16.07 Flight Simulation Laboratory

Assignment I Solution

Issued: Sep 24, 2004
Last Updated: Oct 25, 2004

In this laboratory exercise you were asked to develop a two degrees of freedom model for aircraft motion. Refer to the Assignment I handout for the entire problem text.

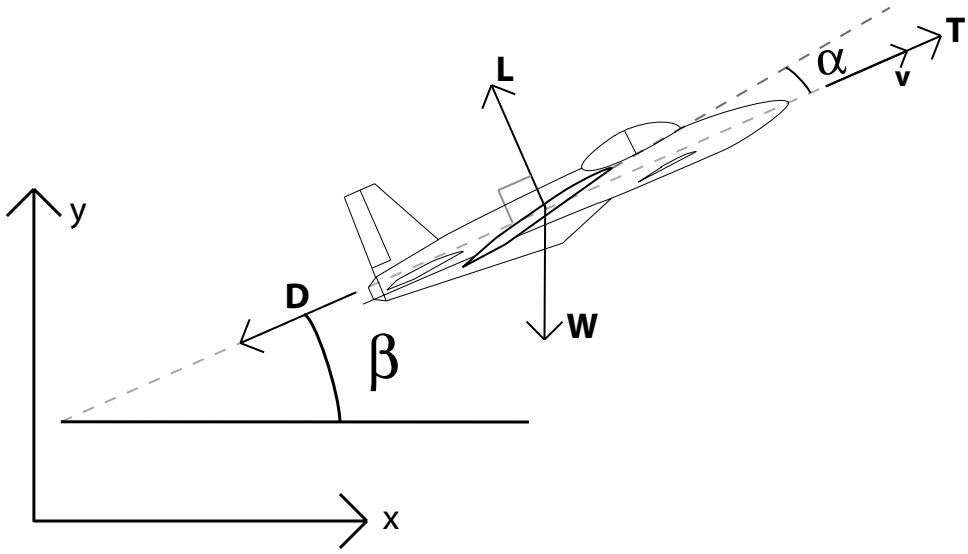

Figure 1: Aircraft and coordinate systems

Two degrees of freedom: the horizontal and the vertical positions, x and y. Four state variables: horizontal position x, vertical position y, horizontal speed v_x, and vertical speed v_y. Grouping them into $\mathbf{X}$:

$$\mathbf{X} = \begin{bmatrix} x \\ y \\ v_x \\ v_y \end{bmatrix} \tag{1}$$

Inputs: the angle of attack α and the thrust T. The equation of motion can then be written as:

$$\dot{\mathbf{X}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{v_x} \\ \dot{v_y} \end{bmatrix} = F(\mathbf{X}, \alpha(t), T(t)), \quad \mathbf{X}(t=0) = \mathbf{X}_0, \tag{2}$$

where t is the time and $\mathbf{X}_0$ is the initial state of the system. Now, let's begin. Governing assumptions:

- Aircraft is a point mass (moments of inertia are negligible)
- One main lifting surface (stabilizers give negligible lift/drag)
- No stall (linear $\delta C_L/\delta\alpha$ curve)
- Thrust T is assumed to be always aligned with the flight direction (in the notation from lecture D3, $\alpha_T = -\alpha$).
- No wind
- No ground friction

1. *"Using Newton's second law, find the explicit form of the function F . . ."*

   Putting the "function F" in words will help you understand what is supposed to happen here. Equation (2), in words, says that *the state derivatives* $\dot{\mathbf{X}}$ *are a function* F *of the current state* $\mathbf{X}$. The state vector $\mathbf{X}$ holds the aircraft's position and velocity, so your vector of state derivatives should contain the aircraft's velocity and acceleration. These are the values your derivatives function should be computing.

   The essence of this part of the lab is in the D3 lecture notes, beginning with page 3. It may not be easy to think of the aircraft as just a point mass, as we all know that the flight path of an aircraft greatly depends on where the thing is pointed. Instead, think of the model as constrained to point along the flight direction at all times, with a moment of inertia so small that there is no resistance to pitching motion (you'll learn more about moments of inertia later). Also in this model, the angle of attack is fixed. Since the airplane always points along its direction of travel, visualize the wing as pitched slightly upward with an angle α relative to the airflow at all times. In other words, the angle of attack is equal to the angle of incidence here. Interestingly enough, that makes the lift coefficient C_L and drag coefficient C_D constant for Part 3 because α is constant.

   Let's pseudocode this up by starting with the data we have to return to the integrator (aircraft velocity and acceleration) and heading backward from there. The accelerations come from Newton's 2nd Law: $\mathbf{F} = m\mathbf{a}$. All of the forces that act on the aircraft are given in Figure (1). Armed with this information:

   (a) We can write a differential equation for the acceleration in x and a differential equation for the acceleration in y:

$$\dot{v_x} = \frac{1}{m}(-Lsin\beta - Dcos\beta + Tcos\beta) \tag{3}$$

$$\dot{v_y} = \frac{1}{m}(Lcos\beta - Dsin\beta + Tsin\beta) - g \tag{4}$$

(b) Equations (3) and (4) present us with three unknowns: lift L, drag D, and flightpath angle β. T is an input to the system, so we don't have to compute it. Lift, drag, and flightpath angle are:

$$L = q_\infty S C_L = \frac{1}{2}\rho v^2 S C_L \tag{5}$$

$$D = q_\infty S C_D = \frac{1}{2}\rho v^2 S C_D \tag{6}$$

$$\beta = tan^{-1}(v_y/v_x) \tag{7}$$

where q_∞ is the dynamic pressure. Note that it's advantageous to use the MATLAB four-quadrant arctangent `atan2(`v_y`,` v_x`)` to compute β, as `atan2()` sweeps from $-\pi$ to π. This will be very important in Assignment II. Now we need v (note that this is the magnitude of the aircraft's velocity vector $\mathbf{v}$), C_L, and C_D:

$$C_D = C_{D_0} + \frac{C_L^2}{\pi \not{R} e} \tag{8}$$

$$C_L = K_1 \alpha \tag{9}$$

$$v = \sqrt{v_x^2 + v_y^2} \tag{10}$$

Everything in equations (9), (8), and (10) is either a constant or a state variable, so our function is complete... on paper.

2. *"Code the model equation (2) into a MATLAB function . . ."*

It's very helpful to look at the solution code for this part. Essentially, all you have to do is take the stuff in Part (1) and put it into MATLAB code in reverse order, so that you start with constants and aircraft state variables and end up with state derivatives. Just unpack the vector of aircraft states, go (10) (8) (9) (7) (6) (5) (4) (3), package the derivatives, and you're just about done. Since the state derivatives vector contains velocities as well as accelerations, what velocities do you store? Simply, the velocities that go into the state derivatives vector are the velocities that you unpack from the state vector. It's a direct one-to-one copy: $v_{output} = v_{input}$.

3. *"Test your code with the set of constant inputs . . ."*

Stack the parameters and inputs up at the top of your state derivatives function, as shown in the `springmass` example. In your main code routine (the script that calls the state derivatives function), write code that will plot the aircraft's position y vs. x. See the code solution for the main routine `lab1.m` to understand how to make the plots. Now give the code a run. Ah yes; the aircraft *does* fall below

the ground surface, doesn't it? You could write code to simulate a normal force given by the ground when the aircraft is on it, but it's easier to write code that simulates the *effects* of the normal force: the aircraft can't go below $y = 0$. This seems easy enough, but there are some subtleties in coding up this condition. If you wrote code to set y equal to 0 if it is less than 0, it wouldn't work because you can't change y directly (it's not returned from the state derivatives function). So what else does "not falling through the ground" mean? Simply, the aircraft's y velocity and y acceleration cannot be less than 0 *as long as it is on the ground*. In code, this condition becomes either a nested `if` statement or an `if` statement with a logical `and` comparison:

```
if (y <= 0) && (v_y < 0)
    v_y = 0;
end
if (y <= 0) && (v_y_dot < 0)
    v_y_dot = 0;
end
```

The `&&` is the MATLAB logical `and` operator. The comparisons on the left and right sides of the `&&` must be true for the whole comparison to evaluate to true. Note that earlier versions of MATLAB don't recognize the `&&` operator, but you can use `and(comparison1, comparison2)` to get the same effect. Place this code *after* the computation of $\dot{v}_y$ and $\dot{v}_x$, but *before* returning the derivatives.

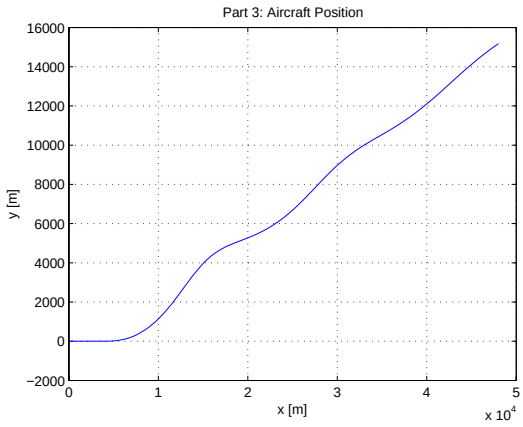

Figure 2: Part 3 Position

The Part 3 hodograph plot in Figure (3) is rather telling: you can guess from the spiral that if the aircraft were simulated for a long time, the velocity v_y would eventually settle around 60 m/s and the velocity v_x would eventually settle around

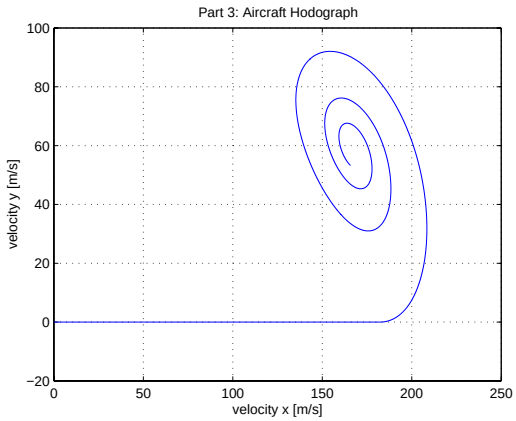

Figure 3: Part 3 Hodograph

170 m/s. Even more interesting: the aircraft doesn't lift off from the ground until it reaches almost 180 m/s! Perhaps unrealistic, but anyway...

4. *"Find a set of* time-varying *inputs α and T such that the airplane takes off from the airport, climbs to a certain altitude, and continues on a steady-state level flight . . ."*

   First off, you want to write a whole new state derivatives function to hold the code for Part 4. Don't worry; almost all of the code will be copied verbatim from the derivatives function for Part 3. Having it in a separate file makes logical sense because you are essentially running a whole new simulation with different conditions, and there is no way to pass switches or arguments to your state derivatives routines anyway.

   Next, you want to figure out the steady state flight conditions ("cruise conditions") on paper. There are actually an infinite number of steady-state flight conditions; they all depend on what steady-state speed you want your aircraft to fly at. You must pick a speed! Pick something reasonable (the solution uses 178 m/s) and then do a force balance to get the thrust T and wing angle-of-attack α. In steady-state level flight:

$$L = W \tag{11}$$
$$T = D \tag{12}$$

   Expanding these out:

$$\frac{1}{2}\rho v^2 S K_1 \alpha = mg \tag{13}$$

$$T = \frac{1}{2}\rho v^2 S(C_{D_0} + \frac{(K_1\alpha)^2}{\pi \cancel{A} Re}) \tag{14}$$

Now you know what the thrust and angle of attack should be during steady-state level flight for a given speed. How do you get there? Think of how a plane takes off and climbs out in reality: full power on take-off, with up-elevator to get the nose off the ground and begin the climb. Once the plane is in the air, the pilot backs off on the elevator and brings the thrust down a bit. Bringing the angle of attack and thrust to the cruise conditions will cause the aircraft to *eventually* settle into cruise. These steps are good enough for this assignment, and are what the solution follows. "Up-elevator" is simulated by increasing the angle-of-attack of the wing.

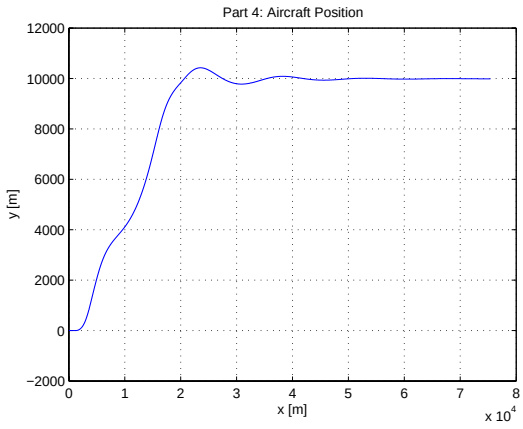

Figure 4: Part 4 Position

The hodograph looks funny due to the amount of oscillations the aircraft undergoes, but you can see clearly where the plane settles down into the cruise phase.

5. *". . . This is undamped phugoid motion."*

Again, start a new state derivatives function for Part 5. Most of the code will be a copy of what you've already written. After computing the drag D, set $T = D$ as requested. Head back to paper to figure out the angle of attack you need to maintain cruise at 170 m/s by doing a force balance like you did for the cruise phase in Part 4. α turns out to be 0.046 radians here. In your main routine, construct a new initial vector of states based on the given information and use it to simulate the flight for Part 5. Why do we ask you to use the conditions for a 170 m/s cruise on an aircraft that's started at 185 m/s? The point is that the aircraft will oscillate about the cruise speed and a particular rest height, but because $T = D$ at all times, there

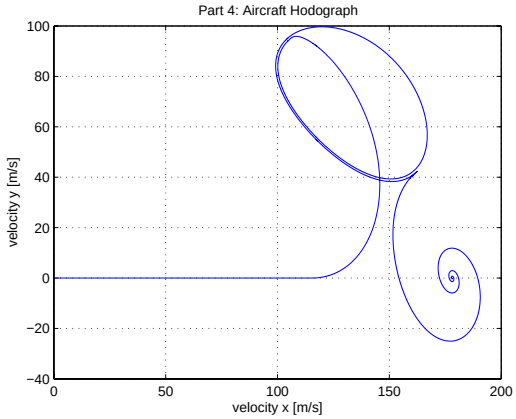

Figure 5: Part 4 Hodograph

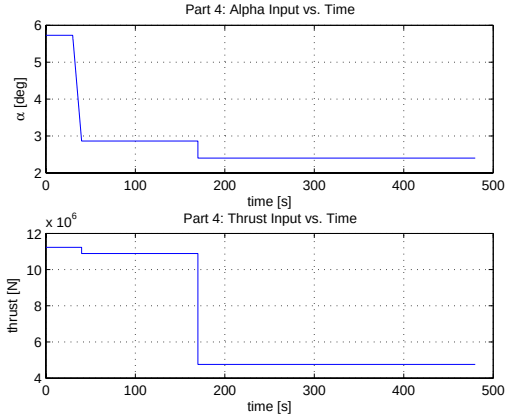

Figure 6: Part 4 Inputs

is no dissipation in the system. Lack of dissipation leads to undamped oscillation. Think of a spring vibrating back and forth without any damping: it will oscillate about the rest point forever.

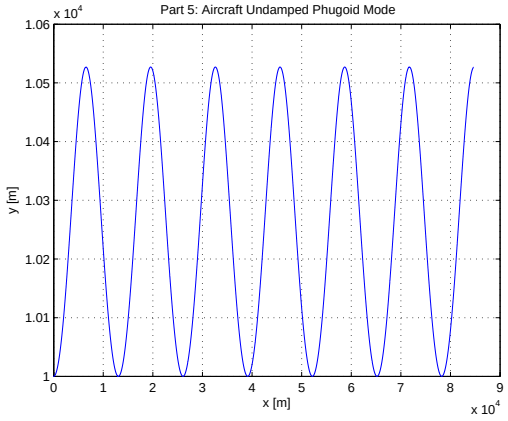

Figure 7: Part 5 Position