Verification Challenges in F-16 Ground Collision Avoidance and Other Automated Maneuvers

Peter Heidlauf[1], Alexander Collins[1], Michael Bolender[1], and Stanley Bak[2]

[1] Air Force Research Laboratory, Aerospace Systems Directorate
{peter.heidlauf.1, alexander.collins.3, michael.bolender}@us.af.mil
[2] Safe Sky Analytics
stanleybak@gmail.com

Abstract

Benchmark Proposal: The F-16 Fighting Falcon is a highly-maneuverable aircraft in production since the 1970s. Since then, several studies and books have investigated the aircraft's performance and created simulation models. In this paper, we present some of these models as a verification challenge, providing MATLAB and Python code to simulate an F-16 performing ground collision avoidance as well as other autonomous maneuvers. The aircraft model and inner-loop controller has 16 continuous variables with piecewise nonlinear differential equations. Autonomous maneuvers are performed by an outer-loop controller using finite-state machines with guards involving the continuous variables. Pass-fail specifications are provided based on the aircraft flight limits and boundaries of the model. This model aims to be a starting point for analyzing detailed behaviors of aerospace systems.

1 Introduction

Continuous and hybrid system verification models strive to analyze safety-critical systems governed by predictable continuous dynamics and possibly discrete logic. Aircraft dynamics and performance are prime examples for verification as they are clearly safety-critical and can be modeled with ordinary differential equations. Verification involving aircraft motion [8, 5], however, has often been done on very simplified models which, for example, assume aircraft are point masses or follow Dubins paths [1].

This benchmark presents simulation code for the next level of complexity for aircraft models, based on the well-studied F-16 system. The aerospace engineering models we present here can simulate aircraft behavior in 3D space, dealing with the position, velocity and orientation of the aircraft.

DISTRIBUTION A: Approved for public release; distribution unlimited. Case Number: 2018-0257 (original case number(s): MSC/PA-2018-0056; 88ABW-2018-1310), completed on 18 Apr 2018.

We provide both MATLAB and Python versions of the simulation code on github[1]. The code simulates different scenarios, providing visualizations as well as checking specifications at each time step.

We first present the plant and inner-loop controller models in Section 2. Then, Section 3 presents some outer-loop autonomous controllers and associated safety specifications for various aircraft maneuvers, including automated ground-collision avoidance.

2 F-16 and Inner-Loop Controller Model

Studies as early as the 1970s characterized the performance of the F-16 aircraft [4]. The basis of the benchmark we provide is a standard model used in aerospace engineering and described extensively in a textbook [7]. A shortened and accessible introduction which uses the same base model is also available [6]. Here, we provide the basic details, and strongly recommend the interested reader access the reference material for additional model information.

We model an F-16 aircraft with 6 degrees of freedom (DoF) nonlinear equations of motion. Three equations each are used to model the forces, kinematics, moments, and position of the aircraft, for a total of 12 governing equations. To capture the dynamics of the F-16 turbojet engine, a 13th equation models the thrust lag state.

Three variables define the aircraft position in a global coordinate system. We assume a flat-earth model, thus the position variables are northward displacement `pn`, eastward displacement `pe`, and altitude `alt`. This is referred to as the north-east-down, or NED reference frame. This reference frame is commonly used in aircraft dynamics to ensure right-hand orthogonality with the aircraft nose, right wing, and landing gear defining the body-frame axes. The altitude variable is inverted for ease of use.

The velocity of the aircraft is defined by the air speed `Vt`, angle of attack α, and angle of sideslip β. See Figure 1 for a visualization of these parameters.

Three more variables, Euler angles, define the orientation of the aircraft. These Euler angles are ϕ, θ, and ψ, the roll, pitch, and yaw angles of the aircraft respectively. An additional three variables, `P`, `Q`, and `R`, define the angular roll rates of the aircraft about the Euler angles.

An aircraft has four standard inputs: one for the engine throttle command δ_t, and three for the main control surfaces, the ailerons δ_a, the elevator δ_e, and the rudder δ_r. Each of the control surfaces can be used individually to provide moments along a different axis of the aircraft, as shown in Figure 2. In modern aircraft, however, the pilot does not typically provide commands directly to the control surfaces. Instead the pilot controls the aircraft in a "fly-by-wire" manner [2], actuating the control surfaces indirectly through an inner-loop controller.

The F-16 was designed to be slightly aerodynamically unstable in order to improve maneuverability. This means that if left uncontrolled in a non-trimmed state, the aircraft will experience increasing oscillations. To alleviate pilot exhaustion and simplify control of the aircraft, an inner-loop controller uses two decoupled Linear-Quadratic Regulators (LQR). Integral tracking control is applied to the upward acceleration `Nz`, the stability roll rate `ps`, and sum of the side acceleration and yaw rate `Ny` + `R`. These three terms are commanded to produce the desired maneuvers.

The inner-loop controller is created by linearizing the nonlinear F-16 model about a selected steady-state trim point. The trim conditions for both the aircraft state and controls are saved for later use. The longitudinal and lateral modes are then decoupled into two linear models, as the cross-coupling terms are negligible, and the process of tuning the regulators is simplified. The

[1]The MATLAB version of the benchmark code is available at https://github.com/pheidlauf/AeroBenchVV and the Python version is at https://github.com/stanleybak/AeroBenchVVPython.

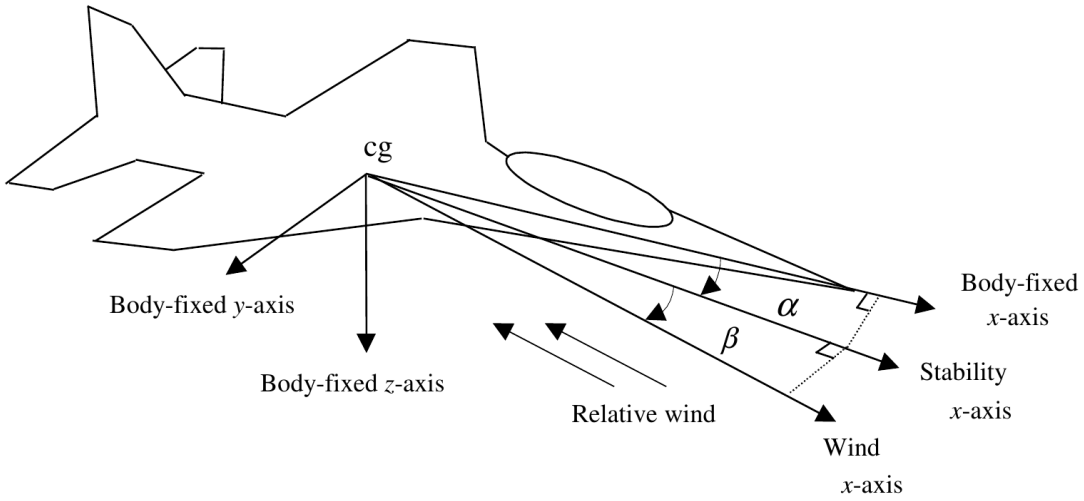

Figure 1: The velocity variables are given in the body frame, and are defined by the air speed `Vt`, angle of attack α, and angle of sideslip β. Image was taken from Seto [6].

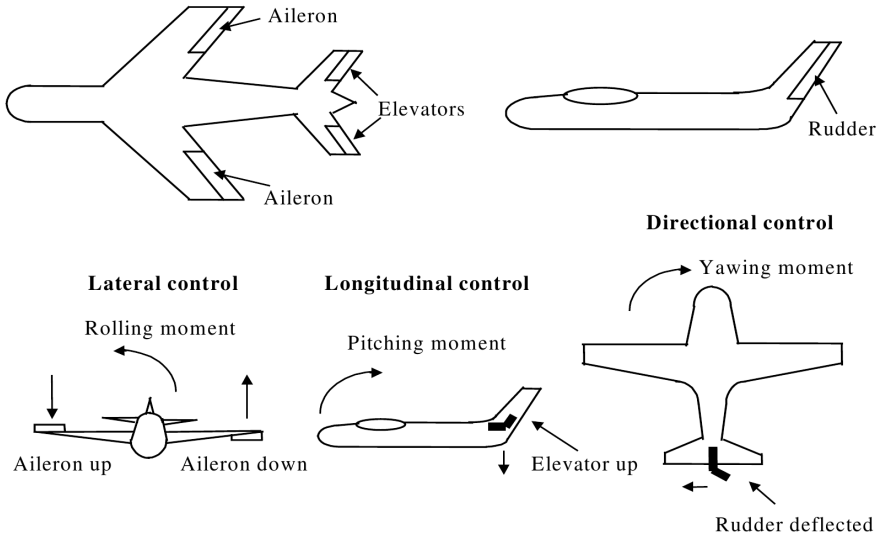

Figure 2: The F-16 has four main inputs: the ailerons, elevator, and rudder, and the throttle command to the engine. Image was taken from Seto [6].

decoupled longitudinal mode consists of the states α and `Q`, the elevator input, and the outputs α, `Q`, and `Nz`. To determine the LQR gain matrix, appropriate weighting matrices for state error, Q, and control effort, R, are selected and tuned to achieve a desired frequency response with a loop transfer gain crossover frequency of 10 rad/s. These desired response characteristics are typically expected by F-16 pilots. The throttle, δ_t, is left under manual control of the pilot. The decoupled lateral mode consists of the states β, `P`, and `R`, the aileron and rudder inputs, and the outputs β, `Q`, `R`, `ps`, and `Ny + R`. Again, the LQR weighting matrices Q and R are

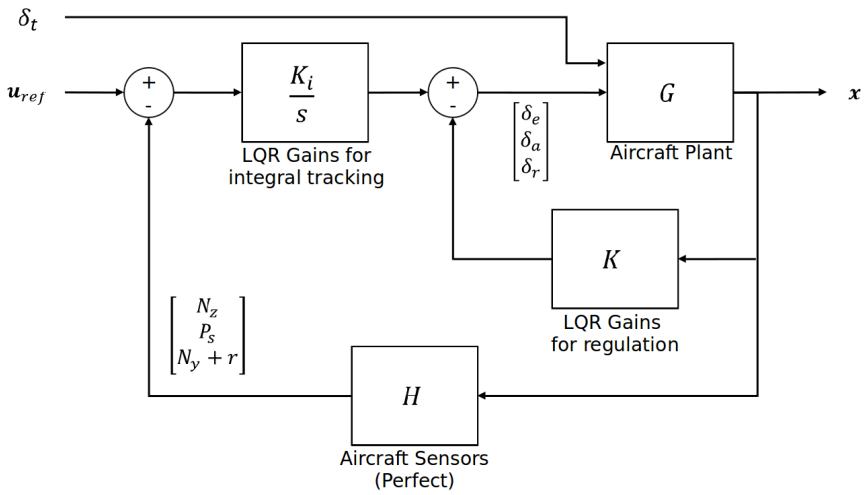

Figure 3: The inner-loop controller takes in throttle setting δ_t and reference input values u_{ref} (`Nz`, `ps`, and `Ny + R`), and produces actuation commands for the individual control surfaces.

tuned to achieve desired frequency response with a loop transfer gain crossover frequency of 10 rad/s. In the lateral mode, both `Ps` and `Ny + R` are controlled. In all standard maneuvers, the `Ny + R` term is regulated to zero, as combinations of side acceleration and yaw rate are undesirable by pilots. A block overview of the controller is given in Figure 3. The tuning code is provided as part of the benchmark, in `BuildLqrControllers.py` in the Python repository and `BuildLateralLqrCtrl.m` and `BuildLongitudinalLqrCtrl.m` in the Matlab version.

An improved response is possible by repeating the linearization and LQR design process at multiple setpoints, and then interpolating the gain matrices at runtime based on the current aircraft state. This is called gain scheduling, and is typically done based on the ratio of the static and dynamic pressure of the aircraft, or the ratio of the airspeed and the altitude. To simplify the benchmark controller, we did not implement gain scheduling.

The inner-loop controller, which takes in the commands for `Nz`, `ps`, and `Ny + R`, uses an integrator for each to eliminate steady-state error. These integrators add three more states to the aircraft model, bringing the total number of variables to 16. A summary of the variables, in the order they are used in the benchmark code, is given in Table 1.

The plant dynamics are defined by considering the forces and moments based on the aircraft orientation and flight environment. Rather than using a first-principles design which would depend on intricate details of the shape of the surfaces of the aircraft, aerospace engineers often instead build an aircraft (or a scaled model of an aircraft), and then use a wind tunnel to measure the lift, drag, and various moments experienced for different angles of attack α and sideslip β. To simulate the system, this data is used to create look-up tables that are interpolated at runtime. This sort of setup makes the right-hand side of the differential equations non-smooth. Our benchmark code can use either this look-up table version to compute the aerodynamic coefficients (the `Stevens` model [7]), or a polynomial fit computed from the look-up table data (the `Morelli` model [3]).

Table 1: State Variables in the Aircraft Model and Inner-Loop Controller

Variable	Symbol	Meaning
x[0]	`Vt`	Air Speed
x[1]	α	Angle of Attack
x[2]	β	Angle of Sideslip
x[3]	ϕ	Roll
x[4]	θ	Pitch
x[5]	ψ	Yaw
x[6]	`P`	Roll Rate
x[7]	`Q`	Pitch Rate
x[8]	`R`	Yaw Rate
x[9]	`Pn`	Northward Displacement
x[10]	`Pe`	Eastward Displacement
x[11]	`alt`	Altitude
x[12]	`pow`	Engine Power Lag
x[13]	`Nz` (integrator)	Upward Accel
x[14]	`Ps` (integrator)	Stability Roll Rate
x[15]	`Ny + r` (integrator)	Side Accel and Yaw Rate

There are other sources of complexity in the dynamics that make the ODEs non-smooth. For example, the engine performance is a function of the dynamic pressure and the Mach number, which depends on the density of the air, the speed of the aircraft, and the speed of sound. The speed of sound depends on the temperature, which will depend on the weather as well as the altitude. The relationship between temperature and altitude is linear up to the stratosphere (about 36000 feet), where it basically becomes constant. Further, the engine power is also qualitatively different depending on whether the throttle is below or above 70%, which is when the afterburner becomes active. Finally, the engine model also uses look-up tables derived from empirical data as part of its power calculation, which is likely simpler than a first-principles model of an aircraft engine.

The aircraft state is most easily updated in the body frame, but the position needed for verification is given in terms north-east-down coordinates. Sine and cosine therefore need to be used in the dynamics in order to convert between the body frame and the north-east-down frame. This could be made more complicated without assuming a flat-earth model, using an earth-centered inertial (ECI) frame, which would require a further conversion. For simplicity, we do not do this in the benchmark code. Portions of the model use degrees, so conversions to radians are sometimes needed as well.

To further complicate matters, the F-16 is an American aircraft, so its design, control, and simulation is most often done in imperial units. Temperature in the model, for example, is measured in degrees Rankine, an absolute scale for Fahrenheit comparable to the Kelvin-Celsius relationship.

Table 2: Verification Cases for Engine Controller

Case	Initial Altitude `alt` (ft)	Initial Speed `Vt` (ft/sec)	Setpoint `sp` (ft/sec)
1A	10000	1000	[1200, 1220]
1B	10000	1100	[1200, 1220]
1C	10000	[1000, 1100]	[1200, 1220]
1D	[10000, 11000]	[1000, 1100]	[1200, 1220]
1E	20000	1000	[2200, 2220]
1F	40000	1000	[2200, 2220]
1G	[20000, 40000]	1000	[2200, 2220]
1H	[5000, 30000]	[1000, 2000]	[1000, 2000]

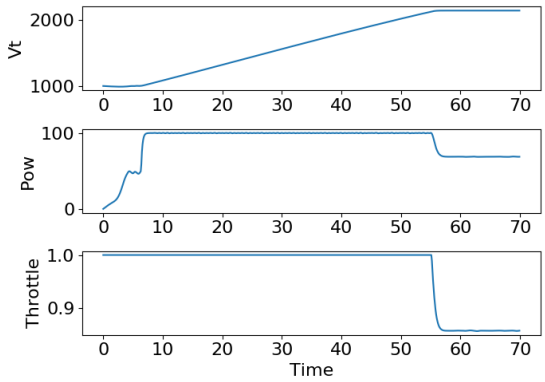

Figure 4: A plot of the simulation from Case E of the the engine control benchmark meets the settling time specification.

3 Outer-Loop Control Automation and Specifications

We now present three categories of verification problems that use the F-16 plant and inner-loop controller model. For each specification, it could be considered using either the look-up table version of the dynamics (`Stevens`), or the polynomial interpolation version (`Morelli`).

3.1 Easy: Engine Control (2 Variables)

The first benchmark examines the engine performance under fixed conditions. The model uses only two variables: the air speed `Vt`, and the engine power lag state `pow`, with all other variables fixed to constants (which may be parameters). Proportional control is done on the air speed `Vt` to drive it to a setpoint, using the throttle input. We consider a settling-time specification for this system: can you achieve within 5% of the desired setpoint within 60 seconds? The specification can be considered for various initial altitudes `alt`, speeds `Vt`, and setpoints `sp`, as shown in Table 2.

Although simulations can likely detect violations of some of the specifications, since the model is nonlinear, sampling values within the ranges is likely insufficient to guarantee the absence of violations without further reasoning.

Table 3: Verification Cases for Longitudinal Dynamics

Case	Initial States
2A-2H	Same as 1A-1H using the airspeed controller and specs
2I	`Vt` = 560, `alt` = [550, 560], `sp`=500
2J	`Vt` = 560, `alt` = [590, 600], `sp`=[500, 505]
2K	`Vt` = [560, 600], `alt` = [590, 600], `sp`=[500, 505]
2L	`Vt` = [560, 500], `alt` = [600, 500], `sp`=[700, 800]
2M-2P	Same as 2I-2L with $\alpha = [-0.1, 0.1]$, $\theta = \alpha$, `Q` = [-0.1, 0.1]

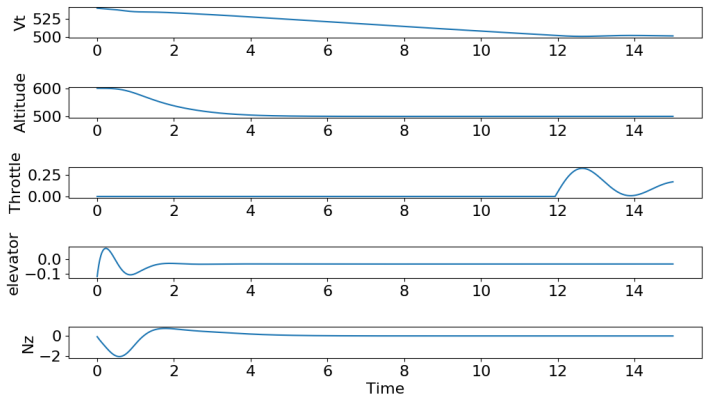

Figure 5: A simulation of the longitudinal model violates the g-limit specification (`Nz` goes below -2 at time 0.5).

This problem can be simulated and plotted using `check_engine.py` within the benchmark code. An output of running the script for a value in Case E is shown in Figure 4. Running the script in the case produces the output in the terminal: `Simulation Conditions Passed: True`.

3.2 Medium: Longitudinal Control (7 Variables)

The longitudinal dynamics of an aircraft model the up-down movement without the side-to-side dynamics (see Figure 2). Thus, in this model, we assume the aircraft is flying straight without any roll or yaw. The inputs considered are the engine throttle and the elevator. This model uses 7 state variables: the air speed `Vt`, angle of attack α, pitch angle θ, pitch rate `Q`, altitude `alt`, engine power lag `pow`, and the `Nz` integrator.

The main specification we check is that the g-force experienced by the pilot remains within the range $[-2, 9]$. If the aircraft experiences g-forces outside of this range, structural integrity may be compromised, requiring grounding of the aircraft for extensive inspections and servicing. Additionally, if the upper threshold of g-force is exceeded for more than a short time, blood can rush away from the pilot's brain causing greyout (loss of color vision), tunnel vision (loss of peripheral vision), blackout (complete loss of vision), and eventually G-LOC (temporary loss of consciousness). On the other end, large negative g-forces are extremely uncomfortable for pilots (imagine the feeling of descending on a commercial flight) and can cause excessive

Table 4: Verification Cases for GCAS system

Case	Initial States
3A-3P	Same as 2A-2P using the 16-dimensional model
3Q	`alt` = [3600, 3700], `xcg` ±5%
3R	Same as 3Q with $\phi = [0, \frac{\pi}{4}]$
3S	Same as 3R with $\theta = [-\frac{3\pi}{5}, -\frac{2\pi}{5}]$
3T	Same as 3S with `xcg` ±25%
3U	`xcg` ±5%, `cxt`, `cyt`, `czt`, `clt`, `cmt`, `cnt` ±40%
3V	Same as 3U and 3R
3W	Same as 3U and 3S
3X	`cxt`, `cyt`, `czt`, `clt`, `cmt`, `cnt` ±40%
3Y	`cxt`, `cyt`, `czt`, `clt`, `cmt`, `cnt` ±45%
3Z	`clt`, `cmt`, `cnt` ±55%

blood pressure in the brain and eyes, leading to redout (blood entering the field of vision). The asymmetric range on safe g-forces is the reason why, when F-16 pilots want to reduce their altitude, they prefer to roll the aircraft upside down and pull back, experiencing positive g-forces, rather than directly pitching the nose downward, which would cause a negative g-force.

The outer-loop logic we use for this model is a simple altitude hold controller that tries to guide the aircraft to a specific altitude. If the difference between the initial altitude and target altitude is too large, the g-limit can be exceeded. An example of such a violation, produced by the `check_longitudinal.py` simulation code, is shown in Figure 5.

The longitudinal model can also be used to check the earlier engine specifications, now with the additional dynamics that will account for altitude changes. A table of specific verification cases for this model is given in Table 3. Unless specified, initially all of the other seven state variables in the model are 0.

3.3 Hard: Ground-Collision Avoidance (16 Variables)

A recent upgrade to modern F-16s is an automated ground collision avoidance system (GCAS). The system was developed by Lockheed Martin, NASA, and the Air Force Research Laboratory[2]. As of the start of 2018, the system has been confirmed as saving six aircraft (at least $25 million each) and seven lives.

The GCAS system detects when a ground collision is imminent and performs a recovery maneuver. This can happen for a number of reasons. As mentioned before, due to the high maneuverability of F-16s, during high-speed turns the g-forces can cause blood to rush away from a pilot's head and cause the pilot to pass out (G-LOC). If the aircraft happens to turn towards the ground when the pilot is passed out, a ground collision may result. A first-person video of this situation happening during a training mission, and the GCAS system saving the pilot's life, has recently been declassified[3].

[2]This academic benchmark code and recovery logic are in no way connected to the code or methodology used for the real GCAS system.

[3]http://aviationweek.com/air-combat-safety/auto-gcas-saves-unconscious-f-16-pilot-declassified-usaf-footage

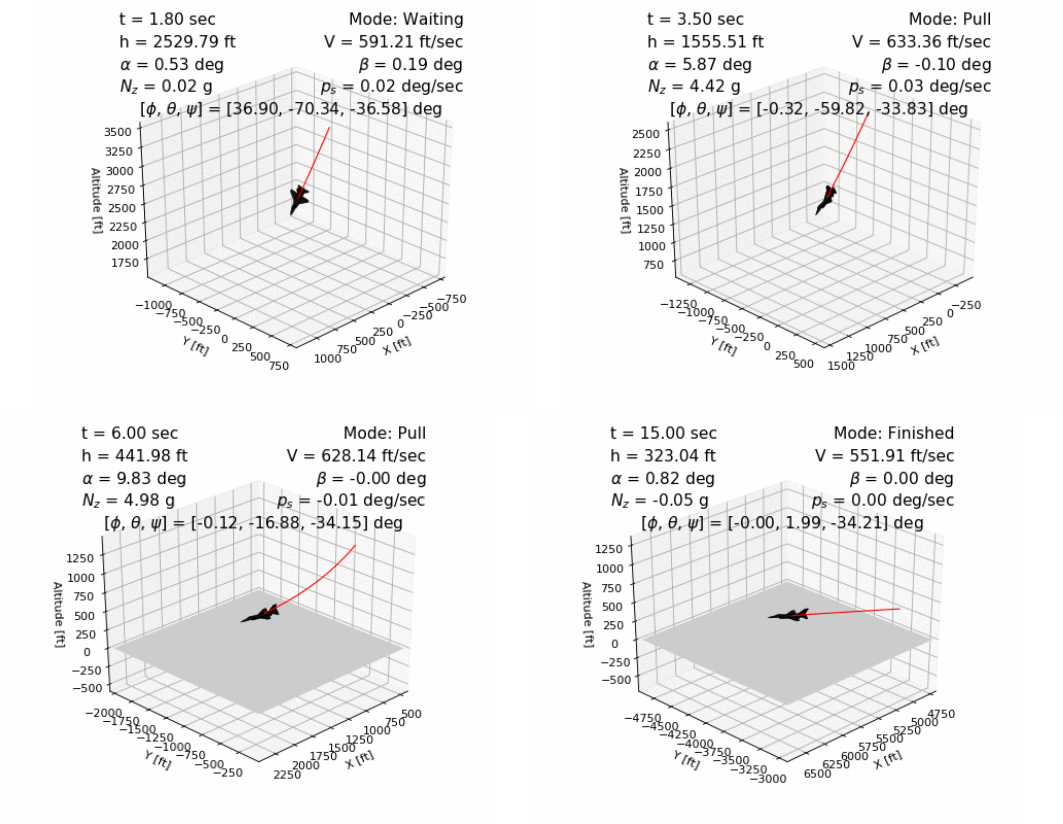

Figure 6: The `check_gcas.py` script can produce an animation of the GCAS recovery maneuver.

A simplified version of the GCAS recovery maneuver can be modeled as a state machine. The aircraft first rolls until the wings are level, then begins a 5-G pull up until the nose is above the horizon, at which point control is returned to the pilot. In the benchmark, we also have a short time delay before the maneuver begins. The final verification problem uses the full 16-variable model (all the variables listed in Table 1) to check if a recovery can be guaranteed.

The main specification is that the aircraft stays above the ground (in this case `alt` ≥ 0), although other limits on variables check that the aircraft model remains valid. Further, as before, the full model can also be used for checking the earlier specifications. For cases 3A-3P, assume unspecified state variables are initially zero. For 3Q and later, unless specified, use the initial states `pow` $= 9$, $\alpha = 2.1215$ degrees, `alt` $= 3600$, `Vt` $= 540$, $\phi = \frac{\pi}{4}$, $\theta = -\frac{2\pi}{5}$, $\psi = -\frac{\pi}{4}$. Some cases consider variations on the model. In particular we consider variations on the x position of the center of gravity `xcg`, which is normally 0.35 in the model. We also consider modifying the aerodynamic force and moment coefficients, `cxt`, `cyt`, `czt`, `clt`, `cmt`, `cnt`, since those elements are approximations computed from interpolating look-up tables and may vary depending on the specific payload being carried by the aircraft. In this code, these adjustments are made in `subf16_model.py`, although the multipliers are exposed in the top-level script, `check_gcas.py`. The specific test cases are given in Table 4.

An additional challenge may be to determine the boundary of when the recovery is successful, as a function of the 16 state variables. This boundary could be used at runtime to initiate the recovery logic.

The `check_gcas.py` script can be used to simulate the GCAS system and check the specifications. It can also produce an animation of the maneuver. Several frames of the animation are shown in Figure 6, while the full animation is viewable on the benchmark's github page.

4 Conclusion

We have presented three benchmark scenarios of increasing complexity dealing with various aspects of control of an F-16 aircraft. For each benchmark, several sets of initial states were provided, and each benchmark can be run with either the `Stevens` or `Morelli` version of the F-16 dynamics. The code to simulate the benchmarks is available both in MATLAB and Python.

Some possible enhancements to the model would be to increase fidelity by adding sensor and actuator models. Further, beyond the scenarios and specifications in mentioned in this paper, the code is structured so that it can be easily modified to simulate more complex autonomous maneuvers, or duplicated to check air-to-air collision avoidance, multi-aircraft coordination or swarm control algorithms.

References

[1] L. E. Dubins. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of mathematics*, 79(3):497–516, 1957.

[2] C. Favre. Fly-by-wire for commercial aircraft: the airbus experience. *International Journal of Control*, 59(1):139–157, 1994.

[3] E. A. Morelli. Global nonlinear parametric modelling with application to f-16 aerodynamics. In *American Control Conference, 1998. Proceedings of the 1998*, volume 2, pages 997–1001. IEEE, 1998.

[4] L. T. Nguyen. *Simulator study of stall/post-stall characteristics of a fighter airplane with relaxed longitudinal static stability*, volume 12854. National Aeronautics and Space Administration, 1979.

[5] A. Platzer and E. M. Clarke. Formal verification of curved flight collision avoidance maneuvers: A case study. In *International Symposium on Formal Methods*, pages 547–562. Springer, 2009.

[6] D. Seto, E. Ferreira, and T. F. Marz. Case study: Development of a baseline controller for automatic landing of an F-16 aircraft using linear matrix inequalities (LMIs). Technical report, Carnegie-Mellon University Software Engineering Institute, 2000.

[7] B. L. Stevens and F. L. Lewis. Aircraft control and simulation. *Aircraft Engineering and Aerospace Technology*, 76(5), 2004.

[8] C. Tomlin, G. J. Pappas, and S. Sastry. Conflict resolution for air traffic management: A study in multiagent hybrid systems. *IEEE Transactions on automatic control*, 43(4):509–521, 1998.