4.5 Multicopter Modeling and Simulation

Chapter Outline

Implement and Verify a Multicopter Motion Model Step by Step

Chapter 4
Vehicle Motion Modeling and Simulation

4.1 Modeling Fundamentals and Kinematics
4.2 Rigid-Body Dynamics and Solvers
4.3 Propulsion, Forces, and Observations
4.4 Trim, Linearization, and Dynamics
4.5 Multicopter Modeling and Simulation
4.6 Measurement and Static Identification
4.7 Dynamic Identification and Refinement
4.8 Typical Vehicles and Extensions
4.9 Modeling Tools and Deployment
4.10 API and SDK Modeling Interfaces
4.11 Examples and Integrated Practice
4.12 Exercises and Course Projects

6 Units in This Section

4.5.1 Program Objects, States, and Hierarchy
Textbook: Secs. 4.1.3.1–4.1.3.2, 4.2.3

4.5.2 A Minimal Vertical-Motion Program
Textbook: Sec. 4.1.4.1, Translational Dynamics

4.5.3 Progressive Single-Axis Rotation
Textbook: Secs. 4.1.4.1, 4.1.4.4

4.5.4 Three-Dimensional Rigid-Body Integration
Textbook: Secs. 4.1.4.1, 4.1.4.4

4.5.5 Propulsion, Environment, and Observation Extensions
Textbook: Secs. 4.1.4.2, 4.1.4.3, 4.1.4.6

4.5.6 Template Comparison and Acceptance Checks
Textbook: Secs. 4.2.3, 4.2.4

Section Introduction

Turn Four Model Layers into a Checkable Program

- **Isolate each problem:** Start with prescribed thrust and vertical motion, add pitch, then connect 3D attitude, propulsion, and the environment.
- **One calculation chain:** The input function supplies propeller commands; the load function sums forces and torques; the derivative function computes rates; the integrator updates the state.
- **One evidence set:** Log states, loads, and model time at every step so hand calculations, plots, and code can be matched.

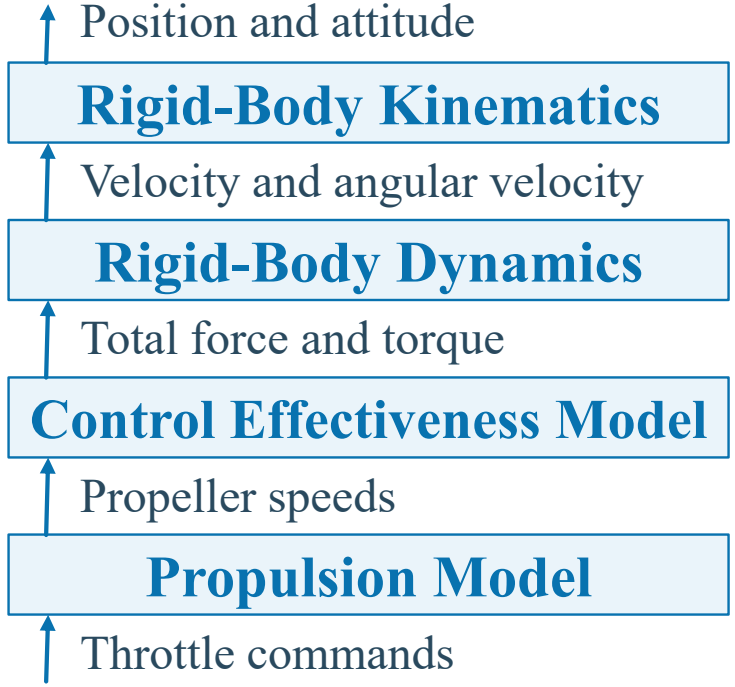


Figure: Four-layer motion model

Each program layer returns physical quantities the next layer needs and can verify.

4.5.1 Program Objects, States, and Hierarchy: Learning Entry Point

Find the Standalone Learning Example First

- **Directory:** [installation directory]/RflySimAPIs/4.RflySimModel/1.BasicExps/e6_ModelingLearningLab/. Read its Readme.pdf first.
- **Main program:** 01_VerticalRigidBody (read its Readme.pdf). Set this as the MATLAB current folder and run the four stages with modeling_stages('vertical'), etc. vertical_beginner and simulation.py are separate beginner routes.
- **M0 checks:** 05_M0Checks/m0_offline_checks.m copies e0_MinModelTemp for offline cases and requires Simulink. The staged program needs base MATLAB only; Python needs NumPy.
- **Outputs/diagnosis:** On normal completion, metrics show checks_pass = true. Stage folders under outputs contain CSV, JSON, and PNG files. If the directory is read-only, pass a writable output directory.

Programs, data, and instructions together make a reproducible modeling experiment.

4.5.1 Program Objects, States, and Hierarchy: Program Parameters

Reuse the Example Parameters, Initial State, and Frames

- **Initial state:** Level body, nose north; zero linear and angular velocity. Vehicle and simplifications: "Parameters and Initial Conditions," slide deck 4.1.
- **Teaching parameters:** $m = 2\,\text{kg}$, $g = 9.8\,\text{m/s}^2$; all rotor moment arms are 0.25 m; pitch inertia $J_y = 0.05\,\text{kg}\,\text{m}^2$.
- **Inputs/outputs:** Set front, right, rear, and left propeller thrusts; compute position, velocity, attitude, and angular velocity. Command-to-thrust model: slide deck 4.3.

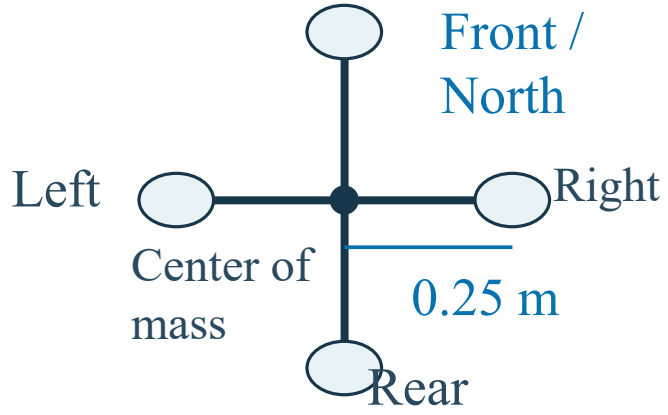


Figure: Top-view layout and rotor moment arms

- **Simplifications:** Initially set drag, rotational damping, rotor lag, and ground contact to zero for analytical vertical/single-axis checks. Unit 4.5.5 adds motor lag and relative airflow.
- **Ground origin below start; down positive:** $z_D = -1\,\text{m}$, $H = -z_D = 1\,\text{m}$.

Define the vehicle and operating case before predicting its response to an input.

4.5.1 Program Objects, States, and Hierarchy: Function and State Interfaces

Keep Interfaces Fixed as Internal Physics Grows

- **Call:** modeling_stages(stage,outDir,dt) selects the stage, output directory, and time step. The default step is 0.002 s.
- **Returned state:** The first 13 columns of result.state always contain position, velocity, quaternion, and body angular velocity, in that order. The extended stage appends four rotor-speed columns.
- **Returned loads:** force_body_N excludes gravity; moment_body_Nm is about the center of mass; acceleration_NED_mps2 uses the ground NED frame.

$$\mathbf{x} = [\mathbf{p}_e^{\mathrm{T}}, \mathbf{v}_e^{\mathrm{T}}, \mathbf{q}_{eb}^{\mathrm{T}}, \boldsymbol{\omega}_b^{\mathrm{T}}]^{\mathrm{T}}$$

- **Subscripts:** e denotes the ground north–east–down (NED) frame; b denotes the body forward–right–down (FRD) frame.

Fixed return quantities let simple and complex stages use the same comparison methods.

Chapter Outline

Implement and Verify a Multicopter Motion Model Step by Step

Chapter 4
Vehicle Motion Modeling and Simulation

4.1 Modeling Fundamentals and Kinematics
4.2 Rigid-Body Dynamics and Solvers
4.3 Propulsion, Forces, and Observations
4.4 Trim, Linearization, and Dynamics
4.5 Multicopter Modeling and Simulation
4.6 Measurement and Static Identification
4.7 Dynamic Identification and Refinement
4.8 Typical Vehicles and Extensions
4.9 Modeling Tools and Deployment
4.10 API and SDK Modeling Interfaces
4.11 Examples and Integrated Practice
4.12 Exercises and Course Projects

6 Units in This Section

4.5.1 Program Objects, States, and Hierarchy
Textbook: Secs. 4.1.3.1–4.1.3.2, 4.2.3

4.5.2 A Minimal Vertical-Motion Program
Textbook: Sec. 4.1.4.1, Translational Dynamics

4.5.3 Progressive Single-Axis Rotation
Textbook: Secs. 4.1.4.1, 4.1.4.4

4.5.4 Three-Dimensional Rigid-Body Integration
Textbook: Secs. 4.1.4.1, 4.1.4.4

4.5.5 Propulsion, Environment, and Observation Extensions
Textbook: Secs. 4.1.4.2, 4.1.4.3, 4.1.4.6

4.5.6 Template Comparison and Acceptance Checks
Textbook: Secs. 4.2.3, 4.2.4

4.5.2 A Minimal Vertical-Motion Program: States and Equations

Use Two Vertical Equations to Choose the Stored States

- **Conditions:** Level body, vertically upward thrust, and a small-perturbation baseline near hover in near-calm air. Drag is zero in this stage.

- **Variables:** Down position z_D is positive downward. Initially $z_D = -1\ \mathrm{m}$ and $v_D = 0$; height is $H = -z_D$.
- **Thrust effect:** At total thrust $T = mg$, acceleration is zero; when $T > mg$, $a_D < 0$, so the vehicle accelerates upward.

$$\dot{z}_D = v_D, \qquad \dot{v}_D = g - \frac{T}{m}$$

Velocity is the rate of position; net force divided by mass is the rate of velocity.

4.5.2 A Minimal Vertical-Motion Program: Piecewise Input

Accelerate, Decelerate, Then Hold

- **Schedule:** Hover for the first 0.5 s; accelerate upward at 0.5–1.5 s; decelerate at 1.5–2.5 s; then hold.
- **Equal propellers:** 5.4 N each gives 21.6 N total; 4.4 N each gives 17.6 N total; the equilibrium value is 4.9 N each.

```
forces = 4.9*ones(4,1);
if t>=.5 && t<1.5
    forces(:)=5.4;
elseif t>=1.5 && t<2.5
    forces(:)=4.4;
end
```

Returning thrust to weight does not remove the upward velocity already accumulated.

4.5.2 A Minimal Vertical-Motion Program: Runnable Program Core

A Single Loop Implements Forward Euler for Vertical Motion

- **Variables:** t is time, T total thrust, zD down position, and vD down velocity. Initialize zD(1)=-1 and vD(1)=0; dt is the time step.

```
for k = 1:numel(t)-1
    aD = g - T(k)/m;
    zD(k+1) = zD(k) + dt*vD(k);
    vD(k+1) = vD(k) + dt*aD;
end
```

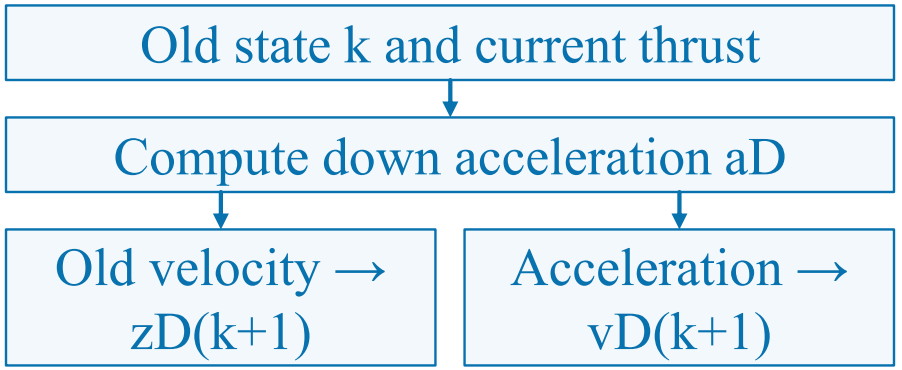


Figure: Both updates read k and write k+1

- **Read the code:** Each iteration calculates acceleration from net force, then updates from the old position and velocity. Both right-hand sides read index k. The position and velocity arrays retain every step.
- **Plot:** plot(t,-zD) shows height; plot(t,-vD) shows upward velocity.

4.5.2 A Minimal Vertical-Motion Program: Interpreting the Run

Read the Thrust, Velocity, and Height Curves

- **Entry:** Revisit the vertical_beginner exercise run in slide deck 4.1; read its plots alongside this program.
- **Baseline:** 21.6 N accelerates for 1 s, 17.6 N brakes for 1 s; 19.6 N otherwise. The comparison omits braking.
- **At 3 s:** baseline 2 m, zero velocity; comparison about 3 m and still climbing at about 1 m/s.
- **Read plots:** thrust sets acceleration; integrate to velocity, then height. Upward velocity peaks at 1.5 s.

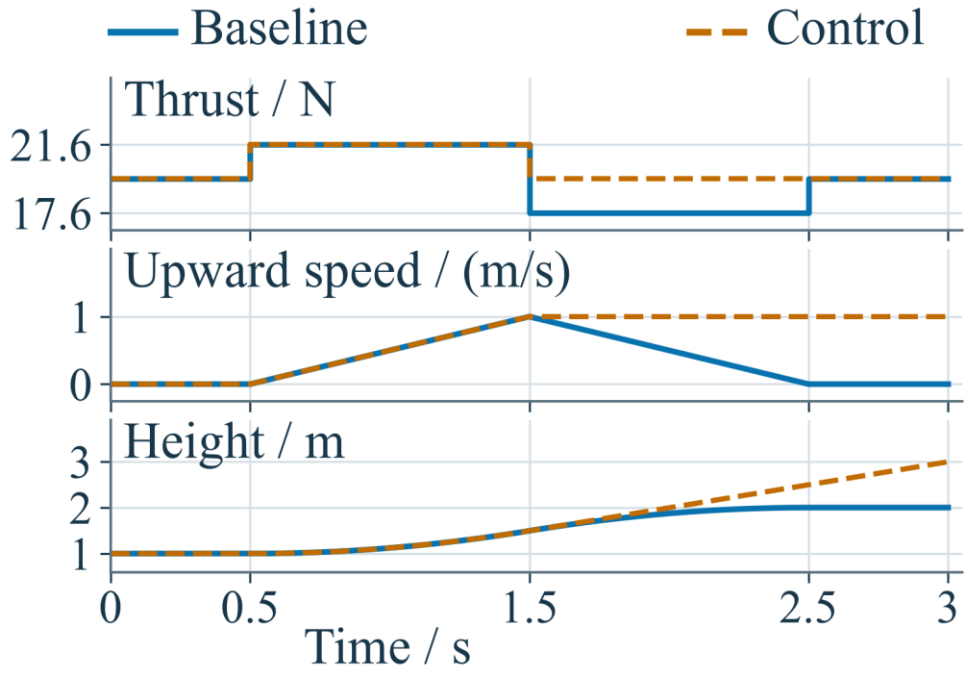


Figure: Baseline vs comparison (teaching; not platform measurements)

Check the program with a hand-calculable case before adding model complexity.

4.5.2 A Minimal Vertical-Motion Program: Change One Parameter

Change One Parameter: Predict First, Then Run

- **Mass change:** Set m to 2.2 kg. Weight 21.56 N > thrust 19.6 N gives about 0.89 m/s² down acceleration in the hover interval. At 3 s, height is about −2.1 m (no ground-contact model).
- **Time step:** Keep 2 kg; reduce dt from 0.01 s to 0.005 s. Maximum position error halves: 5 mm → 2.5 mm.
- **Save separately:** Change only one quantity and use a new output directory for every run.

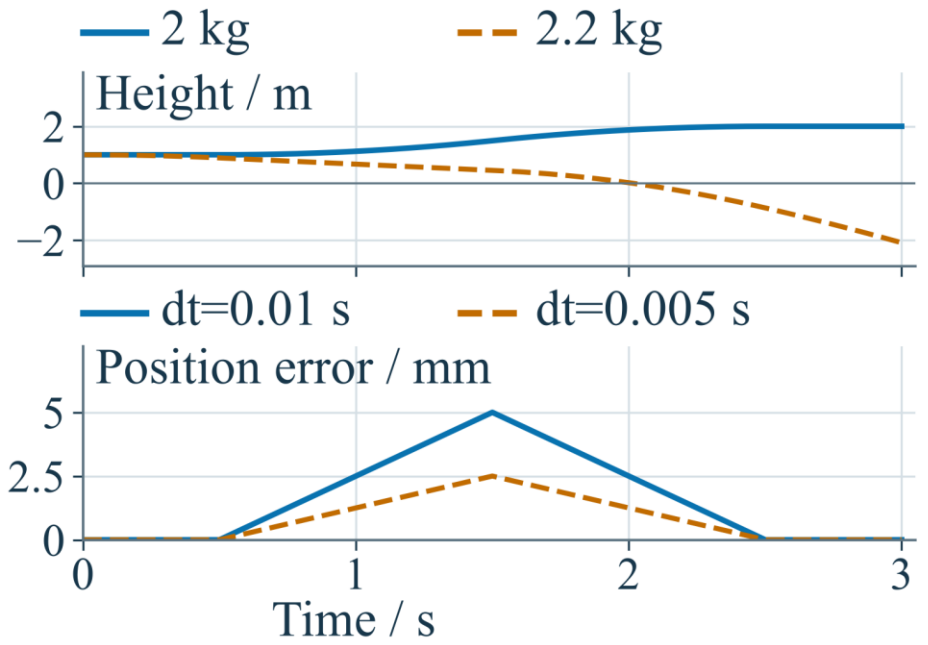


Figure: Mass/time-step changes (teaching; not platform measurements)

Changing one parameter at a time makes its effect identifiable.

4.5.2 A Minimal Vertical-Motion Program: Endpoint and Curve Checks

Check the Endpoint, Then Explain the Whole Curve

- **Switch:** Run modeling_stages('vertical'); output: outputs/vertical. It integrates the same equations and input by fourth-order Runge–Kutta (RK4; Sec. 4.5.4).
- **At 3 s:** height 2.000 m, down velocity zero. The curve steepens, then flattens.
- **Input check:** Height is constant before 0.5 s and after 2.5 s. Acceleration and braking give opposite curve curvature.

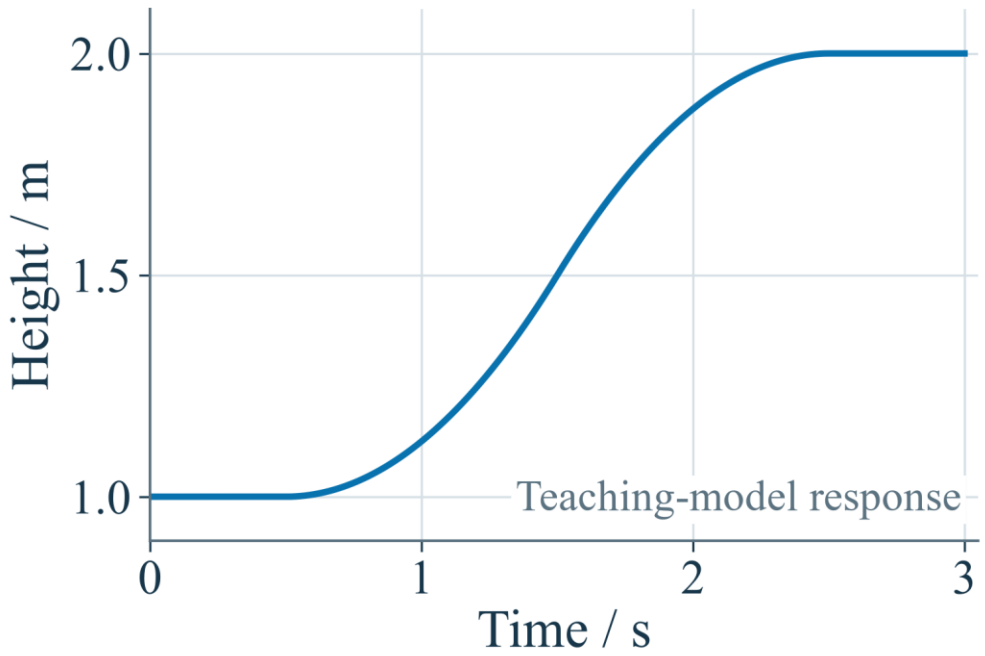


Figure: Computed height in the vertical stage

The correct endpoint meets both target height and zero vertical velocity.

Chapter Outline

Implement and Verify a Multicopter Motion Model Step by Step

Chapter 4
Vehicle Motion Modeling and Simulation

4.1 Modeling Fundamentals and Kinematics
4.2 Rigid-Body Dynamics and Solvers
4.3 Propulsion, Forces, and Observations
4.4 Trim, Linearization, and Dynamics
4.5 Multicopter Modeling and Simulation
4.6 Measurement and Static Identification
4.7 Dynamic Identification and Refinement
4.8 Typical Vehicles and Extensions
4.9 Modeling Tools and Deployment
4.10 API and SDK Modeling Interfaces
4.11 Examples and Integrated Practice
4.12 Exercises and Course Projects

6 Units in This Section

4.5.1 Program Objects, States, and Hierarchy
Textbook: Secs. 4.1.3.1–4.1.3.2, 4.2.3

4.5.2 A Minimal Vertical-Motion Program
Textbook: Sec. 4.1.4.1, Translational Dynamics

4.5.3 Progressive Single-Axis Rotation
Textbook: Secs. 4.1.4.1, 4.1.4.4

4.5.4 Three-Dimensional Rigid-Body Integration
Textbook: Secs. 4.1.4.1, 4.1.4.4

4.5.5 Propulsion, Environment, and Observation Extensions
Textbook: Secs. 4.1.4.2, 4.1.4.3, 4.1.4.6

4.5.6 Template Comparison and Acceptance Checks
Textbook: Secs. 4.2.3, 4.2.4

4.5.3 Progressive Single-Axis Rotation: Single-Axis Boundary

Fix Translation to Isolate Pitch

- **Physical setup:** An ideal support fixes position but allows pitch about the center of mass. This constrained experiment is not free flight.
- **Differential input:** Front thrust 4.7 N, rear thrust 5.1 N; total thrust remains 19.6 N. Right and left propellers stay at 4.9 N.
- **Retained states:** Integrate angular velocity and quaternion normally. Set only position and linear-velocity derivatives to zero to isolate pitch.

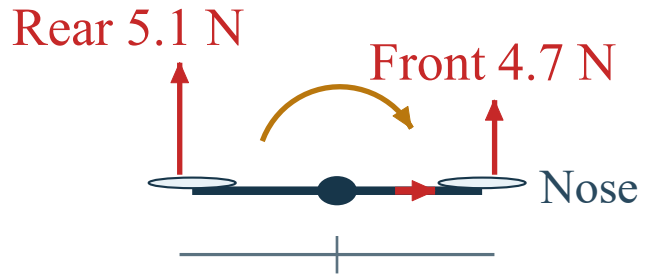


Figure: Weaker front, stronger rear → nose down

- **Rotate and brake:** Apply negative torque for 0.3 s, then equal positive torque for 0.3 s. Angular velocity returns to zero; pitch stops at −0.18 rad (about −10.31°). See “A Single-Axis Calculation” in slide deck 4.2.

Reducing degrees of freedom isolates rotation; the constraint must be explicit in the model.

4.5.3 Progressive Single-Axis Rotation: Moment Arm and Torque

Compute Pitch Torque from Rotor Locations

- **Teaching layout:** Front, right, rear, and left rotors lie along $+x_b, +y_b, -x_b, -y_b$, respectively; arm length is $L = 0.25$ m.
- **Negative-pitch interval:** Front thrust is 4.7 N and rear thrust is 5.1 N, giving $M_y = L(T_f - T_r) = -0.10$ N m.
- **Prediction:** $J_y = 0.05\ \mathrm{kg\,m^2}$; single-axis motion has zero gyroscopic coupling, giving angular acceleration $-2\ \mathrm{rad/s^2}$.

$$\mathbf{M}_b = \sum_i \boldsymbol{\rho}_i \times [0,0,-T_i]^{\mathrm{T}}$$

- **Symbols:** $\boldsymbol{\rho}_i$ is rotor i's position relative to the center of mass; $[0,0,-T_i]^{\mathrm{T}}$ is its upward thrust in FRD.

Torque signs follow the position–force cross product; motor numbers alone do not define them.

4.5.3 Progressive Single-Axis Rotation: Three Torque Intervals

Three Torque Intervals Return Angular Velocity to Zero

- **Input schedule:** −0.10 N·m at 0–0.3 s; +0.10 N·m at 0.3–0.6 s; zero torque thereafter.

```
if t<.3
   forces([1 3])=[4.7;5.1];
elseif t<.6
   forces([1 3])=[5.1;4.7];
end
```

$$M_y(t) = \begin{cases} -0.10\ \text{N m}, & 0 \le t < 0.3\ \text{s} \\ +0.10\ \text{N m}, & 0.3 \le t < 0.6\ \text{s} \\ 0, & t \ge 0.6\ \text{s} \end{cases}$$

- **Read the code:** forces([1 3]) selects front and rear, in that order; left and right remain at 4.9 N. Call this branch at the midpoint of each time-step interval.

Reverse torque first removes angular velocity; attitude then stops at a new angle.

4.5.3 Progressive Single-Axis Rotation: Angular-Velocity Check

Check Torque Integration with Angular Velocity First

- **Run:** r = modeling_stages('single_axis'); results are in outputs/single_axis.

- Slope is angular acceleration.
  Angular velocity reaches −0.6 rad/s at 0.3 s, then returns linearly to zero. Rotation stops after 0.6 s.
- Area is pitch-angle change.
  The two triangles each contribute −0.09 rad: total −0.18 rad (about −10.31°).

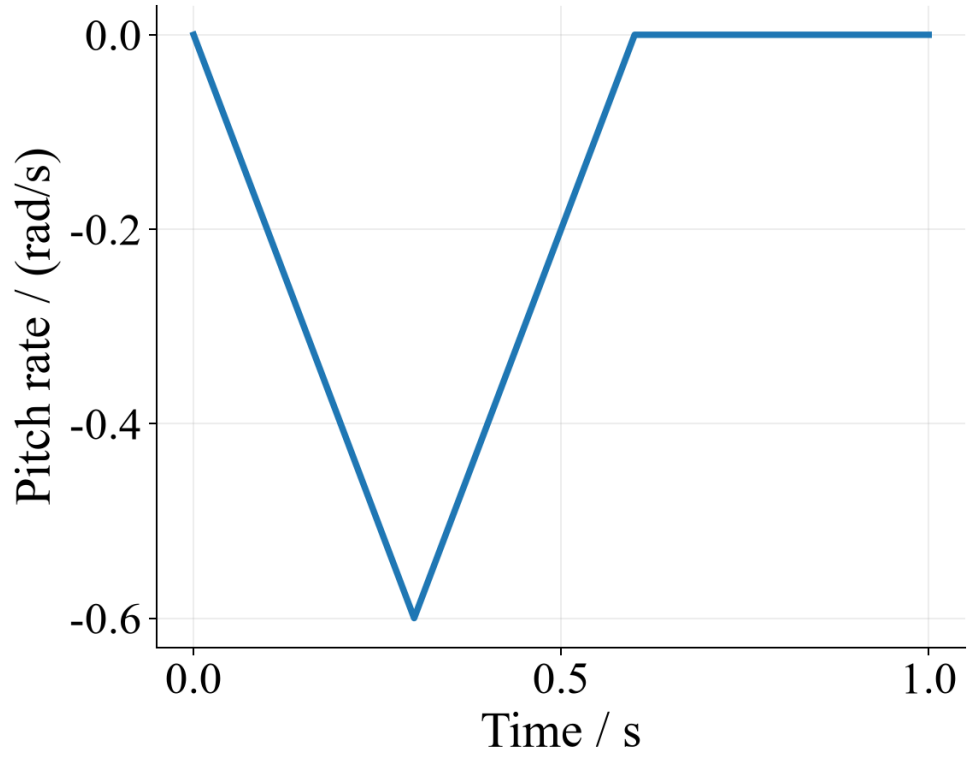


Figure: Body pitch angular velocity in the single-axis stage

4.5.3 Progressive Single-Axis Rotation: Final Pitch Angle

Zero Angular Velocity Leaves a New Pitch Angle

- **Final pitch:** about −10.31°; error from analytical −0.18 rad is below 10^{-6} rad.

- After 0.3 s, the nose keeps lowering more slowly. Rotation stops at 0.6 s when angular velocity reaches zero. Restoring level attitude requires another reverse maneuver.

- **Parameter change:** Double pitch inertia at fixed torque to halve the angle. In a copy, set P.J(2,2)=0.1; change both 0.18 values (analytical value and error expression) to 0.09.

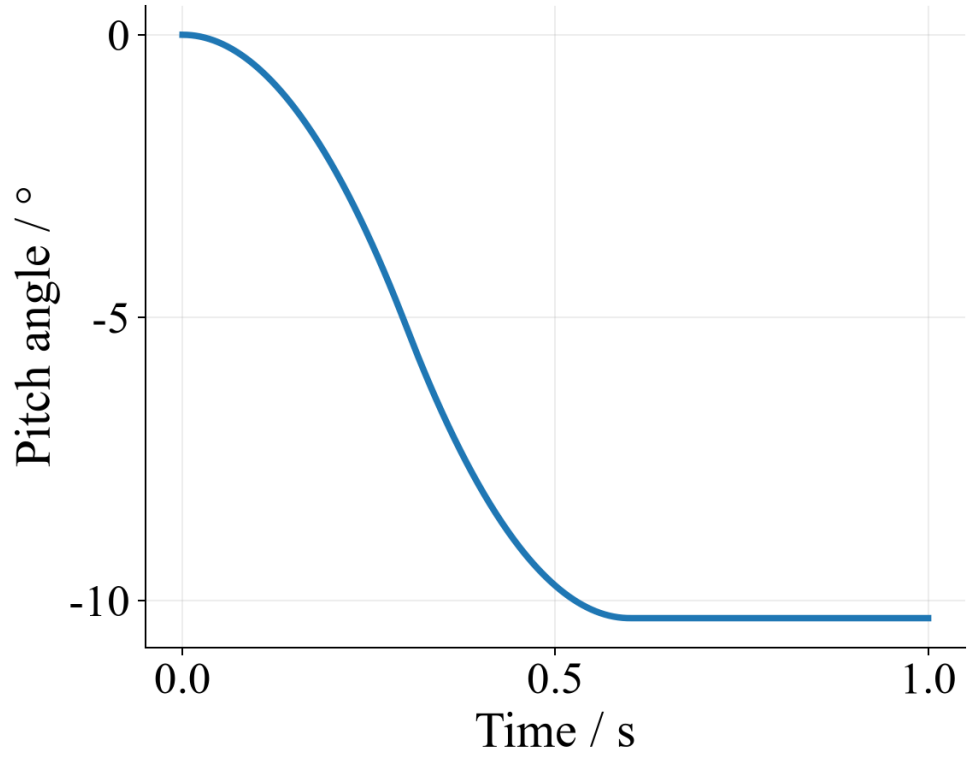


Figure: Pitch in the same single-axis experiment

Chapter Outline

Implement and Verify a Multicopter Motion Model Step by Step

Chapter 4
Vehicle Motion Modeling and Simulation

4.1 Modeling Fundamentals and Kinematics
4.2 Rigid-Body Dynamics and Solvers
4.3 Propulsion, Forces, and Observations
4.4 Trim, Linearization, and Dynamics
4.5 Multicopter Modeling and Simulation
4.6 Measurement and Static Identification
4.7 Dynamic Identification and Refinement
4.8 Typical Vehicles and Extensions
4.9 Modeling Tools and Deployment
4.10 API and SDK Modeling Interfaces
4.11 Examples and Integrated Practice
4.12 Exercises and Course Projects

6 Units in This Section

4.5.1 Program Objects, States, and Hierarchy
Textbook: Secs. 4.1.3.1–4.1.3.2, 4.2.3

4.5.2 A Minimal Vertical-Motion Program
Textbook: Sec. 4.1.4.1, Translational Dynamics

4.5.3 Progressive Single-Axis Rotation
Textbook: Secs. 4.1.4.1, 4.1.4.4

4.5.4 Three-Dimensional Rigid-Body Integration
Textbook: Secs. 4.1.4.1, 4.1.4.4

4.5.5 Propulsion, Environment, and Observation Extensions
Textbook: Secs. 4.1.4.2, 4.1.4.3, 4.1.4.6

4.5.6 Template Comparison and Acceptance Checks
Textbook: Secs. 4.2.3, 4.2.4

4.5.4 Three-Dimensional Rigid-Body Integration: Thirteen States

Store Thirteen States in Four Groups

- **Position and velocity:** Columns 1–3 are NED position in m; columns 4–6 are NED linear velocity in m/s.
- **Attitude:** Columns 7–10 are the Hamilton quaternion in wxyz order, mapping body FRD to ground NED. The unit-norm constraint removes one independent degree of freedom.

- **Angular velocity:** Columns 11–13 are body-frame angular velocity in rad/s, not the derivatives of the three Euler angles.
- **Initial state:** Set all entries to zero, then set $x_3 = -1$ and $x_7 = 1$: height 1 m, level, facing north, and at rest.

State order is part of the program interface; every function must follow it.

4.5.4 Three-Dimensional Rigid-Body Integration: Load Function

The Load Function Sums Loads Without Advancing Motion

- **Inputs:** forces contains four upward thrusts; P.rotor_position contains positions relative to the center of mass. Matching rows refer to the same motor.
- **Outputs:** Fb is total body-frame force; Mb is total torque about the center of mass. The symmetric teaching commands cancel yaw reaction torques between counter-rotating pairs.

```
fi = [zeros(4,2), -forces(:)];
Fb = sum(fi,1)';
Mb = sum(cross(P.rotor_position,fi,2),1)';
```

Separate load mapping from integration to check each rotor's contribution.

4.5.4 Three-Dimensional Rigid-Body Integration: Translational Equations

Transform Force, Then Add Gravity Once

- **Attitude matrix:** rotation_matrix reads the wxyz quaternion and returns R, the body-to-ground rotation matrix.
- **Force convention:** Fb excludes gravity. Add down gravitational acceleration once in the derivative function; do not add it again in the load function.

```
R = rotation_matrix(x(7:10));
dx(1:3) = x(4:6);
dx(4:6) = [0;0;P.g] + R*Fb/P.mass;
```

$$\dot{\mathbf{p}}_e = \mathbf{v}_e, \quad \dot{\mathbf{v}}_e = g\,\mathbf{e}_3 + \mathbf{R}\mathbf{F}_b/m$$

- **Code mapping:** [0;0;P.g] is $g\,\mathbf{e}_3$, with $\mathbf{e}_3$ the down unit vector; R*Fb/P.mass is $\mathbf{R}\mathbf{F}_b/m$.

Transform body thrust to NED before combining it with ground-frame gravity.

4.5.4 Three-Dimensional Rigid-Body Integration: Rotation and Attitude Equations

Dynamics and Attitude Kinematics Share Body Angular Velocity

- **Dynamics:** Retain angular-momentum coupling $\boldsymbol{\omega} \times (J\boldsymbol{\omega})$ in 3D motion. It vanishes only in the single-axis check.
- **Kinematics:** The quaternion derivative uses the Hamilton product of q and the pure angular-velocity quaternion. Frame conventions determine left/right multiplication order.

```
omega = x(11:13);
dx(7:10) = .5*quat_product(x(7:10),[0;omega]);
dx(11:13) = P.J\(Mb-cross(omega,P.J*omega));
```

$$\mathbf{J}\dot{\boldsymbol{\omega}}_b = \mathbf{M}_b - \boldsymbol{\omega}_b \times (\mathbf{J}\boldsymbol{\omega}_b)$$

$$\dot{\mathbf{q}} = \frac{1}{2}\,\mathbf{q} \otimes [0, \boldsymbol{\omega}_b^{\mathrm{T}}]^{\mathrm{T}}$$

- **Code mapping:** P.J\(…) solves a linear system without inversion; quat_product implements $\otimes$.

Net torque changes body angular velocity, which then changes attitude.

4.5.4 Three-Dimensional Rigid-Body Integration: Hamilton Product

Compute Scalar and Vector Parts of the Hamilton Product

- **Groups:** a(1) and b(1) are scalar parts; a(2:4) and b(2:4) are vector parts. dot and cross are the dot and cross products.
- **Call:** In attitude differentiation, b=[0;omega]. Use angular velocity in rad/s, with the preceding 0.5 factor in the quaternion derivative.

```
q = [a(1)*b(1)-dot(a(2:4),b(2:4)); ...
     a(1)*b(2:4)+b(1)*a(2:4) ...
     + cross(a(2:4),b(2:4))];
```

$$\mathbf{a} \otimes \mathbf{b} = \begin{bmatrix} a_0 b_0 - \mathbf{a}_v \cdot \mathbf{b}_v \\ a_0 \mathbf{b}_v + b_0 \mathbf{a}_v + \mathbf{a}_v \times \mathbf{b}_v \end{bmatrix}$$

- **Mapping:** a_0 is a(1), and $\mathbf{a}_v$ is a(2:4). The first row is scalar; the second is a 3D vector.

The cross product preserves rotation order; elementwise multiplication cannot replace it.

4.5.4 Three-Dimensional Rigid-Body Integration: Four Evaluations

Four Evaluations Advance One Time Step

- **Fourth-order Runge–Kutta (RK4):** f(x) abbreviates the derivative function with this step's input and parameters fixed; h is the step duration. Combine four evaluations with weights.

```
k1 = f(x);
k2 = f(x+h*k1/2);
k3 = f(x+h*k2/2);
k4 = f(x+h*k3);
next = x+h*(k1+2*k2+2*k3+k4)/6;
next(7:10) = next(7:10)/norm(next(7:10));
```

$$\mathbf{x}_{k+1} = \mathbf{x}_k + \frac{h}{6}(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4)$$

- **Slopes:** k1 uses the start; k2 and k3 use two midpoint trials; k4 uses an endpoint trial. Weight them 1:2:2:1.

Fourth-order RK improves accuracy with trial states and commits the actual state once per step.

4.5.4 Three-Dimensional Rigid-Body Integration: Input Discontinuities

Input Switching Times Are Part of the Numerical Setup

- **Problem:** If one RK step crosses a thrust switch, intermediate evaluations may mix inputs from both sides and bias the endpoint.
- **Implementation:** Include events such as 0.5, 1.5, and 2.5 s explicitly in the time grid. Use the piecewise constant command selected at the interval midpoint.
- **States and logs:** States remain continuous at an input switch. Logged derivatives use the new input at the switch, so load curves may jump.
- **Time-step comparison:** When halving dt, retain the same events, initial state, and input, and compare at identical model times to assess integration error.

Split steps at input events so each integration interval uses one control input.

4.5.4 Three-Dimensional Rigid-Body Integration: North-Velocity Check

Run the Full Rigid Body and Track North Velocity

- **Run:** r = modeling_stages('rigid_body'); output has 13 states and is integrated to 5.8 s.
- **Schedule:** Nose-down motion starts at 3.0 s; tilt is held at 3.6–4.2 s, followed by a reverse maneuver to restore level attitude.
- **Computed result:** After leveling at 4.8 s, north velocity is still about 2.10738 m/s. This baseline has no drag, so northward motion continues.

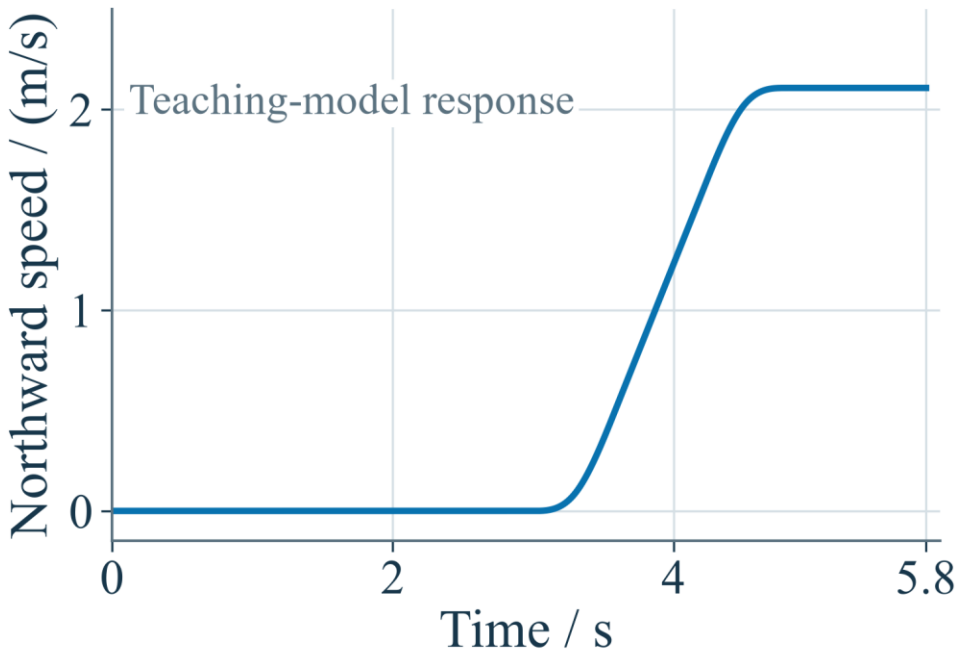


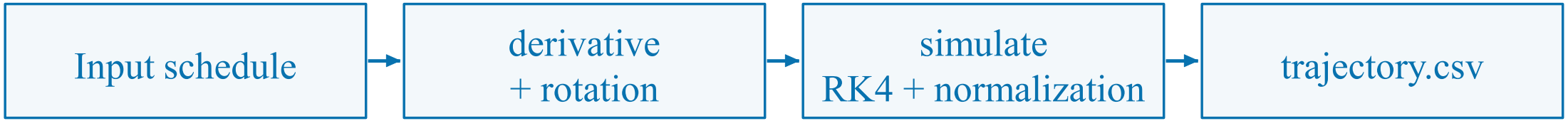
Figure: North velocity, full rigid-body stage

Restoring 3D translation lets tilted thrust produce horizontal acceleration and accumulated velocity.

4.5.4 Three-Dimensional Rigid-Body Integration: 3D Program and Results

Run the 3D Program and Read Three Quantities from One CSV

- **Entry point:** simulation.py (Python and NumPy required) shares the rigid_body parameters, initial state, and input: 2 kg, initial height 1 m, zero drag and rotational damping.



- **Run Python:** Write trajectory.csv

```
python simulation.py --out-dir outputs
```

- **Read in MATLAB:** Plot height from this CSV

```
D = readtable('outputs/trajectory.csv');
plot(D.t_s,D.height_m);
```

height_m (height)	pitch_deg (pitch angle)	v_N_mps (north velocity)
2.5 s: 2 m, zero velocity	3.6 s: about −10.31°, rotation stopped	4.8 s: about 2.11 m/s, still northward

Figure: Function roles and three observables from the same CSV

Chapter Outline

Implement and Verify a Multicopter Motion Model Step by Step

Chapter 4
Vehicle Motion Modeling and Simulation

4.1 Modeling Fundamentals and Kinematics
4.2 Rigid-Body Dynamics and Solvers
4.3 Propulsion, Forces, and Observations
4.4 Trim, Linearization, and Dynamics
4.5 Multicopter Modeling and Simulation
4.6 Measurement and Static Identification
4.7 Dynamic Identification and Refinement
4.8 Typical Vehicles and Extensions
4.9 Modeling Tools and Deployment
4.10 API and SDK Modeling Interfaces
4.11 Examples and Integrated Practice
4.12 Exercises and Course Projects

6 Units in This Section

4.5.1 Program Objects, States, and Hierarchy
Textbook: Secs. 4.1.3.1–4.1.3.2, 4.2.3

4.5.2 A Minimal Vertical-Motion Program
Textbook: Sec. 4.1.4.1, Translational Dynamics

4.5.3 Progressive Single-Axis Rotation
Textbook: Secs. 4.1.4.1, 4.1.4.4

4.5.4 Three-Dimensional Rigid-Body Integration
Textbook: Secs. 4.1.4.1, 4.1.4.4

4.5.5 Propulsion, Environment, and Observation Extensions
Textbook: Secs. 4.1.4.2, 4.1.4.3, 4.1.4.6

4.5.6 Template Comparison and Acceptance Checks
Textbook: Secs. 4.2.3, 4.2.4

4.5.5 Propulsion, Environment, and Observation Extensions: Module Interfaces

Add Motors and Environment Through the Same Rigid-Body Interfaces

- **New propulsion states:** Four motor speeds become states 14–17. Thrust follows actual speed squared and no longer equals the command immediately.
- **New environment:** A 1 m/s northward wind changes relative airflow. Signed quadratic drag on three axes enters the total body-frame force.
- **Keep comparisons aligned:** Retain the thrust-target schedule, 2 kg parameters, and initial state to observe how propulsion and environment alter the baseline.

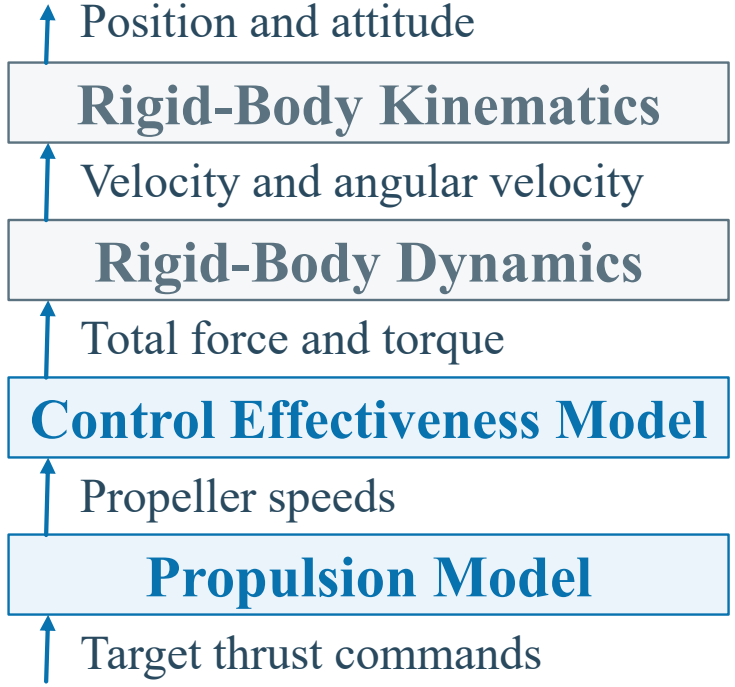


Figure: Four-layer motion model

Each extension should identify where new states, inputs, and loads enter the calculation chain.

4.5.5 Propulsion, Environment, and Observation Extensions: Motor States

Four Rotor-Speed States Represent Propulsion Lag

- **Target and actual:** Thrust coefficient $c_{\mathrm{T}} = 10^{-5}\ \mathrm{N/(rad/s)^2}$ (defined in slide deck 4.3; Ct in code), motor time constant $\tau_m = 0.05\ \mathrm{s}$. Derive target speed from target thrust.
- **Aligned initial state:** Start each propeller at the steady speed for 4.9 N, matching the initial ideal rigid-body case.

$$\tau_m \dot{\omega}_i + \omega_i = \omega_{t,i}, \quad T_i = c_{\mathrm{T}}\, \omega_i^2$$

```
motor = x(14:17);
target = sqrt(max(command,0)/P.Ct);
dx(14:17) = (target-motor)/P.motor_tau;
forces = P.Ct*motor.^2;
```

Current speed sets current thrust; target speed sets how fast and in which direction speed changes.

4.5.5 Propulsion, Environment, and Observation Extensions: Relative Air Velocity

Subtract Velocities in the Same Frame First

- **Subtract first:** vNED and wind_NED are both in ground NED. Their difference gives air-relative velocity, then transform it to the body frame.
- **Calculate force:** Apply a signed quadratic law on each axis. abs supplies magnitude; the leading minus sign opposes motion. drag_k is a teaching coefficient.

```
vAirBody = R'*(vNED-P.wind_NED);
Fdrag = -P.drag_k*vAirBody.*abs(vAirBody);
Fb = Fb + Fdrag;
```

Drag opposes motion relative to air; zero ground speed does not imply zero air-relative speed.

4.5.5 Propulsion, Environment, and Observation Extensions: Stationary in Wind

Check Direction and Magnitude While Stationary in Wind

- **Conditions:** Level body facing north, zero ground velocity, wind 1 m/s north. The drag coefficient on each axis is 0.10 N/(m/s)².
- **Direction:** Vehicle velocity relative to air is southward, with body-forward component −1 m/s. Drag is northward, +0.10 N.
- **Acceleration:** At 2 kg, initial north acceleration is +0.05 m/s², matching the initial value in the extended modeling_stages stage.

$$F_{D,N} = -0.10(-1)|-1| = 0.10\ \text{N}, \quad a_N = F_{D,N}/2 = 0.05\ \text{m/s}^2$$

One hand-calculable wind case checks both the frame transform and the drag sign.

4.5.5 Propulsion, Environment, and Observation Extensions: Load Curves

Propulsion Lag Changes the Load Curve First

- **Run:**
  modeling_stages('actuator_environment');
  output contains 17 states.
- **Read the plot:** Targets are unchanged, but motors make actual upward force change gradually at switches. Total force also includes body drag.
- **Local check:** For one step input, rotor-speed change after 0.05 s should reach about 63.2% of its total change; then check the speed-squared thrust.

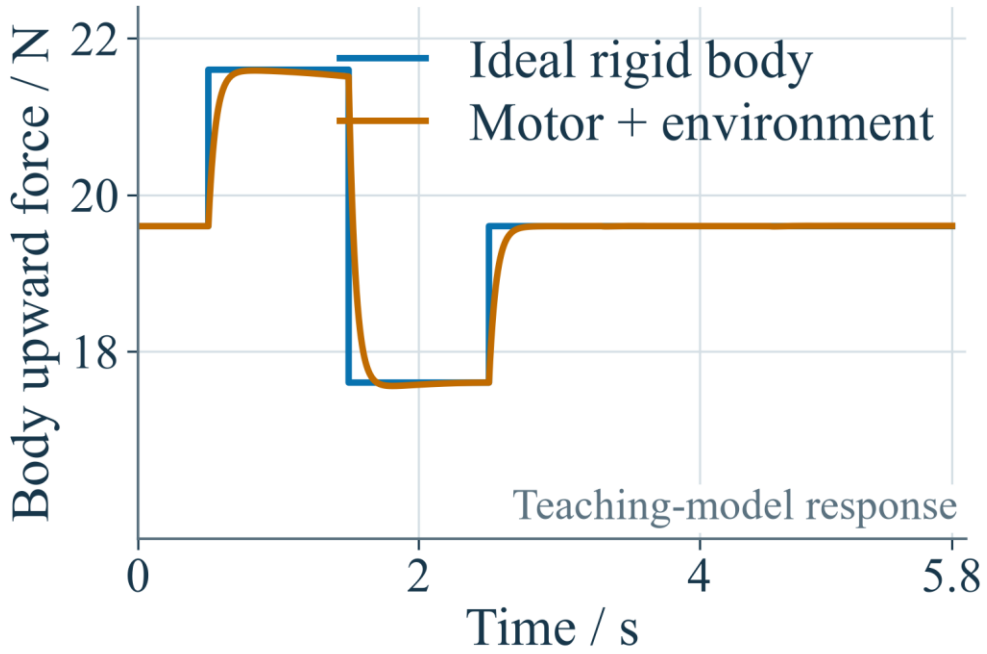


Figure: Total upward force under the same commands

Propulsion lag changes loads first, then velocity and trajectory through integration.

4.5.5 Propulsion, Environment, and Observation Extensions: Endpoint Comparison

The Same Command Schedule Can Produce Different Endpoints

- **At 5.8 s:** Ideal rigid-body height is about 1.68103 m; the motor-and-environment extension gives about 1.54633 m.
- **Why:** During tilt, vertical thrust ≈19.28 N is below weight 19.6 N; down velocity persists after leveling. The extension adds propulsion lag, wind, and drag.
- **Tests:** Disable wind/drag separately or shorten the motor time constant. Save each run separately. Removing a dominant cause should move the extended curve toward the ideal.

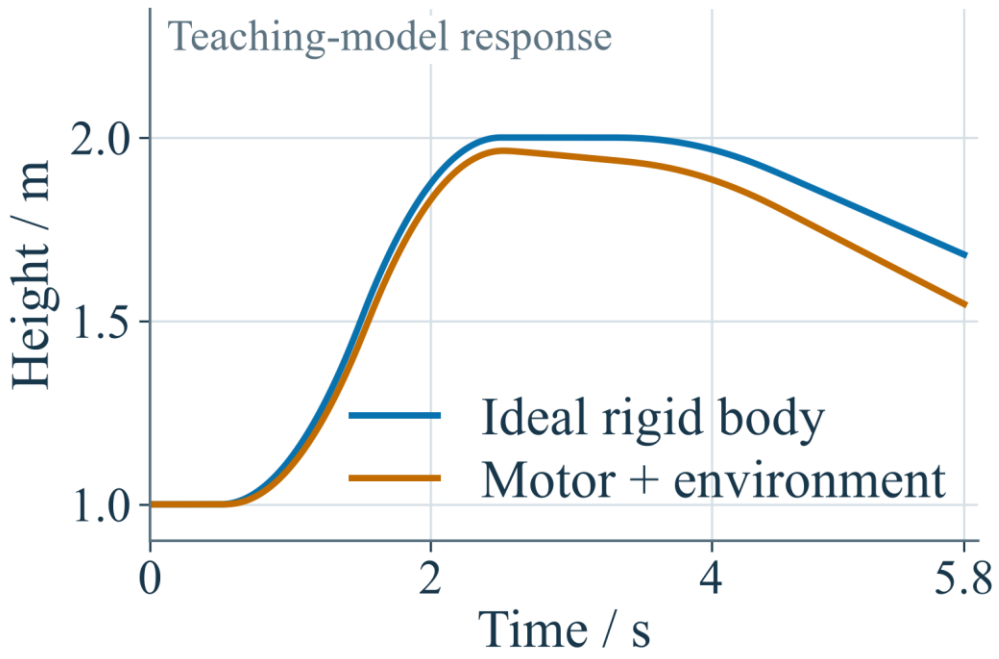


Figure: Height in the ideal and extended models

After adding physics, recheck endpoint conclusions that depended on earlier simplifications.

4.5.5 Propulsion, Environment, and Observation Extensions: Ideal Inertial Observations

Compute Ideal Inertial Readings from Motion States

- **Sensor location:** Start with an ideal sensor at the center of mass, its axes aligned with FRD. Extend later to offset mounting and noise.
- **Specific force:** Subtract gravity from NED acceleration, then transform to the body frame. In hover, this gives $[0,0,-g]^{\mathrm{T}}$.
- **Gyroscope:** Ideal angular velocity is x(11:13). If sensor and body axes differ, transform it with the mounting matrix.

$$\mathbf{f}_b = R^{\mathrm{T}}(\mathbf{a}_e - [0,0,g]^{\mathrm{T}}), \qquad \boldsymbol{\omega}_{\mathrm{gyro}} = \boldsymbol{\omega}_b$$

Sensor readings derive from motion truth; ground acceleration is not body specific force.

Chapter Outline

Implement and Verify a Multicopter Motion Model Step by Step

Chapter 4
Vehicle Motion Modeling and Simulation

4.1 Modeling Fundamentals and Kinematics
4.2 Rigid-Body Dynamics and Solvers
4.3 Propulsion, Forces, and Observations
4.4 Trim, Linearization, and Dynamics
4.5 Multicopter Modeling and Simulation
4.6 Measurement and Static Identification
4.7 Dynamic Identification and Refinement
4.8 Typical Vehicles and Extensions
4.9 Modeling Tools and Deployment
4.10 API and SDK Modeling Interfaces
4.11 Examples and Integrated Practice
4.12 Exercises and Course Projects

6 Units in This Section

4.5.1 Program Objects, States, and Hierarchy
Textbook: Secs. 4.1.3.1–4.1.3.2, 4.2.3

4.5.2 A Minimal Vertical-Motion Program
Textbook: Sec. 4.1.4.1, Translational Dynamics

4.5.3 Progressive Single-Axis Rotation
Textbook: Secs. 4.1.4.1, 4.1.4.4

4.5.4 Three-Dimensional Rigid-Body Integration
Textbook: Secs. 4.1.4.1, 4.1.4.4

4.5.5 Propulsion, Environment, and Observation Extensions
Textbook: Secs. 4.1.4.2, 4.1.4.3, 4.1.4.6

4.5.6 Template Comparison and Acceptance Checks
Textbook: Secs. 4.2.3, 4.2.4

4.5 Multicopter Modeling and Simulation

4.5.6 Template Comparison and Acceptance Checks: Actual Model Entry Point

The Actual Example Connects These Equations into a Dynamics Model

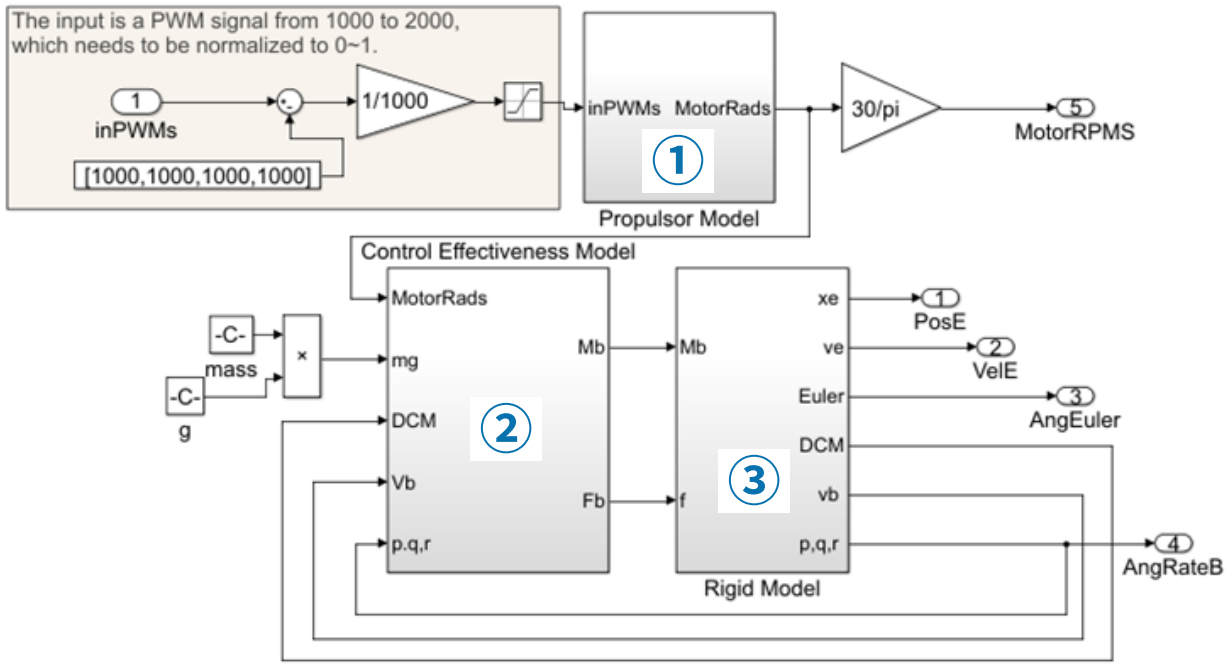


Figure: Overall quadcopter dynamics model in e12

- ① **Propulsion:** Normalize pulse-width modulation (PWM) commands to obtain each motor's speed.
- ② **Control effectiveness:** Speed, mass, and gravity give total force (gravity included) and torque.
- ③ **Rigid body:** Integrate position, velocity, and attitude. A direction cosine matrix (DCM), like **R**, represents attitude.
- **Conventions:** This 1.4 kg model integrates body velocity; its parameters differ from the 2 kg teaching example.

Actuators generate forces and torques; the rigid-body module integrates position and attitude.

4.5.6 Template Comparison and Acceptance Checks: Files and Modules

Match Example Files to This Section's Equations

- **Example directory:** [installation directory]/RflySimAPIs/4.RflySimModel/2.AdvExps/e12_ModelTempId/3.ModelingTemp/code/.



Figure: From files to equations

- **Run:** In MATLAB, run Init.m, then open and run dynamics.slx. ode4 is fourth-order Runge–Kutta, with step 0.001 s (0.002 s here). Total force includes gravity.
- **Behavior and diagnosis:** With pwm = 0, the body falls from 100 m; drag gradually reduces downward acceleration. For compilation errors, check mex compiler configuration first.

Understanding the states and equations is the starting point for reading a simulation model.

4.5.6 Template Comparison and Acceptance Checks: Subsystem Comparison

Match Teaching Functions to Template Subsystems

- **M0:** Exp1_MinModelTemp.slx in 1.BasicExps/e0_MinModelTemp is the deployment starting point. It uses a six-degree-of-freedom (6DOF) Euler-angle block, mass 1.515 kg, and its own initialization script for hover.
- **D0:** 2.AdvExps/e12_ModelTempId/3.ModelingTemp/code/dynamics.slx supports equation-by-equation comparison and requires the matching version's advanced resources.
- **Trace the layers:** Find motor states, thrust/torque synthesis, then rigid-body derivatives and integration. Check ports and units against the actual connections.
- **Comparison conditions:** Self-check the teaching model, M0, and D0 separately. To compare curves, align parameters, initial state, input, environment, and model time.

Match equations, signals, and parameter sources; the template and teaching model remain distinct.

4.5.6 Template Comparison and Acceptance Checks: Free-Fall Prediction

Experiment 1: Predict Free Motion Without Thrust

- **Ideal conditions:** $z_0 = -100\,\text{m}$, $g = 9.8\ \text{m/s}^2$, zero initial velocity; no thrust, drag, or contact.
- **At 1 s:** down velocity 9.8 m/s, displacement 4.9 m, $z_D = -95.1\,\text{m}$. The positive-down NED coordinate increases.
- **Program:** The zero case starts at $z_D = -10\,\text{m}$ with g=9.81 m/s². At 1 s, $z_D = -5.095\,\text{m}$ and down velocity 9.81 m/s. Run: "Zero Input," teaching unit 4.11.2.

$$v_D = gt, \quad z_D = z_0 + 1/2\, gt^2$$

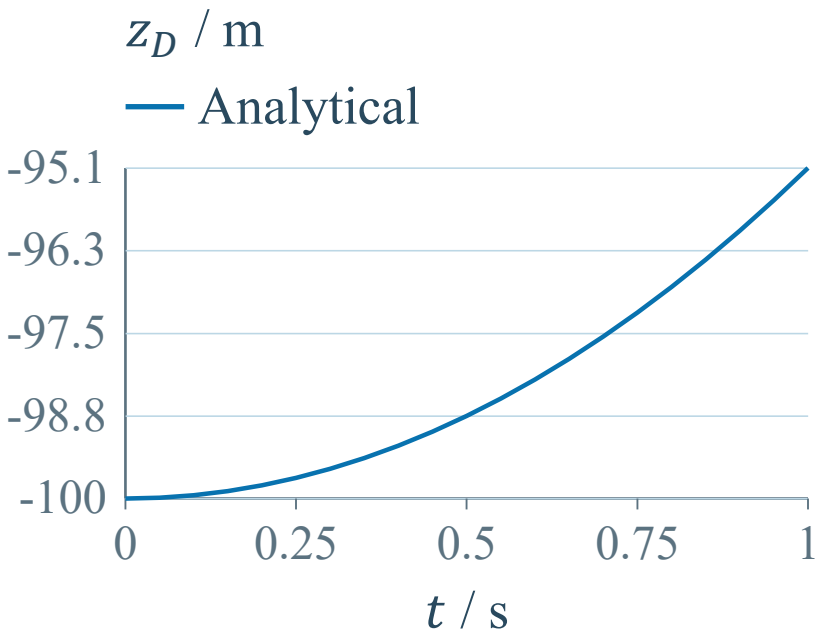


Figure: Analytical free-fall baseline (teaching calculation)

With no contact or drag, downward velocity increases linearly in ideal free fall.

4.5.6 Template Comparison and Acceptance Checks: Free-Fall Diagnosis

Check Loads First When Free Fall Deviates

- **No fall:** Check whether ground contact already balances gravity, or test connections hold position or velocity fixed.
- **Velocity levels off:** Check whether drag grows with speed. A physical model retaining drag need not maintain gravitational acceleration indefinitely.
- **Wrong direction:** Check NED signs, frame transforms, and duplicate gravity. Changing the time step cannot fix these structural errors.
- **Full diagnosis:** See “Fault Isolation” in teaching unit 4.11.2 for layer-by-layer M0 checks.

Check contact, drag, and gravity conventions before assessing free-fall integration error.

4.5.6 Template Comparison and Acceptance Checks: Hover Hand Calculation

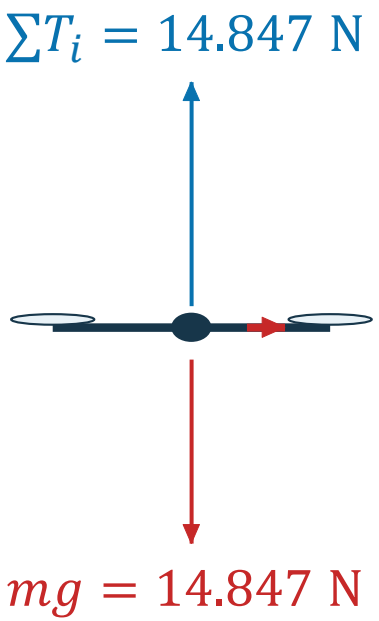
Experiment 2: Estimate M0 Hover by Hand

- **Hand estimate:** 1.515 kg, with $g = 9.8\ \mathrm{m/s^2}$ on this slide; total thrust 14.847 N, or 3.71175 N per propeller.
- **Invert:** use $c_T = 1.681 \times 10^{-5}$ for speed, then the M0 map $\omega = 842.1u + 22.83$ (slide deck 4.3) for input. Program g=9.81 gives 470.14 rad/s and u≈0.53118.

$$\omega_h = \sqrt{3.71175/(1.681 \times 10^{-5})} = 469.90\ \mathrm{rad/s}$$

$$u_h = (469.90 - 22.83)/842.1 = 0.53090$$

- **Zero-speed rotors:** motor transient, not immediate hover. Both initial states: “Balance and Startup,” teaching unit 4.11.2.



Level, four symmetric rotors

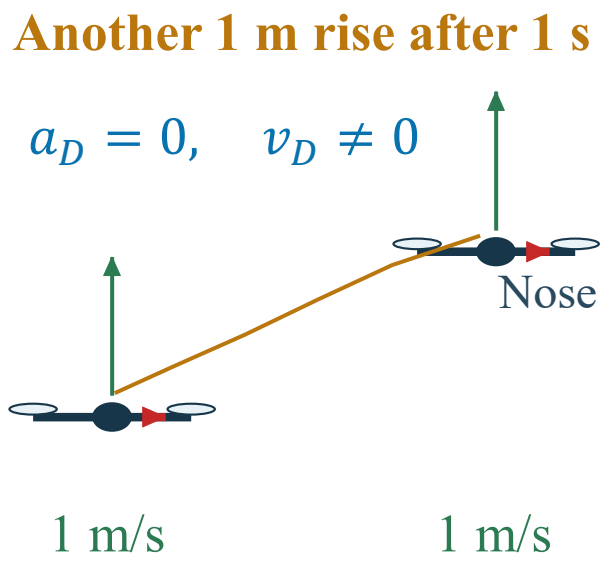
Static hover requires zero net force, zero net torque, and zero initial velocity.

4.5.6 Template Comparison and Acceptance Checks: Symmetric Inputs

Why Do Symmetric Input Increases Mainly Affect Vertical Motion?

- **Symmetric thrust increase:** Torques cancel for four level, symmetric rotors. When total thrust exceeds weight, down acceleration is negative and the aircraft accelerates upward.
- **Counterexample:** Once climbing at 1 m/s, restore thrust to weight. Net force is zero, yet upward speed remains 1 m/s after 1 s and height rises another 1 m.
- **Diagnosis:** Restoring balanced thrust does not brake motion. If velocity instantly becomes zero, check for a controller, state reset, or a target mistaken for a state.

$$a_D = g - \frac{\sum_i T_i}{m}$$



With level, symmetric rotors, net thrust changes while torques cancel on every axis.

4.5.6 Template Comparison and Acceptance Checks: Single-Motor Direction

Experiment 3: Predict a Single-Motor Increment's Direction

- In body FRD, rotor position is $(x_i, y_i, 0)$ and upward thrust is $(0,0,-T_i)$. Position crossed with force gives roll and pitch torques.
- **Signs:** More right-side thrust produces negative roll torque; more front thrust produces positive pitch torque. Reaction torque also depends on rotation direction.
- **M0:** Check X-layout positions and spin directions in slide deck 4.3; its numbering differs from this section's "+" layout.
- **Program:** motor1 adds 0.03 to input 1 from equilibrium rotor speed. See "One-Motor Increment," teaching unit 4.11.2.

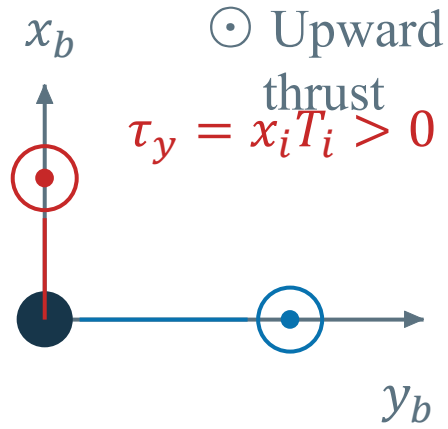


Figure: Rotor position and moment-arm torque directions

Motor position and thrust direction determine torque signs; numbering alone does not.

4.5.6 Template Comparison and Acceptance Checks: Input and State Timeline

Align Inputs, Components, and States on One Timeline

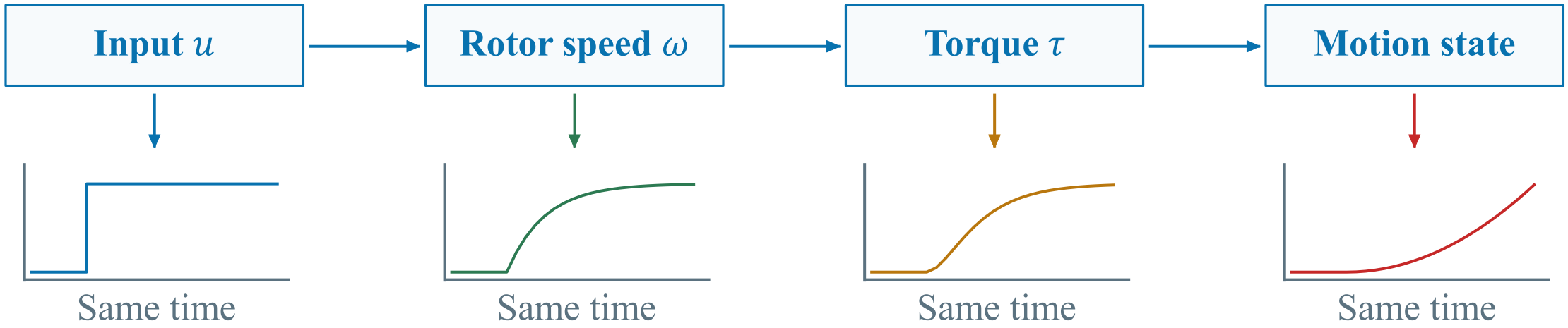


Figure: Log along the signal chain; waveforms show causal shapes only

- **Diagnosis:** Input changes, speed does not → check thresholds/connections. Correct speed, wrong torque → check layout.
- **Next:** Correct torque, wrong angular velocity → check inertia/frames. Use the same simulation time for all logs.

Finding the first layer that deviates diagnoses causes better than final position alone.

4.5.6 Template Comparison and Acceptance Checks: First-Order Motor and Time Step

Compare Time-Step Error Against a First-Order Motor Baseline

- **M0 motor:** $\tau_m = 0.0214\,\mathrm{s}, \eta = 1 - e^{-t/\tau_m}$ (slide deck 4.3); 63.2% at one time constant.
- **Same Euler equation:** One 10 ms step gives 0.4673; two 5 ms steps give 0.4127. Both cover 10 ms.
- **Error:** 0.3733 analytically; error 0.0940 → 0.0394, roughly halved (first order; slide deck 4.2).
- **M0 time-step comparison:** teaching unit 4.11.2.

$$\eta_{k+1} = \eta_k + \frac{h}{\tau_m}(1 - \eta_k)$$

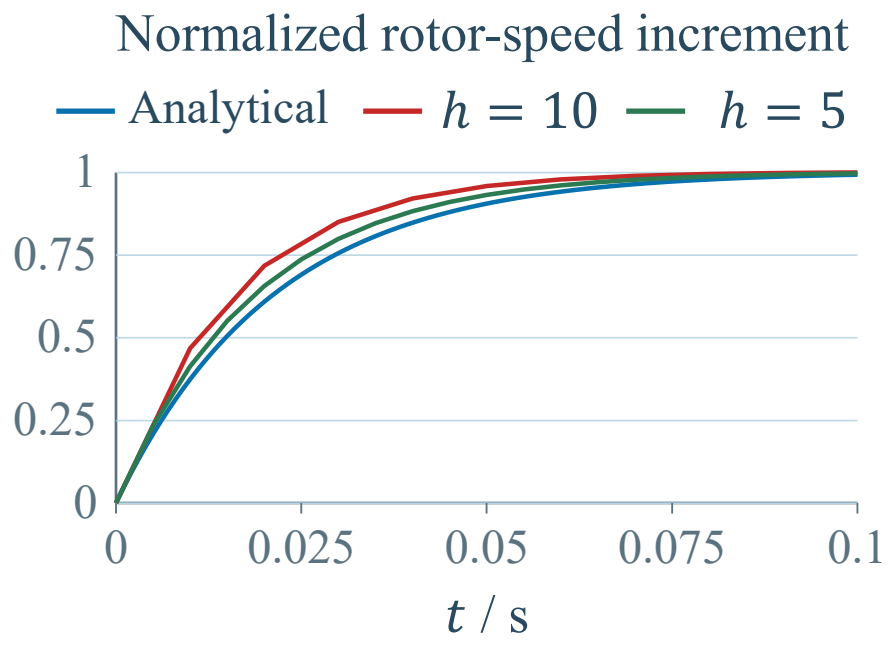


Figure: Time steps in ms; teaching calculations

Fix the physical problem first, then compare numerical results at the same simulation times.

4.5.6 Template Comparison and Acceptance Checks: Integrated Model Boundary

An Integrated Model Combines Vehicle Motion and a Controller

- **Textbook Sec. 4.2.4:** An integrated model implements "vehicle model + controller." The 2 kg teaching program and M0/D0 compute motion only and are not yet integrated models.
- **Implementation:** Embed control in Simulink. The generated dynamic-link library (DLL) handles commands and motion. Check control interfaces, state definitions, and scheduling periods together.
- **Scope:** Software-in-the-loop (SITL; textbook SIL) and hardware-in-the-loop (HITL; textbook HIL) close the PX4 loop in the actual control workflow. Integrated models can run without PX4 for accelerated, parallel training; they cannot replace flight-controller validation.
- Unreal Engine (UE) Blueprint integrated models use UE physics for contact/collision and a simplified high-level controller. This differs from the DLL implementation.

A motion model computes physical response; an integrated model also closes the control loop.

4.5.6 Template Comparison and Acceptance Checks: Three Task Levels

Complete Baseline, Modification, and Actual-Model Tasks

- **Baseline:** Run all four stages; save CSV files, metrics, and plots. Check vertical endpoints, single-axis angular velocity, and angle against analytical values.
- **Independent change:** Change only mass, inertia, or motor time constant; predict first. Doubling pitch inertia halves the single-axis angle. In a copy, replace both 0.18 values with 0.09, retaining signs.
- **Actual model:** Run m0_offline_checks as in teaching unit 4.11.2. Use report.json parameters to predict and compare free fall, hover, and single-motor direction.
- **Deliverables:** Record program version, parameters, input, initial state, time step, and output path. Include at least one hand calculation, one state curve, and one error measure.

Code, numbers, and physical explanations must agree for a reproducible modeling implementation.

Classroom Exercise

Predict New Cases Before Running Them

- **Q1—Mass:** Retain the 4.5.2 thrust schedule (19.6, 21.6, 17.6, 19.6 N) and set mass to 2.4 kg. Will it rise during the 21.6 N interval? What is its height at 3 s?
- **Q2—Duration:** Retain ± 0.10 N·m and pitch inertia 0.05 kg·m² in the single-axis stage, with each interval lasting 0.4 s. Find peak angular velocity and final pitch.
- **Q3—Time constant:** In a modeling_stages copy, set drag_k and wind_NED to zero; change motor_tau from 0.05 s to 0.1 s. When is speed 63.2%? At 1.5 s, is height above or below the ideal rigid body?
- **Procedure:** Predict first; save separately. Q1: use vertical_beginner's mass argument. Q2: change 0.3/0.6 → 0.4/0.8 in both command_forces and events. Replace −0.18 (single_axis analytical value) and +0.18 (error expression) with your prediction, retaining signs.

Writing the prediction first lets program results test your physical reasoning.

Reference Points

Predictions and Criteria for the Classroom Exercise

- **Q1:** Weight is 23.52 N. Even at 21.6 N, downward acceleration is 0.8 m/s², so it descends throughout. At 3 s, down velocity is 4.9 m/s and height about −5.5 m (no ground-contact model).
- **Q2:** Angular acceleration remains ±2 rad/s²; peak angular velocity is −0.8 rad/s. Angle scales with duration squared: final pitch about −0.32 rad (−18.3°), with zero angular velocity.
- **Q3:** Speed reaches 63.2% at 0.1 s and 95.0% at 0.3 s. Climb is delayed: height at 1.5 s is about 1.41 m (ideal: 1.50 m; at $\tau_m = 0.05$ s: about 1.45 m), and about 1.99 m at 3 s.
- **Criteria:** Check direction and scale before numerical differences. If the trend opposes prediction, check signs, units, and input timing first. Q3 values are offline teaching calculations.

Identify whether a parameter changes net force, action duration, or response speed.

Section Summary

Progressive Modeling: Method, Criteria, and Common Errors

- **Four stages:** Vertical motion → single-axis pitch → 3D rigid body → motors, wind, and ideal observations. Keep interfaces and the 13-column state order fixed; add one physical process at a time.
- **Checks:** Vertical motion—endpoint height and zero velocity; single axis—angular-velocity slope M_y/J_y and angle; 3D—north velocity retained after leveling; extensions—initial derivatives and causes of curve differences.
- **Error 1:** NED z_D is positive down; height is $H = -z_D$. Thrust excludes gravity; add gravity once in the derivative function.
- **Error 2:** Initialize the quaternion to [1,0,0,0]; an all-zero quaternion is invalid. No RK4 step may cross a thrust-switching time.
- **Error 3:** Target thrust and target speed are inputs, not states. Normalized speed compares response shapes; the thrust model needs actual speed in rad/s.