircraft is realistically modeled, C_{Do} will be greater than zero.

2. Periodic Method

Flat plate theory states that for a perfect, 2-dimensional flat plate, the lift and drag are curves are described by Eqs. 17 and 18.[9]

$$C_L = 2\sin\alpha\cos\alpha \tag{17}$$

$$C_D = 2\sin^2\alpha \tag{18}$$

This implies, however, that the lift and drag at α = 0 will be zero, which is not realistic for all aircraft. Furthermore, the slope of the lift and drag curves will vary depending on the results of the point determination of coefficients. Therefore, Eqs. 17 and 18 should be modified to reflect these facts and is shown in Eqs. 19 and 20.

$$C_L = C_{Lo} + A\sin\alpha\cos\alpha \tag{19}$$

$$C_D = C_{Do} + B\sin^2\alpha \tag{20}$$

As before, this system can be solved using two known points of [α, C_L, C_D]. Here, drag is a function of α and not C_L. Furthermore, the lowest point in the drag curve (C_{Do}) will always be at α = 0. While this is not always the real-world case, this method better approximates performance over the range of α, while the linear method is a better model of lift and drag at low angles of attack. One may also observe that the greatest lift is achieved at α = 45°, consistent with 2-dimensional flat plate theory. In reality, aircraft tend to stall at angles of attack of approximately 15°. The sinusoidal curves also do not capture the stall event in terms of the discontinuous changes in lift or drag slope seen with stall.

3. Post-Stall Method

Viterna and Corrigan[10,11] proposed a model for post stall lift and drag on an airfoil. The method is based on curve-fitting for α greater than stall but less than 90°. In the original method, the pre-stall regime is assumed to be linear in lift and parabolic in drag. Wind tunnel data typically provides the pre-stall points. The post-stall calculation is paired with pre-stall data to form a continuous curve. Here, the pre-stall data is obtained using the linear method. Points beyond α=90 are assumed to be zero for both lift and drag. The combination of these methods results in a continuous function for lift and drag in [-180 < α > 180].

This method uses the aspect ratio, AR to find the maximum C_D, as shown in Eq. 21.

$$C_{D,max} = 1.11 + 0.018AR \tag{21}$$

where,

$$AR = \frac{B^2}{S}$$

The post-stall lift and drag curves are established using Eqs. 22 and 23. Note that the C_L and C_D at stall must be known. These can be determined using Eqs. 15 and 16 at the stall angle of attack. For [$\alpha_{stall} \le \alpha \le 90$ °], Eqs. 22 and 23 are applied to find C_L and C_D.

$$C_L = A_1 \sin 2\alpha + A_2 \frac{\cos^2\alpha}{\sin\alpha} \tag{22}$$

$$C_D = B_1 \sin^2\alpha + B_2 \cos\alpha \tag{23}$$

where,

$$A_1 = B_1/2, \qquad A_2 = \left(C_{L,stall} - C_{D,\max}\sin\alpha_{stall}\cos\alpha_{stall}\right)\frac{\sin\alpha_{stall}}{\cos^2\alpha_{stall}},$$

$$B_1 = C_{D,\max}, \qquad B_2 = \frac{C_{D,stall} - C_{D,\max}\sin^2\alpha_{stall}}{\cos\alpha_{stall}}$$

4. A Comparison of Methods

All three of the above modeling methods have advantages and disadvantages in different situations. The linear method is consistent with the established definitions of lift and drag. For small angles of attack, this method will best approximate experimental results. However, for excursions beyond this flight regime, this method quickly diverges from reality. The periodic method is more realistic for all α, since the maximum drag is reached at $\alpha = 90°$ and lift becomes negative for $\alpha > 90°$. However, this method results in minimum drag at $\alpha = 0$ for all cases, which is not true in all cases. Furthermore, stall is not modeled using this method, so maximum lift is achieved at $\alpha = 45°$. The sinusoidal nature of the curves are good approximations of the straight-line lift and parabolic drag found by the linear method at low angles of attack. The post-stall method is the most accurate at high angles of attack, but to be continuous in α, the pre-stall regime is modeled using the linear method, so large negative values of α will result in unrealistic lift and drag values.

C. Example

An example is provided in order to show the methods of this section applied to a notional aircraft. The aircraft is described in Table 1. Each flight condition (cruise and power-on stall) are used to find aerodynamic data points using the method shown in Section A. The airspeed is the speed relative to the body x-axis of the aircraft and climb rate is the speed relative to the negative world z-axis.

For the data in Table 1, the following results are obtained:

For the cruise condition: $\alpha = 0$; $C_L = 0.675$; $C_D = 0.163$

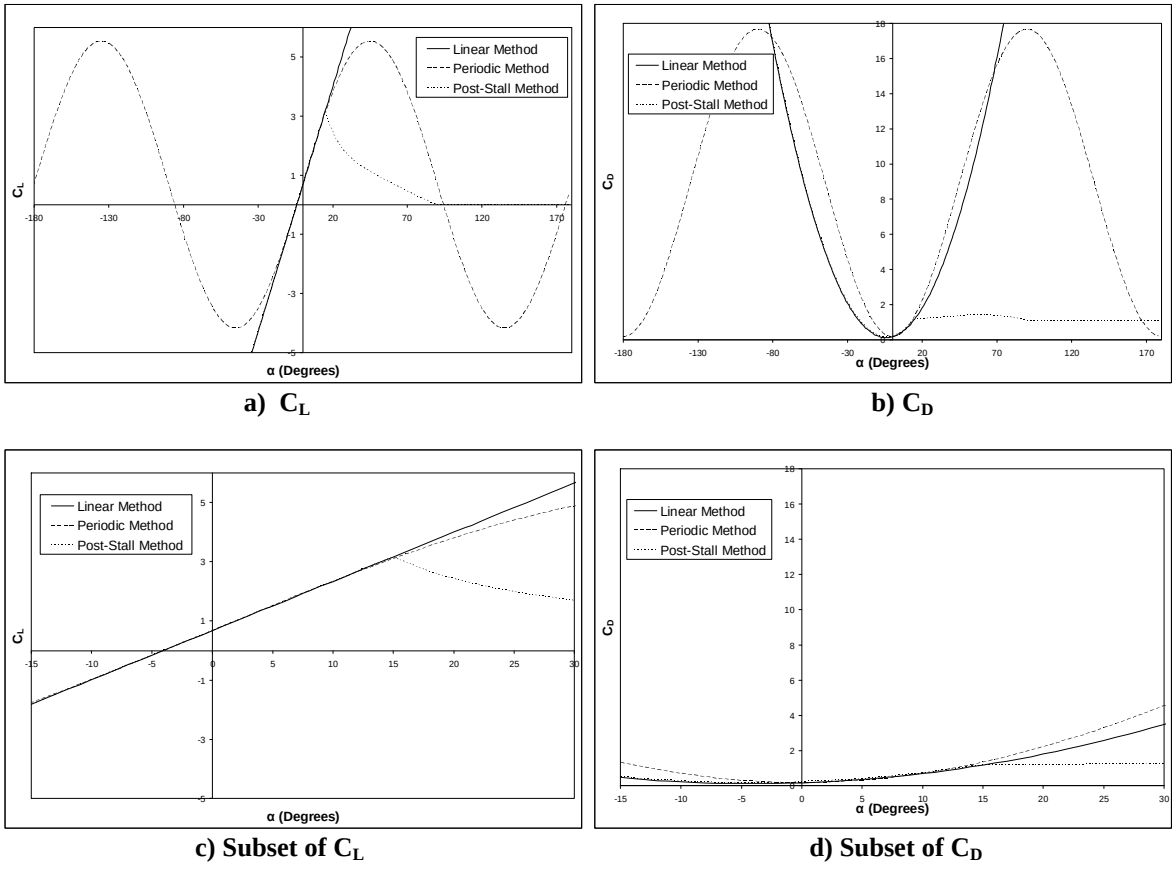
For the stall condition: $\alpha = 10$; $C_L = 2.335$; $C_D = 0.691$

Figure 3 shows a comparison of the lift and drag curves based on the above results. Note that the linear method shows extreme results for both C_L and C_D at high α. The lift curve is also discontinuous across the 180 to -180 threshold. The periodic method results in a more reasonable curve that is continuous for all α. However, the post-stall model captures the fall-off in lift after the stall (which was assumed to be 15°). All three methods are in close agreement at low angles of attack as shown in subfigures c) and d) in Fig. 3.

Approximating lift and drag for an entire aircraft in steady state conditions can be accomplished using two flight conditions as data points. This first-order approximation may be adequate for lower fidelity simulation when complete and accurate aircraft data are not available. The method of interpolation for lift and drag along the entire angle of attack range also depends on the desired level of fidelity. For high-α conditions, the post-stall model may be more accurate. However, the periodic method provides reasonable results over the range of α. The linear method can be applied in situations when the accuracy of small α values is important.

Table 1. Aircraft Description. *This description is used in the example calculation.*

Wingspan	90 ft
Wing area	1000 ft^2
Weight	140,000 lbs
Max Thrust	40,000 lbs
<u>Cruise Condition</u>	
Pitch	0°
Airspeed	480 kts
Climb rate	0 fpm
% Max thrust	85%
Altitude	35000 ft
<u>Power-on Stall Condition</u>	
Pitch	10°
Airspeed	135 kts
Climb rate	0 fpm
% Max thrust	100%
Altitude	0 ft



a) C_L **b) C_D**

c) Subset of C_L **d) Subset of C_D**

Figure 3. Curve Fitting. *The results of curve fitting methods for the example aircraft. a) and b) show curves over the entire range of α. c) and d) show the same curves in the normal operating range of an aircraft.*

V. Automatic Flight

A. Autopilot Design

Autopilot design is usually highly dependent on the airframe being considered. Although most aircraft include the standard primary control surfaces (ailerons, elevator, and rudder), the control of those surfaces to achieve a desired effect can be a very complicated task. Movement of control surfaces induces a rotational acceleration on the airframe. Some factors that affect rotational acceleration are: size and deflection of the control surface, the moments of inertia of the aircraft, airspeed, and the orientation of the aircraft. There is also coupling between the axes of rotation, such as yaw that is induced by deflection of the ailerons, or roll induced by deflection of the rudder.

The area of autopilot control and automatic flight is well studied. For full-featured aerodynamic models, advanced autopilots employing PID control have been developed[4]. As stated earlier, future concepts, trade space analysis, or simply a lack of data results in the need for simple effects-level flight models. A simplified airframe model calls for a simplified autopilot. A method is presented which bypasses the control surface deflection aspect of rotational control. By bypassing the lowest level of control, a more robust autopilot system can be employed that does not depend on individual weights applied to multiple, cascading controllers, resulting in autopilot control that is stable for a wide range of aircraft.

A simplified model of rotational and translational dynamics is used to develop a basic 3-axis autopilot. Here, the autopilot functions are limited to:

1) Vertical Speed Hold
2) Altitude Hold
3) Heading Hold
4) Orbit (Constant Bank)

5) Speed Control

Each controller is assumed to be independent of other controllers, with the exception of (1) and (2). Also, each controller affects only one axis of rotation. In the case of (5), the speed is controlled via a throttle command, so has no direct impact on rotational rates. The autopilot components with the exception of (5) control affect the body rotational rates. This system ensures perfect tracking of commanded rate, without considering the rotational inertia of the aircraft. In this model, the body rotational rates are often related to the Euler angles. Therefore, coordinate system conversion must be performed in order to relate the current Euler angles to body axis rotation and vice versa.

1. Vertical Speed Hold

Vertical speed hold is an essential component in maintaining climb angle and also plays a role in holding a constant altitude. In actuality, vertical speed is a function of several factors, including the angle of attack, forward speed, and thrust setting. For the purposes of developing a simplified autopilot, the vertical speed is assumed to be a function of the pitch angle and forward speed of the aircraft. Referring to Fig. 2, the vertical velocity, V_Z, is the parameter of interest in a vertical speed hold autopilot. It is easy to see that w is a function of α and that V_Z is a function of α and θ. Assuming that α is small, which is the case in normal flight conditions, we can assume that $V_\infty \approx u$. Further assuming that the aircraft is in a near zero roll angle ($\phi = 0$) and negligible yaw rate, Eq. 24 describes a simplified relationship between the body pitch rate, q, and the difference in vertical velocity.

$$\Delta V_Z \approx uq \tag{24}$$

Rearranging Eq. 24, Eq. 25 shows how the commanded pitch rate is found. The change in V_Z is the difference between the commanded and current values of V_Z. Note that there is a term based on the acceleration in the z-direction. This term, combined with the time constant, t_θ, helps damp oscillations in pitch. In practice, t_θ of zero can still yield stable autopilot response for many designs.

$$q_{cmd} = \frac{V_{Z,cmd} - V_Z - \frac{dV_Z}{dt} t_\theta}{u} \tag{25}$$

Limits must be placed on q_{cmd} to insure that the aircraft does not pitch too high or low, or change pitch too quickly. Recall that this autopilot bypasses the control surfaces, so the commanded values for rotations become instantaneous rotations for the aircraft. By placing limits on pitch rate, the aircraft will change pitch more realistically. A common parameter for setting a pitch rate limit is the load factor, n, which is stated as a multiple of the earth gravitational acceleration measured in the body-z-axis of the aircraft. Referring back to Fig. 2, and the simplifying assumptions that were used for Eqs. 24-25, the vertical acceleration (or g-force) that the aircraft experiences is a function of vertical plunging of the aircraft (change in w over time) and the pitch rate. Since ΔV_Z over a given time step is equivalent to the acceleration in the world-z axis, we can assume that for small θ and ϕ, Eq. 26 is valid. Restating this in terms of n, and solving for the maximum q, we get Eq. 27, where the quantity g is the average gravitational acceleration of the Earth at its surface. Eq. 27 can also be used to find the minimum pitch rate based on a minimum n limit. Equation 28 shows this relationship.

$$q = \frac{a_{z,world}}{u} \approx \frac{a_{z,body}}{u} \tag{26}$$

$$q_{\max} = \frac{n_{\max} g}{u} \left(\frac{\theta_{\max} - \theta}{\theta_{\max}} \right) \tag{27}$$

$$q_{\min} = \frac{n_{\min} g}{u} \left(\frac{\theta_{\min} - \theta}{\theta_{\min}} \right) \tag{28}$$

Note that in Eq. 27, the maximum q is multiplied by a quantity based on the maximum and current pitch angles. This ensures both that the aircraft does not pitch up (or down) too far while maintaining a smooth deceleration

toward the maximum pitch angle. By specifying a maximum pitch angle, excessive pitch will never be commanded by the autopilot.

When the aircraft is turning, there is a significant change in the amount of q needed to compensate for the tendency for the aircraft to slip downwards. To maintain altitude in a turn, Eq. 29 is added to the commanded q value. The r value shown is the yaw rate.

$$q_{turn} = r \tan \phi \tag{29}$$

2. Altitude Hold

The altitude hold system drives the vertical speed hold. The commanded vertical speed is based on the difference between the commanded altitude and the current altitude. Equation 30 shows how commanded vertical speed is determined. The weight, k_{alt} is simply a weight chosen to provide a desired scale of $V_{Z,cmd}$ for the aircraft. If the MKS system is used, a k_{alt} of less than 1.0 works well in practice. The $V_{Z,cmd}$ is typically limited to a maximum and minimum value of vertical speed.

$$V_{Z,cmd} = k_{alt}\left(alt_{cmd} - alt\right) \tag{30}$$

3. Heading Hold

The heading hold system controls the aircraft bank angle (ϕ) to correct for the difference between commanded and current heading. In some ways this is more complicated than the vertical speed or altitude hold systems, because it couples a change in the yaw axis with a change in the roll axis. This system assumes small slip angles ($\beta \approx 0$).

$$\Delta\psi = \psi_{cmd} - \arctan\left(\frac{V_{east}}{V_{north}}\right) \tag{31}$$

or,

$$\Delta\psi = \psi_{cmd} - \psi$$

The method begins by finding the difference between the aircraft heading and the bearing to the waypoint, $\Delta\psi$. Equation 31 shows two methods for determining $\Delta\psi$. The first method uses aircraft ground track, where V_{east} and V_{north} are the components of the horizontal earth axis velocity. The second relationship uses the aircraft heading directly. One may wish to use the ground track in order to compensate for wind effects. The result would be an aircraft that flies to a waypoint with an offset heading, or *crab angle*.

The heading difference determines the commanded bank angle, ϕ_{cmd}, as shown in Eq. 32. V_∞ is included to adjust the bank rate based on speed. As the speed increases, there is less of a change in ψ for a given bank angle. Including V_∞ in the equation adjusts the commanded bank angle to maintain a more consistent $\Delta\psi$ throughout the speed range. The maximum change in bank rate ($\dot{p}_{max}$) helps to compensate for the roll-out time when approaching ψ_{cmd}.

$$\phi_{cmd} = k_\phi \Delta\psi V_\infty \dot{p}_{max} \tag{32}$$

The weight, k_ϕ, is determined by the user's requirement for bank angle rate. Large weights would result in oscillations around the commanded ψ while weights that are too small would result in slow bank response. Usually, k_ϕ is less than 1.0.

Once ϕ_{cmd} is found, the aircraft is rotated around its longitudinal axis at the rate, p_{cmd} using Eq. 33. This is clamped to the maximum bank rate, and accounts for the sign of ϕ_{cmd}.

$$p_{cmd} = \min\left(\left|\phi_{cmd} - \phi\right|, p_{max}\right) sgn(\phi_{cmd}) \tag{33}$$

4. Orbit

Here, an orbit hold system is simply a condition of constant turning at a constant commanded bank angle. The heading hold system is engaged with a continuously changing commanded heading value that is +/- 179° from the current ψ.

5. Speed Control

A simplified auto-throttle is presented. In this system, thrust is the only speed control mechanism used. Equation 34 shows how the difference in speed, Δu, is determined. The forward speed, u, is the speed that is of interest in this controller. The value t_{spool} is the time that the engine requires to change thrust from 0 to 100%. Taking the acceleration ($\dot{u}$) into account minimizes overshoot.

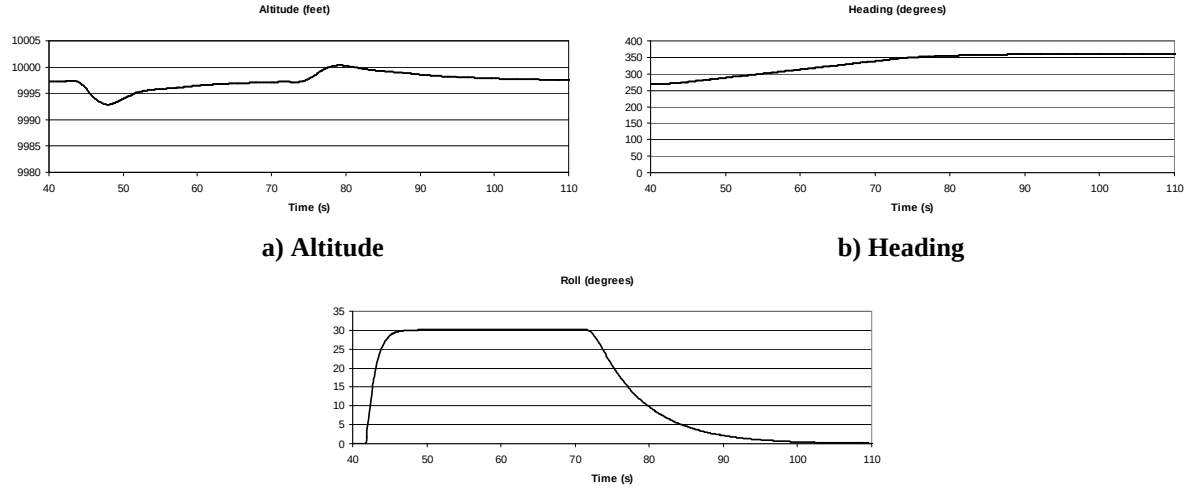
$$\Delta u = u_{cmd} - u - \dot{u}t_{spool} \tag{34}$$

The commanded throttle setting is shown below, where Δt is the simulation time step. This method ensures a smooth transition in throttle as u approaches u_{cmd}. The spool time, t_{spool} should be is always greater than 0 to prevent division by zero.

$$throttle = throttle + \frac{\Delta u \Delta t}{t_{spool}} \tag{35}$$

B. Navigation

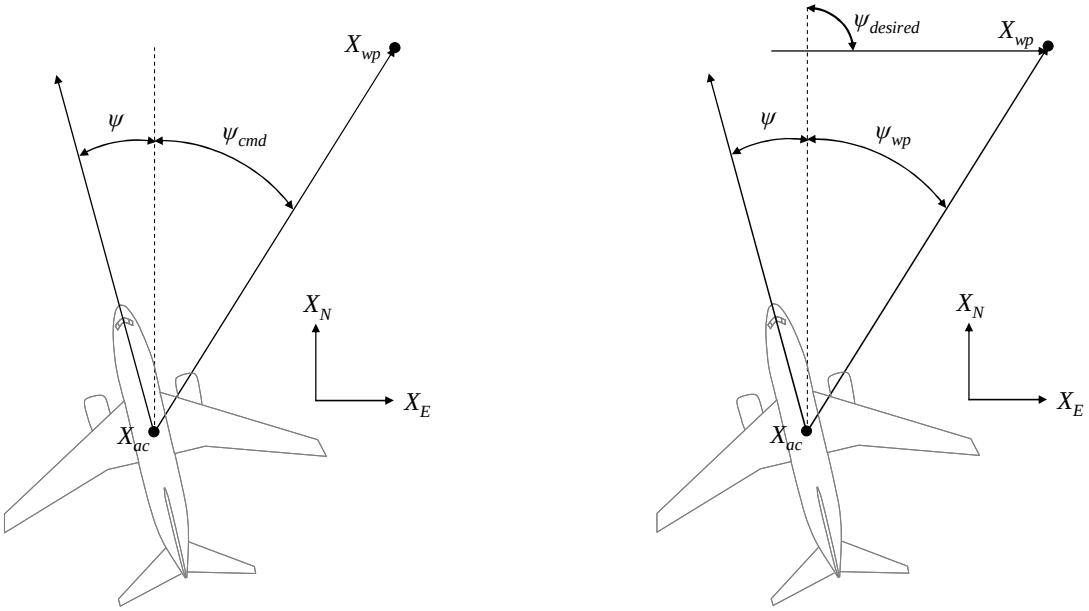
Waypoint navigation is often a requirement in simulations, especially those of UAVs or in simulations involving many entities. *SimpleFlight* implements a basic navigation computer in addition to its autopilot in order to statisfy the requirement for automatic flight. In addition to basic waypoint following, a system is presented which commands a specific heading to a waypoint. The following is a description of a basic navigational computer that will enable waypoint following when used in conjuction with a basic 3-axis autopilot such as that described in Section A.



a) Altitude **b) Heading**

c) Roll

Figure 4. Autopilot Performance. *An example of a heading change maneuver while maintaining altitude. Note the realistic transition in heading and roll. The altitude deviation is less than 10 feet with a 30° bank and 250 knots of airspeed.*



a) Direct to Waypoint **b) Heading Constraint**

Figure 5. Waypoint Following. *Diagrams showing how heading is calculated for waypoint following. In a) the commanded heading is the azimuth between the vehicle and the waypoint. In b) an additional constraint of ψ_{cmd} is added. In this case, the aircraft will intercept the waypoint at ψ=90°.*

1. Direct-to-Waypoint Flight

A system is described which allows an aircraft to fly to geo-referenced waypoints. Typically, a series of waypoints is combined in order to form a route which the vehicle follows. In this system, the aircraft is commanded to f ly directly to the waypoint, then turn toward the next waypoint once the current waypoint is over-flown. Altitude can be included in the waypoint definition. Commanded altitude is passed directly to the autopilot.

The azimuth to the waypoint is passed to the autopilot by the flight computer. The autopilot heading hold function determi nes the amount and direction of bank, as shown in Section A. Figure 5(a) shows this scenario. Note that the autopilot can use the aircraft heading, ψ, or use the ground track of the aircraft to determine the angle of the flight path. The latter would correct for wind as the aircraft flies toward the waypoint. This is shown in Eq. 36, where *x* and *y* are the linear distances in the East and North directions, respectively.

$$\psi_{cmd} = \psi_{wp} = \arctan\left(\frac{X_{E,wp} - X_{E,ac}}{X_{N,wp} - X_{N,ac}}\right) \tag{36}$$

Once the aircraft passes the assigned waypoint, the next waypoint is loaded by the flight computer. Passage of the waypoint is usually based on two conditions, the first being a given closeness to the waypoint, and the second being an aspect angle, Δψ, greater than 90°.

2. Heading Constraint

A vehicle may be required to fly along a given heading to a waypoint. Examples include flying to a VOR station along a specified radial, landing using ILS, or assigned to follow a specific path to a GPS waypoint. In these cases, an extra constraint is applied to the technique described previously. Using Fig. 5(b) and Eq. 37, one can find the commanded heading for this condition. As with waypoint following, ψ can be the actual heading of the aircraft, or based on the ground track.

$$\psi_{cmd} = 2\psi_{wp} - \psi - \psi_{desired} \quad \text{for} \quad \left|\psi_{wp} - \psi\right| < 90^{\circ} \tag{37}$$

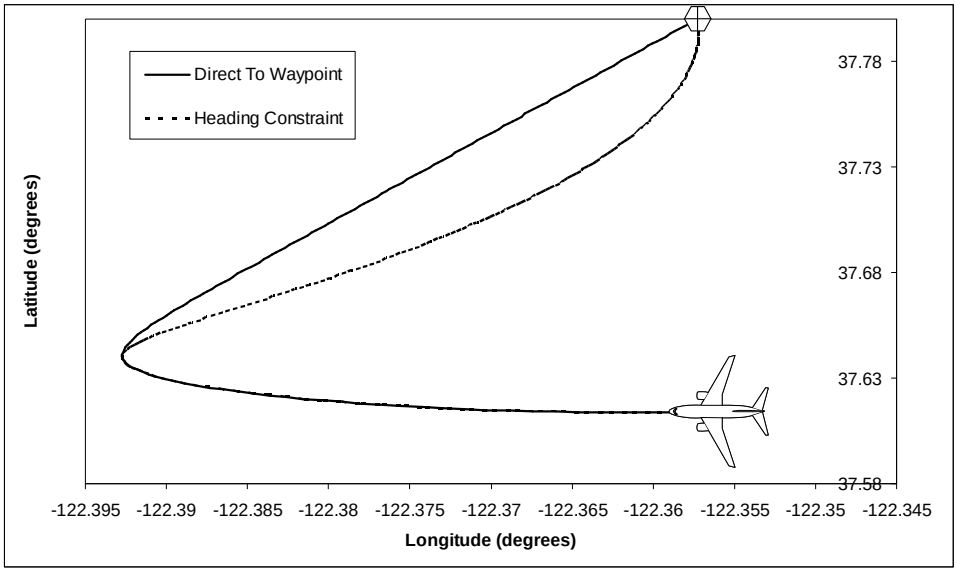


Figure 6. Waypoint Following Comparison. *The vehicle ground track (in geodetic coordinates) is shown. In the case of direct waypoint following, the aircraft intercepts the waypoint at a heading of approximately 45°. In the second case, the aircraft is commanded to intercept the waypoint at ψ=0.*

Equation 37 should only be used for aspect angles less than 90°. For larger angles, the algorithm causes the aircraft to turn in the opposite direction of the waypoint. When $\Delta\psi$ is greater than 90°, apply the method shown in Eq. 36. Figure 6 shows how a heading constraint changes the flight path when flying to a waypoint. The aircraft starts at a heading of 270°. Both methods follow the same path for a period of time. However, with a heading constraint of 0° applied to the waypoint, the second method adjusts the flight path to intercept the waypoint at the commanded heading.

VI. Implementation

As stated earlier in this paper, a goal of *SimpleFlight* was to construct a flight model using modern languages and techniques. Many flight models in use today are based on legacy computer languages and platforms. It is not uncommon to find flight models written in FORTRAN, utilizing custom format text files as input. *SimpleFlight*

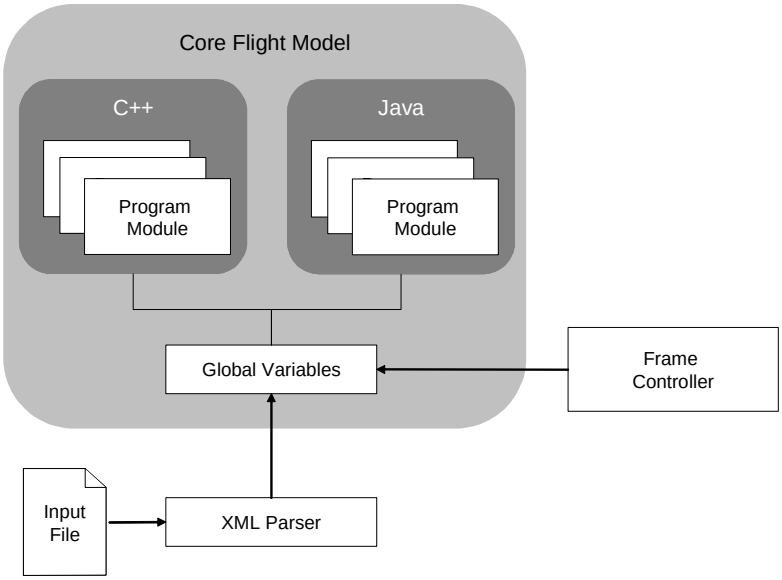


Figure 7. Program Structure. *SimpleFlight utilizes a simple module-based architecture combined with a common repository of data . C++ and Java modules interact through the "globals" repository.*

presents a complete framework for vehicle modeling that utilizes C++ and Java programming languages, and uses XML input for configuration. The design is object oriented, allowing users to create new models by extending existing models or base classes. Models of various aircraft systems are written as *modules*. Each module accesses a central repository of variables that are shared among all modules in the program. Figure 7 shows the structure of *SimpleFlight*.

A. Computer Languages

It may seem odd that *SimpleFlight* was written in two different computer languages, namely Java and C++. This approach was taken to allow for increased flexibility by developers in writing modules. C++ was chosen for its wide use in existing applications. There are often situations where multiple models are compiled or linked with a larger application framework. Many existing applications include extensive C++ interfaces for doing so. The core flight model, as well as several additional modules, were implemented in C++ for this reason. The flight model can be compiled as a single C++ library which is linked into other applications.

Java has shown a rapid increase in use in recent years, and modern Java runtimes show performance increases as well. Since a goal of *SimpleFlight* was to develop a flight model using modern languages, Java compatibility was an important consideration in its development. Java offers a host of features that are difficult to find in other languages. Java code is more resistant to inefficiencies in coding, due to its lack of pointers and automatic garbage collection. Java also has a very extensive API for development, including graphics, user interfaces, media, networking, and XML handling. All of these features in addition to its ubiquity make Java an attractive language for application development. Most of *SimpleFlight* development took place in the Java language. Once modules were developed and tested in Java, many modules were ported to C++ for the reasons stated above.

SimpleFlight connects Java and C++ to a single repository of global variables (see Fig. 7) using the Java Native Interface (JNI)[12]. The details of the JNI system and its implementation in *SimpleFlight* is beyond the scope of this paper.

```
<?xml version="1.0" encoding="UTF-8"?>

<SimpleFlight>

    <Modules Rate="60">
        <Module Class="EOMFiveDOF"/>
        <Module Class="InverseDesign"/>
        <Module Class="SimpleAutopilot"/>
        <Module Class="simpleflight.io.FGInterface" Rate="30" />
    </Modules>

    <Name>ExampleAircraft</Name>
    <Version>1.0</Version>

    <InitialConditions>
        <Position Latitude="37" Longitude="-122" Altitude="10000"/>
        <Orientation Heading="270.0" Pitch="0.0" Roll="0.0"/>
        <Airspeed>250</Airspeed>
        <Throttle>0.8</Throttle>
        <Weight>140000</Weight>
        <Fuel>20000</Fuel>
    </InitialConditions>

    <AutoPilot>
        <Component Type="HeadingHold" MaxBank="30" MaxBankRate="30" BankWeight="0.05" />
        <Component Type="VSHold" MaxG="1.5" MinG="0.5" MaxPitchUp="15" MaxPitchDown="-10" />
        <Component Type="AltitudeHold" MaxVS="2000" AltWeight="0.2" />
        <Component Type="AutoThrottle" MaxThrottle="1.0" MinThrottle="0.01" SpoolTime="10" />
    </AutoPilot>

    <Design>
        <Engine>
            <Type>Simple</Type>
            <StaticThrust>40000</StaticThrust>
            <ThrustAngle>0</ThrustAngle>
        </Engine>

        <FlightConditions WingArea="1000" WingSpan="90" Weight="140000">
            <FlightCondition Altitude="35000" Pitch="0" Throttle="0.85" VS="0" Airspeed="480"/>
            <FlightCondition Altitude="0" Pitch="10" Throttle="1" VS="0" Airspeed="135"/>
        </FlightConditions>
    </Design>

</SimpleFlight>
```

Figure 8. XML Example. *SimpleFlight uses XML as its default configuration language. However, developers are free to use other methods to configure modules and global variables.*

B. Timing

Referring to Fig. 7, a frame controller is used in conjunction with the flight model. This is shown outside of the flight model in *SimpleFlight.* The flight model container includes functions to step the model according to an externally specified time step. This allows users to use any external timing source to update *SimpleFlight*. The model passes the change in time through the list of modules. A Java-based frame controller was developed for *SimpleFlight.*

C. Configuration

Configuration of *SimpleFlight* is a process of loading initial states into the global variable repository and passing any initialization instructions and data through each module. XML was chosen as the input language due to its wide-spread acceptance as well as its ability to describe data in a tree-like manner. An example of an XML input file is shown in Fig. 8. The XML configuration file is loaded at runtime. A section of the file describes which modules should be loaded into the flight model. After the modules are loaded, the XML file is passed to each module for initialization. Users can add custom nodes to the XML file to add additional configuration data. Although XML was chosen as the default input language, all of the global variables can be accessed directly, so the model can be configured using methods other than the XML input file.

VII. Application of *SimpleFlight*

A. Examples of Modules

As stated previously, *SimpleFlight* is designed to be a variable fidelity flight dynamics model. As such, there are many possible applications of *SimpleFlight*. However, a key motivation behind *SimpleFlight* was the desire to create a set of simplified modules for users who have a small set of performance parameters for creating a vehicle model. A list of several key modules and their description follow.

1) **Five-DOF motion**. This is the core equations of motion module. Motion is modeled as six degrees of freedom in the world axis, with five degrees of freedom in the body axis. Section III describes the implementation of this model by forcing zero sideslip.
2) **Inverse design**. This provides basic aerodynamics using the methods shown in Section IV. Basic models for different engine types are also included.
3) **Simple auto pilot**. This is an autopilot based on Section V.
4) **Waypoint follower.** This is a simple waypoint follower routine that ties to the autopilot module. It is described in Section V.B.
5) **Atmosphere.** A module that models the pressure, density, and temperature of the 1976 standard atmosphere.

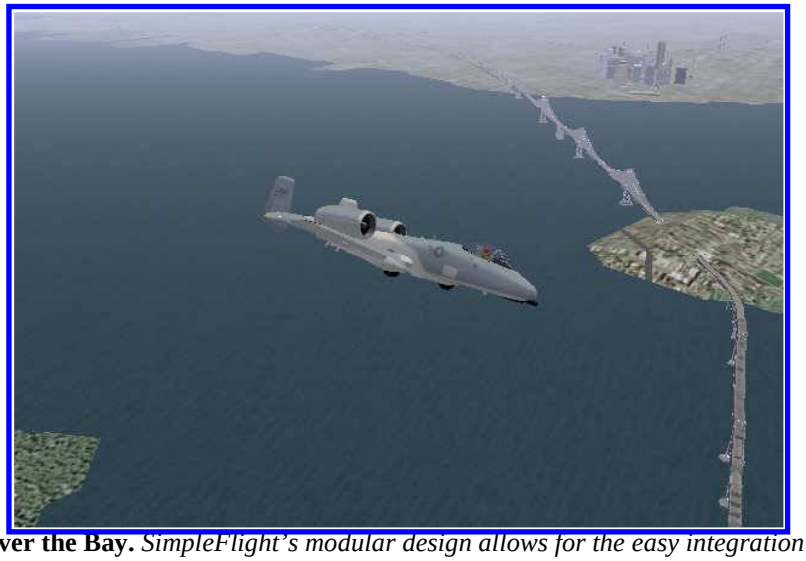


Figure 9. A-10 Over the Bay. *SimpleFlight's modular design allows for the easy integration of interface code. Here, The Flightgear flight simulator is receiving state updates over Ethernet to set the position and orientation of an A-10 visual model.*

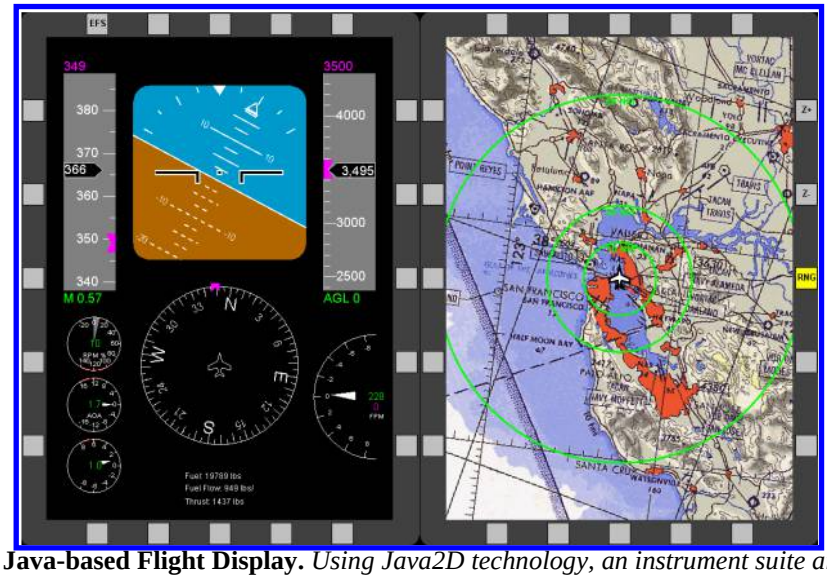


Figure 10. Java-based Flight Display. *Using Java2D technology, an instrument suite and moving map display were created for SimpleFlight. The ease of use and extensive API in Java allows for detailed instrumentation to be easily created.*

6) **Flightgear interface.** This provides an interface to the *Flightgear* flight simulator. (See Fig. 9)
7) **Instrument panel.** A module that provides aircraft instrumentation using Java 2D. (See Fig. 10)

B. Example Applications

1. Design Trade space

In two separate studies, constructive simulations were created to examine the performance of different aircraft designs performing the same mission. Each aircraft varied significantly in weight, speed, and range. *SimpleFlight*, loaded with the modules listed above, was used as the flight model in the studies. In both studies, a stable controllable vehicle was assumed, but the speed and range performance of the vehicle was critical to the outcome of the mission. Using *SimpleFlight*, engineers were able to model the various aircraft at an appropriate level of fidelity for the experiments.

2. Small UAV performance in a wind field

To examine the difference in flight path following and sensor viewing in a wind field, a small UAV was modeled in *SimpleFlight* and tested in an virtual simulation environment. Researchers were interested in varying the cruise speed of the UAV as it flew through a turbulent wind field. *SimpleFlight* allowed a the rapid integration of the wind field data by writing a module that loaded wind speed, direction into the *SimpleFlight* program. Also, by using the design techniques described earlier in this paper, vehicle performance parameters were easily changed to allow for different design cruise speeds while maintaining stable performance.

3. Multi-crew aircraft virtual simulation

A operator-in-the-loop simulator was constructed by AFRL for the purpose of immersive virtual simulation for multi-crew operations. The flight simulator features side-by-side seating with wide field-of-view collimated visuals. *SimpleFlight* was used as a preliminary model for demonstrations due to its easy configuration and flight stability. Visitors to the simulator could fly the aircraft using stick and rudder inputs. A *SimpleFlight* module translated inputs into stable body rotation rates that provided realistic aircraft dynamics for a wide range of flight conditions.

C. Other Applications and Future Work

Other experimental modules have been created for testing capabilities beyond those available in the modules listed earlier in this section. For example, higher fidelity aerodynamics have been modeled using table-based input files. The aerodynamics were combined with mass properties for an existing aircraft to model rotational acceleration. An autopilot system was developed based on PID controllers, connecting control surfaces to autopilot control. Currently, a helicopter model is under test. Like the fixed wing version, the helicopter model is intended to be stable and realistic, while requiring a minimum of input data.

VIII. Caveats

As with any aircraft dynamics model, *SimpleFlight* makes many assumptions in order to be useable and computationally efficient. Models were developed to satisfy the need for an easily configurable flight model, but despite its appearance of realistic 6-DOF motion, *SimpleFlight* is limited by its models and data. While real-world aircraft can be used as the basis for the creation of a *SimpleFlight* model, care should be taken in the application of *SimpleFlight* to any particular purpose. Making decisions based on any simulation model, without careful consideration of modeling assumptions, can lead to false conclusions.

IX. Conclusion

SimpleFlight provides an object-oriented modular architecture for the development of custom flight models. This framework can provide the opportunity for engineers to modify and expand flight model capabilities. Using a combination of modeling fidelity can better focus research and development. Using a flight model that allows the inclusion of several languages also reduces time and cost. Java and C++ have shown to be efficient, reliable languages to use in the development of *SimpleFlight*.

A set of modules were developed in order to allow users to model realistic aircraft dynamics with a minimum of input data. These modules have been used successfully in simulation studies and demonstrations, using both constructive and virtual simulation environments. *SimpleFlight* has shown that it is possible to create representative aircraft models using a minimum of input data. When used appropriately, this can help save countless hours and greatly reduce cost, allowing engineers to focus on other areas of research and development. *SimpleFlight* is a work-in-progress, but early versions have been used successfully in AFRL simulations. It is envisioned that *SimpleFlight* will be expanded and maintained for use in future simulations.

Acknowledgments

The author would like to acknowledge Mr. Dan Caudill of the Air Force Research Laboratory and Mr. Chris Linhardt of General Dynamics Corporation. Their requirements for a simpler, reconfigurable flight model inspired the development of *SimpleFlight*. The author would also like to thank Mr. Dave O'Quinn of General Dynamics Corporation, who worked as an integrator and beta-tester for *SimpleFlight*.

References

[1]Nelson, R. C., *Flight Stability and Automatic Control*, 2nd Ed, McGraw-Hill, 1998, pp. 96-130, 398-399

[2]Persing, T. R., Dube, T., and Slutz., G. J., "Impact of Aircraft Flight Dynamics Modeling Technique on Weapon System Beyond-Visual-Range Combat Effectiveness." *AIAA Modeling and Simulation Technologies Conference and Exhibit*, Austin, TX, August 2003. AIAA 2003-5689.

[3]Zyskowski, M. K., "Aircraft Simulation Techniques Used in Low-cost, Commercial Software." *AIAA Modeling and Simulation Technologies Conference and Exhibit*, Austin, TX, August 2003. AIAA 2003-5818.

[4]Berndt, J. S., "JSBsim: An Open Source Flight Dynamics Model in C++." *AIAA Modeling and Simulation Technologies Conference and Exhibit*, Providence, RI, August 2004. AIAA 2004-4923.

[5]Unmanned Dynamics, LLC. Aerosim Blockset Version 1.2 User's Guide, URL: http://www.u-dynamics.com [cited: 1 Jun 2007].

[6]Stevens B. L. and Lewis F. L. *Aircraft Control and Simulation*, John Wiley & Sons, Inc., 1992.

[7]Rogers R. M. *Applied Mathematics in Integrated Navigation Systems*, American Institute of Aeronautics and Astronautics, Inc., 2000.

[8]Anderson, J. D., *Aircraft Performance and Design*. McGraw-Hill, 1999, pp. 150-175.

[9]Tangler, J. and Kocurek, J. "Wind Turbine Post-stall Airfoil Performance Characteristics Guidelines for Blade-element Momentum Methods". Technical Report NREL/CP-500-36900, National Renewable Energy Laboratory, Oct. 2004.

[10]Viterna, L .A. and Janetzke, D. C., "Theoretical and Experimental Power from Large Horizontal-axis Wind Turbines." Technical Report TM-82944, NASA, Sept. 1982.

[11]Viterna, L .A. and Corrigan, R. D. "Fixed Pitch Rotor Performance of Large Horizontal Axis Wind Turbines." In DOE/NASA Workshop on Large Horizontal Axis Wind Turbines, Cleveland, Ohio, July 1981.

[12]Liang, S., *Java™ Native Interface: Programmer's Guide and Specification*. Prentice Hall, 1999.