Chapter Outline

External Control Examples and Experimental Validation

Chapter 6
External Control and Trajectory Planning

6.1 External Control Fundamentals
6.2 Control Principles and Modes
6.3 Trajectory Models and Generation
6.4 Planning and Obstacle Avoidance
6.5 Tracking and Aggressive Flight Control
6.6 Frontier Control Technologies
6.7 Platform and Communication Interfaces
6.8 Examples and Experimental Validation
6.9 Exercises and Classroom Discussion

5 Units in This Section

6.8.1 Remote Control and Mission-Mode Experiments
Textbook: Secs. 6.1.2.1–6.1.2.2, 6.3.1

6.8.2 Trajectory Execution and Result Validation
Textbook: Secs. 6.2.2.5, 6.3.1

6.8.3 Learning and Vision Control Experiments
Textbook: Secs. 6.1.4.2, 6.3.1

6.8.4 Foundation-Model Control Experiments
Textbook: Secs. 6.1.4.3, 6.3.1

6.8.5 Advanced Applications and Real-Vehicle Transfer
Textbook: Secs. 6.3.2, 6.2.2.6

6.8.1 Remote Control and Mission-Mode Experiments: Simulation Run

Start, Run, Observe, and End the Example

Start: run PX4RcCtrlAPITest.bat as administrator. It opens QGroundControl (QGC) and RflySim3D, terminates any running CopterSim, then restarts it.

Run: wait for CopterSim to report GPS 3D fixed & EKF initialization finished (EKF: extended Kalman filter). Open Python38Run.bat; enter python PX4RcCtrlAPITest.py.

Observe: after arming, the aircraft climbs for about 60 s in RflySim3D. Check that channel values, actual mode, and altitude changes agree.

End: after 60 s, radio-control (RC) override stops and stopRun forces disarming in flight. Simulation only; see Section 6.7, “Termination and Exceptions.” Press any key in the software-in-the-loop (SITL; SIL in the textbook) window; the bat file closes CopterSim, QGC, and RflySim3D.

Troubleshooting: if arming fails, check CopterSim initialization and the QGC geofence. For communication loss, check whether another program is using the port.

6.8.1 Remote Control and Mission-Mode Experiments: Route-Planning Example

Draw, Upload, and Fly a Mission in QGC

Installation directory: [Installation directory]/RflySimAPIs/2.RflySimUsage/1.BasicExps/e16_RoutePlanning/: Readme.pdf (HITL; requires a Pixhawk 6C and RC transmitter)

Prepare: upload the compiled firmware, select HIL Quadcopter X, and enable hardware-in-the-loop (HITL; HIL in the textbook).

Start: run HITLRun; enter the autopilot serial-port number. Wait for CopterSim initialization.

Plan: in Plan, add takeoff (e.g., 50 m), waypoints and return-to-launch; click Upload Mission.

Expected: slide to arm in Fly; the aircraft takes off, follows the route, returns, and lands.

Acceptance: actual mode is Mission; the flown path matches the plan. Confirm disarming after landing.

Troubleshooting: uploaded mission idle—check takeoff, waypoints and Mission mode.

Side effects: firmware is flashed and HITL enabled. Before real flight, disable HITL and recheck the airframe.

Chapter Outline

External Control Examples and Experimental Validation

Chapter 6
External Control and Trajectory Planning

6.1 External Control Fundamentals
6.2 Control Principles and Modes
6.3 Trajectory Models and Generation
6.4 Planning and Obstacle Avoidance
6.5 Tracking and Aggressive Flight Control
6.6 Frontier Control Technologies
6.7 Platform and Communication Interfaces
6.8 Examples and Experimental Validation
6.9 Exercises and Classroom Discussion

5 Units in This Section

6.8.1 Remote Control and Mission-Mode Experiments
Textbook: Secs. 6.1.2.1–6.1.2.2, 6.3.1

6.8.2 Trajectory Execution and Result Validation
Textbook: Secs. 6.2.2.5, 6.3.1

6.8.3 Learning and Vision Control Experiments
Textbook: Secs. 6.1.4.2, 6.3.1

6.8.4 Foundation-Model Control Experiments
Textbook: Secs. 6.1.4.3, 6.3.1

6.8.5 Advanced Applications and Real-Vehicle Transfer
Textbook: Secs. 6.3.2, 6.2.2.6

6.8.2 Trajectory Execution and Result Validation: Running and Acceptance

Two-Step Line Run: A Completion Message Does Not Prove Arrival

Directory: [Installation directory]/RflySimAPIs/6.RflySimExtCtrl/2.AdvExps/e4.TrajPlanExps/1.LineTrajMotion/

Entries: Readme.pdf (steps and FAQ), SITLRun.bat, Python38Run.bat, LineWPCtrl.py

Start: run SITLRun.bat (requests elevation; terminates existing CopterSim). Wait for GPS 3D fixed & EKF initialization finished.

Run: in Python38Run.bat, enter python LineWPCtrl.py. initOffboard() automatically arms and changes link-loss parameters; see Section 6.7, “SDK Side Effects.”

Observe: after about 3.3 s, “轨迹完成！” (“Trajectory complete!”) prints; 3 s later, endOffboard is called. D and T show state and path. This ends the reference loop; verify arrival by fresh position error and hold time (Section 6.7).

Troubleshooting: no response—check CopterSim initialization and QGC OFFBOARD mode. Wrong path—restart simulation and wait for EKF convergence.

End: endOffboard only stops sending. For final-state checks, see Section 6.7, “Termination and Exceptions.” Press any key in SITLRun to close CopterSim, QGC and RflySim3D.

6.8.2 Trajectory Execution and Result Validation: Basic e9 Entries

Compare Responses of Two Models in the Basic Tracking Experiment

Directory: [Installation directory]/RflySimAPIs/6.RflySimExtCtrl/
1.BasicExps/e9_TrajectoireFollowing_Segment/e3.1/

Entry	Operation and Output
Readme.pdf **startSimulation.m**	Read dependencies; initialize sim1.0 and sim2.0 separately.
sim1.0/ (TF) **Transfer-function model**	Open e3_1_TF_TrajectoireFollowing_Segment.slx.
sim2.0/ (Model) **Nonlinear physics model**	Open e3_1_Model_TrajectoireFollowing_Segment.slx.

Run: connect Control Input to the T = 10 s circle, then run. Records: “Saving and Comparing.”
Quick reproduction: run Plot_Compare_cercle.m in sim2.0; see Section 6.1, “Running the Example.”

6.8.2 Trajectory Execution and Result Validation: Saving and Comparing

Copy Both Model Records into One Directory for a Shared Plot



Files: sim1.0 saves TF data as N4_Step1_cercle_comparaison_T10tf.mat. The sim2.0 plotting script reads both MAT files only from its own directory.

Expected: similar XY response shapes. In the historical records, both Model and TF amplitudes exceed the reference; see “Paired-Time Error Results” and “Time-Series Comparison.”

Troubleshooting: skipping Step 3 uses the old bundled TF MAT. The _2017b model names in the instructions no longer exist; open the .slx files listed under “Basic e9 Entries.”

6.8.2 Trajectory Execution and Result Validation: Evaluation Metrics

Pair Reference and Feedback at the Same Time Before Computing Error

$$\mathbf{e}_k = \mathbf{p}_{d,k} - \mathbf{p}_k, \quad E_{\mathrm{RMS}} = \sqrt{\frac{1}{N}\sum_{k=1}^{N} \|\mathbf{e}_k\|^2}, \quad E_{\max} = \max_k \|\mathbf{e}_k\|$$

Time pairing: align clocks and sample times, then interpolate the reference; never shift curves to make error look smaller.

N is the sample count; $E_{\mathrm{RMS}}, E_{\max}$ are root-mean-square (RMS) and peak errors, both in m.

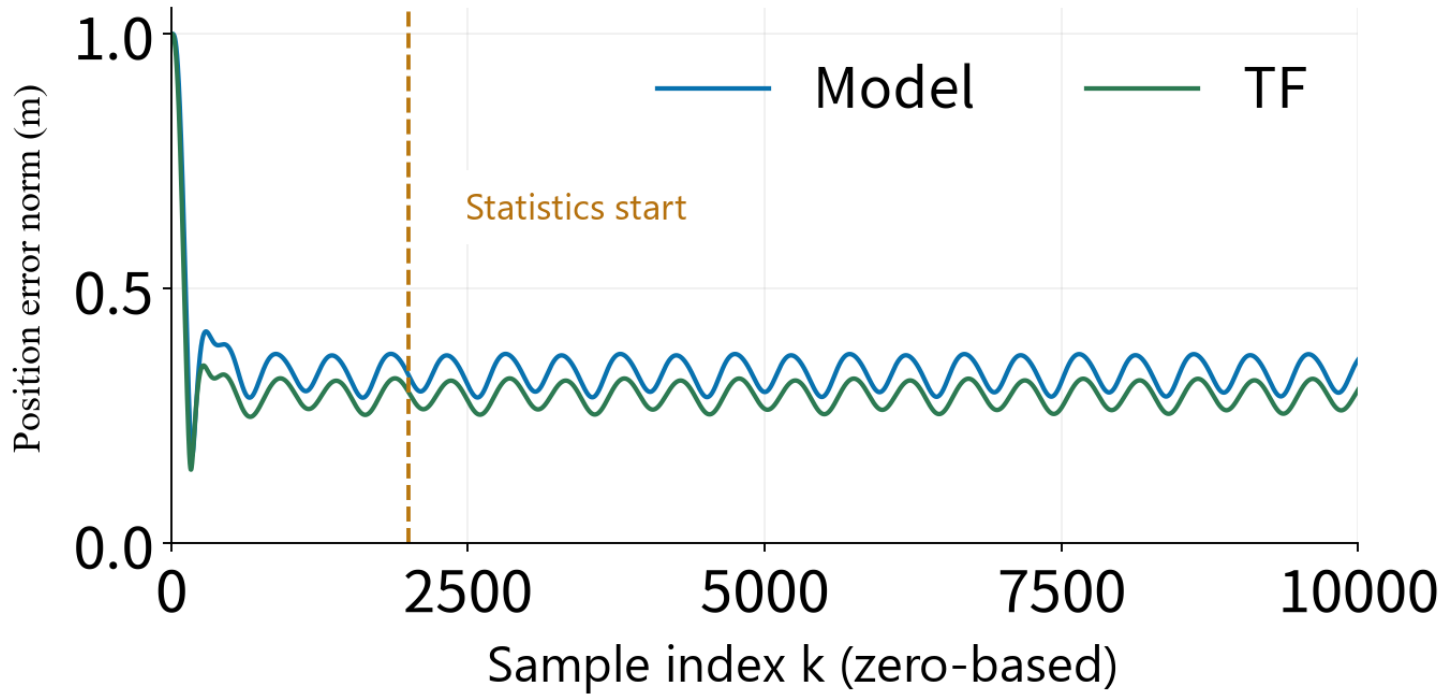
Historical MAT: 10001 rows each, with reference X/Y and output X/Y; the two reference arrays are identical.

Scope: no time vector. Pair equal rows assuming matching samples; evaluate only 2D position error.

New records: save position-sample times, reference times, actual commands, and mode; retain startup and termination.

6.8.2 Trajectory Execution and Result Validation: Paired-Time Error Results

Historical Steady-State Error Is About 0.3 m; Model Exceeds TF



Indices 2000–10000

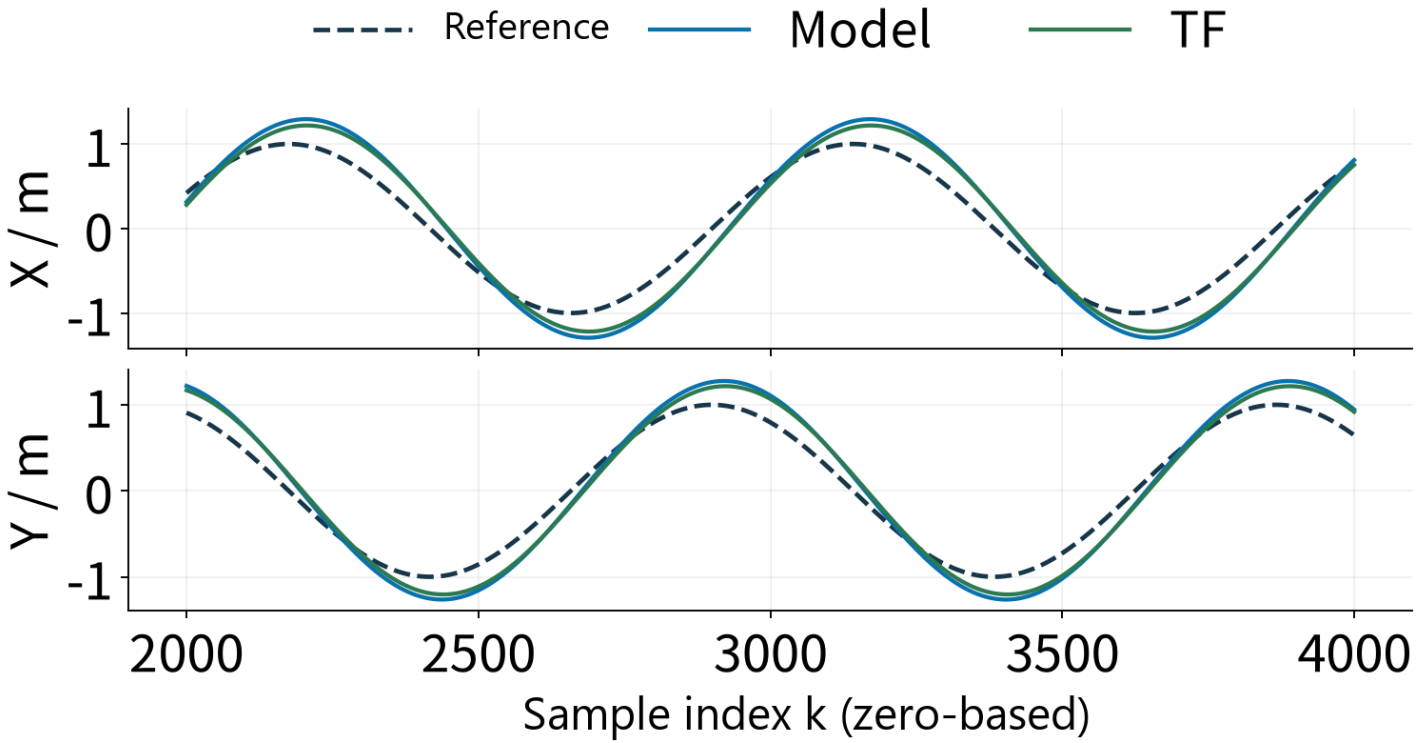
Model
RMS 0.334 m
Peak 0.371 m

TF
RMS 0.290 m
Peak 0.323 m

The initial 1 m error occurs because output starts at the origin. Statistics pair equal rows as the same time.

6.8.2 Trajectory Execution and Result Validation: Time-Series Comparison

Amplitude Overshoot and Phase Lag Can Occur Together



Amplitude: Model and TF both exceed the reference.

Phase: peaks arrive later; lag differs between axes.

Evaluation: smaller position error need not mean less delay.

6.8.2 Trajectory Execution and Result Validation: Limits of Delay Estimation

Phase Lag Is Reproducible; Seconds Require a Sampling Assumption

Record / Axis	Amplitude Ratio	Equivalent Lag / Samples	At 0.01 s per Sample
Model X	1.291	28.94	0.289 s
Model Y	1.269	20.04	0.200 s
TF X	1.219	31.05	0.311 s
TF Y	1.211	23.07	0.231 s

Least squares over indices 2000–10000: $s_k = A\sin(\Omega k) + B\cos(\Omega k) + C$

$\phi = atan2(B, A), \quad n_{\text{lag}} = (\phi_{\text{ref}} - \phi_{\text{out}})/\Omega$; $\Omega = 0.0065$ rad/sample.

If sampling is 0.01 s, lag is $\tau = n_{\text{lag}} \times 0.01$ s; period 9.666 s (966.64 samples).

A, B, C are fitted coefficients. Phase lag includes closed-loop dynamics, as well as link delay.

6.8.2 Trajectory Execution and Result Validation: Simulation Steps

Correct a Copy of the Python Source Before Simulation

Directory: [Installation directory]/RflySimAPIs/6.RflySimExtCtrl/2.AdvExps/e4.TrajPlanExps/ 3.CirularTrajMotion/ or 4.SpiralTrajMotion/ (Readme.pdf, both bat files, and UAVCtrl.py share the folder)

1 Prepare a copy: make phase derivatives consistent in UAVCtrl.py (signs: Section 6.5, “Circular Trajectory”); add logging, deceleration, and an actual-start-position check.

2 Initialize: open SITLRun.bat in the same folder (elevation, existing CopterSim termination, firmware build).
Wait for GPS 3D fixed & EKF initialization finished in CopterSim.

3 Run: in the same-folder Python38Run.bat window, enter python UAVCtrl.py.

4 End: decelerate, then call endOffboard(). Confirm landing and disarming separately.

Side effects: initOffboard() arms and changes failsafe parameters; see Section 6.7, “SDK Side Effects.”

6.8.2 Trajectory Execution and Result Validation: Expectations and Troubleshooting

Assess Shape, Logs, and Errors Paired at the Same Time

Circle: arm, climb to 10 m, reach (6, 0, −10), then fly a 6 m radius circle for about 30 s.

Spiral: start at (5, 0, −10); ascend on a 5 m radius spiral at 0.6 rad/s for 40 s, with Z from −10 m to −25 m.

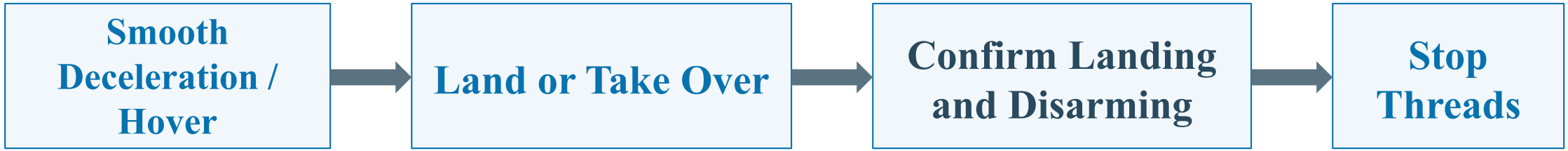
Observe: press D, then T in RflySim3D to display state values and the flight path.

Acceptance: after correction, feedback should approach the reference. Pair times to compute RMS and peak error; see “Evaluation Metrics.”

Troubleshooting: no initialization log—first launch needs Internet access to download maps; then inspect logs and blocked-component records, allowing trusted components as instructed. No connection—check simulation and firewall. Wrong path—check radius, angular speed, altitude, and the 30 Hz loop.

6.8.2 Trajectory Execution and Result Validation: Termination and Exceptions

Confirm Landing and Disarming as Evidence of the Final State



Normal end: smoothly transition the reference to hover, then use a validated landing or takeover procedure.

Example criterion: after endOffboard(), still confirm landing and disarming.

Cleanup prerequisite: landed and disarmed, or an offline experiment that was never armed. Only then call stopRun().

Exception: use a prevalidated exit or takeover strategy. Full function semantics: Section 6.7, “Termination and Exceptions.”

6.8.2 Trajectory Execution and Result Validation: Records and Acceptance

Log Position-Message Time and Reception Time Separately

Autopilot Clock	**time_boot_ms:** position timestamp within the message; save with position and velocity.
Host Clock	**Monotonic reception time:** arrival intervals, dropped frames, and scheduling; differs from the autopilot clock.
Reference Clock	**Reference time:** position, velocity, acceleration (PVA), pre/post-limit values, and send time; updates at about 30 Hz.

Final state and exceptions: log mode, landed and armed states, timeouts, and fallback events.

Acceptance: assess RMS/peak error, saturation fraction, message intervals and final state together; set task-specific thresholds.
uavTimeStmp updates only with ATTITUDE reception; uavMsg holds only the latest message.

Chapter Outline

External Control Examples and Experimental Validation

Chapter 6
External Control and Trajectory Planning

6.1 External Control Fundamentals
6.2 Control Principles and Modes
6.3 Trajectory Models and Generation
6.4 Planning and Obstacle Avoidance
6.5 Tracking and Aggressive Flight Control
6.6 Frontier Control Technologies
6.7 Platform and Communication Interfaces
6.8 Examples and Experimental Validation
6.9 Exercises and Classroom Discussion

5 Units in This Section

6.8.1 Remote Control and Mission-Mode Experiments
Textbook: Secs. 6.1.2.1–6.1.2.2, 6.3.1

6.8.2 Trajectory Execution and Result Validation
Textbook: Secs. 6.2.2.5, 6.3.1

6.8.3 Learning and Vision Control Experiments
Textbook: Secs. 6.1.4.2, 6.3.1

6.8.4 Foundation-Model Control Experiments
Textbook: Secs. 6.1.4.3, 6.3.1

6.8.5 Advanced Applications and Real-Vehicle Transfer
Textbook: Secs. 6.3.2, 6.2.2.6

6.8.3 Learning and Vision Control Experiments: Example Steps

Optional Example 1: Read the Single-Axis RL Model; Check Tracking

Directory: [Installation directory]/RflySimAPIs/6.RflySimExtCtrl/
2.AdvExps/e5.RLExps/1.Simulink/Readme.pdf

Needs MATLAB R2022b or later, Simulink, RL (reinforcement learning) and deep-learning toolboxes.

Weights: initialization loads UAVDPPG.mat from the same folder. Optional training: train/rluav_init.m.

Start: run RflyUdpMavlinkRealSim.bat; wait for CopterSim to report GPS 3D fixed & EKF initialization finished, then run rlhighDynamicAcc.slx.

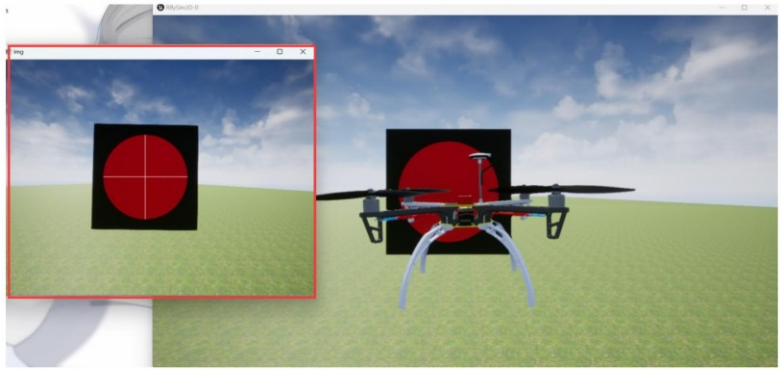
Operate: selector Offboard → Arm → Takeoff → Flying; set desired X to about 1 m.
Expected: QGC still shows Flying; the aircraft moves fairly smoothly to desired X in RflySim3D.

End: center slider → Land → Disarm after landing → stop model → exit simulation.
Troubleshooting: no takeoff—check GPS; cache error—delete slprj; out of view—use scroll wheel.

6.8.3 Learning and Vision Control Experiments: HSV Following Steps

Circular-Board Following with Hue–Saturation–Value (HSV) Detection

Directory: [Installation directory]/RflySimAPIs/6.RflySimExtCtrl/2.AdvExps/e6.IsaacSimExps/
RL_attctrl/: circle_follow.bat, Python38Run.bat, circle_follow.py



Following scene (Readme)

Start: run circle_follow.bat and wait for simulation initialization.

Run: in Python38Run.bat, enter python circle_follow.py. The script calls initOffboard and arms; see Section 6.7, “SDK Side Effects.”

Expected: move the board with arrow keys and Ctrl+↑/↓; the aircraft follows it.

End: press Enter in the bat window. It closes CopterSim, QGC, RflySim3D and forcibly terminates ALL local python.exe and cmd.exe processes. Save other Python work first.

Troubleshooting: no image—check Config.json: SendProtocol = 0 (shared memory; Windows only).

Chapter Outline

External Control Examples and Experimental Validation

Chapter 6
External Control and Trajectory Planning

6.1 External Control Fundamentals
6.2 Control Principles and Modes
6.3 Trajectory Models and Generation
6.4 Planning and Obstacle Avoidance
6.5 Tracking and Aggressive Flight Control
6.6 Frontier Control Technologies
6.7 Platform and Communication Interfaces
6.8 Examples and Experimental Validation
6.9 Exercises and Classroom Discussion

5 Units in This Section

6.8.1 Remote Control and Mission-Mode Experiments
Textbook: Secs. 6.1.2.1–6.1.2.2, 6.3.1

6.8.2 Trajectory Execution and Result Validation
Textbook: Secs. 6.2.2.5, 6.3.1

6.8.3 Learning and Vision Control Experiments
Textbook: Secs. 6.1.4.2, 6.3.1

6.8.4 Foundation-Model Control Experiments
Textbook: Secs. 6.1.4.3, 6.3.1

6.8.5 Advanced Applications and Real-Vehicle Transfer
Textbook: Secs. 6.3.2, 6.2.2.6

6.8.4 Foundation-Model Control Experiments: Installed Examples

Four Installed Entries Produce Four Different Outputs

Directory: [Installation directory]/RflySimAPIs/6.RflySimExtCtrl/
0.ApiExps/e20.LLMUsage/ (Readme.pdf in each subdirectory)

Directory and Entry	Actual Output	Check First
1.CloudAPIinvocation **LLMDSApiTest.py**	Model text response	Service configuration and response format
2.LLMGenUAVCtrlCode **UAVCtrl.py**	Control program	Interfaces, frames, units, and exit branches
3.LLMDiretCtrlUAV **llmCtrluav.py**	Parse a command list, then call the position interface	Parameter checks, results, and final state
4.LocalLLMUAVCtrl **smoothAvoid_LLM_sim.slx**	Local command parsing + cloud route reasoning	WSL2-GPU add-on (about 50 GB) and API key

6.8.4 Foundation-Model Control Experiments: Direct-Control Steps

Language Direct Control: Start Simulation, Then the Interface

Directory: [Installation directory]/RflySimAPIs/6.RflySimExtCtrl/0.ApiExps/e20.LLMUsage/3.LLMDiretCtrlUAV/: SITLRun.bat, Python38Run.bat, llmCtrluav.py

Prepare: in 文档/Ogre/.env (Documents folder), set the DeepSeek key, endpoint, and model name; format: 1.CloudAPIinvocation.

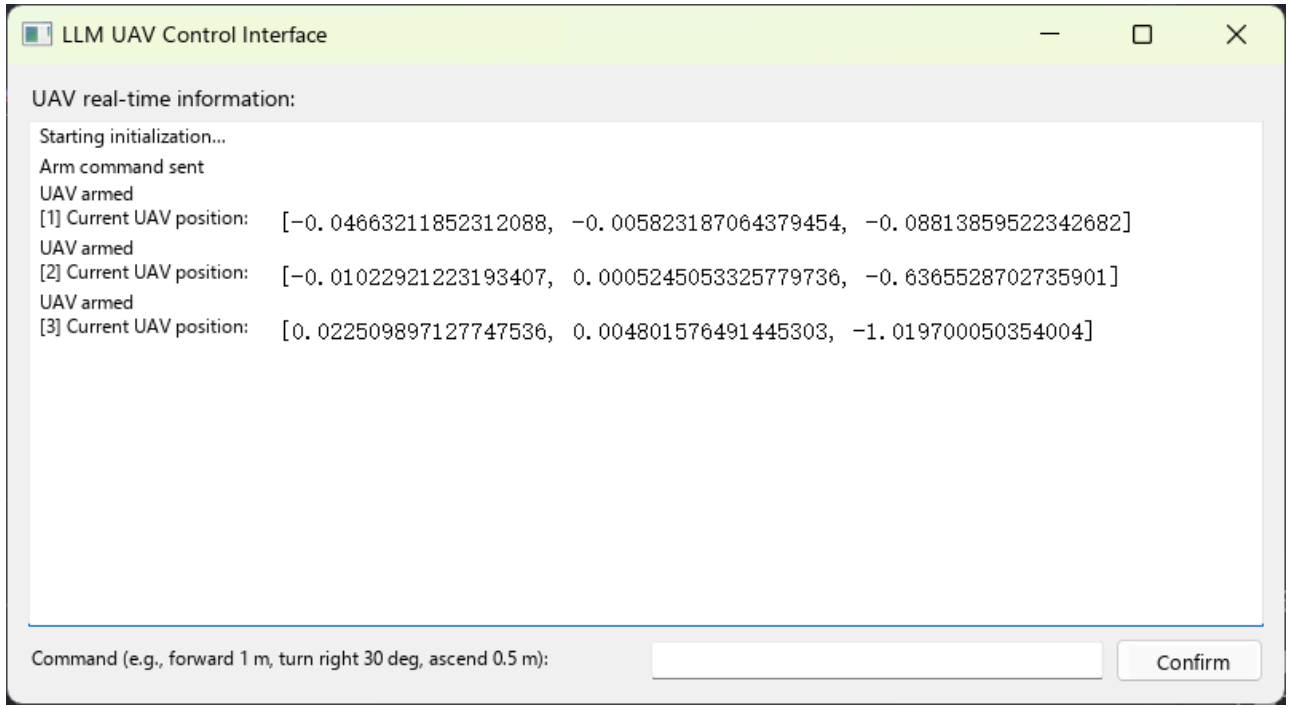
Start: run SITLRun.bat to open QGC, CopterSim, and RflySim3D. Wait for GPS 3D fixed & EKF initialization finished.

Run: enter python llmCtrluav.py in Python38Run.bat. After interface initialization, the aircraft automatically climbs to hover at 1.7 m and waits for commands (initOffboard: Section 6.7, “SDK Side Effects”).

End: closing the interface calls endOffboard, then stopRun; closing it in flight forces disarming. Press any key in the SITL window to close the three applications.

6.8.4 Foundation-Model Control Experiments: Direct-Control Observations

The Interface Echoes Parsing; Telemetry Establishes Completion



Direct-control interface output (Readme screenshot)

Expected: enter “向前飞行10米” (“fly forward 10 m”), for example. The interface prints parsed commands, position, and execution state.

Acceptance: use position and telemetry; response text does not establish completion.

Troubleshooting: no takeoff—check initialization logs and API key; commands not executed—check model configuration in .env.

6.8.4 Foundation-Model Control Experiments: Other Examples

Running, Observing, and Troubleshooting the Other Three Examples

Directory: [Installation directory]/RflySimAPIs/6.RflySimExtCtrl/
0.ApiExps/e20.LLMUsage/ (Readme.pdf in each subdirectory)

Run order: start simulation/model, then Python. Each Readme.pdf has step-by-step screenshots.

Directory and Entry	Run and Expectation	Common Issues
1.CloudAPIinvocation LLMDSApiTest.py	Run in Python38Run.bat; Terminal prints the model reply	Authentication error: check .env key, quota, and model name
2.LLMGenUAVCtrlCode UAVCtrlSILRun.bat	Template + prompt → code; take off and fly a simulated circle	No response: check EKF logs; wrong circle: check north-east-down (NED) and radians
4.LocalLLMUAVCtrl smoothAvoid_LLM_sim.slx ServerFile/WinWSL2.bat	Enter “所有飞机起飞” (“all aircraft take off”); six aircraft take off, then test both modes	Requires WSL2-GPU and a key; no acknowledgment (Ack): check User Datagram Protocol (UDP) ports

6.8.4 Foundation-Model Control Experiments: Termination and Cleanup

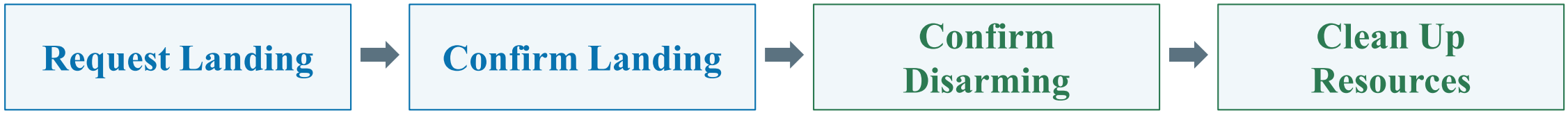
Control Termination and Resource Cleanup Have a Required Order

Side effects: see Section 6.7, “SDK Side Effects.” initOffboard changes failsafe parameters and automatically arms.

End sequence: see Section 6.7, “Termination and Exceptions.” endOffboard only stops the setpoint stream; it does not land.

Criterion here: tool success only confirms arrival. End the task only when fresh state confirms landed and disarmed.

Exception branch: stopRun forces disarming when armed. Window-close and exception handlers must not call it unconditionally.



Chapter Outline

External Control Examples and Experimental Validation

Chapter 6
External Control and Trajectory Planning

6.1 External Control Fundamentals
6.2 Control Principles and Modes
6.3 Trajectory Models and Generation
6.4 Planning and Obstacle Avoidance
6.5 Tracking and Aggressive Flight Control
6.6 Frontier Control Technologies
6.7 Platform and Communication Interfaces
6.8 Examples and Experimental Validation
6.9 Exercises and Classroom Discussion

5 Units in This Section

6.8.1 Remote Control and Mission-Mode Experiments
Textbook: Secs. 6.1.2.1–6.1.2.2, 6.3.1

6.8.2 Trajectory Execution and Result Validation
Textbook: Secs. 6.2.2.5, 6.3.1

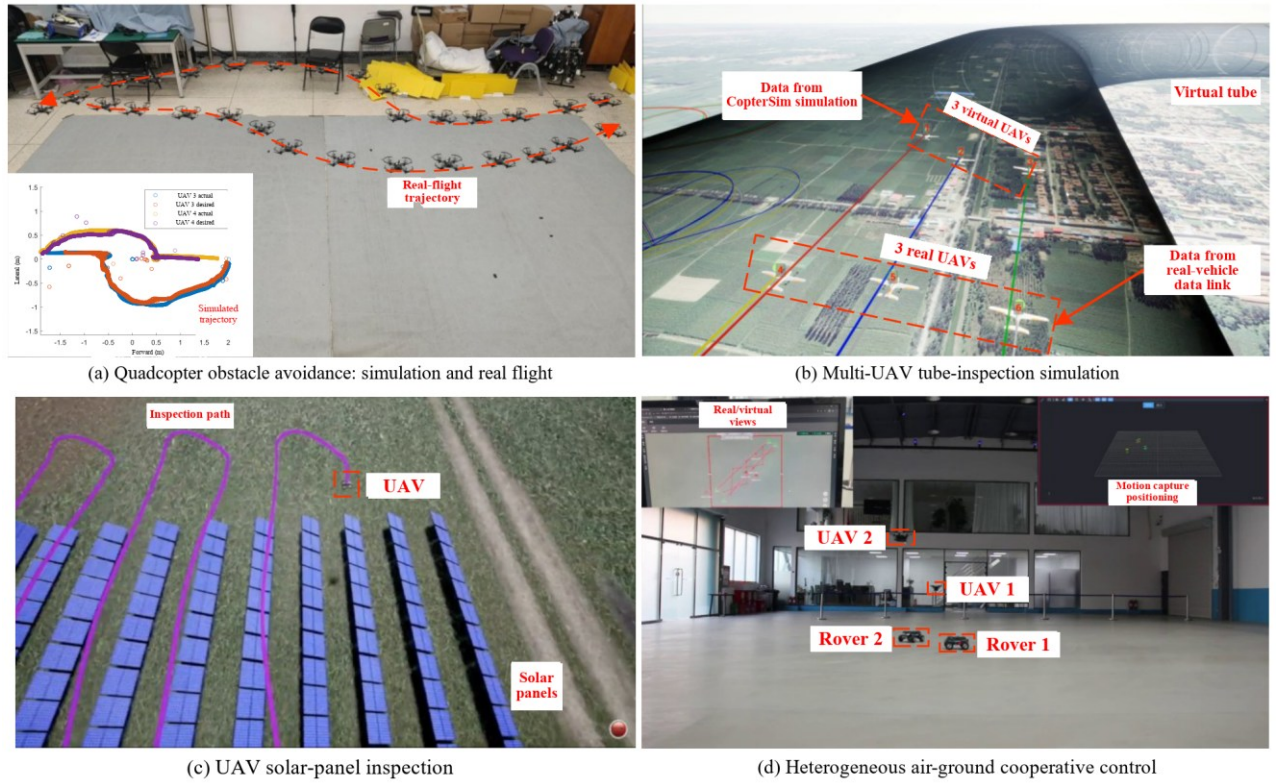
6.8.3 Learning and Vision Control Experiments
Textbook: Secs. 6.1.4.2, 6.3.1

6.8.4 Foundation-Model Control Experiments
Textbook: Secs. 6.1.4.3, 6.3.1

6.8.5 Advanced Applications and Real-Vehicle Transfer
Textbook: Secs. 6.3.2, 6.2.2.6

6.8.5 Advanced Applications and Real-Vehicle Transfer: Advanced Case Overview

Advanced Cases Combine Perception, Planning, and Control



(a) Quadcopter obstacle avoidance: simulation and real flight

(b) Multi-UAV tube-inspection simulation

(c) UAV solar-panel inspection

(d) Heterogeneous air-ground cooperative control

Figure 6.12 Advanced Application Examples

Avoidance: replanning and tracking response.

Multiple aircraft: separation in space/time and task coordination.

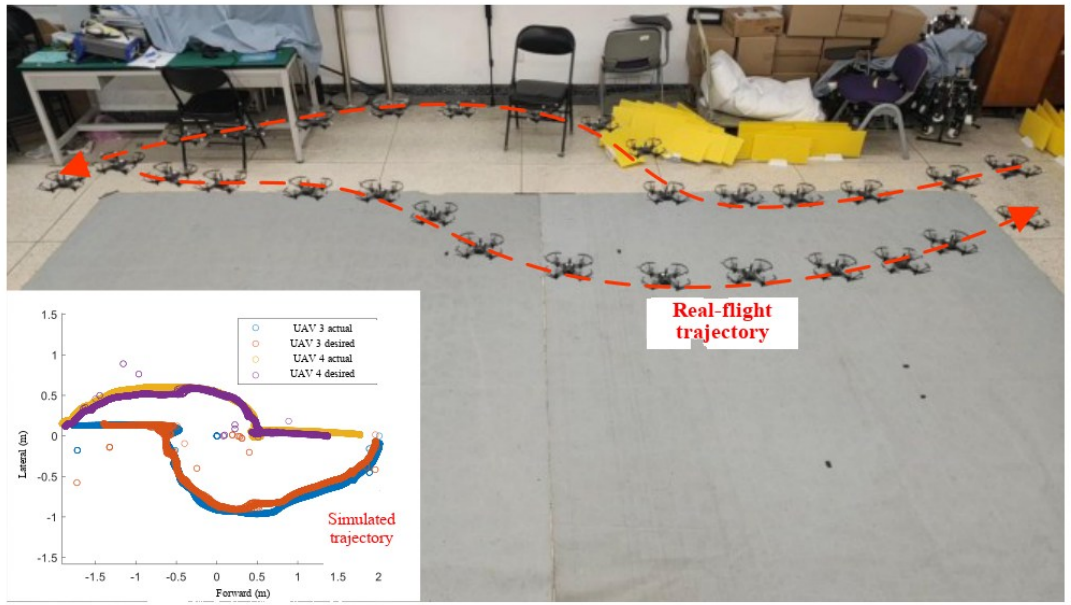
Inspection: region detection and coverage trajectories.

Air–ground collaboration: shared coordinates and states.

Textbook Sec. 6.3.2: these platform cases do not prescribe a learning method; use them to test the earlier units' feedback, interface, and evidence requirements.

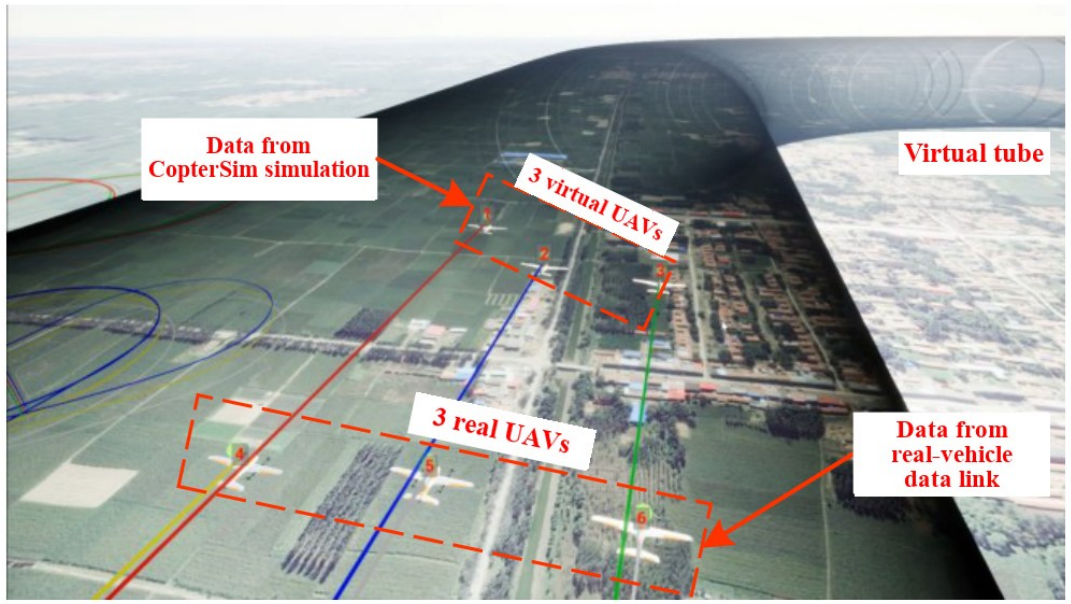
6.8.5 Advanced Applications and Real-Vehicle Transfer: Case Details 1

Avoidance and Multi-Aircraft Inspection Face Different Constraints



(a) Quadcopter obstacle avoidance: simulation and real flight

Figure 6.12(a) Avoidance: check obstacle clearance, trajectory replanning, and tracking error.

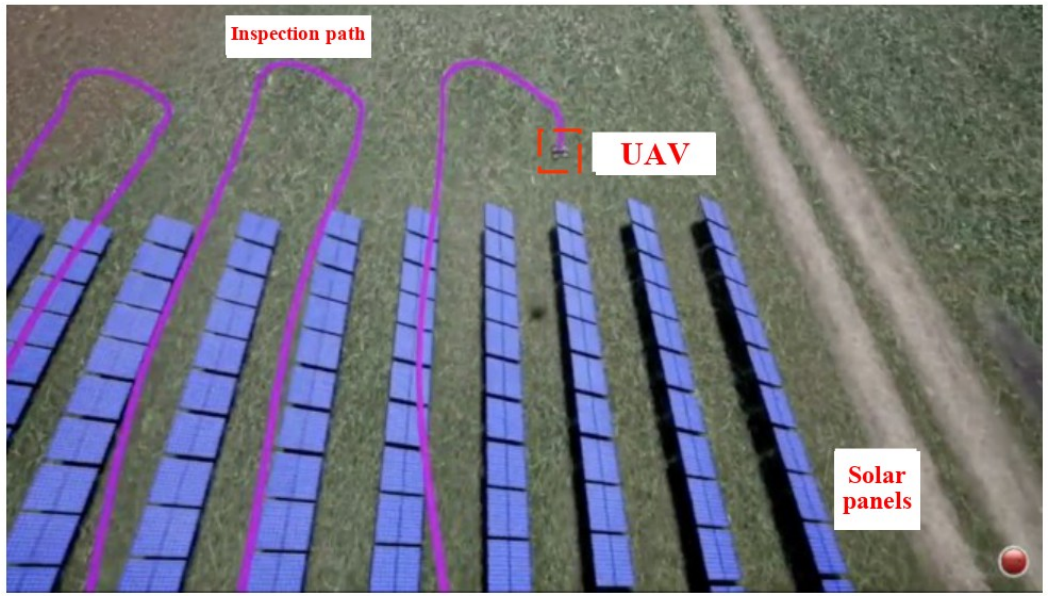


(b) Multi-UAV tube-inspection simulation

Figure 6.12(b) Multi-aircraft inspection: check timing, inter-aircraft separation, and feedback to centralized planning.

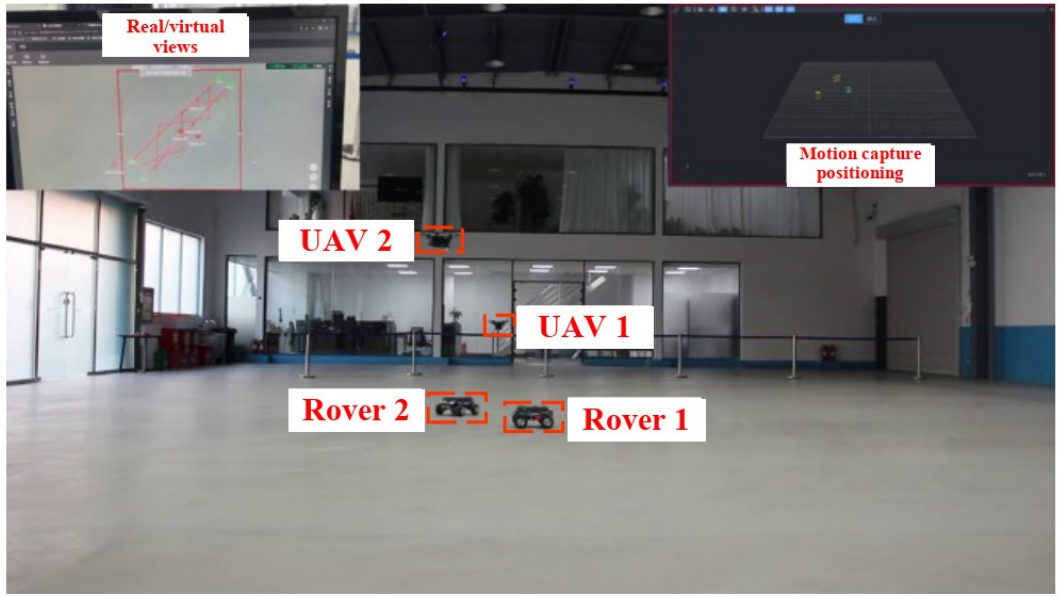
6.8.5 Advanced Applications and Real-Vehicle Transfer: Case Details 2

Inspection and Heterogeneous Collaboration Need Shared Task Meaning



(c) UAV solar-panel inspection

Figure 6.12(c) Detected region → coverage trajectory;
record gaps, overlap and image quality.



(d) Heterogeneous air-ground cooperative control

Figure 6.12(d) Share coordinates/task state.
Text: LiDAR mapping/localization; image: motion capture.

6.8.5 Advanced Applications and Real-Vehicle Transfer: Reading Results

What Can a Trajectory Crop Establish?

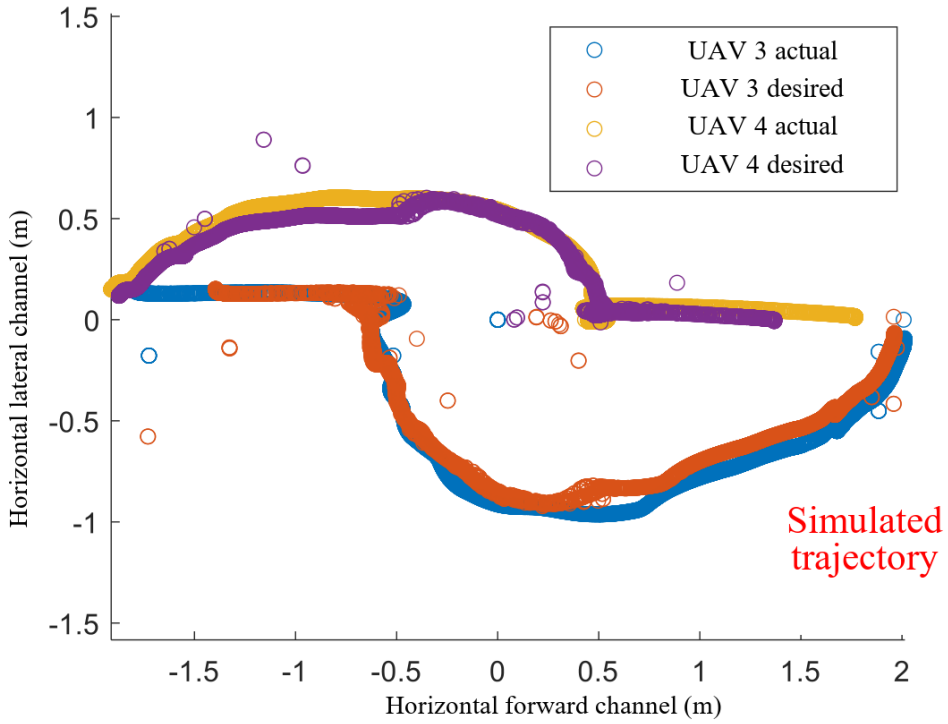


Figure 6.12(a) Enlarged Simulation-Trajectory Detail

Vehicle 3 actual (blue)

Vehicle 3 desired (orange)

Vehicle 4 actual (yellow)

Vehicle 4 desired (purple)

Visible: desired and actual paths for two vehicles; coordinate units are meters.

No time: similar geometric curves do not show on-time arrival at every instant.

Full evaluation: align reference and feedback times; check errors, constraints and final state.

6.8.5 Advanced Applications and Real-Vehicle Transfer: Real-Vehicle Experiments

Revalidate Timing and Exit Behavior When Deploying to Real Vehicles

Expand validation step by step, keeping the same record format and error statistics.

Numerical Computation
Fix reference, control law, and limits; save an error baseline.

Model-in-the-Loop
Replace the reduced-order model with a six-degree-of-freedom model; keep record format. Check Coder licenses.

SITL / HITL
PX4 via MAVROS and CopterSim relay; software-in-the-loop, then hardware-in-the-loop. Check message updates and clock synchronization.

Real Vehicle
Onboard or ground deployment; MAVLink Router forwarding. Match serial baud rates (e.g., 57600/115200). Recalibrate thrust mapping; verify takeover, landing and parameter restoration.

See Chapter 10 for multi-vehicle time synchronization and coordination. First retain complete single-vehicle error and exit records.

Classroom Exercise

Judge Tracking with Calculations, Source Code, and Records

Question	Key Points
Can a 3 m radius circle at 5 m/s meet a 35° tilt limit?	Centripetal acceleration: 8.33 m/s²; tilt: 40.3°. Exceeds 35°: infeasible. Speed must be ≤4.54 m/s.
How do you check the negative-direction circle? Does program exit prove landing?	Differentiate position component by component; original X velocity repeats the direction factor. endOffboard does not land; confirm landing and disarming.
For a 5 m radius circle with amplitude ratio A = 1.2 and lag φ = 0.1 rad, how large is error? What dominates?	$\|e\| = R\sqrt{(1 + A^2 - 2A\cos\varphi)} \approx 1.14$ m. Amplitude alone: 1.0 m; lag alone: 0.5 m. Amplitude dominates.

Classroom Exercise

Stopping Margin and Conclusions from Plots

Question	Key Points
A teaching vehicle starts at 2 m/s, decelerates at 1 m/s², and has 0.3 s total sensing/actuation delay. What minimum stopping distance is needed?	Delay travel: $2 \times 0.3 = 0.6$ m; braking: 2 m. Total: 2.6 m, plus a localization-error margin.
In Figure 6.12(a), desired and actual paths are geometrically close. Is tracking error small?	Not yet established: the plot has no timestamps. Align reference and feedback at the same time, then compute error.

Section Summary

Completion Requires a Closed Loop and Final State, Not Just Sending

Consistent reference: derive position, velocity, and acceleration from one trajectory; use consistent frames, times, and units.

Effective feedback loop: feedforward supplies known motion demand; feedback corrects residual error. Keep execution margin.

Interface adaptation: attitude and thrust are coupled; calibrate normalized commands and compare algorithms under stated conditions.

Credible results: assess error, amplitude, phase, and final state together; printed references and exit statements cannot replace feedback.

Pitfalls: multiplying the direction factor twice; negative vertical velocity means ascent; program exit does not mean landing.

Timed references, executable control, and verifiable results together close the task.