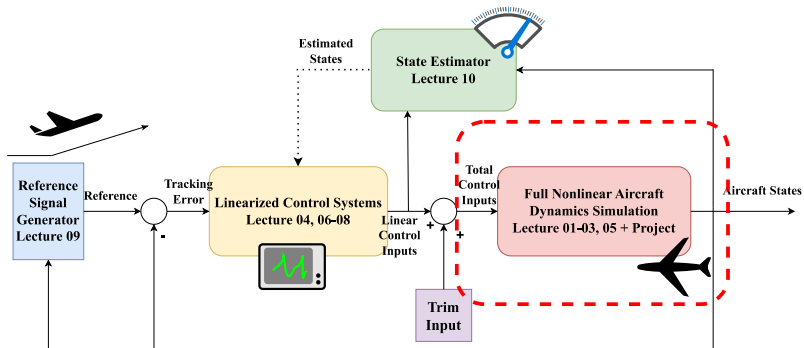


Lecture 05

Flight Simulation with Simulink

October 14, 2026

Acknowledgement: These slides were developed with reference to course materials by Professor Hugh H. T. Liu.



Contents

① Controller Design in MATLAB

② Simulink Basics

Three Representations of the Same System

MATLAB stores a linear model in whichever form you build it, and converts freely between them.

```
s = tf('s'); G = 1/(s^2 + 2*s + 5); % transfer function
G = tf([1 2],[1 3 2]); % numerator, denominator
sys = ss(A,B,C,D); % state space
G = zpk(-2,[-1 -3],5); % zeros, poles, gain

tf(sys)      ss(G)      zpk(G)      % convert
```

Inspect a model with pole, zero, dcgain, order and damp.

The three are mathematically equivalent but *not* numerically equivalent. A high-order tf is ill-conditioned: its coefficients span many orders of magnitude and root-finding loses digits. Build and interconnect in ss, and convert to tf only to display a result.

From the Aircraft Model to a SISO Channel

For $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}$, $\mathbf{y} = \mathbf{C}\mathbf{x}$, the transfer matrix is

$\mathbf{G}(s) = \mathbf{C}(s\mathbf{I} - \mathbf{A})^{-1}\mathbf{B}$. A classical design needs one output and one input, so select the channel first. For the longitudinal ordering

$$\mathbf{x} = [u \ w \ q \ \theta]^T, \quad \mathbf{u} = [\delta_e \ \delta_T]^T,$$

$$G_{ue} = G(1, 1),$$

$$G_{uT} = G(1, 2),$$

$$G_{\theta e} = G(4, 1),$$

$$G_{\theta T} = G(4, 2).$$

```
G = ss(A,B,eye(4),zeros(4,2)); % all four states as outputs
Gte = tf(G(4,1)); % theta <- delta_e
GuT = tf(G(1,2)); % u <- delta_T
dcgain(Gte) % CHECK THE SIGN FIRST
```

Indexing is $G(\text{output}, \text{input})$.

Under the course convention a positive elevator perturbation gives a negative pitch response, so $\text{dcgain}(Gte)$ is negative and the positive design plant is $P_\theta = -G_{\theta e}$. Fixing this sign before designing saves chasing an inverted root locus later.

Interconnection, and Why the Order Grows

```
L = C*P;           % or series(C,P)
T = feedback(L,1);  % unity negative feedback, L/(1+L)
S = feedback(1,L);  % sensitivity, 1/(1+L)
```

Products and sums of `tf` objects multiply denominators *without* cancelling. A loop assembled by hand therefore comes back at a far higher order than the system really has: the same mode appears in both numerator and denominator, two or three times over.

This is bookkeeping, not dynamics. The step response and every margin are unaffected — but `pole` will list spurious repeats, and a design verified against that pole list is being read wrongly. Either interconnect in `ss`, where the order never inflates, or reduce afterwards.

Why an interconnection comes back too big

Build a loop from transfer functions and MATLAB multiplies denominators without cancelling. Take $G_1 = \frac{1}{s+1}$, $G_2 = \frac{1}{s+2}$ and

$$H = G_1 + \frac{G_1 G_2}{1 - G_2}.$$

By hand, $\frac{G_1 G_2}{1 - G_2} = \frac{1}{(s+1)(s+2)} \cdot \frac{s+2}{s+1} = \frac{1}{(s+1)^2}$, so

$$H = \frac{1}{s+1} + \frac{1}{(s+1)^2} = \frac{s+2}{(s+1)^2}, \quad \text{order 2.}$$

MATLAB returns order **4**: poles $\{-2, -1, -1, -1\}$ against zeros $\{-2, -2, -1\}$. The extra $(s+2)$ and $(s+1)$ appear in both lists, so they cancel exactly. The order is bookkeeping, not dynamics.

`minreal` returns a *minimal realization*: the smallest model with the same input-output behaviour. On a state-space object it removes uncontrollable or unobservable states; on a `tf` or `zpk` object it cancels pole-zero pairs lying within tolerance.

minreal: Minimal Realization

```
Pr = minreal(P);           % default tolerance sqrt(eps) ~ 1.5e-8  
Pr = minreal(P,1e-3);     % looser: cancels near pairs too
```

`minreal` returns the smallest model with the same input–output behaviour. On a state-space object it removes uncontrollable or unobservable states; on a `tf` or `zpk` object it cancels pole–zero pairs lying within tolerance.

Do not guess the tolerance. Whether it is compared against the raw separation $|z - p|$ or against a scaled one such as $|z - p|/|z|$ is an implementation detail, and the two differ by orders of magnitude when the roots are far from the unit circle. List the actual separations with `zpkdata` and look for a clear gap in them.

Choosing the minreal Tolerance

Separations for the speed channel of Assignment 2, Q2:

pair	$ z - p $	as a fraction of $ z $
duplicated modes	$3 \times 10^{-5} \dots 9 \times 10^{-4}$	$10^{-6} \dots 3 \times 10^{-5}$
genuine near pair	3.8×10^{-2}	5.8×10^{-3}

Two clusters, decades apart. Put the tolerance in the empty space between them and the reduction is the same on either measure.

Cancelling a merely-close pair is a model *reduction*, not an identity: it discards dynamics that are small in this channel but still present. Choose the tolerance by inspecting the actual gaps, and verify the finished design on the unreduced model.

Deterministic Lead Design and Exact Verification

For unity feedback with a type-0 plant,

$$e_{ss} = \frac{1}{1 + KP(0)}.$$

Fixing an error target therefore fixes K before the transient design. For the lead compensator

$$C(s) = K \frac{Ts + 1}{\alpha Ts + 1}, \quad 0 < \alpha < 1,$$

use the specified phase-margin target and allowance to determine $\phi_{\max}$, then

$$\alpha = \frac{1 - \sin \phi_{\max}}{1 + \sin \phi_{\max}}, \quad |KP(j\omega_{\max})|_{\text{dB}} = 10 \log_{10} \alpha,$$

$$T = \frac{1}{\omega_{\max} \sqrt{\alpha}}.$$

When several magnitude crossings exist, the design must state which crossing is used. Finally, verify the full model with `margin`, `pole`, `dcgain`, and `stepinfo`. Phase margin guides the design; the exact step response determines whether the overshoot requirement is met.

Close the Pitch Loop Before Designing Speed Control

With the pitch command held at zero and the elevator sign convention

$$\delta_e = -C_e(\theta_c - \theta) = C_e\theta,$$

the coupled longitudinal channels satisfy

$$\theta = G_{\theta e}\delta_e + G_{\theta T}\delta_T,$$

$$u = G_{ue}\delta_e + G_{uT}\delta_T.$$

Eliminating θ and δ_e gives the plant seen by the outer speed controller:

$$P_u(s) = G_{uT}(s) + \frac{G_{ue}(s)C_e(s)G_{\theta T}(s)}{1 - C_e(s)G_{\theta e}(s)}.$$

This reduction preserves the effect of the closed pitch loop. Designing on G_{uT} alone instead describes a different architecture and can give the wrong low-frequency sign.

A Reproducible Filtered-PID Construction

Suppose justified pole-zero cancellation reduces the outer-loop plant to

$$\hat{P}(s) = k \frac{s + z}{(s + p_1)(s + p_2)}, \quad p_1, p_2, z > 0.$$

For a specified bandwidth ω_b , choose

$$C(s) = \frac{\omega_b}{k} \frac{(s + p_1)(s + p_2)}{s(s + z)}.$$

Then $C\hat{P} = \omega_b/s$, giving a nominal first-order closed loop with zero step error and no overshoot. To write the proper controller as

$$C(s) = K_p + \frac{K_i}{s} + \frac{K_d s}{T_f s + 1},$$

set $a = \omega_b/k$ and use

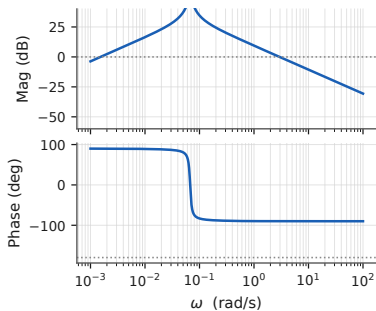
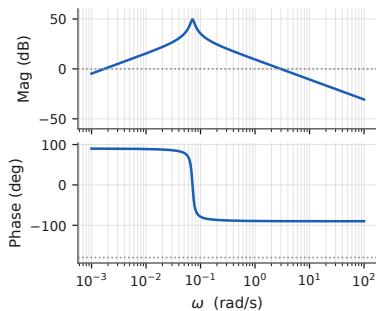
$$T_f = \frac{1}{z},$$

$$K_i = \frac{ap_1 p_2}{z},$$

$$K_p = \frac{a(p_1 + p_2)}{z} - K_i T_f,$$

$$K_d = \frac{a}{z} - K_p T_f.$$

Pole-zero cancellation is a design approximation. Stability and performance must be checked on the exact interconnected model.

(a) $G_{u\delta_p}$ (b) $G_{u\delta_p}^{ph}$

From control design perspective, due to the nature of $G_{u\delta_p}$ has zero $s = 0$, it is expected that a simple proportional control will not be able to regulate the speed to track the command properly, as we can show that steady state error under a command step function u_c is 100%. This fact is shown in Fig. 1. This can be mitigated by using a PI control, where the integral term contributes to reduce the steady error, as shown in Fig. 2.

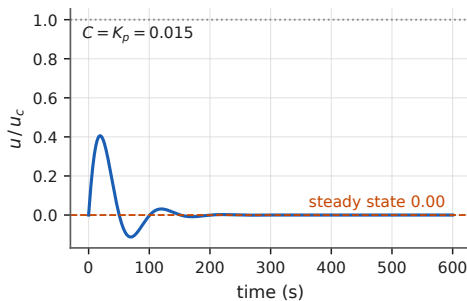


Figure 1: Step Response of Closed-loop System $\frac{CG}{1+CG}$, $C = K_p = 0.015$

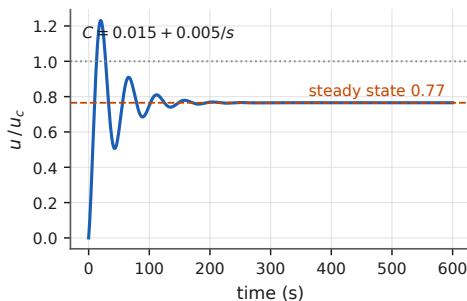


Figure 2: Step Response of Closed-loop System $\frac{CG}{1+CG}$,
 $C = K_p + K_i/s = 0.015 + 0.005/s$

In fact, from the approximation transfer function, one can estimate the steady error

$$e_{ss} = \frac{1}{1 + (K_p + \frac{K_i}{s})G_{u\delta_p}} = \frac{-\frac{g}{m} \frac{Z_u}{U_e}}{-\frac{g}{m} \frac{Z_u}{U_e} + K_i \frac{X_{\delta_p}}{m}} = 0.2 \quad K_i = 0.005$$

What Simulink Is

A graphical environment for modelling and simulating dynamical systems. It shares a workspace with MATLAB, so the matrices and compensators built in Part 1 are read straight into the blocks.

- ▶ A diagram is a *model*, not a script: the blocks carry no execution order of their own, and the solver derives one from the signal flow.
- ▶ Nonlinearity, saturation, rate limits and sampling are drawn in the diagram — which is why a linear design is verified here rather than in `lsim`.
- ▶ A recorded signal enters through a From Workspace block, which interpolates between its samples; the solver step and the data step are independent.

Assignment 2, Q3 is exactly this: the Euler kinematics

$[\dot{\phi} \ \dot{\theta} \ \dot{\psi}]^T = \mathbf{S}_B^{-1} \underline{\omega}_B$, a nonlinear input-driven right-hand side,

integrated from rest over a recorded $\underline{\omega}_B$ history.

How to Solve ODEs in Simulink?

A block diagram is not a solution — it is a picture of the right-hand side of

$$\dot{\underline{x}} = \underline{f}(t, \underline{x}, \underline{u}), \quad \underline{x}(0) = \underline{x}_0$$

The Integrator block is the state: input $\dot{\underline{x}}$, output $\underline{x}$. The path from that output back into $\underline{f}$ is what makes the diagram an ODE.

- ▶ One Integrator per state, its initial condition set in the block rather than in the workspace.
- ▶ A State-Space block is the same construction for $\dot{\underline{x}} = \underline{A}\underline{x} + \underline{B}\underline{u}$; a 6-DOF model needs the explicit form.
- ▶ The solver drives the diagram — it proposes $(t, \underline{x})$, evaluates every block for $\dot{\underline{x}}$, and advances, several times per step.
- ▶ A feedback path with no integrator is an *algebraic loop*: $\underline{x}$ is implicit and must be solved for at every step. Break it with a state.

Runge-Kutta Method in Simulink

The default solvers are explicit Runge–Kutta. Fixed-step ode4 is classical RK4: from $\underline{x}_n$ at t_n with step h ,

$$\underline{k}_1 = \underline{f}(t_n, \underline{x}_n), \quad \underline{k}_2 = \underline{f}\left(t_n + \frac{h}{2}, \underline{x}_n + \frac{h}{2}\underline{k}_1\right)$$

$$\underline{k}_3 = \underline{f}\left(t_n + \frac{h}{2}, \underline{x}_n + \frac{h}{2}\underline{k}_2\right), \quad \underline{k}_4 = \underline{f}(t_n + h, \underline{x}_n + h\underline{k}_3)$$

$$\underline{x}_{n+1} = \underline{x}_n + \frac{h}{6} \left(\underline{k}_1 + 2\underline{k}_2 + 2\underline{k}_3 + \underline{k}_4 \right)$$

Four evaluations of the whole diagram per step, and global error $O(h^4)$. Variable-step ode45 is Dormand–Prince 5(4). It takes a fifth-order step alongside an embedded fourth-order one and treats their difference as the local error estimate, then adjusts h until that estimate falls below $\max(\text{RelTol} |x_i|, \text{AbsTol}_i)$ — by default 10^{-3} and 10^{-6} . A plot that changes when you tighten RelTol is not converged.

Choosing the Solver and the Step Size

Applied to $\dot{x} = \lambda x$, RK4 is stable only for λh inside its stability region: $|\lambda h| < 2.785$ along the negative real axis and $|\lambda h| < 2\sqrt{2} = 2.828$ along the imaginary axis. Accuracy binds long before stability does — aim for $|\lambda| h \lesssim 0.1$ on the fastest mode you care about.

- ▶ The B747 longitudinal eigenvalues span $|\lambda| = 0.061$ (phugoid) to 0.962 (short period), a ratio of only 16: the model is not stiff, and $h = 0.01$ s resolves it with room to spare.
- ▶ Adding a 20 rad/s actuator or a fast inner loop is what actually sets h . The ratio, not the absolute speed, decides whether an explicit solver is the right choice.
- ▶ Use ode45 while exploring, fixed-step ode4 when you need a uniform grid. A From Workspace record carries its own rate — $\Delta t = 0.02$ s in A2 Q3 — so pick a step that divides it.
- ▶ If h collapses and the simulation crawls, the model is stiff. Move to ode15s or ode23t rather than forcing a smaller step.

Basic Vector Calculus

$$\underline{\mathbf{a}} + \underline{\mathbf{b}}, \quad \lambda \underline{\mathbf{a}}, \quad \underline{\mathbf{a}}^T \underline{\mathbf{b}}, \quad \underline{\mathbf{a}} \times \underline{\mathbf{b}}, \quad \|\underline{\mathbf{a}}\|$$

Matrix Product

$$\underline{y} = \underline{A}\underline{x}, \quad \underline{v}_b = \underline{C}_{ba} \underline{v}_a$$

Integration

$$\underline{\mathbf{x}}(t) = \underline{\mathbf{x}}_0 + \int_0^t \dot{\underline{\mathbf{x}}}(\tau) d\tau$$

State-Space Models

$$\dot{\underline{x}} = \underline{A}\underline{x} + \underline{B}\underline{u}, \quad \underline{y} = \underline{C}\underline{x} + \underline{D}\underline{u}$$

Feedback Control

$$\underline{u} = -\underline{K}\underline{x}, \quad u = C(s)(r - y)$$

Summary

① Controller Design in MATLAB

② Simulink Basics