Chapter Outline

RflySim: From Algorithm Development to Real-Vehicle Validation

Chapter 1 Introduction

1.1 Concepts and Applications
1.2 History and Trends
1.3 Components and Architecture
1.4 Coordinate Systems and Control Principles
1.5 Assembly and Debugging
1.6 RflySim Toolchain
1.7 Learning Path and Experiments
1.8 Exercises and Discussion

4 Units in This Section

1.6.1 Background and Platform Role
Textbook: Sec. 1.4

1.6.2 Design Philosophy and Features
Textbook: Sec. 1.4.2

1.6.3 Representative Cases
Perception, collaboration, disturbance rejection, cross-vehicle tests

1.6.4 Teaching, Research, and Learning Resources
Textbooks, documentation, cases, and papers

1.6.1 Background and Platform Role: Experiments and Simulation

Real-Vehicle Tests Are Direct but Costly; Simulation Depends on Models

Table 1.1 Experiment-Based vs. Simulation-Based Development and Testing

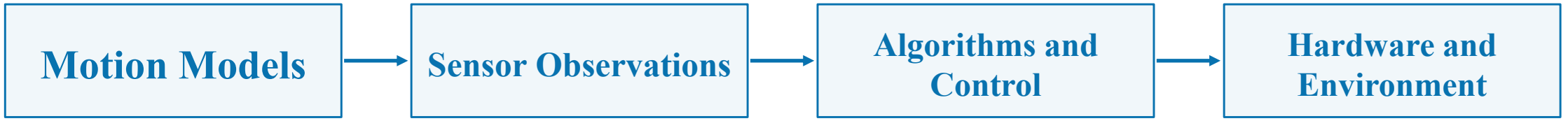
Aspect	Experiment-Based Development and Testing	Simulation-Based Development and Testing
Safety	Fast-spinning rotors make experiments hazardous, especially for students.	Software-only: lower injury and damage risks. Hardware requires safety checks.
Space	Indoor space is limited; outdoor airspace is hard to obtain.	No airspace limits; simulate any condition. Inject faults; access ground truth at any time.
Time	Debugging and testing take considerable time, mostly spent on hardware rather than algorithms.	Available at any stage of development.
Cost	Expensive hardware, frequent crashes, and rapid obsolescence.	Low-cost, efficient parallel tests; uncertain credibility. Mainly for development and functional tests.

Use simulation for repeatable functional tests and real vehicles for key validation.

1.6 RflySim Toolchain

1.6.1 Background and Platform Role: Practical Simulation Challenges

Building Experimental Platforms Diverts Effort from Algorithm Research



Interfaces exchange model states, observations, and control commands

- **Interface alignment:** Model states, sensor data, and control commands must share units, coordinates, timing, and formats.
- **Environment integration:** Simulators, renderers, development tools, and hardware have separate dependencies.

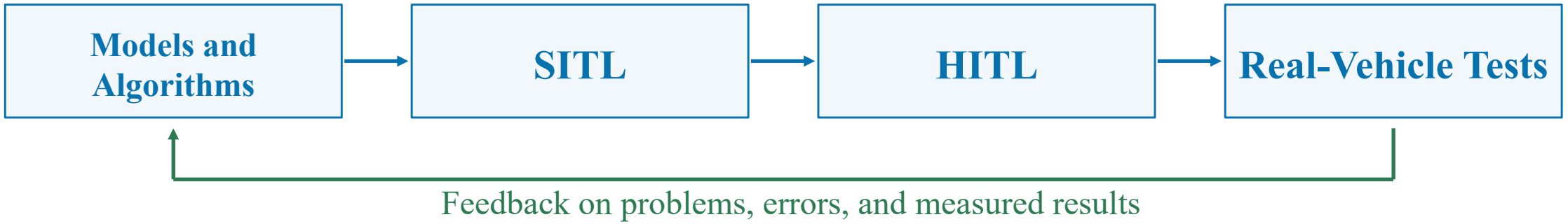
A working program does not guarantee a connected system.

- **Reuse:** Integrated models, interfaces, and examples reduce setup work for the next experiment.

Connect models, observations, algorithms, and hardware to run system experiments.

1.6.1 Background and Platform Role: Simulation Verification Requirements

Simulation Reveals Problems Before Deployment



- **Repeat conditions:** Reset initial states and disturbances to compare motion before and after algorithm changes.
- **See inside:** Read model states and ground truth to locate error sources.
- **Extend tests:** Create abnormal cases in software-in-the-loop (SITL; textbook: SIL), then connect a real autopilot for hardware-in-the-loop (HITL; textbook: HIL) simulation.

Feed test results back into models and algorithms to improve development.

1.6.1 Background and Platform Role: Existing Platforms and Limits (Optional)

Common Simulation Platforms Have Different Strengths and Limits

Platform	Main Strengths	Main Limitations
X-Plane	Realistic flight dynamics and HITL support.	Closed source; weaker support for small aircraft such as multicopters.
FlightGear	Mature open-source community; formation and networked flight.	Limited rendering quality and physical accuracy.
Gazebo	Multiple physics engines; simulated sensors. Robot Operating System (ROS) integration.	Modest rendering quality; complex scene construction.
jMAVSim	Official lightweight PX4 simulator; easy deployment.	Simplified scenes and sensor models; limited support for complex vision.
AirSim	Realistic imagery rendered with Unreal Engine (UE).	Scene construction requires the UE editor and powerful hardware.

Dynamics, imagery, or networked flight: choose the platform for the task.

1.6 RflySim Toolchain

1.6.1 Background and Platform Role: Unification and a Common Framework

RflySim: A Toolchain for Unmanned Systems and Embodied Intelligence

RflySim: Reliable Flight Simulation, now extended to diverse embodied unmanned systems



Aerial and Ground Platforms: Real-Vehicle Tasks



Ground Vehicle: Driving Simulation

Extend flight-system modeling, simulation, and validation to other vehicles.

1.6 RflySim Toolchain

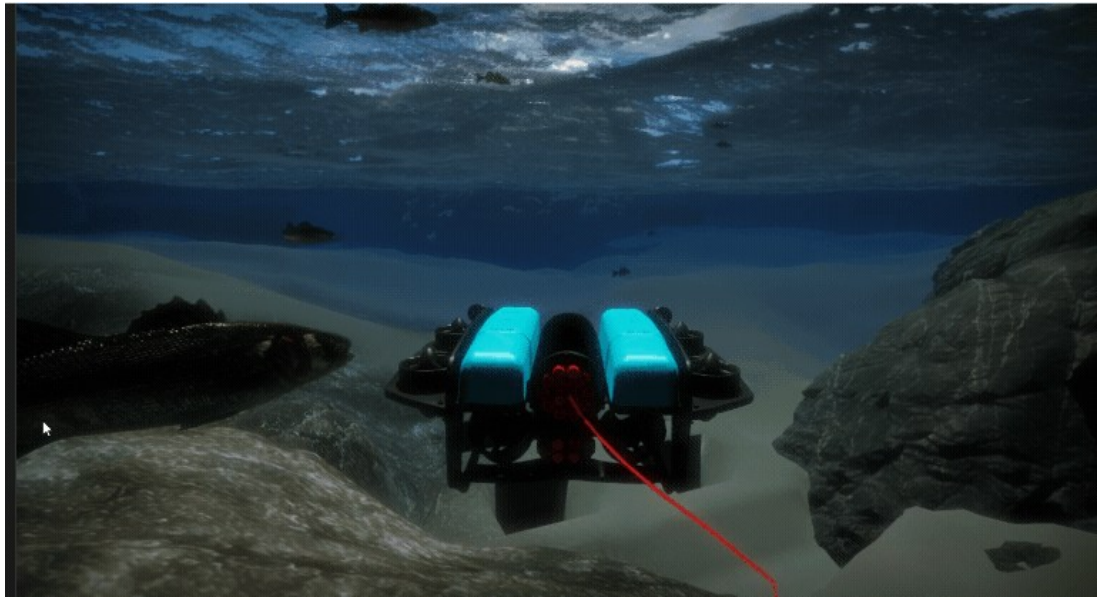
1.6.1 Background and Platform Role: Unification Across Vehicle Types

Embodied Locomotion and Underwater Exploration Use the Same Workflow

Vehicle forces, constraints, and sensing differ; interfaces link models and algorithms to one framework.



Quadruped: Embodied Motion Simulation



Underwater Exploration: 3D Motion Simulation

Share the development and validation workflow; adapt models and task algorithms.

1.6.1 Background and Platform Role: Unification and Vehicle Models

One Modeling Framework Provides Models for Many Vehicle Types



Fig. 1.17 3D Simulation Models for Multiple Vehicle Types

Top: Quadrotor, Fixed-Wing, Hybrid Vertical-Takeoff Aircraft
Bottom: Unmanned Surface Vehicle, Helicopter, Unmanned Ground Vehicle

- **Modeling:** Build graphical MATLAB Simulink models from built-in general motion models.
- **Model generation:** Change mass, dimensions, and propulsion parameters, or identify a new model from measured data.
- **Shared workflow:** Change vehicles while keeping the modeling-to-evaluation process.

Reuse model libraries; identify parameters to adapt models to each vehicle.

1.6.1 Background and Platform Role: Functional Completeness

Unmanned Systems: From Assembly Tests to Real-Vehicle Deployment

The toolchain links modeling, perception, diagnosis, planning, control, networking, and swarm modules. Artificial intelligence (AI) helps write and check code.



Assembly-Test Example: Virtual Unmanned Aircraft Disassembly and Assembly

Assembly Tests →	**Algorithm Development** →	**HITL Simulation** →	**Real-Vehicle Deployment**
Components and Wiring	Models, Code, Metrics	Interfaces and Timing	Measured Data; Refinement

Use AI to rapidly design, assemble, and validate virtually, then rapidly transfer to real vehicles.

1.6 RflySim Toolchain

1.6.1 Background and Platform Role: A Running Example

One Autonomous Navigation Task Spans Development and Validation



Indoor Autonomous Motion and Collaboration (Real Vehicles)

- **Task:** Sense the environment, avoid obstacles, and reach the target.
- **Roles:** Localization estimates position, mapping represents the environment, planning selects a path, and control drives motion.
- **Validation:** Check arrival, trajectory error, obstacle clearance, and response to environmental changes.

Environment Sensing → **Localization and Mapping** → **Path Planning** → **Motion Control**

Validate sensing, planning, and control together across the full task workflow.

Chapter Outline

RflySim: From Algorithm Development to Real-Vehicle Validation

Chapter 1 Introduction

1.1 Concepts and Applications
1.2 History and Trends
1.3 Components and Architecture
1.4 Coordinate Systems and Control Principles
1.5 Assembly and Debugging
1.6 RflySim Toolchain
1.7 Learning Path and Experiments
1.8 Exercises and Discussion

4 Units in This Section

1.6.1 Background and Platform Role
Textbook: Sec. 1.4

1.6.2 Design Philosophy and Features
Textbook: Sec. 1.4.2

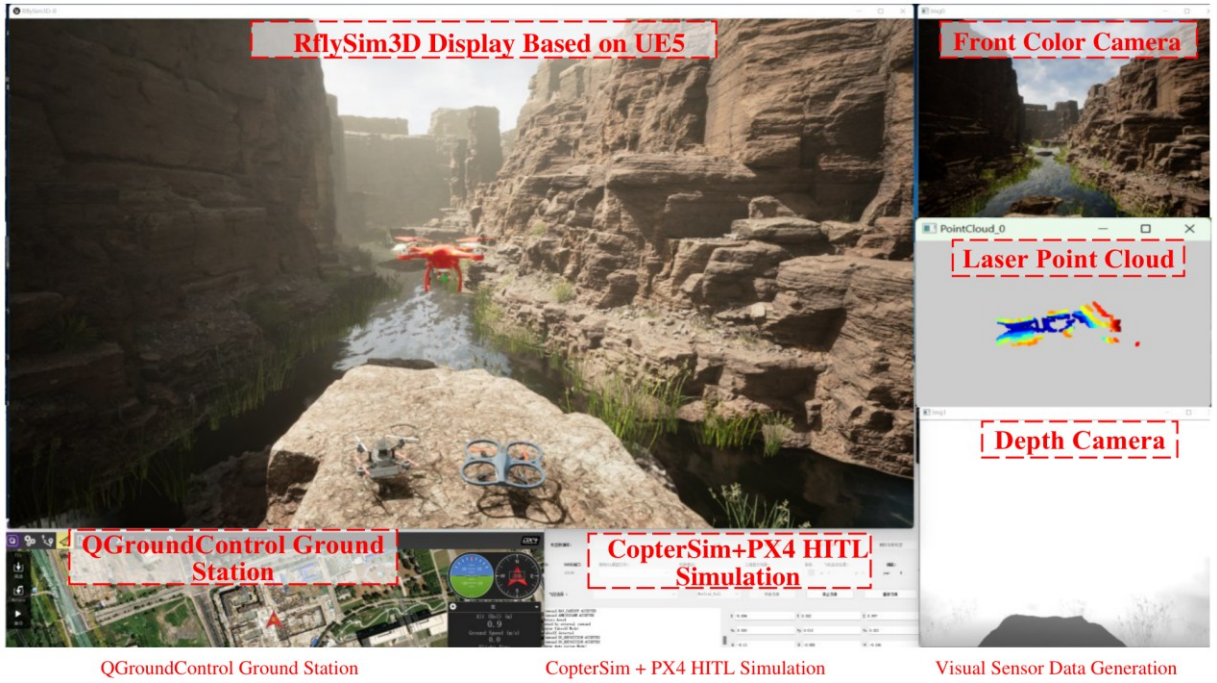
1.6.3 Representative Cases
Perception, collaboration, disturbance rejection, cross-vehicle tests

1.6.4 Teaching, Research, and Learning Resources
Textbooks, documentation, cases, and papers

1.6 RflySim Toolchain

1.6.2 Design Philosophy and Features: Tool Components

3D Rendering, Motion Simulation, and a Ground Station Work Together



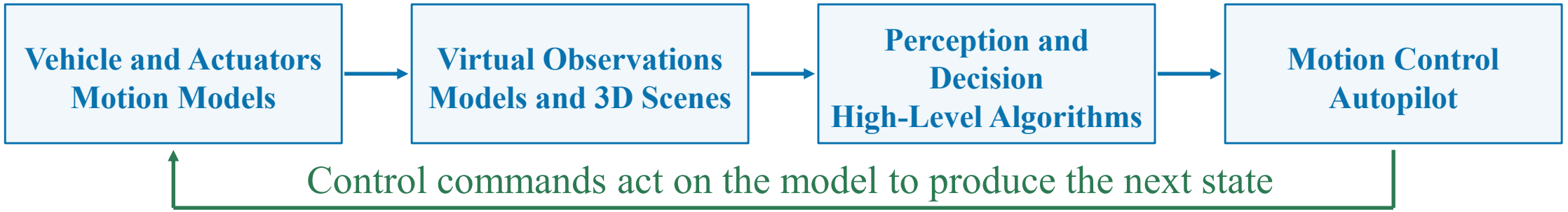
Main Software Windows in a RflySim Simulation

- **3D rendering:** RflySim3D uses Unreal Engine (UE) to render scenes and generate camera, point-cloud, and depth data.
- **Motion simulation:** CopterSim runs the vehicle model and exchanges sensor data and motor commands with the PX4 autopilot.
- **Ground station:** QGroundControl plans missions and displays position and attitude.

Algorithm actions must change the next observation, beyond displaying motion.

1.6.2 Design Philosophy and Features: Module Roles

Models, Algorithms, Scenes, and Hardware Form a Development Loop

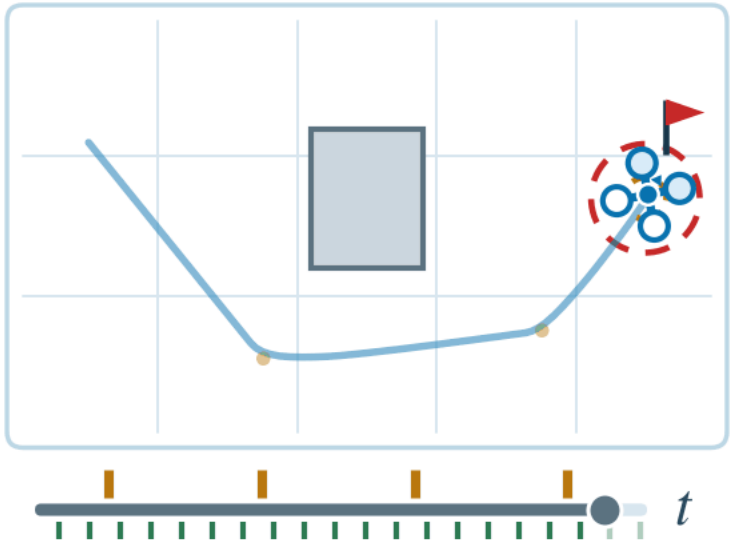


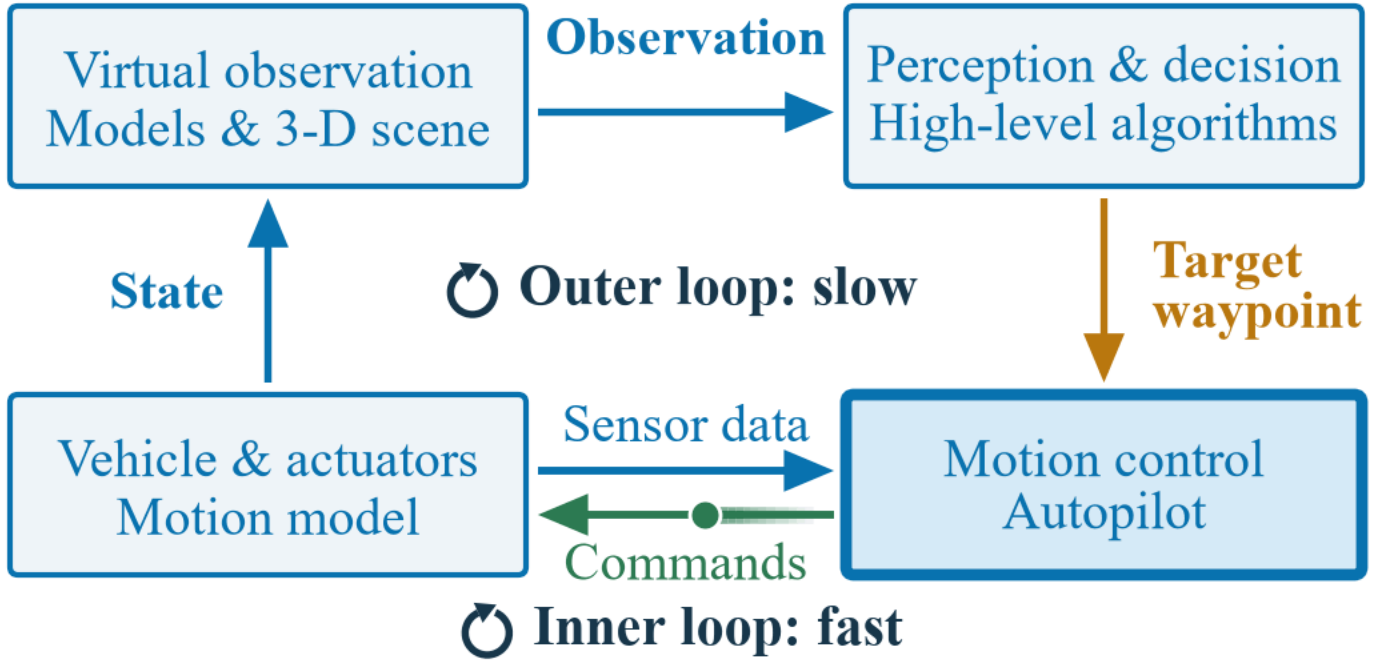
- **Motion:** CopterSim updates position, attitude, and velocity from control outputs.
- **Observation and decision-making:** RflySim3D generates images and point clouds for navigation algorithms to plan paths.
- **Commands and monitoring:** The autopilot turns path targets into actuator outputs. The ground station configures missions and displays states.

States generate observations; algorithm commands then change those states.

1.6.2 Design Philosophy and Features: Closed-Loop Timing

The Inner Loop Corrects Motion Between Outer-Loop Target Updates



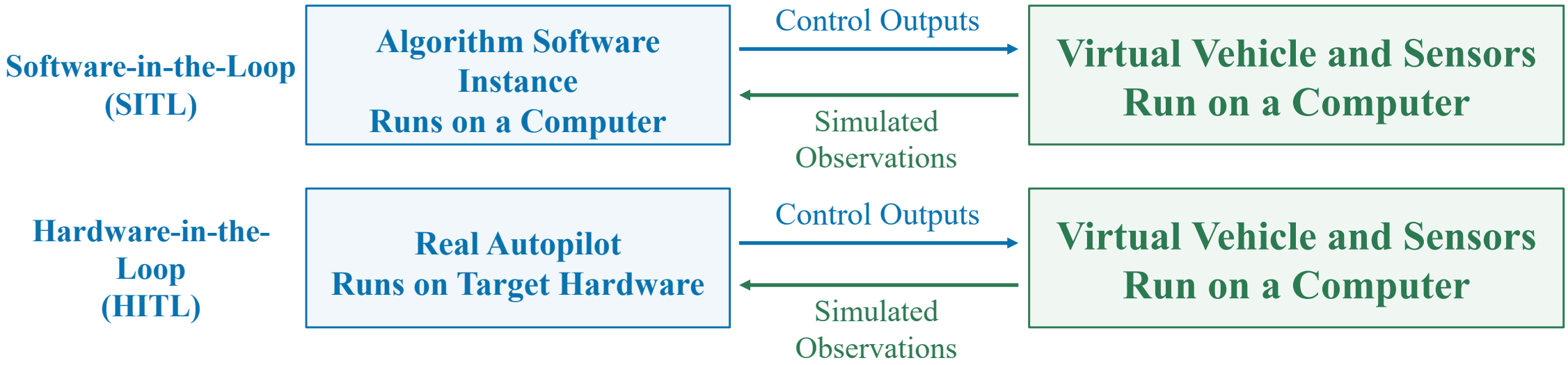


Teaching Simulation of a Multirate Loop: Example Period Ratio 5：1

The outer loop updates targets; the inner loop corrects motion between updates.

1.6.2 Design Philosophy and Features: In-the-Loop Verification

SITL and HITL Change Where the Tested Code Runs



- **SITL:** Run algorithms and the virtual vehicle on a computer; compare avoidance routes and tracking errors first.
- **HITL (semi-physical simulation):** Connect a real autopilot to check interfaces and real-time execution.

Real hardware reveals how algorithm implementations execute on the target.

1.6.2 Design Philosophy and Features: High Efficiency and the Test Workflow

Validate in Stages: SITL, HITL, and Real-Vehicle Experiments

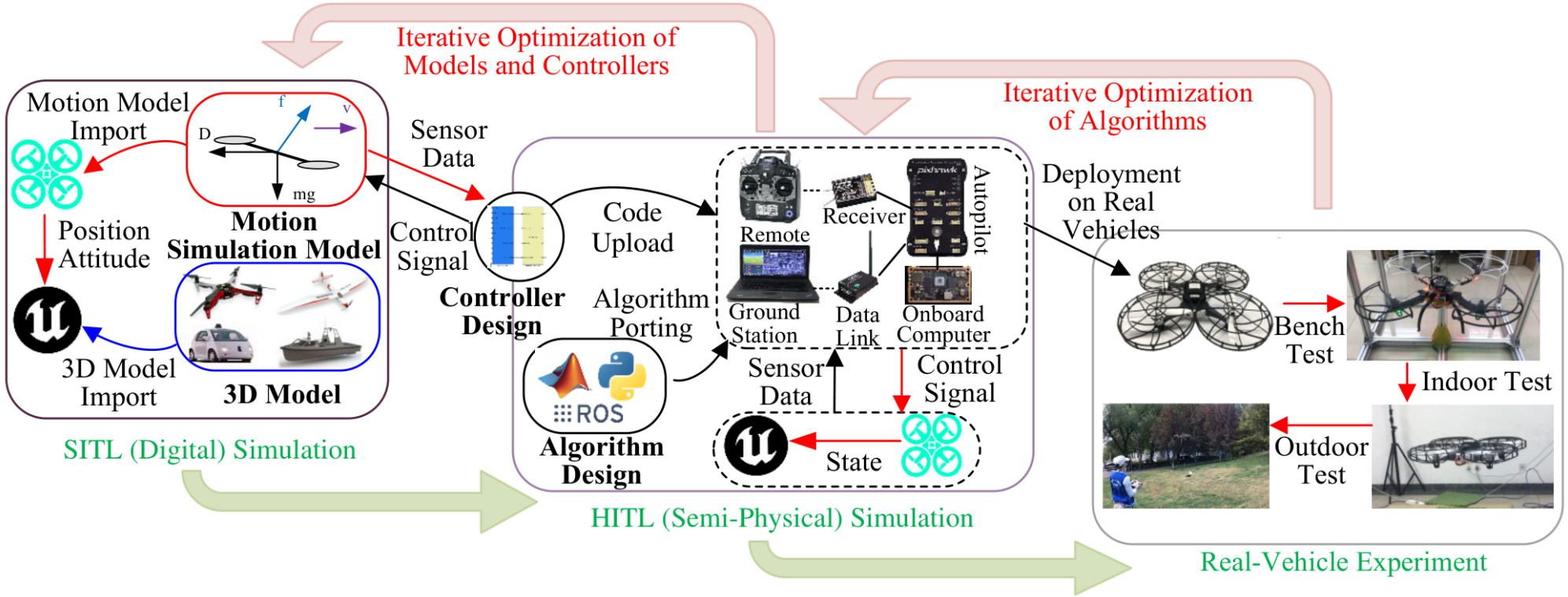


Fig. 1.18 RflySim Efficient Development and Testing Framework

1.6.2 Design Philosophy and Features: High Efficiency and a HITL Close-Up

HITL Connects Real Control Hardware to a Simulated Closed Loop

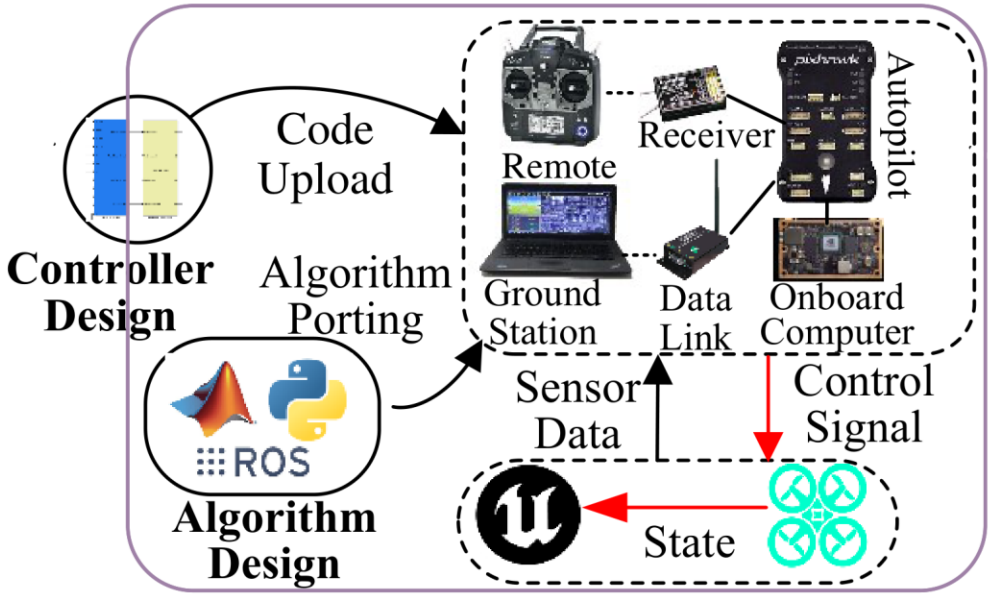


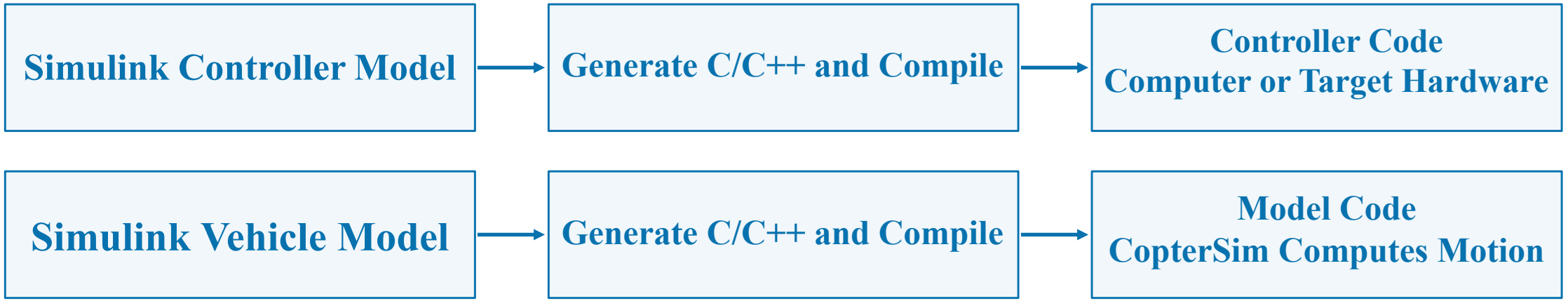
Fig. 1.18 (Detail) HITL (Semi-Physical) Simulation

- **Real equipment:** Autopilot, remote controller and receiver, ground station, telemetry radios, and onboard computer.
- **Simulation supplies:** Vehicle motion and sensor data with synchronized 3D display.
- **Code upload:** Generate Simulink controller code and upload it to the autopilot.
- **Algorithm porting:** Run Python/ROS high-level algorithms on the onboard computer.

Real hardware supports repeatable checks of interfaces and execution timing.

1.6.2 Design Philosophy and Features: High Efficiency

Automatic Code Generation Reduces Manual Model-to-Code Conversion

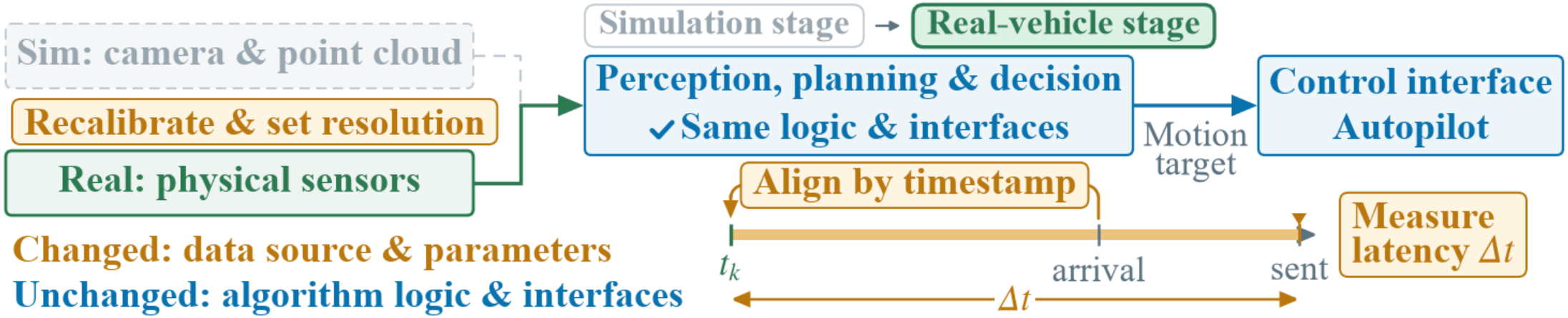


- **Controller code:** Generate C/C++ from Simulink blocks; compile and run it on a computer or autopilot.
- **Vehicle-model code:** Generate code from the Simulink vehicle model. CopterSim computes motion from thrust and torque.
- **Iteration:** After model changes, regenerate and compile code; compare errors and execution timing under the original conditions.

Controller code issues commands; vehicle-model code computes motion.

1.6.2 Design Philosophy and Features: Portability

Shared Interfaces Support Perception and Decision Algorithm Transfer



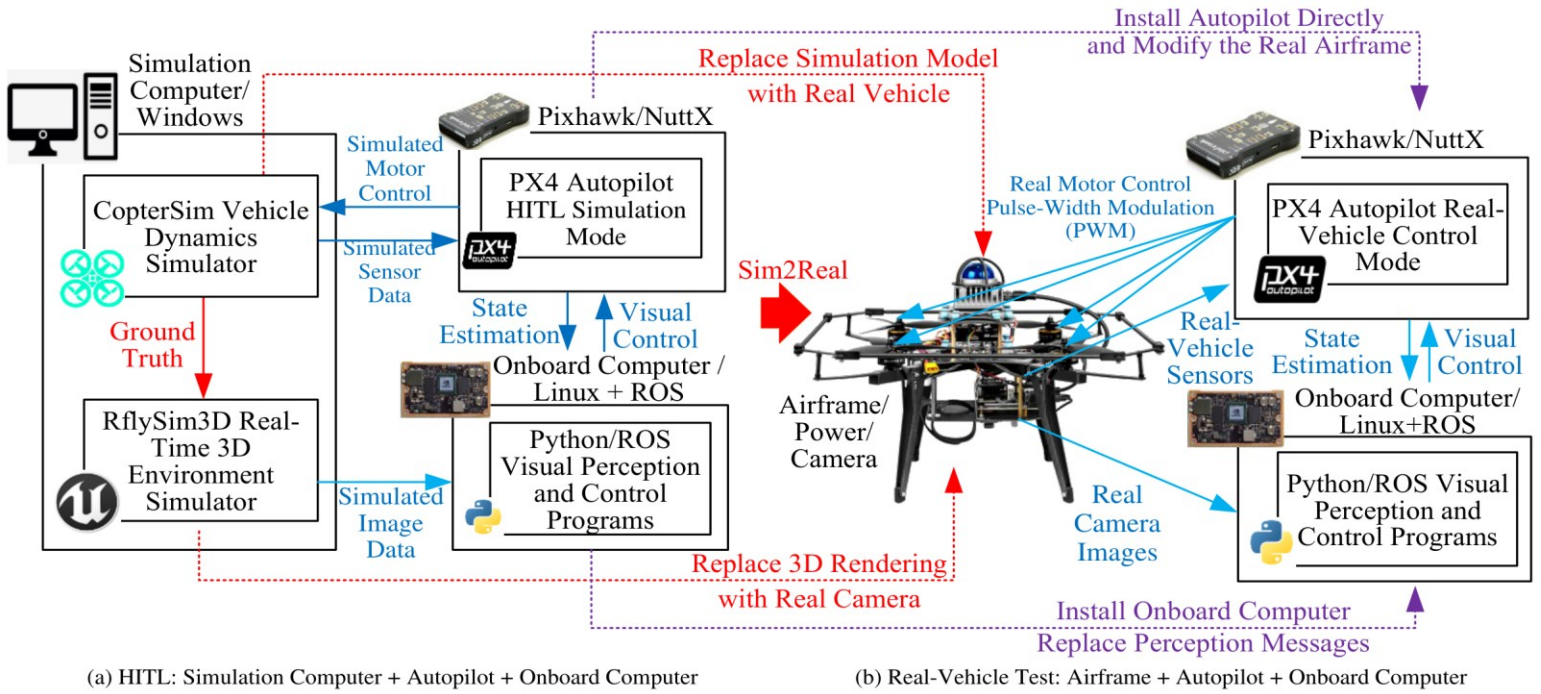
Simulation-to-Real Observation-Source Replacement and Latency Comparison (Schematic)

- **Observation source:** Replace the virtual camera with a real one; keep image inputs and motion-target outputs.
- **Algorithm structure:** Keep detection, planning and control interfaces, coordinates, units and message meanings.
- **Real differences:** Reset camera calibration and resolution for the hardware; align timestamps and measure end-to-end latency.

Retain algorithm logic and interfaces to reduce rewrites from simulation to real vehicles.

1.6.2 Design Philosophy and Features: Portability and Sim2Real

Sim2Real: New Data Sources, Same Algorithm Logic



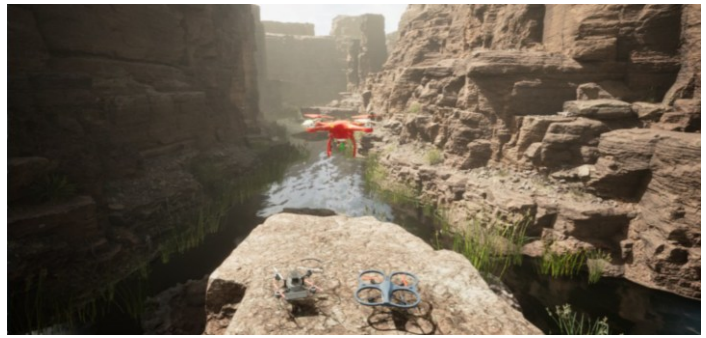
- **(a) HITL:** CopterSim computes motion; RflySim3D generates images.
- **Red dashed lines:** Replace the model with a real airframe and rendering with a real camera.
- **Purple dashed lines:** Install the autopilot and onboard computer directly on the vehicle.

Fig. 1.20 Rapid Sim2Real Transfer of Perception and Decision Algorithms

Use a real airframe and camera, switch autopilot mode, and retain perception and decision algorithms.

1.6.2 Design Philosophy and Features: High Fidelity in Scenes and Sensors

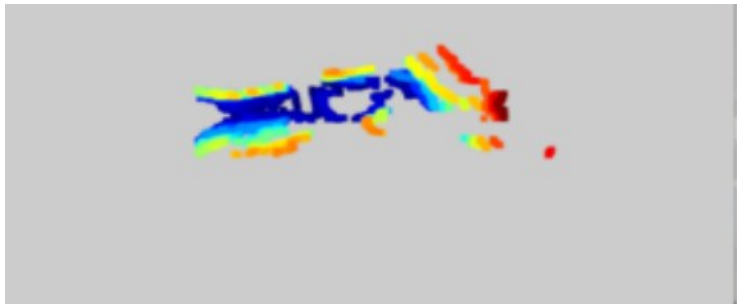
Realistic Textures and Multimodal Sensor Simulation



3D Scene with Realistic Textures



Color Image



Laser Point Cloud



Depth Image

- **Realistic textures:** Unreal Engine renders rock and water textures and lighting close to real scenes.
- **Multimodal data:** Color images, laser point clouds, and depth images are available to high-level algorithms.
- **Algorithm development:** These inputs support object recognition, localization and mapping, and path planning.

Complementary sensor views of the same scene enable perception-fusion tests.

1.6.2 Design Philosophy and Features: High Fidelity in Infrared Imaging

Infrared Pod and Imaging Simulation in a Battlefield Scene



Battlefield (Right); False-Color and Grayscale Infrared Views (Left)

- **Infrared imaging:** Thermal radiation makes larger temperature differences stand out.
- **False color and grayscale:** Two displays of the same infrared intensity.
- **Occlusion:** Pod rotation and vehicle motion change the visible area.

A shared 3D scene links motion and images for repeatable viewpoint and occlusion tests.

1.6.2 Design Philosophy and Features: Swarm Scale

Distributed Execution Supports Shared Multi-Vehicle Experiments

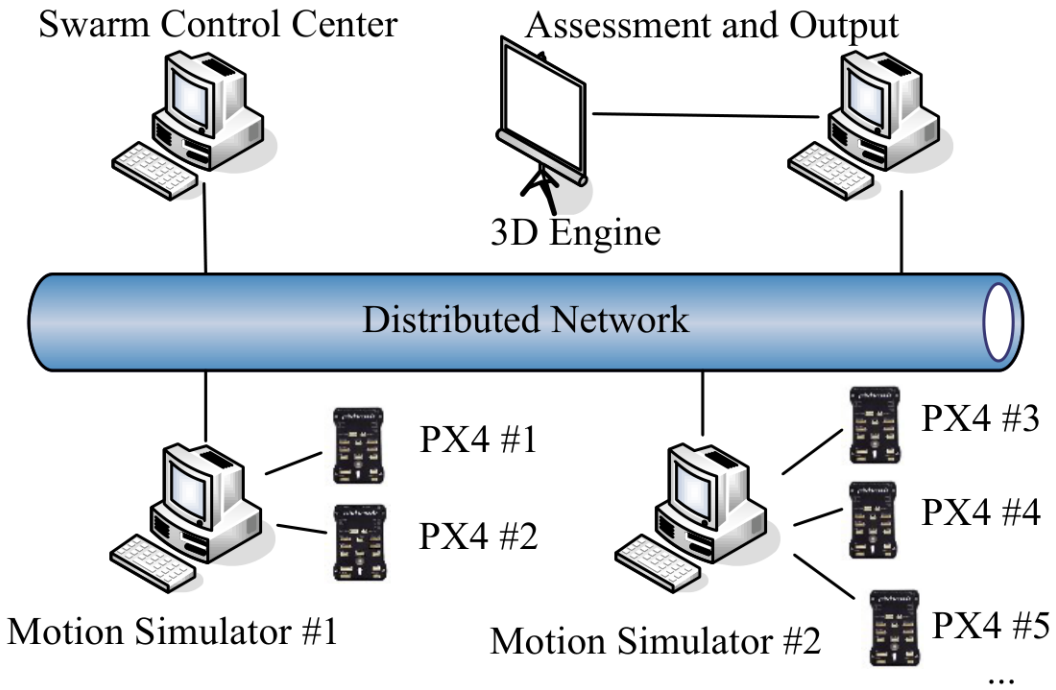


Fig. 1.21 Distributed Simulation on Multiple Computers

- **Motion simulators:** Each computer connects to multiple PX4 autopilots and computes their vehicles' motion.
- **3D engine:** Display all vehicles in a shared scene and generate visual data.
- **Command and evaluation:** A command center assigns common tasks; an evaluation computer records results.
- **Scale limits:** Image rates, model complexity, and network latency constrain real-time scale.

Distribute computation and exchange states to test multiple vehicles in one task.

1.6.2 Design Philosophy and Features: Ease of Use

An Integrated Environment and Examples Reduce Setup Work

- **Integrated environment:** One-click Windows deployment includes Ubuntu in Windows Subsystem for Linux (WSL) and the PX4 build environment.
- **Examples:** Basic and advanced examples already connect models, algorithms, interfaces, and scenes. Run one first as a baseline.
- **Controlled changes:** Keep baseline conditions. Change one module or parameter at a time,
  then compare paths, errors, and runtime.
- **Ongoing reuse:** Save configurations, code, and records. Reuse interfaces and test conditions for the next algorithm.

Reuse environments, interfaces, and examples to focus on algorithms and comparisons.

1.6.2 Design Philosophy and Features: Seven Features

Seven Features Support a Continuous Development Workflow

- **Unification:** A shared development and validation framework for different vehicle types.
- **Ease of use:** Integrated environments and examples reduce repeated setup.
- **High efficiency:** Generate code from models and validate through SITL/HITL stages to compare changes faster.
- **High fidelity:** Unreal Engine renders realistic scenes and generates color, depth, point-cloud, and infrared data.
- **Portability:** Continuous algorithm implementations and interfaces reduce deployment rewrites.
- **Swarm scale:** Distributed execution expands shared multi-vehicle experiments.
- **Functional completeness:** Modeling, perception, diagnosis, planning, control, networking, and swarm modules connect.

Connect reusable components to support continuous development and validation.

Chapter Outline

RflySim: From Algorithm Development to Real-Vehicle Validation

Chapter 1 Introduction

1.1 Concepts and Applications
1.2 History and Trends
1.3 Components and Architecture
1.4 Coordinate Systems and Control Principles
1.5 Assembly and Debugging
1.6 RflySim Toolchain
1.7 Learning Path and Experiments
1.8 Exercises and Discussion

4 Units in This Section

1.6.1 Background and Platform Role
Textbook: Sec. 1.4

1.6.2 Design Philosophy and Features
Textbook: Sec. 1.4.2

1.6.3 Representative Cases
Perception, collaboration, disturbance rejection, cross-vehicle tests

1.6.4 Teaching, Research, and Learning Resources
Textbooks, documentation, cases, and papers

1.6.3 Representative Cases: Task Groups and Shared Structure

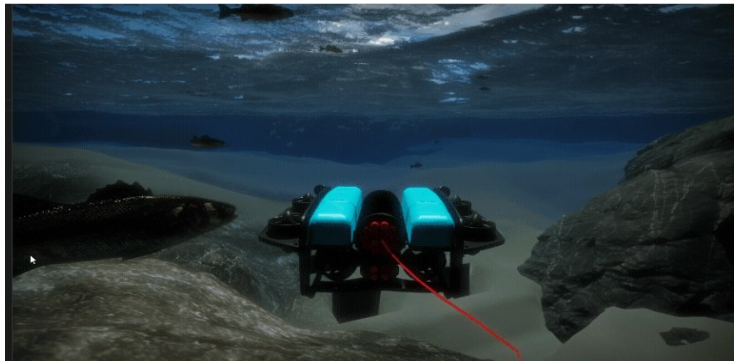
Tasks Set Validation Priorities; Vehicles Set Model Adaptations



Aircraft
Real Flight



Quadruped
Simulation



Underwater Vehicle Simulation

Perception → Decision-Making → Control → Environment Interaction

- **Perception/motion:** Compare ring-traversal and simultaneous localization and mapping (SLAM) observations and paths.
- **Collaborative tasks:** Landing, transport, and formation compare relative motion and collaboration.
- **Faults and disturbances:** Motor failure and gusts compare deviations and recovery.
- **Across vehicles:** Adapt force models, observations, and interfaces for tail-sitters, quadrupeds, and underwater vehicles.

Tasks and vehicles vary; observation, decision, action, and feedback stay connected.

1.6.3 Representative Cases: Perception and Motion Coupling

Agile Traversal Needs Perception, Planning, and Control



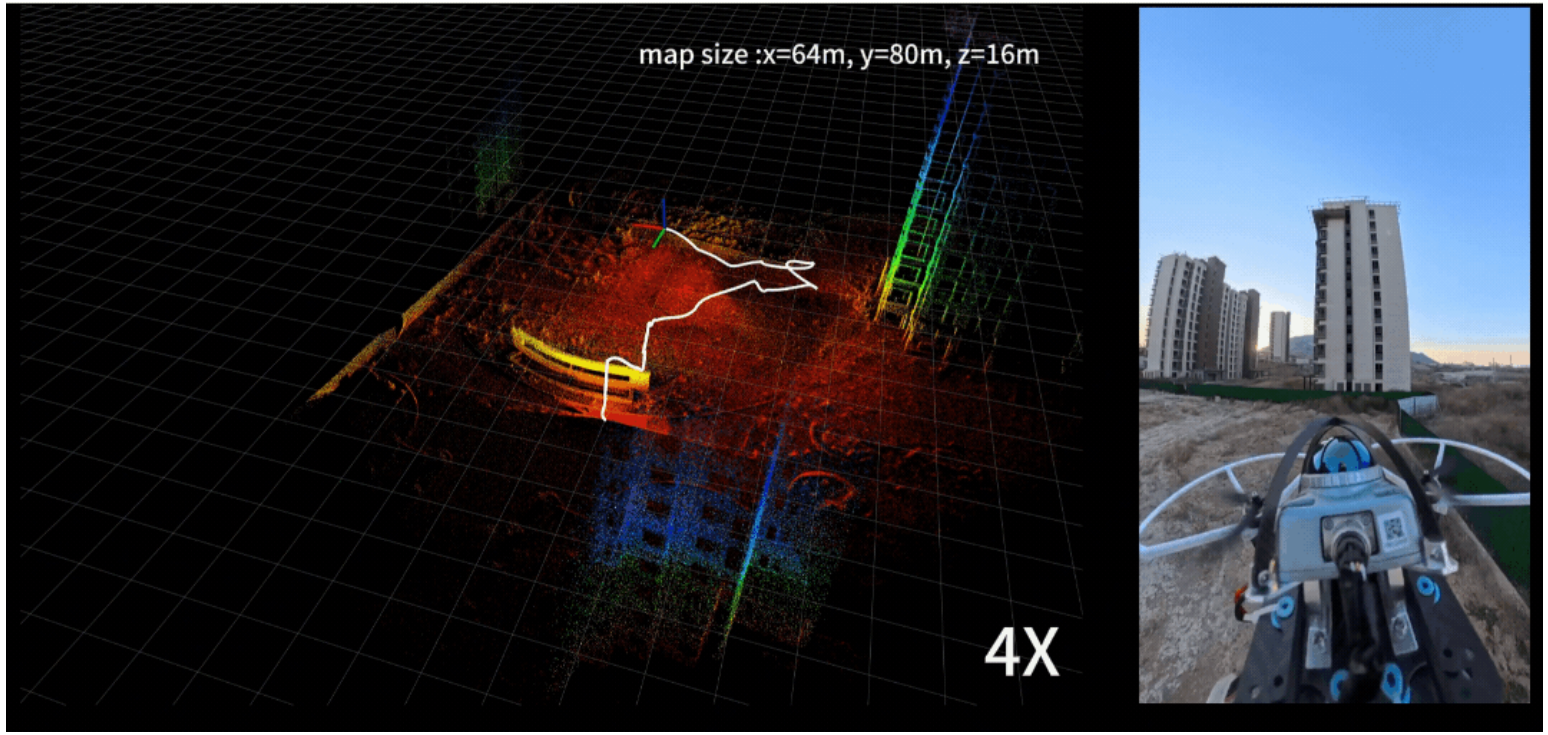
Continuous Multicopter Ring Traversal (Real Flight)

- **Behavior:** A multicopter traverses successive rings, changing motion direction and body attitude with the path.
- **Coordination:** Observations constrain the path; control tracks it and handles deviations.
- **Simulation:** Compare paths, speeds, and disturbances before real-flight validation.

Continuous traversal links sensing, trajectory planning, and motion control.

1.6.3 Representative Cases: Perception and Motion Coupling

Localization and Mapping Turn Sensor Observations into Maps



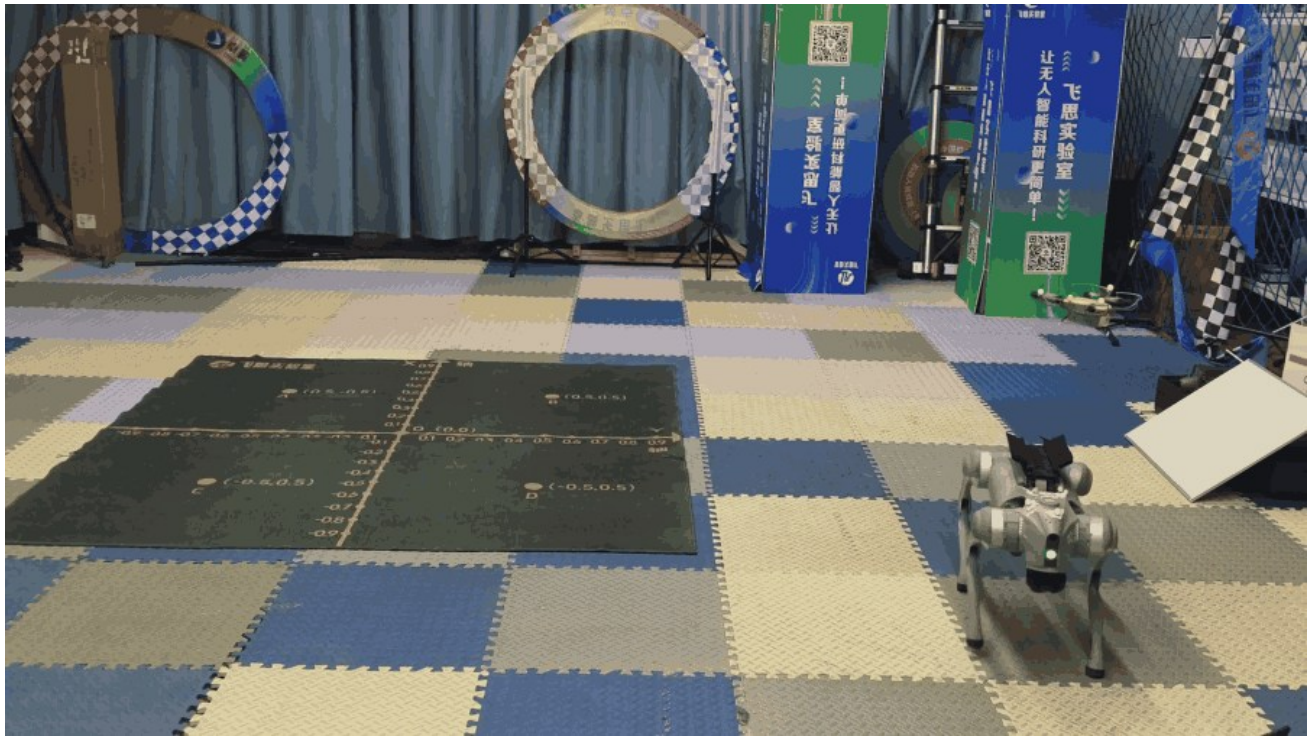
Real-Scene Observations and Localization and Mapping (4× Original Speed)

- **Accumulating observations:** Motion brings new views of spatial structure.
- **Localization and mapping:** Simultaneous localization and mapping (SLAM) jointly estimates vehicle position and the environment map.
- **Validation:** Set scenes and observations in simulation; compare estimates with reference states.

Position estimates and maps support each other and guide autonomous motion.

1.6.3 Representative Cases: Cross-Platform and Payload Coupling

Aerial–Ground Collaboration Requires Relative-Motion Control



Multicopter Landing on a Moving Quadruped (Real Vehicles)

- **Platform combination:** An aircraft and ground robot provide aerial and ground mobility.
- **Relative state:** Relative position, velocity, and heading affect approach and landing.
- **Joint validation:** Vary platform motion in a shared scene; compare tracking and collaborative-control responses.

Ground-platform motion makes the collaborative-control target change over time.

1.6.3 Representative Cases: Cross-Platform and Payload Coupling

Two Aircraft Must Coordinate a Shared Suspended Load



Two-Aircraft Suspended-Load Transport and Adjustment (Real Flight)

- **Shared load:** Cables connect two aircraft to one load, coupling their motion.
- **Collaborative objectives:** Keep both aircraft controlled while adjusting load position, attitude, and swing.
- **Modeling:** Multibody and constraint models describe cable, load, and aircraft interactions.

A shared load couples both aircraft; collaborative control also targets load attitude.

1.6.3 Representative Cases: Faults and New Configurations

Motor Failure and Gusts Produce Different Abnormal Responses



Single-Motor Failure and Gust Simulation

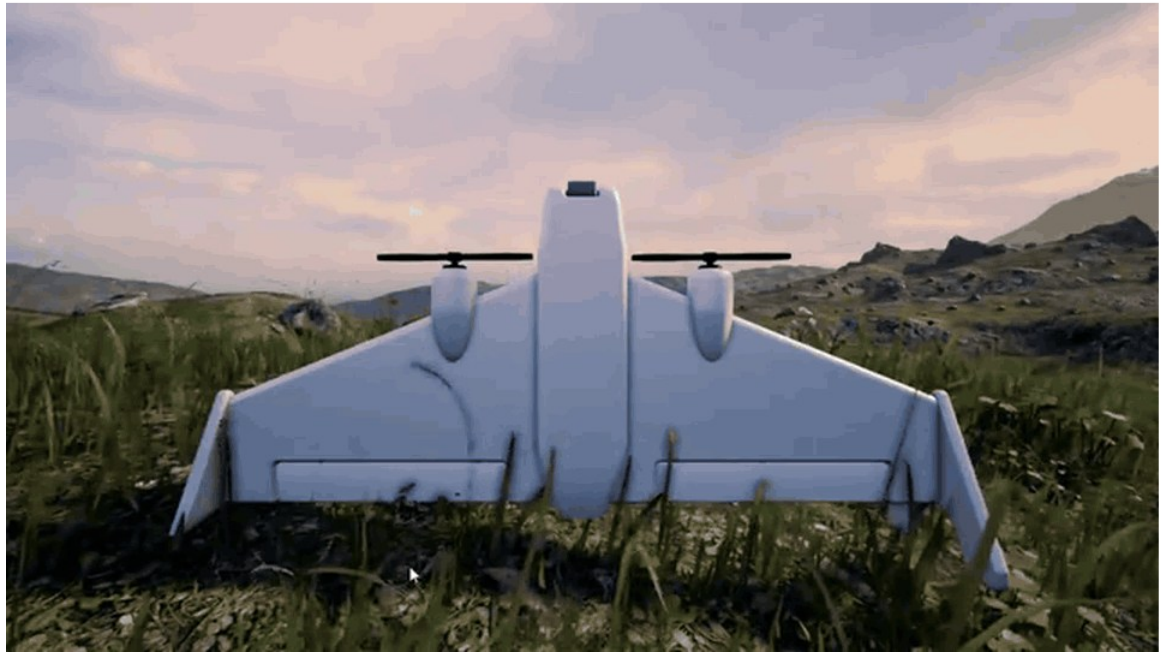
- **Two conditions:** Single-motor failure is shown first, then motion under gusts.
- **Different effects:** Motor failure changes actuator capability; gusts add external disturbances.
- **Repeatable tests:** Set faults or disturbances; record attitude, trajectory, and control responses.

Actuator failure changes propulsion; disturbances change forces. Test both repeatedly.

1.6.3 Representative Cases: Different Vehicle Types

Tail-Sitter Simulation and Tethered Tests

Simulation Vertical Takeoff



Real Vehicle
Indoor Tethered Test



Tail-Sitter Simulated Takeoff and Indoor Tethered Test

- **Configuration:** The nose points up for vertical takeoff and landing; body attitude changes for forward flight.
- **Simulated takeoff:** The virtual airframe lifts off and climbs to examine vertical-stage motion and control.
- **Tethered tests:** A real vehicle moves under cable constraints; check its response under those conditions.

Simulation computes flight response; tethered tests show constrained real-hardware motion.

1.6.3 Representative Cases: Vehicle Types and Multi-Vehicle Extension

A Quadruped Combines Locomotion with Arm Manipulation



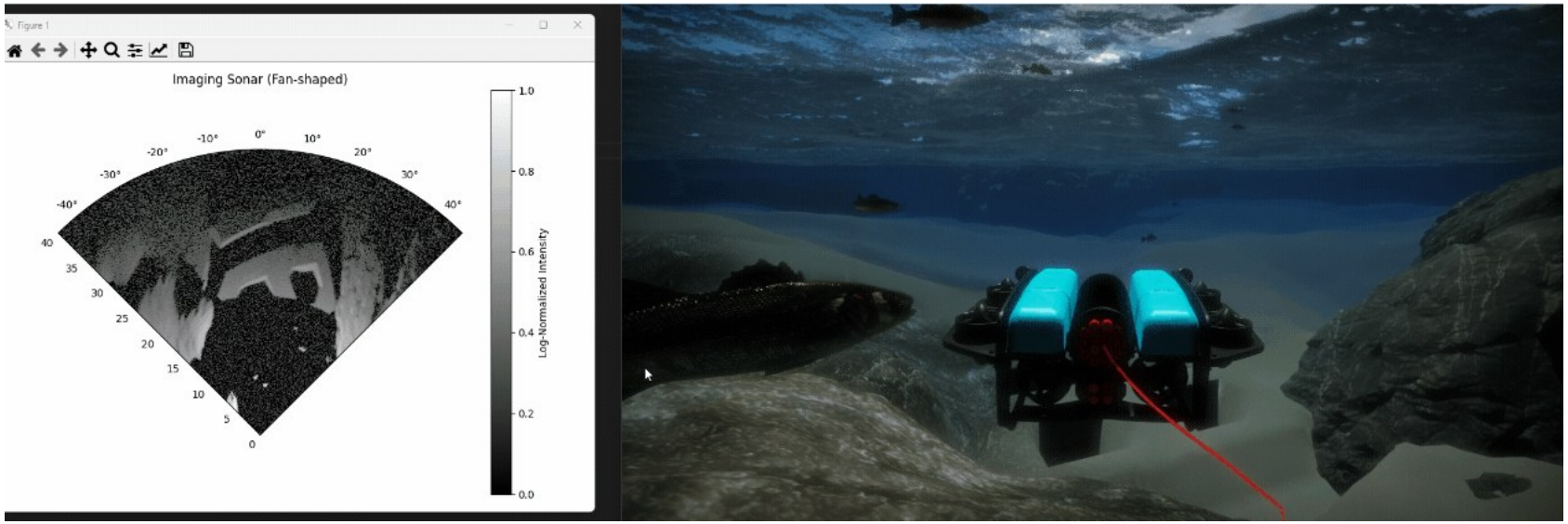
Quadruped Manipulation (Real Robot)

- **System:** The quadruped base provides mobility and support. The arm adjusts end-effector position and attitude relative to the environment.
- **Action sequence:** Approach the panel, stand and raise the arm, then retract it, linking locomotion and manipulation.
- **Motion relationships:** Body position and attitude change the arm's reachable region and contact position.
- **Shared method:** Models, observations, control, and environment interaction still form the development and validation framework.

Embodied systems combine sensing, locomotion, and manipulation in complete tasks.

1.6.3 Representative Cases: Vehicle Types and Multi-Vehicle Extension

Underwater Vehicles Also Need Perception and Motion Validation



Imaging-Sonar Fan View (Left) and Underwater 3D Scene (Right), Simulated

- **Reading sonar:** Azimuth is horizontal, about ±40°. Radius gives range; brightness gives echo strength.
- **Adaptation and reuse:** Remodel buoyancy, hydrodynamics, and sensing; reuse the framework, interfaces, and test workflow.

Reuse the workflow; adapt motion and sensor models to each vehicle's environment.

1.6.3 Representative Cases: Collaborative Tasks

Formation Flight Extends Single-Vehicle Control to Coordinated Tasks



Multi-Aircraft Formation in Real Flight

- **Individual control:** Each aircraft needs controlled motion before a common task.
- **Group relationships:** Formation, relative positions, and task allocation add coordination needs.
- **Extended validation:** Compare node and communication changes in distributed simulation before real formation-flight tests.

Multi-vehicle tasks require individual motion control and coordination among vehicles.

1.6.3 Representative Cases: Comparing Results

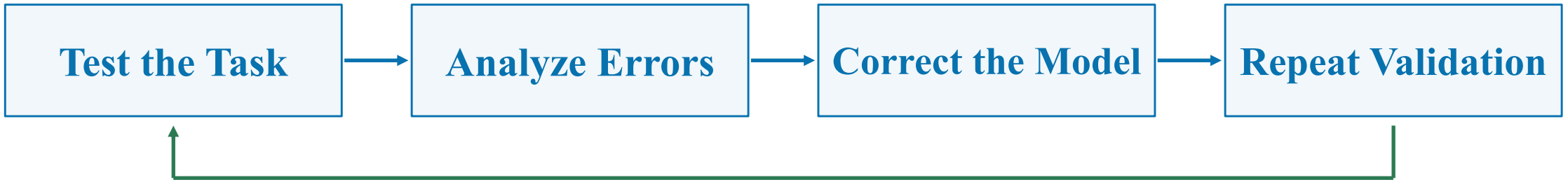
Explicit Metrics Make Task Results Comparable

- **Continuous traversal:** Trajectory error, minimum obstacle clearance, and task success rate measure path execution.
- **Aerial–ground collaboration:** Relative position and velocity errors measure tracking and approach quality.
- **Collaborative transport:** Load position error, attitude deviation, and swing amplitude measure shared-task performance.
- **Fault response:** Attitude deviation, recovery time, and trajectory changes measure motion after motor failure or gusts.

Use comparable task metrics to guide algorithm refinement.

1.6.3 Representative Cases: Credible Validation

Models and Measurements Together Support Validation Conclusions



Compare measured data and algorithm outputs together

- **Consistent implementation:** Continuous code and interfaces reduce implementation differences and support reuse of existing tests.
- **Model correction:** Adjust parameters and error descriptions using measured motion, sensor, and actuator results.
- **Condition coverage:** Test within defined speed, payload, and environmental ranges, including key abnormal conditions.

Correct models from measured differences; repeat tests to improve conclusion coverage.

Chapter Outline

RflySim: From Algorithm Development to Real-Vehicle Validation

Chapter 1 Introduction

1.1 Concepts and Applications
1.2 History and Trends
1.3 Components and Architecture
1.4 Coordinate Systems and Control Principles
1.5 Assembly and Debugging
1.6 RflySim Toolchain
1.7 Learning Path and Experiments
1.8 Exercises and Discussion

4 Units in This Section

1.6.1 Background and Platform Role
Textbook: Sec. 1.4

1.6.2 Design Philosophy and Features
Textbook: Sec. 1.4.2

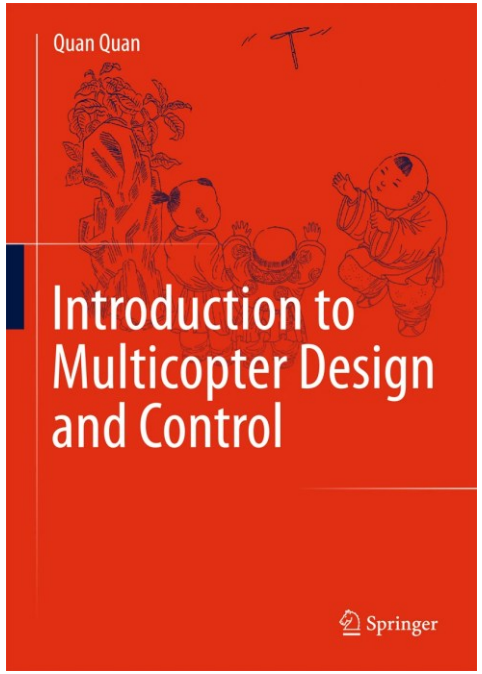
1.6.3 Representative Cases
Perception, collaboration, disturbance rejection, cross-vehicle tests

1.6.4 Teaching, Research, and Learning Resources
Textbooks, documentation, cases, and papers

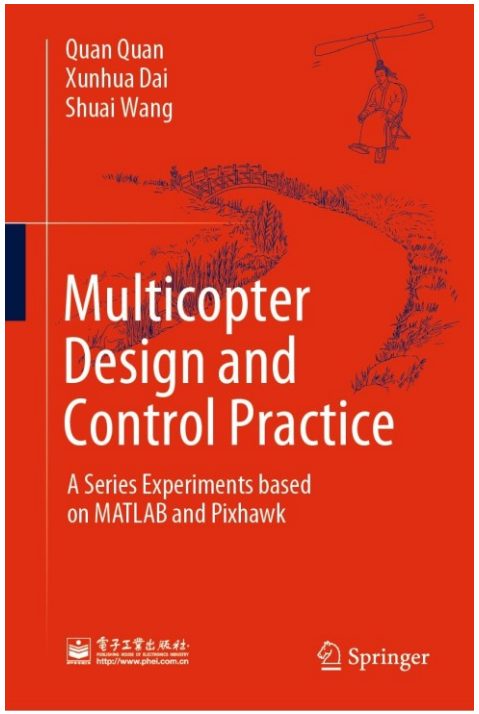
1.6 RflySim Toolchain

1.6.4 Teaching, Research, and Learning Resources: Textbook Series

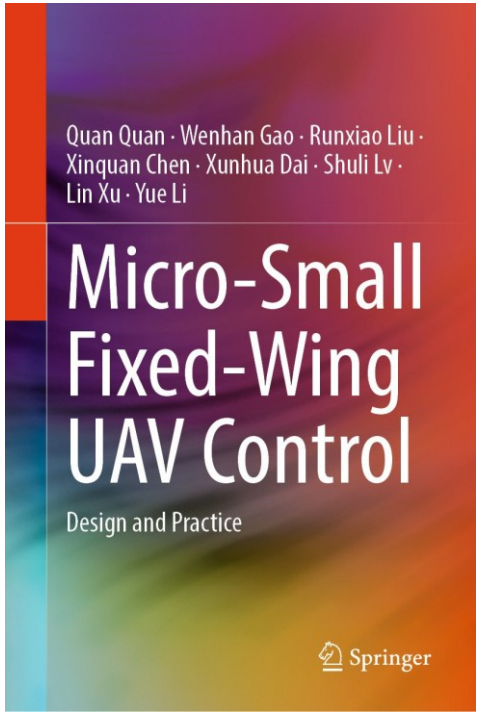
The Textbook Series Connects Aircraft Principles to Unmanned-System Practice



Introduction to Multicopter Design and Control



Multicopter Design and Control Practice



Micro-Small Fixed-Wing UAV Control Design and Practice



Intelligent Unmanned System Control Practice (Coming soon.)

Cross-check vehicle principles, control methods, and simulation practice from a task.

1.6.4 Teaching, Research, and Learning Resources: Official Resources

Official Documentation, Case Videos, and Research Papers

Toolchain Documentation

Find the framework, quick start, manuals, and development interfaces.

https://rflysim.com/doc/zh/index.html

Case Videos

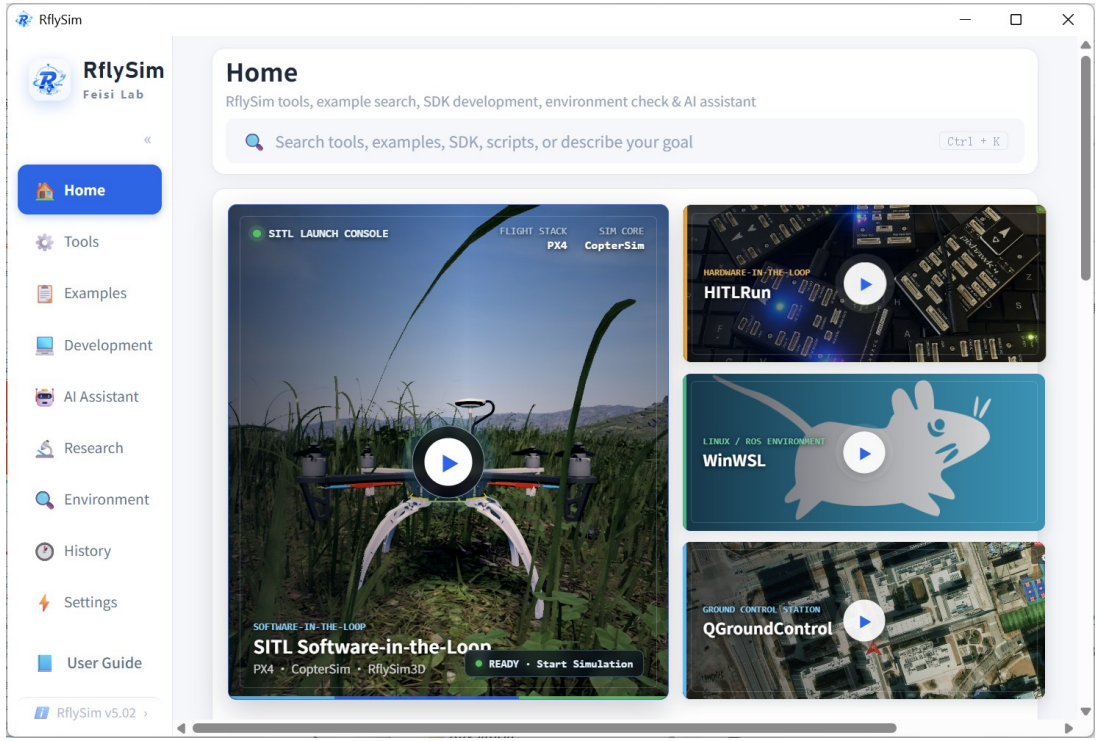
See task performance in simulation, HITL, or real-vehicle tests.

https://rflysim.com/doc/zh/0.Show/videos.html

Research Papers

Read about problems, methods, experimental conditions, and results.

https://rflysim.com/doc/zh/0.Show/papers.html



RflySim Unified Launcher Home Page

Section Summary

Repeatable Validation Runs Through Unmanned-System Development

- **Motivation:** Reduce platform setup and trial-and-error effort, leaving more time for algorithm research and system understanding.
- **Core mechanisms:** Connect models, observations, algorithms, interfaces, and hardware for staged validation and feedback.
- **Tasks:** From traversal and mapping to collaborative work and embodied systems, the systems studied vary while validation methods remain reusable.
- **Next steps:** Sec. 1.7 links background knowledge and research goals to learning paths and accompanying experiments.

Start repeatable experiments early, then validate on real systems.