Chapter Outline

Examples and Integrated Practice

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

5 Units in This Section

4.11.1 Published Resources and Practice Routes

Textbook: Sec. 4.3.1

4.11.2 The M0 Offline Baseline

Textbook: Sec. 4.2.3

4.11.3 Code Deployment and Interface Experiments

Textbook: Secs. 4.2.3 and 4.2.4

4.11.4 Identification and Vehicle Practice

Textbook: Secs. 4.2.5.1, 4.2.5.3, and 4.3.1–4.3.2

4.11.5 Integrated Changes and Reproducible Delivery

Four cases, replacement calculations, package delivery

4.11.1 Published Resources and Practice Routes: Baseline and Branches

Establish M0, Then Choose an Extension Route

Common baseline: M0 is Section 4.9's deployment baseline, Exp1_MinModelTemp.slx (1.515 kg).

Shared M0 baseline: four cases, two steps; then choose one route

Parameter identification (4.11.4)

Resources: teaching dynamic data.
With Advanced data, also fit real static data.
Deliver: estimates and independent validation.

Deployment and interfaces (4.11.3)

Resources: DLL / initial-state interface.
Deliver: deployment or request records;
new state readback from the target instance.

Vehicle migration (4.11.4)

Resources: vehicle baseline.
Deliver: comparison of inputs, constraints,
and motion response.

Each route saves inputs, initial states, and logs with time and units.

4.11.1 Published Resources and Practice Routes: Resource Root

Open Chapter 4 Examples in Your Own Installation

- Open [installation directory]/RflySimAPIs/4.RflySimModel. Read Intro.pdf, Index.pdf, and API.pdf, then check dependencies in each example's Readme.pdf.

- **Copy the complete project:** SLX defines structure, initialization M files define parameters, DLL is the build artifact, and BAT configures launch. Record full paths for identically named files.
- Direct DLL entry: 0.ApiExps/12.DllModelImport/4.ModelLoadMannual/ModelLoadApiTest.py. See unit 4.11.3.

SLX	Structure and ports
M	Parameters and initial state
DLL	Compiled artifact
BAT	Launch configuration

Match models, parameters, and launch files to one build so results remain traceable.

4.11.1 Published Resources and Practice Routes: Resource Roles

Interface, Basic, Advanced, and Custom: Roles

- 0.ApiExps organizes interface experiments: confirm inputs/outputs, units, coordinate frames, rates, and timing.
- 1.BasicExps supplies minimal templates and basic vehicles for reading equations, changing parameters, generating DLLs, and establishing baselines.
- 2.AdvExps supplies advanced models. 3.CustExps is available only in the Full Edition and above; some examples contain compiled artifacts only. Check each Readme for dependencies.

API	Confirm interface conventions
Basic	Establish a minimal baseline
Advanced	Extend modeling capability
Custom	Choose for project conditions

Roles of the four example groups

Resources are grouped by teaching purpose; folder numbering is not a required execution order.

4.11.1 Published Resources and Practice Routes: Introductory Lab

Five Introductory Lab Folders, Five Experiment Roles

Entry: [installation directory]/RflySimAPIs/4.RflySimModel/1.BasicExps/e6_ModelingLearningLab/Readme.pdf
In the chosen subfolder, follow Readme.pdf; save to outputs or the supplied writable folder.

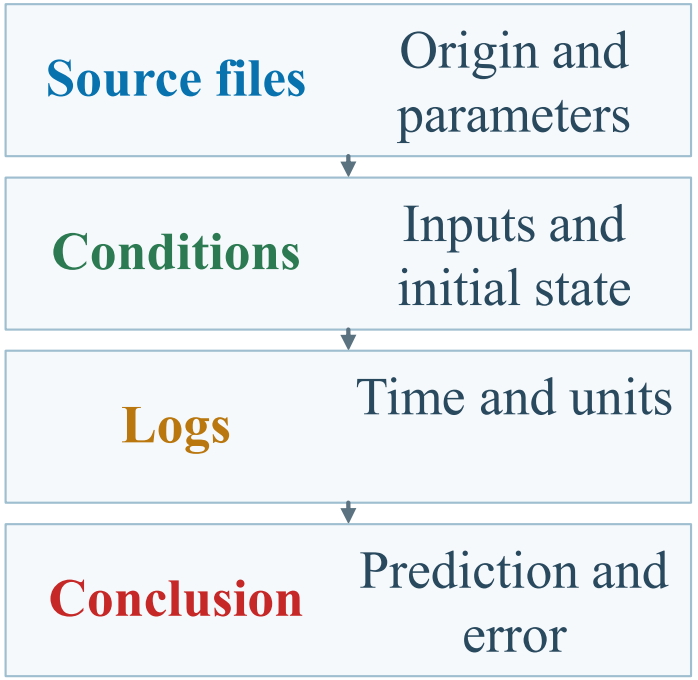
Subfolder	Experiment	Dependencies
01_VerticalRigidBody	Staged motion modeling	Base MATLAB
02_ComponentChecks	Component self-checks	Base MATLAB or Python
03_Identification	Parameter and dynamic identification	Base MATLAB; real static fitting also needs advanced data
04_VehicleKinematics	Kinematics comparison	Base MATLAB
05_M0Checks	M0 offline: four cases	Simulink and model libraries

Copy the full lab: code, guides, teaching CSVs. Each experiment explains its parameters/data.

4.11.1 Published Resources and Practice Routes: Fixed Conditions

Write an Experiment-Condition Record Before Running

- Record source path, mass/inertia, and modifications; then input dimensions/units, initial state, wind, and ground. Together they define one experiment.
- Save CSVs, comparison plots, and explanations for all four cases and both time steps. Tie conclusions to specific columns and physical times, beyond screenshots.
- Choose one extension and retain the baseline. Distinguish the validity of the drag-free baseline, full model, synthetic data, and closed-loop conditions.



Explicit inputs, initial states, and comparison quantities make predictions testable.

Chapter Outline

Examples and Integrated Practice

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

5 Units in This Section

4.11.1 Published Resources and Practice Routes

Textbook: Sec. 4.3.1

4.11.2 The M0 Offline Baseline

Textbook: Sec. 4.2.3

4.11.3 Code Deployment and Interface Experiments

Textbook: Secs. 4.2.3 and 4.2.4

4.11.4 Identification and Vehicle Practice

Textbook: Secs. 4.2.5.1, 4.2.5.3, and 4.3.1–4.3.2

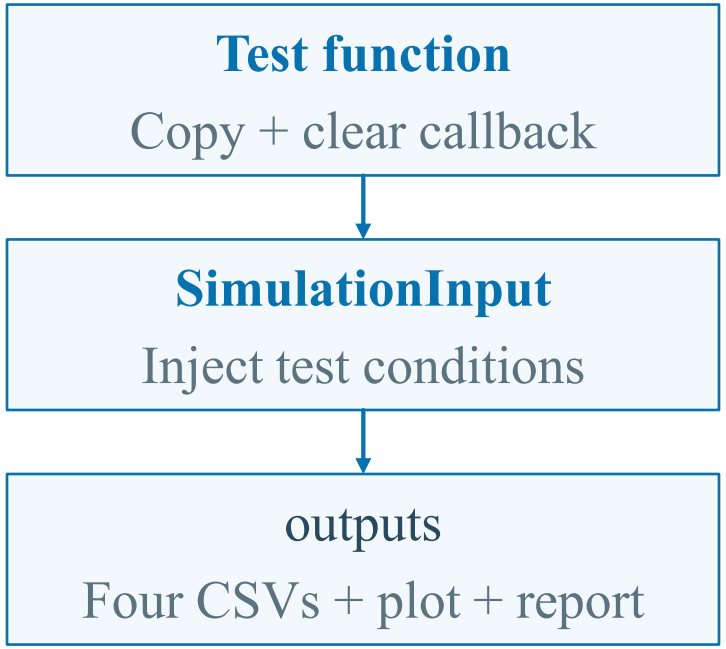
4.11.5 Integrated Changes and Reproducible Delivery

Four cases, replacement calculations, package delivery

4.11.2 The M0 Offline Baseline: Entry and Result Files

Required M0 Experiment: Open the Supplied Test Function

- M0 is Exp1_MinModelTemp.slx in 1.BasicExps/e0_MinModelTemp. In introductory lab folder 05_M0Checks, run report = m0_offline_checks.
- The function creates an M0 test copy, then runs zero, balanced, startup, and motor1. Parameter sources and modifications are recorded in report.json.
- outputs contains four CSVs, positions.png, and report.json. model_copy holds the test copy; the installed source model remains unchanged.

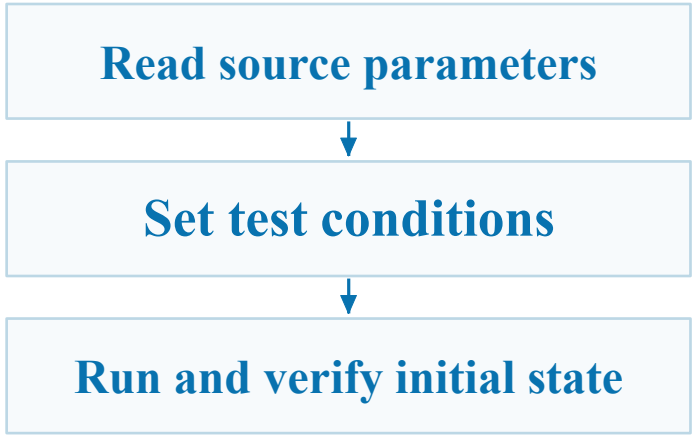


One run saves four cases. Read inputs and initial values before checking responses.

4.11.2 The M0 Offline Baseline: Initialize Once

Keep Startup Initialization from Overwriting Test States

- The source template's InitFcn runs its matching initialization file. Workspace edits to initial states may be overwritten at startup.
- The function copies the SLX and clears the copy's InitFcn. After reading source parameters, it supplies case settings and root inputs through SimulationInput.
- Launch through m0_offline_checks to supply all case parameters. Opening the copy and clicking Run alone omits these settings.
- Close any previously opened M0_OfflineLab copy before rerunning; otherwise the function's initial check stops with an error.



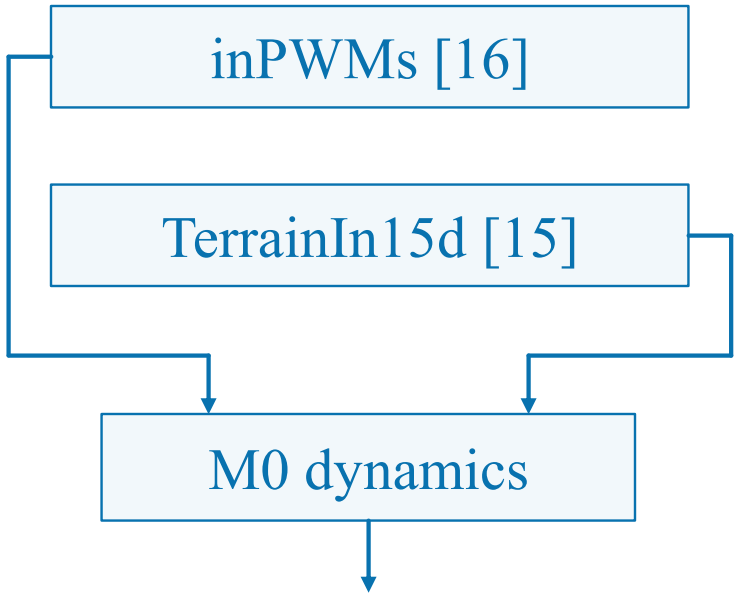
Initialization sequence in the test copy

If a startup callback overwrites the case, the result no longer represents the intended initial state.

4.11.2 The M0 Offline Baseline: Two Root Inputs

Supply Reproducible Signals to Both Root Ports

- inPWMs has 16 elements: the first four drive four motors; the rest are zero. 0.53 is a normalized command, not microseconds.
- TerrainIn15d has 15 elements: the first is ground down-coordinate 1000 m; the rest are zero. It differs from vehicle position p_D.
- Dataset contains two timeseries in root-port order. Equal endpoint values hold each input constant from time 0 through the stop time.



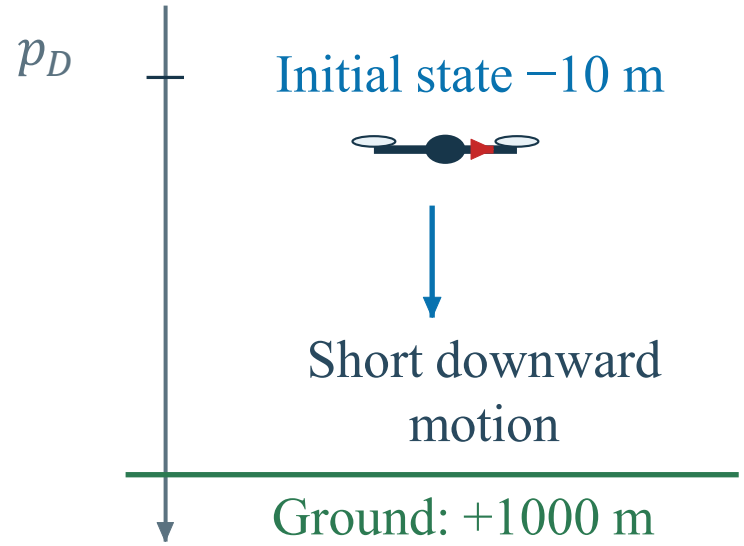
M0's two root inputs

Explicit time series supply both normalized actuator inputs and terrain inputs.

4.11.2 The M0 Offline Baseline: Analytical Test Conditions

M0 Baseline Parameters and Physical Boundaries

- M0 mass is 1.515 kg, with $g = 9.81$ m/s². Initial position is $[0,0,-10]$ m; velocity and attitude are zero.
- Ground is 1000 m down. The vehicle falls only about 4.9 m in 1 s, so the contact branch is inactive.
- Set drag and rotational damping to zero for an analytical result. Full M0 retains drag and ground-contact branches.



Initial state and ground in the analytical test copy

Analytical checks use an explicitly simplified M0 copy; the full model retains aerodynamics and contact.

4.11.2 The M0 Offline Baseline: Log Fields

Read the Log Columns That Describe Motion

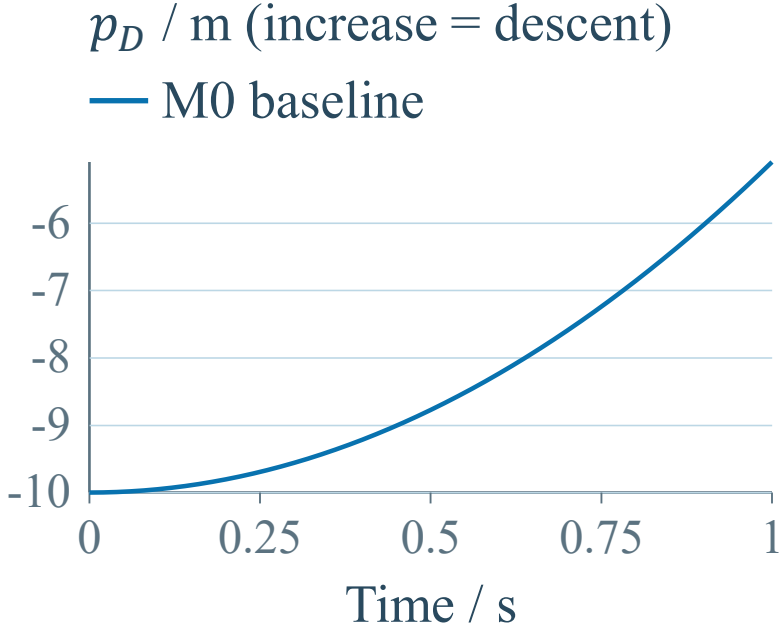
- CSV fields include time_s, pN_m, pE_m, pD_m, and three velocity components. Increasing p_D means downward motion: a rising curve means descent.
- roll_rad, pitch_rad, and yaw_rad are Euler angles. p_rad_s, q_rad_s, and r_rad_s are body rates, generally different from Euler-angle derivatives.
- omega1_rad_s through omega4_rad_s come from MotorRads. Despite its name, ModelInit_RPM also receives rad/s; do not multiply by $60/(2\pi)$ again.
- u1_norm and related fields log inputs. FxB_N and related fields are body-frame resultant forces including gravity; MxB_Nm and related fields are resultant torques. This contact-free baseline's force column is not thrust alone.

Specify frames, units, and physical times so that curves have testable meaning.

4.11.2 The M0 Offline Baseline: Zero Input

Case 1: Predict the Zero-Input Trajectory

- **Conditions:** all 16 inputs and initial motor states are zero. Release from rest at $p_D = -10$ m for 1 s; gravity acts down in north-east-down (NED) coordinates.
- **Prediction:** $v_D = gt, p_D = -10 + gt^2/2$. With $g = 9.81$ m/s², at 1 s $p_D = -5.095$ m.
- **Result:** zero.csv gives 9.81 m/s at 1 s. Maximum position difference from the analytical solution is about 3.2×10^{-14} m; integration agrees with the analytical baseline.



Zero-input free-fall trajectory

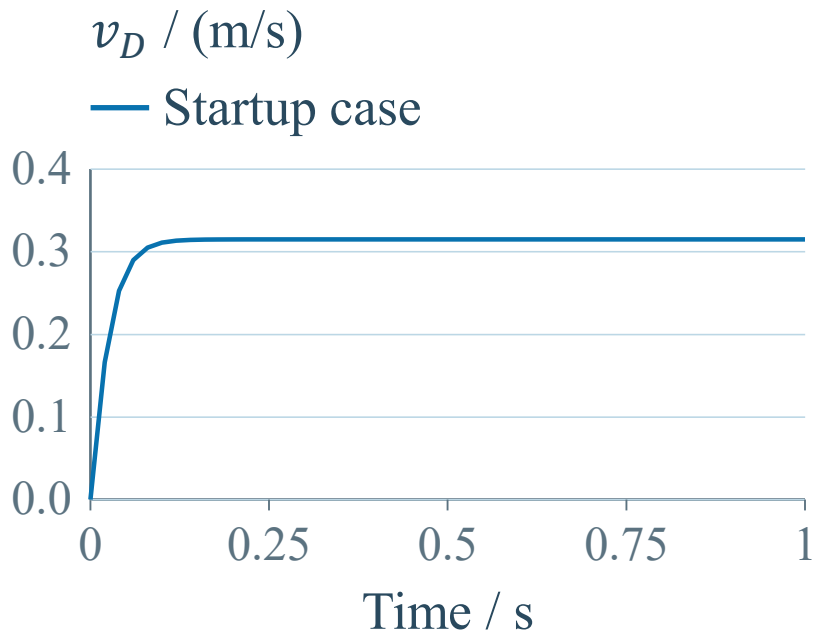
With no contact, no aerodynamics, and zero initial velocity, zero thrust gives free fall.

4.11.2 The M0 Offline Baseline: Balance and Startup

One Balanced Input, Two Initial Motor States

- **Balance:** $4c_T\omega_h^2 = mg \Rightarrow \omega_h = 470.139745$ rad/s, $u_h = (\omega_h - 22.83)/842.1 = 0.53118364$.

- balanced starts at this rotor speed and stays at −10 m. startup starts at zero rotor speed; thrust buildup leaves descent velocity. At 1 s: $p_D = -9.692961$ m, $v_D = 0.314901$ m/s.
- Without drag or altitude control, balanced thrust preserves existing descent velocity. The plot shows the startup log.



Descent speed after motor startup

Steady thrust balance requires both the correct input and matching initial motor states.

4.11.2 The M0 Offline Baseline: Startup Loads and Motion

Restoring Balanced Thrust Does Not Remove Descent Velocity

Full result

$t = 1.000$ s

Same hover input $u = 0.531184$

A: hover-speed start



$T/mg = 1.000$

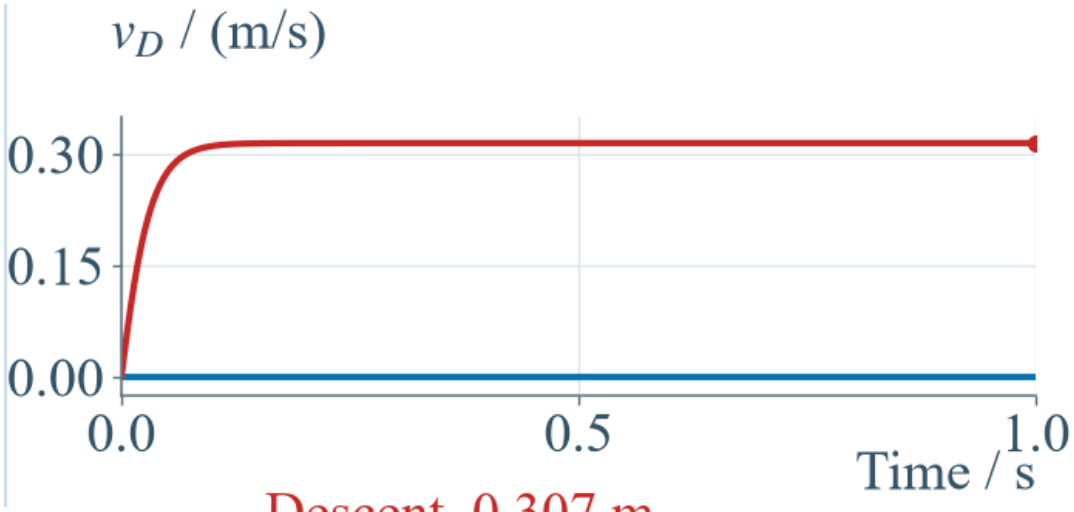
$v_D = 0.000$ m/s

B: zero-speed start



$T/mg = 1.000$

$v_D = 0.315$ m/s



Descent 0.307 m

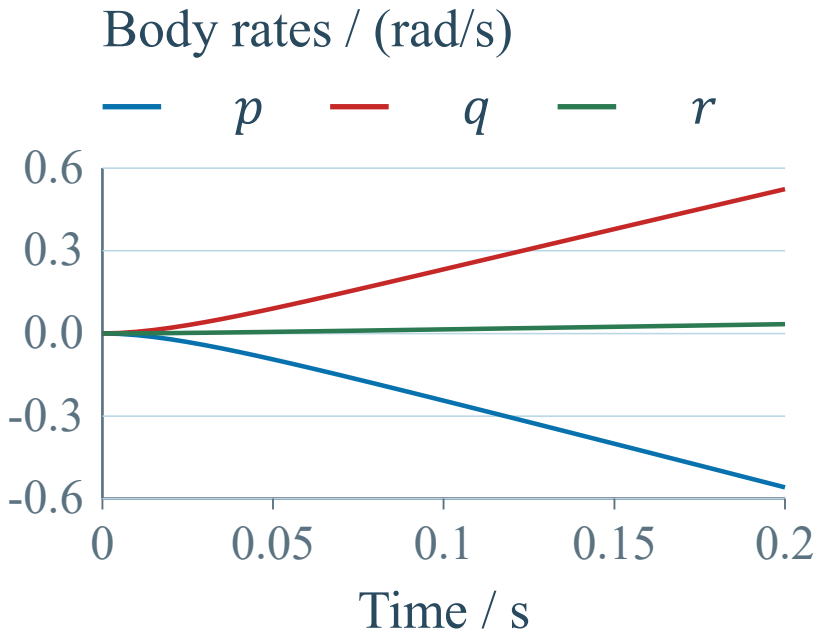
Analytical reconstruction with M0 parameters; no contact, drag, or altitude control

Hover from the initial instant needs balanced input and matching initial motor states.

4.11.2 The M0 Offline Baseline: One-Motor Increment

Case 4: Add a Small Increment to Motor 1

- motor1 starts at balanced rotor speed. Add 0.03 to input 1 only; hold the other three fixed and observe the first 0.2 s.
- **Values at 0.2 s (rad/s):** $p = -0.559288$, $q = 0.523040$, $r = 0.033456$.
- Rotor 1 is front right: $\boldsymbol{r}_1 = [+, +, 0]$. Added thrust $\Delta\boldsymbol{F} = [0,0, -\Delta T]$ gives, through $\boldsymbol{r}_1 \times \Delta\boldsymbol{F}$, $M_x < 0$ and $M_y > 0$; reaction torque gives $M_z > 0$.



Angular rates after a one-rotor increment

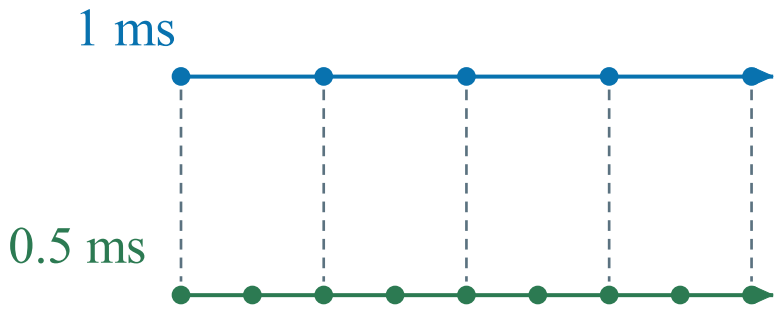
One motor's increment changes both total thrust and torques on multiple axes.

4.11.2 The M0 Offline Baseline: Common-Time Comparison

Change the Time Step; Compare at the Same Physical Time

- **Two runs:** change only output folder and step.
  m0_offline_checks([],'results_1ms',0.001)
  m0_offline_checks([],'results_05ms',0.0005)

- Align by time_s. The 1 s samples are records 1001 and 2001, excluding headers. Equal row numbers do not mean equal times.
- **Result:** startup position differs by about 1.02×10^{-8} m at 1 s; motor1 roll rate differs by about 7.8×10^{-11} rad/s at 0.2 s. These results apply to this short ode4 (fourth-order Runge–Kutta) baseline.



Vertical lines pair equal times

Numerical convergence comparisons require the same model, inputs, and initial state.

4.11.2 The M0 Offline Baseline: Fault Isolation

Trace Unexpected Results in Causal Order

Symptom	First stage to check	Inspect
Wrong initial state	Initialization	First CSV row and InitFcn callback
No zero-input fall	Ground contact	First TerrainIn15d value and contact force
Wrong rotor speed	Input → speed	Thresholds, scales, and channel order
Speed OK; wrong accel.	Speed → net force	Thrust coefficient, mass, and frame
Wrong attitude direction	Torque → body rate	Rotor IDs, arms, spin, and body-frame signs
Step results differ	Numerical solution	time_s alignment and time step

Check time alignment and step size before parameters or structure. Change one hypothesis at a time and repeat comparisons under common conditions.

Check inputs, actuators, loads, and states in sequence to narrow the fault.

4.11.2 The M0 Offline Baseline: Checking Saved Results

Recompute One Conclusion from report and CSV

- **Sources in outputs:** report.json stores parameters and provenance; zero.csv stores time-stamped states. Expect about −5.095, −5.095, and a difference of order 10^{-14}.
- Recompute the 1 s sample against the drag-free prediction. Initial position is −10 m; read gravity 9.81 m/s² from the report.

```
R = jsondecode(fileread('outputs/report.json'));
Z = readtable('outputs/zero.csv');
[~, i] = min(abs(Z.time_s - 1));
assert(abs(Z.time_s(i)-1) < 1e-9);
pred = -10 + 0.5*R.gravity_m_s2*Z.time_s(i)^2;
disp([Z.pD_m(i), pred, Z.pD_m(i)-pred]);
```

Use report parameters and logged times to establish a curve's origin and meaning.

Chapter Outline

Examples and Integrated Practice

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

5 Units in This Section

4.11.1 Published Resources and Practice Routes

Textbook: Sec. 4.3.1

4.11.2 The M0 Offline Baseline

Textbook: Sec. 4.2.3

4.11.3 Code Deployment and Interface Experiments

Textbook: Secs. 4.2.3 and 4.2.4

4.11.4 Identification and Vehicle Practice

Textbook: Secs. 4.2.5.1, 4.2.5.3, and 4.3.1–4.3.2

4.11.5 Integrated Changes and Reproducible Delivery

Four cases, replacement calculations, package delivery

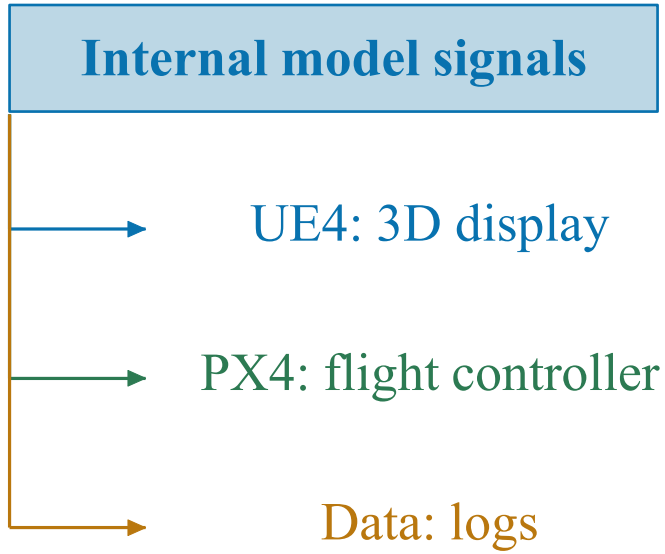
4.11.3 Code Deployment and Interface Experiments: Data Flow

Interface Examples: Identify the Data Flow First

- **0.ApiExps has 12 groups (no number 5):** 1.ApiExpsIntro, overview; 2.UserDefinedC++, code connection; 3.ExtCtrlAPI, interaction; 4.InitAPI, initial-state configuration.

- **Three output routes:** 6.ExtToUE4 sends display data to Unreal Engine 4 (UE4); 7.ExtToPX4 serves the flight controller; 8.OutCopterData logs custom internal quantities.

- Use images to interpret the scene and simulation time to track dynamics. Label units for every element of 8.OutCopterData's 32-element output.



Choose an interface by asking where data originates and which module consumes it.

4.11.3 Code Deployment and Interface Experiments: Input Connections

Received Inputs Must Be Connected to the Model

- 9.inCopterData brings flight-controller states, remote-control inputs, or extension data into the model. 10.InCollisionAPI supplies collision information; force generation still depends on the consuming module.
- 11.inSILAPI receives external integer and floating-point arrays. The current example includes display connections; its first float cannot automatically be interpreted as wind speed or mass.

- To inject wind, define three fields' NED directions and m/s units in a copy, then connect them to the wind model. Save matching input conditions before and after the change.

Model connections determine whether received data changes a physical effect.

4.11.3 Code Deployment and Interface Experiments: Extension Paths

Direct DLL Loading and SIH Use Different Execution Paths

- 12.DllModelImport includes direct and CopterSim loading. The entry for 4.ModelLoadMannual is ModelLoadApiTest.py.
- Simulation-in-hardware (SIH) runs the model on the PX4 side. 13.SIHModelSim has COM, FLY, and SITL branches; software-in-the-loop (SITL) is SIL in the textbook.
- Check each branch's Readme for interpreter, DLL bitness (32/64-bit), and initialization. Whether a DLL includes a controller depends on the model.

Python process (ModelLoad)

Control + dynamics DLL

SIH: PX4 side

Control + motion model

Where the model executes

Identify the process that computes dynamics before choosing launch and logging methods.

4.11.3 Code Deployment and Interface Experiments: Direct DLL Boundary

Read the Direct-DLL Example's Actual Input Boundary

- Open 0.ApiExps/12.DllModelImport/4.ModelLoadMannual. Follow its Readme to run Python38Run.bat, or execute python ModelLoadApiTest.py in a matching environment.
- The script constructs MultiCopterDll, calls SendVelNED and ModelUpdate, and prints position/velocity. The default MulticopterNOpx4.dll contains control loops.
- ModelUpdate specifies requested advancement duration; the step count is truncated to an integer using the DLL time step.
  If its velocity loop works normally, printed velocity should approach the target gradually (inference).
- **Troubleshooting:** first open the bundled environment with Python38Run.bat. For library-load errors, follow the Readme's FAQ to update the PX4PSP toolbox.

Velocity commands belong to an integrated model, rather than bare dynamics motor inputs.

4.11.3 Code Deployment and Interface Experiments: Build Consistency

Keep SLX, Parameters, and DLL from the Same Build

- Generate a DLL from e0_MinModelTemp (Section 4.9), save logs and hashes, and confirm the object actually loaded by the script's DLLModel setting.
- The analytical copy disables drag and contact; the original template differs. Match parameters, initial states, inputs, steps, and output times before comparing SLX and DLL.
- PX4 or an integrated DLL changes the input boundary. Log targets and motor inputs to distinguish control response from bare dynamics.

Bare dynamics input

Motor values → dynamics

Closed-loop system input

Target → control → dynamics

Record one matched set of parameters, model, and build artifacts.

4.11.3 Code Deployment and Interface Experiments: Single-Vehicle SITL

Launch One Vehicle with the Current Script

- SITL: open 1.BasicExps/e0_MinModelTemp, run Exp1_MinModelTemp_SITL.bat, and enter vehicle count 1 when prompted.
- The script sets DLLModel=Exp1_MinModelTemp and copies the DLL. Save ongoing work before launch; it may restart CopterSim.
- In CopterSim, verify the actual model, instance ID, increasing simulation time, and state updates. Follow Readme.pdf to arm and take off.
- **Criterion:** wait for "PX4： GPS 3D fixed & EKF initialization finished." in CopterSim before control. This model fixes terrain height at 0; use flat-ground maps only.

Confirm model, instance, and initial state before applying control commands.

4.11.3 Code Deployment and Interface Experiments: Request and Readback

Check the Connection with One Initial-State Request

- See unit 4.10.3's planar-initial-state request and example-entry pages for format and original-script side effects.
  Keep just one XYyaw request here to check the target ID and readback.
- Criterion: the target ID returns $x = 2$ m, $y = -1$ m, and $\psi = 30°$ (about 0.5236 rad), while simulation time in CopterSim continues increasing.

- Record the target ID's position, yaw, and simulation time before and after the request. The post-request record must differ from the pre-request record and match the requested state; a "sent successfully" printout cannot replace readback.

Matching the requested and returned states confirms the interface reached the target instance.

4.11.3 Code Deployment and Interface Experiments: Minimal Request Copy

Keep One Request and State the Target and Acceptance Check

- Start one SITL vehicle; make your copy of testReqCopterSim.py in its example folder. Keep only:

```
import ReqCopterSim
req = ReqCopterSim.ReqCopterSim()
print(req.getSimIpList())
CopterID = 1; print(req.getSimIpID(CopterID))
req.sendReSimXYyaw(CopterID, [2, -1, 30])
```

Target instance	Request parameters	CopterSim readback	Time and state
Send only to an ID confirmed in the returned list	[2, -1, 30] x/y: m; yaw: °	x ≈ 2 m, y ≈ −1 m yaw ≈ 30°	Record time before/after; confirm fresh time and state

Without a fresh state, mark the connection unconfirmed and send no further reset commands.

4.11.3 Code Deployment and Interface Experiments: Stop, Reset, and Rerun

End Old Instances Before Repeating the Same Conditions

- SITL: land and disarm in QGroundControl (QGC), stop PX4 in the launch window, then follow BAT exit prompts. Cleanup closes all local CopterSim, QGC, and RflySim3D processes; do not share the session with other experiments.
- Initial-state request: record ID, request parameters, and states/times before and after. Restart the single-vehicle BAT for baseline reruns rather than repeatedly resetting an old instance. Recreate the process after a DLL script ends.
- Save SITL log paths, screenshot times, and API readback. ModelLoadApiTest.py only prints position/velocity; save console text. Without logs, do not claim numerical acceptance.

Stop old instances, reset initial states, and save time-stamped records for repeatable comparisons.

Chapter Outline

Examples and Integrated Practice

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

5 Units in This Section

4.11.1 Published Resources and Practice Routes

Textbook: Sec. 4.3.1

4.11.2 The M0 Offline Baseline

Textbook: Sec. 4.2.3

4.11.3 Code Deployment and Interface Experiments

Textbook: Secs. 4.2.3 and 4.2.4

4.11.4 Identification and Vehicle Practice

Textbook: Secs. 4.2.5.1, 4.2.5.3, and 4.3.1–4.3.2

4.11.5 Integrated Changes and Reproducible Delivery

Four cases, replacement calculations, package delivery

4.11.4 Identification and Vehicle Practice: Identification Examples

Propulsion Identification: Follow Data Generation

- e11_PowerSysIdModel covers preparation, static tests, overall/staged models, and dynamic identification. Static relations cannot determine a time constant.
- e12_ModelTempId contains drag, inertia, and D0 modeling templates with parameters different from M0. e13_CopIdModel supplies tools and in-the-loop sweeps.
- dryer2.mat demonstrates the method. Before using flight logs, state whether the identified system includes a controller; distinguish closed-loop and component dynamics.

Static tests

Thrust–speed relation

↓

Dynamic tests

Time constant

↓

Flight frequency sweep

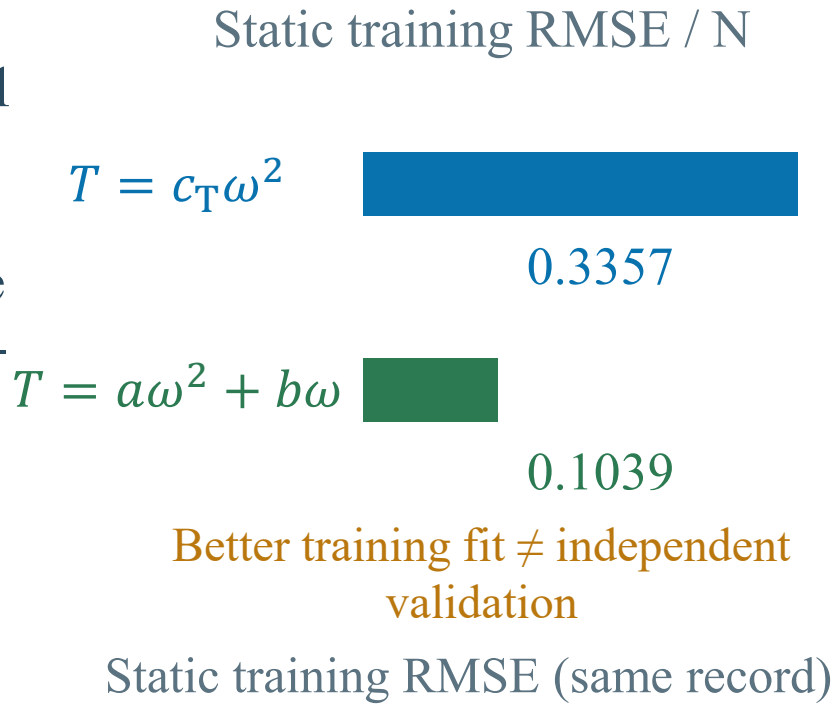
Plant + control boundary

Establish how data was obtained before deciding which model component to fit.

4.11.4 Identification and Vehicle Practice: Real Static and Teaching Dynamic Data

Practice Two Routes: Teaching Dynamics and Real Static Data

- Run identification_beginner in introductory lab folder 03_Identification. If Advanced data is available, read 91 rising-segment points from Static_data.mat.
- Without the static MAT file, skip real fitting and still validate teaching dynamics. Right: real-data static training root-mean-square error (RMSE).
- Teaching dynamic plant: $K = 1000$ rad/s, $\tau = 0.05$ s. Independent-validation RMSE is 2.1483 rad/s and fit (FIT) is 98.9168%; these describe only this teaching plant.



Real static records and the teaching dynamic plant support separate conclusions.

4.11.4 Identification and Vehicle Practice: Basic Example Entries

Choose a Basic Example You Can Explain

Folder	Vehicle	Entry model file	Observe first
e0_MBDIntro	Docs	Readme.pdf	Workflow context
e0_MinModelTemp	Quadcopter	Exp1_MinModelTemp.slx	Interfaces/solution
e1_MinModelTempLib	Quadcopter	Exp1_MinModelTemp.slx	Package/reuse library
e2_FixWingModelCtrl	Fixed-wing	AircraftMathworks.slx	Airspeed and lift
e5_FixedWingManuscriptMdl	Fixed-wing	FixedWingManuscriptMdl.slx	Alternate model
e3_CarAckermanModeCtrl	Ackermann	CarAckerman.slx	Front steer/curvature
e4_CarR1DiffModelCtrl	Diff. drive	CarR1Diff.slx	L/R speed difference

Write the question first: use fixed-wing models for aerodynamics, ground vehicles for steering/contact, and the minimal template for interfaces and solution methods.

Use a basic model to answer one clear question before building a complete application.

4.11.4 Identification and Vehicle Practice: Differential-Drive Migration

Migrate the Car: Match Model, Airframe, and Scene

[installation directory]/RflySimAPIs/4.RflySimModel/1.BasicExps/ — each example's Readme.pdf		
Item	**Differential drive (default)**	**Ackermann (alternative)**
Example	e4_CarR1DiffModelCtrl	e3_CarAckermanModeCtrl
SITL script	CarR1Diff_SITLRun.bat	CarAckerman_SITLRun.bat
PX4 airframe	r1_rover	generic_ground_vehicle

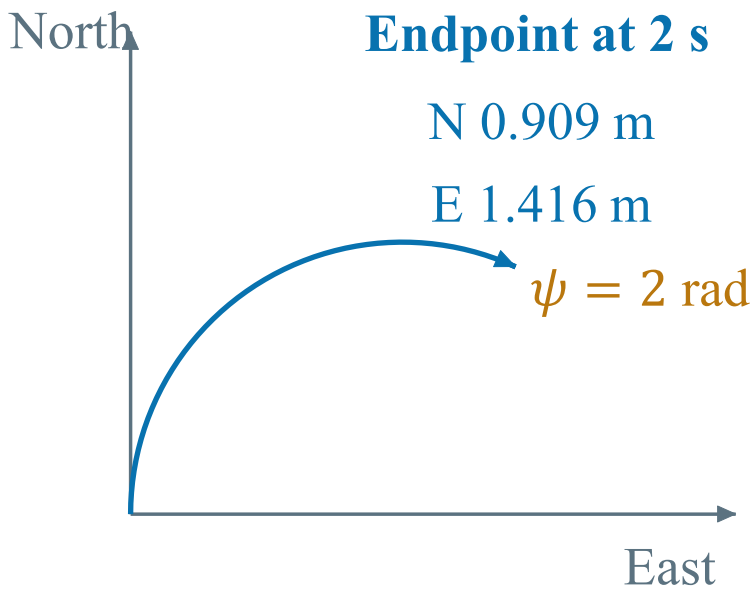
- **Steps:** compile SLX and run GenerateModelDLLFile.p to obtain the DLL → launch the SITL script and enter 1 → arm in QGC and choose "Go to location" on the map.
- **Criterion:** after RflySim3D shows "CopterSim/PX4 EKF 3D Fixed:1/1", the car drives toward the target. Without an airframe setting it starts as a quadcopter. Firmware v1.14+ also needs Actuators-page setup (Readme.pdf).

Establish interfaces and ground contact; compare low-speed straight and turning motion.

4.11.4 Identification and Vehicle Practice: Offline Vehicle Prediction

Use an Ideal Kinematic Endpoint to Predict Vehicle Migration

- The ideal kinematic baseline was run on Section 4.8's offline trajectory comparison page: vehicle_migration_baseline in introductory folder 04_VehicleKinematics.
- Left/right wheel rates 12/8 rad/s give the shown 2 s prediction. Derivation: Section 4.8's differential-drive displacement page.
- For e4_CarR1DiffModelCtrl or e6_CarR1DiffCtrl, first establish whether input means wheel speed or a target. Save actual wheel speeds, yaw, and trajectory; explain contact/slip differences.



Ideal differential-drive trajectory over 2 s

Establish an ideal prediction, then explain contact and dynamics differences in the full vehicle model.

4.11.4 Identification and Vehicle Practice: Hexacopter and Octocopter Entries

Hexacopter, Coaxial Octocopter, or Flat Octocopter

Folder: [installation directory]/RflySimAPIs/4.RflySimModel/2.AdvExps/e2_MultiModelCtrl/			
Subfolder	**Vehicle**	**SITL launch script**	**PX4 airframe**
4.HexModelCtrl	Hexacopter	HexarotorModelCTRL_SITLRun.bat	hexa_x
5.OctoCoxRotor	Coaxial octocopter	OctoCoxRotorSITLRun.bat	octo_cox
6.OctoX	Flat octocopter	OctoX_SITLRun.bat	octo_x

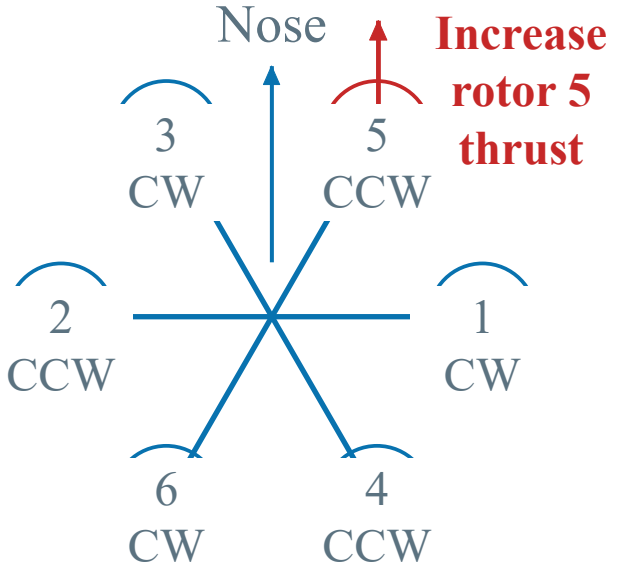
- **Steps:** Read the subfolder's Readme.pdf → run *_init.m → after SLX changes, generate the DLL with GenerateModelDLLFile.p → run the SITL script as administrator, enter 1 → take off in QGC.
- **Checks:** CopterSim reports connection to UE4 and RflySim3D shows normal flight. Otherwise check motor counts and input numbering in BAT/init files. Hardware-in-the-loop (HITL; HIL in the textbook) also needs a flight controller and QGC airframe setup.

Treat the actual model, input mapping, and airframe settings as one layout configuration.

4.11.4 Identification and Vehicle Practice: Advanced Multicopters

Change the Airframe Structure or Control Entry

- e1_MaxModelTemp is a full template; e2_MultiModelCtrl has collision, six/eight-rotor, and tethered branches.
- After changing rotor count, recheck channels, moment arms, and force/torque synthesis. A NoPX4 branch may still contain an internal controller.
- Predict one hexacopter channel's torque signs, then log input and response. Compare layout changes separately from controller changes.
- **Reference:** rotor 5 is front right and CCW. Added thrust gives $M_x < 0$, $M_y > 0$, and $|M_y| \approx 1.73|M_x|$; reaction torque gives $M_z > 0$.



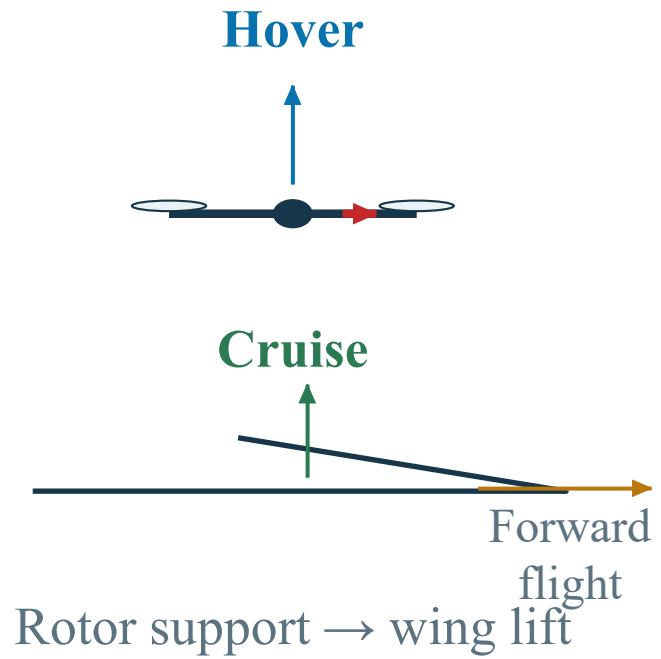
Hexa X IDs; CW/CCW viewed from above

Changing layout and changing the control boundary are different extensions.

4.11.4 Identification and Vehicle Practice: Aerial Vehicles

Advanced Aerial Vehicles: Separate Flight Phases First

- e3_FWingModelCtrl covers fixed-wing aircraft. e4_VTOLModelCtrl includes vertical takeoff/landing, tail-sitters, and mode transitions.
- e8_Helicopter includes helicopter position/velocity control. e10_Wing supplies a flying wing and launch scripts; its force sources differ from a quadcopter's.
- Start fixed-wing checks with level flight. For vertical takeoff and landing (VTOL), validate hover and cruise separately before transition; compare each phase independently.



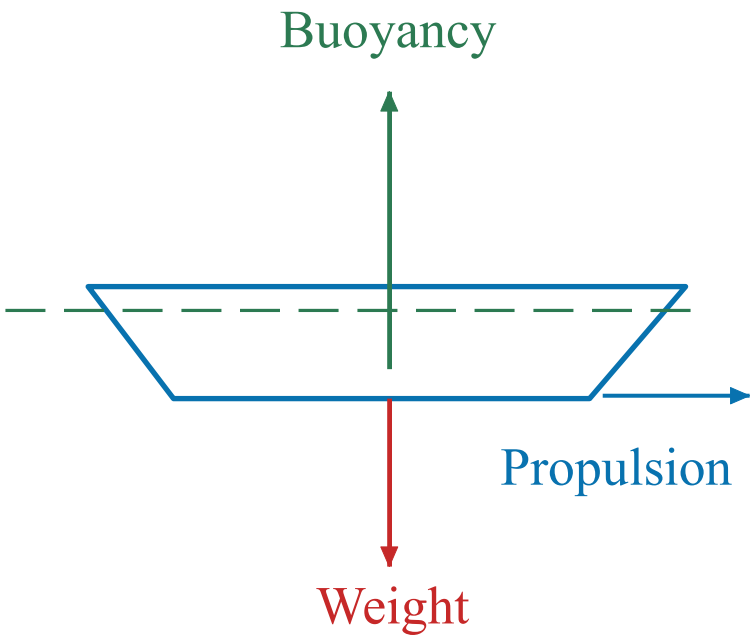
Fixed-wing, VTOL, and rotorcraft models require different primary validation conditions.

4.11.4 Identification and Vehicle Practice: Ground and Marine Vehicles

Contact Conditions and Medium Determine the Force Model

- e5_CarAckermanCtrl and e6_CarR1DiffCtrl provide control branches; e7_TrailerModelCtrl adds towing constraints.

- e9_USV is a surface vessel; e9_UUV is an underwater vehicle. Both use number e9, so specify the complete folder name when running.
- For cars, examine tire–ground contact and slip; for vessels, buoyancy, drag, and propulsion. Even with identical frames, validate external-force models independently.



Across vehicle types, retain the state-equation approach and reinterpret force-generation mechanisms.

Chapter Outline

Examples and Integrated Practice

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

5 Units in This Section

4.11.1 Published Resources and Practice Routes

Textbook: Sec. 4.3.1

4.11.2 The M0 Offline Baseline

Textbook: Sec. 4.2.3

4.11.3 Code Deployment and Interface Experiments

Textbook: Secs. 4.2.3 and 4.2.4

4.11.4 Identification and Vehicle Practice

Textbook: Secs. 4.2.5.1, 4.2.5.3, and 4.3.1–4.3.2

4.11.5 Integrated Changes and Reproducible Delivery

Four cases, replacement calculations, package delivery

4.11.5 Integrated Changes and Reproducible Delivery: Explain Four Cases

Infer Model Conditions from Experimental Results

- With the same input, balanced holds height while startup descends. Identify the different initial state and explain the chain from speed to resultant force to velocity.
- If zero holds height, check motors and ground first. Contact may balance gravity and hold the body stationary; this does not disprove free fall.
- Apply a statically identified replacement propulsion component to an M0 copy. Is the old hover input still valid? Recompute the operating point and propose a validation case.
- **Reference:** ① different initial motor speeds; ② check initial state and ground contact first; ③ see the M0 component-replacement reference page.

Same input, different displacement

Initial motor speed?

Zero input, no motion

Ground contact?

Replace propulsion components

Hover operating point?

Four cases of one model test different parts of the chain.

4.11.5 Integrated Changes and Reproducible Delivery: M0 Component Reference

Recompute M0 Hover After Replacing a Component

- **Conditions:** use only Section 4.6's real static-fit thrust coefficient $c_T = 2.657299 \times 10^{-5}$ N/(rad/s)² as a teaching replacement in M0. Keep mass 1.515 kg, g = 9.81 m/s², four-rotor symmetry, and all other parameters unchanged.
- **New point:** $T_0 = mg/4 = 3.7155375$ N, $\omega_0 = \sqrt{T_0/c_T} \approx 373.930$ rad/s. Original M0 $c_T = 1.681 \times 10^{-5}$ gives about 470.140 rad/s; recompute the old input.
- **New input:** retain M0's steady map $\omega = 842.1u + 22.83$ without crossing thresholds or saturation, giving $u_0 \approx 0.416934$. Start at matching rotor speed for a short contact-free, drag-free run; check near-zero resultant force and constant height. Recalibrate if the map changes.

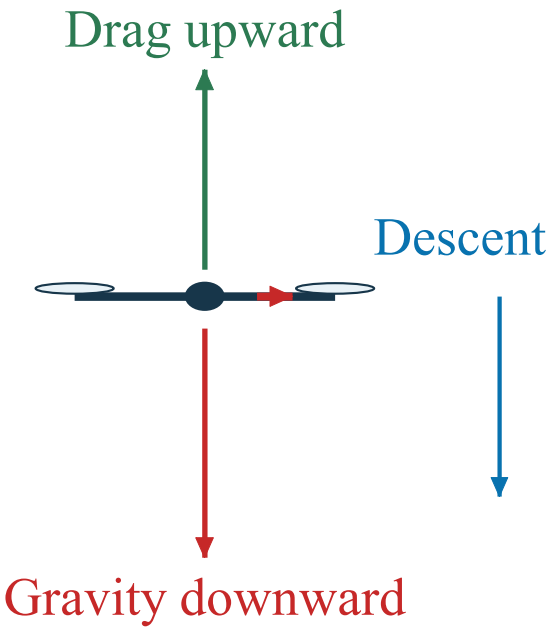
After changing the coefficient, use M0 mass and input mapping; do not reuse the 2 kg result.

4.11.5 Integrated Changes and Reproducible Delivery: Restore Drag

Restore an Environmental Term and Predict Its Direction

- Copy m0_offline_checks.m as m0_drag_check.m, rename its function accordingly, and write results to results_drag.

- Set params.ModelParam_uavCd=0.055 and update report.test_overrides.uavCd. Keep all other initial states and inputs unchanged.
- Replace zero's drag-free analytical assert; retain finite-value, duration, and initial-state checks. Compare velocity and position with original zero.csv at matching times.
- **Prediction at 1 s:** $0 < v_D < 9.81$ m/s and $-10 < p_D < -5.095$ m. In still air, drag acts upward and slows descent relative to baseline.



Restored drag changes the force mechanism. Keep inputs and initial state identical.

4.11.5 Integrated Changes and Reproducible Delivery: Drag-Copy Edits

m0_drag_check: Three Edits and a Comparison Snippet

- **Edits:** ① first-line function name; ② restore translational drag to source value 0.055 and update its record; ③ remove zero's drag-free analytical assertion while keeping other checks.
- **Compare both zero.csv files at the same 1 s sample** after m0_drag_check([],'results_drag').

```
function report = m0_drag_check(apiRoot, outDir, stepSize)   % Rename function on line 1
params.ModelParam_uavCd=0.055;   % Was 0; rotational damping uavCCm stays zero
report.test_overrides.uavCd=0.055;   % After the original test_overrides assignment
% zero branch: remove analytical assert(item.max_pD_error_m<1e-5 && ...)
Z = readtable('outputs/zero.csv');  D = readtable('results_drag/zero.csv');
[~,i] = min(abs(Z.time_s-1));  [~,j] = min(abs(D.time_s-1));
disp([Z.vD_m_s(i) D.vD_m_s(j); Z.pD_m(i) D.pD_m(j)])
```

Keep the baseline and change only drag in the copy to attribute the difference to drag.

4.11.5 Integrated Changes and Reproducible Delivery: Experiment Package

Integrated Task: A Reproducible M0 Experiment Package

Task: run four cases and two time steps with installed M0.
Record drag, damping, and ground settings in the analytical copy.

Material	Files and contents	Reproducibility check
Source files	Test function, provenance, and parameters	Record installed M0 source and modifications
Conditions	Inputs/initial states: four cases, two steps	Inputs and initial states match this experiment
Logs	CSVs and comparison plots	Time and units; common physical times
Conclusions	Report: inputs, predictions, and actual response	Recompute against the original baseline; explain differences

Choose one extension: identification, deployment/interfaces, or vehicle migration.
Keep the common baseline and explain changes caused by the modification.

Self-Check Reference Values

Compare Each Value After Reproducing the M0 Baseline

Conditions: M0 mass 1.515 kg, $g = 9.81$ m/s², 1 ms step; same initial states and inputs.

Case	Reference value	Interpretation
zero	At 1 s: $p_D = -5.095$ m, $v_D = 9.81$ m/s	Free fall
balanced	At 1 s: $p_D = -10$ m, $v_D \approx 0$	No height drift
startup	At 1 s: $p_D = -9.692961$ m, $v_D = 0.314901$ m/s	Startup descent
motor1	At 0.2 s: $p = -0.559288$, $q = 0.523040$, $r = 0.033456$ rad/s	−, +, +
Two steps	startup position difference at 1 s: about 1.02×10^{-8} m	Align by time_s

These are bundled 05_M0Checks offline reference results, not platform or flight records. If model or parameter versions differ, use the current report.json.

Class Exercise

M0 at 1.8 kg: Hover and Startup Descent

- **Conditions:** change M0 mass to 1.8 kg; keep all other parameters ($c_T = 1.681 \times 10^{-5}$, $\omega = 842.1u + 22.83$, motor time constant $\tau = 0.0214$ s, $g = 9.81$ m/s²).
- **Questions:** ① recompute ω_h and u_h; ② in startup (motors initially at zero, input set to the new u_h), is 1 s descent larger or smaller than for 1.515 kg?
- **Reference:** ① $\omega_h = \sqrt{mg/(4c_T)} \approx 512.456$ rad/s, $u_h \approx 0.581435$.
- ② Unchanged, about 0.307 m: $a_D = g[1 - (\omega/\omega_h)^2]$ and, during startup, $\omega/\omega_h = 1 - e^{-t/\tau}$ are mass-independent (provided the input is unsaturated).

A mass change needs a new operating point; startup descent depends on the motor time constant, not mass.

Section Summary

Fix the Baseline, Trace Data, Change One Item

- **Method:** record conditions, then establish a recomputable M0 baseline with four cases and two steps. Trace code artifacts, interface inputs, and returned states during integration. Change one item and compare under common conditions.
- **Key judgments:** the same balanced input gives different results with different initial motor states. Agreement between steps indicates numerical convergence only. “Sent successfully” does not prove a state change. Direct-DLL velocity commands belong to an integrated-model interface.
- **Common errors:** increasing NED p_D means descent. ModelInit_RPM uses rad/s. InitFcn overwrites initial states. Align by time_s, not row number. Check the DLL actually loaded and distinguish bare-dynamics from closed-loop inputs.
- **Next:** Section 4.12 uses exercises and course projects to assess this chapter’s modeling, simulation, and identification methods together.

Traceable baselines, data flow, and individual changes make model differences attributable.