Chapter Outline

Chapter 2 Exercises and Discussion

Chapter 2 Experimental Platform Configuration

2.1 Key Tools

2.2 Installation and First Simulation

2.3 Architecture and Workflows

2.4 API and SDK Interfaces

2.5 Examples and Experimental Routes

2.6 Exercises and Discussion

2 Units in This Section

2.6.1 Concepts and Methods

Q1–6: simulation modes, data flow, and environments

2.6.2 Practice and Transfer

Questions 7–10: first run, logs, and closed-loop experiments

Then: discussion → practice → chapter summary

Textbook: Sec. 2.5 Exercises

2.6 Exercises and Discussion

2.6.1 Concepts and Methods: Textbook Question 2

Q2: Trace a Control Cycle Through the Architecture

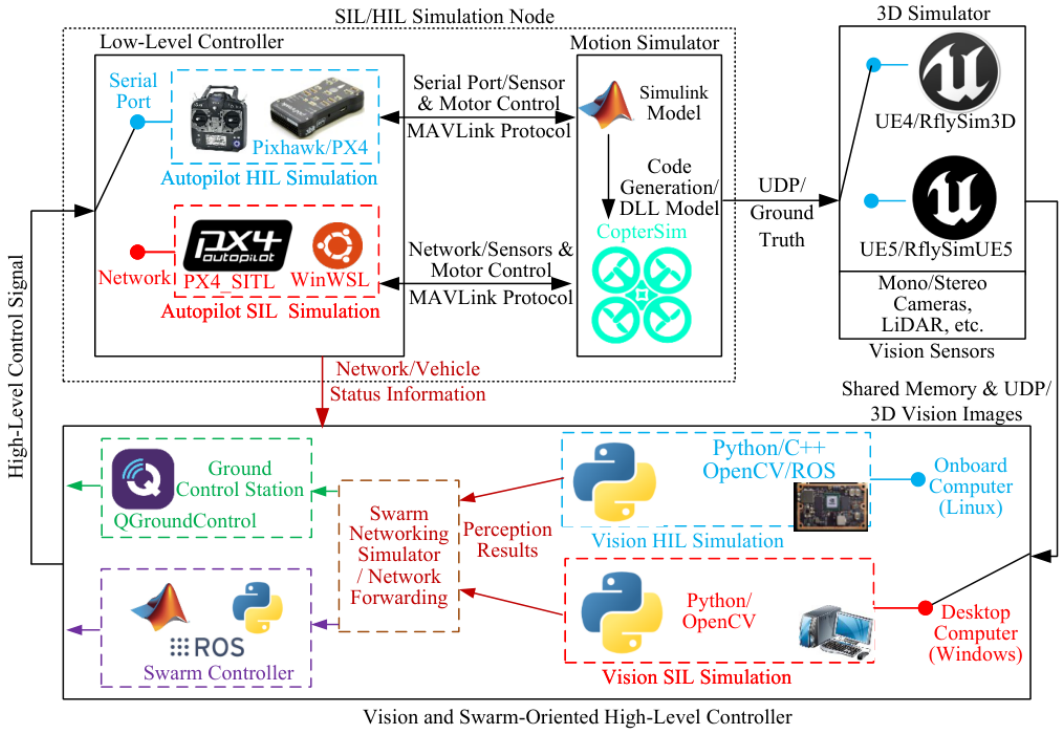


Fig. 2.9 Overall RflySim Toolchain Framework

- **Task:** Assume the default single-PC software-in-the-loop setup (SITL; SIL in the textbook). Explain the four tools in Fig. 2.9.
- **Control and motion:** WinWSL hosts PX4; CopterSim calculates vehicle motion.
- **Scene and monitoring:** RflySim3D generates the scene and observations; QGroundControl (QGC) configures and monitors the vehicle.
- **Trace each interface:** Identify who sends and receives simulated measurements, actuator outputs, pose, and images.

2.6.1 Concepts and Methods: Question 2 Review

Different Data Follow Different Links

Data link	Sender → receiver	Data
Low-level loop	PX4 → CopterSim CopterSim → PX4	Actuator outputs Simulated sensor measurements
Visual feedback	CopterSim → RflySim3D RflySim3D → Python	Pose / simulation ground truth Images / observations
High-level links	QGC／Python ↔ CopterSim ↔ PX4	Configuration, motion setpoints, and state

Interfaces: MAVLink for the low-level loop; UDP for pose; shared memory or UDP for images.

Actuator feedback closes the low-level loop; pose and images connect visual feedback.

2.6.1 Concepts and Methods: Textbook Question 1

Q1: Which Test Needs a Real Control Board?

Same quadrotor model: Team A modifies position control; Team B checks timing on a Pixhawk board. Which mode should each use first?

- **Team A:** Use SITL. PX4 runs on the PC and connects to the model over the network; repeat identical inputs to compare algorithms.
- **Team B:** Use HITL. PX4 runs on Pixhawk and connects to the model over USB serial; inspect execution timing on the board.
- **Both simulate:** The model generates vehicle motion and sensor feedback. Real motor loads, supply fluctuations, and vibration require physical tests.
- **Test stages:** Iterate in SITL, integrate on target hardware in HITL, then test the complete physical system.

Hardware-in-the-loop (HITL; textbook: HIL): real controller, simulated vehicle.

2.6.1 Concepts and Methods: Textbook Question 3

Q3: Code and Firmware Build Environment for Pixhawk 6C

Target: px4_fmu-v6c_default (Pixhawk 6C). Uploading needs a 6C board and USB data cable.

- **Match the environment:** Use PX4Switch to select source and a build target that support 6C. With compatible source already selected, PX4CMD('px4_fmu-v6c_default') changes only the target.
- **Native firmware:** WinWSL builds PX4 source for this target, producing a .px4 firmware file for 6C.
- **Model code:** SimulinkPSP blocks read sensors and publish control through uORB; code generation creates the C/C++ module px4_simulink_app.
- **Build and upload:** Integrate px4_simulink_app into 6C firmware. Connect USB, upload with PX4Upload, then check the version and heartbeat.

Model code computes control; 6C firmware integrates board drivers and startup code.

2.6.1 Concepts and Methods: Textbook Question 4

Q4: WSL1, WSL2, and the Default Setup

Feature	WSL1	WSL2
Implementation	System-call translation; less memory	Full Linux kernel in a virtual machine
File access	Fast access to Windows drives	Fast within Linux; slower across systems
Networking	Shared Windows networking	Mirrored network: Win11 22H2+
Containers and GPU	No Docker or GPU acceleration	Docker and GPU support; more memory
Default choice	RflySim default: builds and SITL	Advanced needs: Docker and GPU

- **Switch and check:** Enter 1 or 2 in RflyTools/WslSwitch2; WSL2 in uname -a identifies WSL2.

Use the validated setup first; choose WSL2 for a Linux kernel or container backend.

2.6.1 Concepts and Methods: Textbook Question 5

Q5: How Does MAVROS Bridge the Two Sides?



- **Protocol and bridge:** MAVLink carries autopilot messages. Robot Operating System (ROS) 1/2 organize nodes through topics and services; ROS 2 uses Data Distribution Service (DDS). MAVROS translates in both directions.
- **State upstream (green):** mode in /mavros/state reports the actual mode; armed reports arming status.
- **Setpoints downstream (blue):** Publish /mavros/setpoint_position/local. /mavros/set_mode only requests a change; read back mode to confirm it.

2.6.1 Concepts and Methods: Textbook Question 6

Q6: Why Docker, and How Is an ARM64 Image Built?

- **Why Docker:** Package the algorithm with its libraries and versions to reduce setup and dependency conflicts. Containers share the host kernel and are lighter than virtual machines.
- **Build across architectures:** Select ARM64 base images and dependencies in the Dockerfile. On x86, use QEMU emulation, an ARM build node, or cross-compilation.

```
docker buildx build --platform linux/arm64 \
 -t course-demo:arm64 --load .
```

- **Deployment:** --load imports into the local image store only. Transfer the image to ARM64 via a registry or file. GPU / peripherals need drivers and device mapping. Check outputs, latency, and logs.

Multi-platform builds: https://docs.docker.com/build/building/multi-platform/

An image packages user-space dependencies; ARM64 programs still depend on the host kernel and drivers.

Chapter Outline

Chapter 2 Exercises and Discussion

Chapter 2 Experimental Platform Configuration

2.1 Key Tools

2.2 Installation and First Simulation

2.3 Architecture and Workflows

2.4 API and SDK Interfaces

2.5 Examples and Experimental Routes

2.6 Exercises and Discussion

2 Units in This Section

2.6.1 Concepts and Methods

Q1–6: simulation modes, data flow, and environments

2.6.2 Practice and Transfer

Questions 7–10: first run, logs, and closed-loop experiments

Then: discussion → practice → chapter summary

Textbook: Sec. 2.5 Exercises

2.6.2 Practice and Transfer: Textbook Question 7

Q7: What Must Be Checked After Installation?

Given: The installer exits normally and TestWSL succeeds, but there are no initialization or takeoff/landing records. Is basic verification complete?

- **Answer:** No. TestWSL checks WSL availability; initialization and takeoff/landing records show that the loop works.
- **Before installation:** Windows 10 1903+ or Windows 11, an English-only path, and at least 100 GB free on the installation drive. Run EnableWSL.bat, then restart.
- **One-click installation:** Use defaults for the first installation; record the path, build target, firmware version, and compiler.
- **Complete first run:** Enter 1 in SITLRun; wait for EKF 3DFixed in the 3D window. In QGC, confirm Hold, select Takeoff, complete takeoff and landing, disarm, and exit normally.

Environment checks prepare the runtime; flight records demonstrate the control loop.

2.6.2 Practice and Transfer: Textbook Question 8

Q8: Four Flight Stages and Flight Logs

Take off	Hover	Fly forward	Land
Climb to 3 m	Hold for 10 s	Move 5 m Keep heading	Touch down and disarm

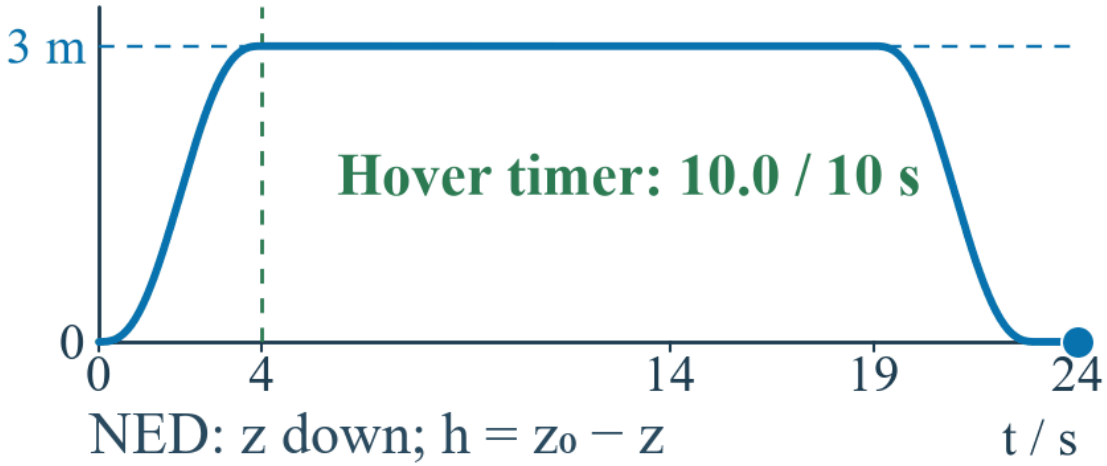
- Conditions: Flat scene; the north–east–down (NED) origin stays at the start. Count the 10 s hover only after reaching 3 m.
- Data: Use e13_Log-Get to record this run's CopterSim ground-truth CSV. For PX4 internal states, obtain the same flight's ULog; align the two in time.
- Criteria: z is positive downward; height $h = z_0 - z$ (z_0 is the initial value). Measure 5 m forward along the initial heading; keep heading, touch down, and disarm.

Task stages, motion response, and logs from the same run should agree in time.

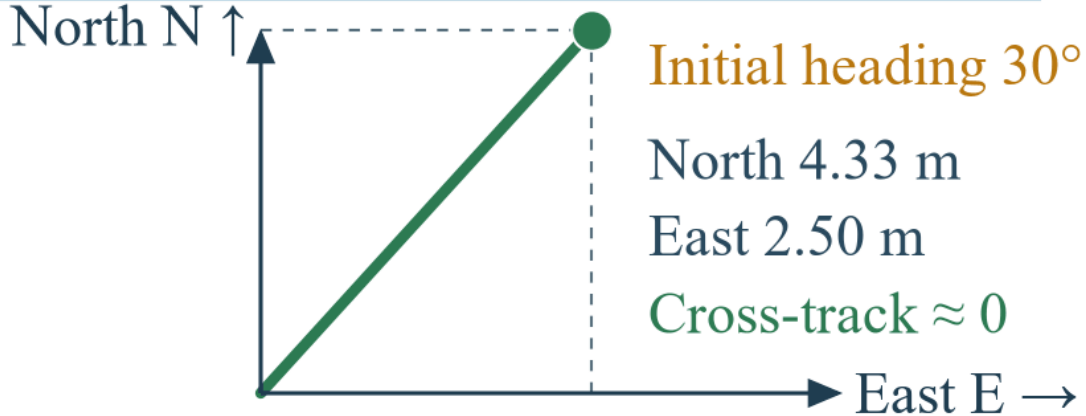
2.6.2 Practice and Transfer: Question 8 Review

Reading the Four Stages on a Timeline

Height h(t): timer starts at 3 m



Move 5 m along initial heading



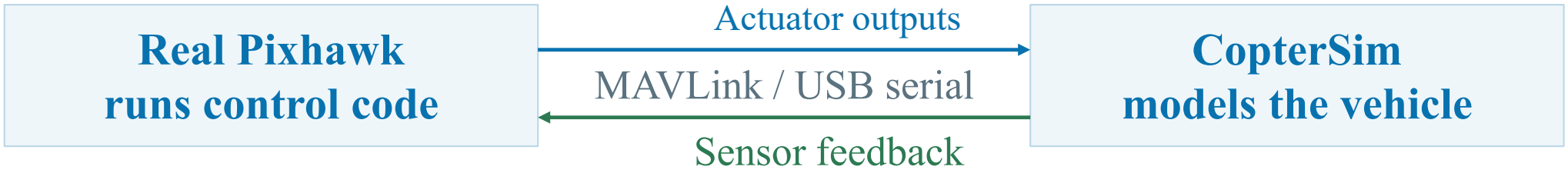
Takeoff	Hover 10 s	Move 5 m	Land / disarm

Four-stage timeline (schematic curves, not simulation data)

Count hovering from arrival at 3 m; measure 5 m along the initial heading. This is a schematic example.

2.6.2 Practice and Transfer: Textbook Question 9

Q9: Transfer the Same Task to HITL



- **Connect hardware:** Link Pixhawk to the simulation PC with a USB data cable. Remove propellers and disconnect motors and other actuators.
- **Firmware and mode:** Upload compatible firmware in QGC and select HITL mode. Run HITLRun, enter the serial port number, and wait for initialization.
- **Verify the flight:** Take off, hover, and land in QGC; save records. After touchdown and disarming, exit and restore changed settings.

HITL adds real autopilot computation and communication; the flying vehicle remains virtual.

2.6.2 Practice and Transfer: Textbook Question 10

Q10: Close the Loop with Vision

Generate images
Camera configured by VisCreate

➡

Process images
Python calculates target error

➡

Control motion
Offboard sends velocity setpoints

After vehicle motion, the 3D scene generates the next image.

- **Start:** Initialize SITL; test image acquisition and Offboard control separately, then take off.
- **Feedback:** Horizontal error sets yaw rate; vertical error sets climb/descent speed. Move forward while target area remains below the threshold.
- **Finish and faults:** On completion or timeout, land; stop communication after touchdown and disarming. A fresh frame without a target permits a timed search. A stale frame means image loss: first send a hold setpoint to replace the old command.

Image error changes motion setpoints; motion changes the next observation, closing visual feedback.

2.6.2 Practice and Transfer: Question 10 Review

Does the Controller Actually Use the Image?

```
ex = p_i[0] - width / 2;  yawrate = K_yawrate * ex
```

Given: 640-pixel forward image, x right; K_yawrate=0.00375; forward–right–down (FRD) body frame.

Observation	Horizontal error	Calculated yaw rate
A: p_i[0]=400 (80 pixels right)	ex = +80 pixels	+0.30 rad/s; turn right
B: p_i[0]=240 (80 pixels left)	ex = −80 pixels	−0.30 rad/s; turn left

- **Compare:** Align target position, final setpoint, and vehicle state in time.
- **Feedback:** Observations change commands; motion changes observations. Finish by landing.

Match observations, transmitted setpoints, and motion in time to explain visual feedback.

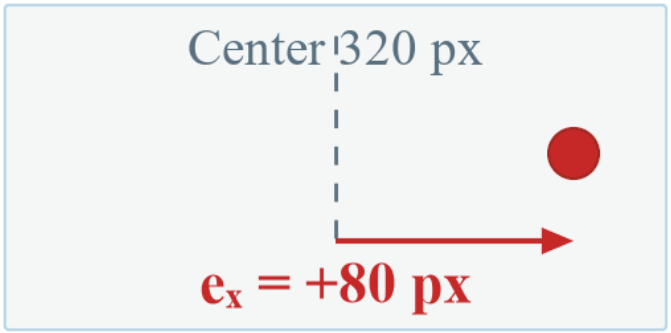
2.6.2 Practice and Transfer: Question 10 Review

How Does Yaw Change the Next Image?

A: target right

Observe | Error | Command | Turn | New image

Camera image



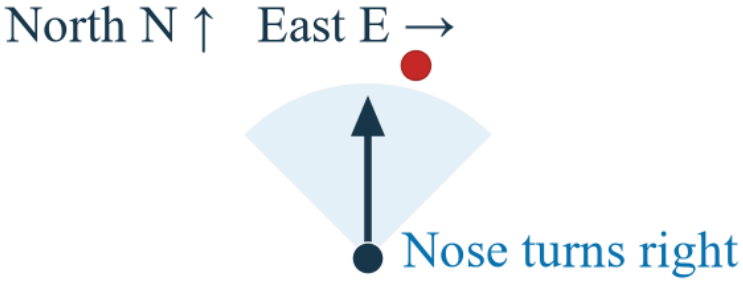
Yaw command

$$yawrate = K \times e_x$$

$$0.00375 \times (+80)$$

+0.30 rad/s

Top: +yaw clockwise



Error magnitude decreases



Proportional yaw control moves the target toward the image center (geometric illustration)

One accurate observation is insufficient; repeated observation and correction keep the target in view.

Classroom Discussion

One Link Works: Which Link Should Be Checked Next?

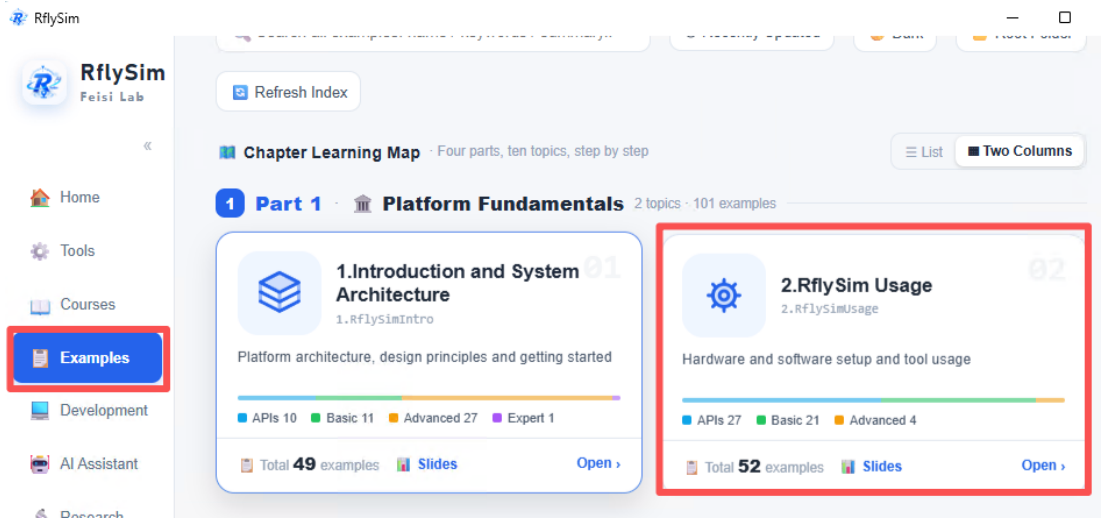
Observed symptom	What it establishes	Check first
3D scene displays **No valid vehicle state**	The scene is rendering	State reception, selected vehicle, and autopilot initialization
Telemetry works **Setpoint not accepted**	The state link carries data	Mode, setpoint format, frame, and continuous sending
Images update **Control ignores observations**	The camera outputs images	Detection, error, and final transmitted setpoint

Choose a case. State the first check, your reason, and the additional records needed.

Independent Practice

Practice Options and Submission

- **Everyone:** Answer the concepts and plans for all ten questions.
- **Core practice:** Questions 7 and 8; attempt the others when prerequisites are available.
- **Submit:** Task, versions, configuration, and changes; screenshots and raw logs; criteria, results, and fault handling.



Chapter 2 example entry point

Entry: Example Gallery → Chapter 2 → experiment instructions

Independent Practice

Examples for Questions 3–6

Q.	Example or tool (actual name)	Purpose
Q3	1.BasicExps\e18_PX4VersionSwitch、e9_Build-Firmware 1.BasicExps\e7_Code-Generation、e10_Firmware-Upload	Select target; compile Generate code; upload
Q4	Desktop RflyTools\WslSwitch2 1.RflySimIntro\2.AdvExps\e13.WinWSL2-GPU	Switch WSL version WSL2 and GPU environment
Q5	0.ApiExps\e10_Mavros\3.Mavros2Offboard	ROS 2 Offboard control
Q6	0.ApiExps\e11_docker	Build a Docker image

Root: [installation]\RflySimAPIs\; 1.BasicExps and 0.ApiExps below are in 2.RflySimUsage\.

Q3 upload: 6C; Q4 mirrored networking: Win11 22H2+; Q5: ROS 2; Q6: WSL2.

Independent Practice

Examples for Questions 7–10

Q.	Example or tool (actual name)	Purpose
Q7 Q9	Desktop RflyTools\SITLRun Desktop RflyTools\HITLRun	First SITL run HITL simulation
Q8	1.BasicExps\e13_Log-Get 1.BasicExps\e4_Log-Reads-Python38Env	Record ground-truth CSV Read ULog in Python
Q10	1.BasicExps\e19.VisCreateUsage 1.RflySimIntro\1.BasicExps\e7_RunPythonProject	Configure visual sensors Track a red ball with vision

Paths are under [installation]\RflySimAPIs\. Paths starting with 1.BasicExps omit 2.RflySimUsage\.

Q9 requires a Pixhawk board; the other three tasks run in single-PC SITL.

Chapter Summary

Chapter 2: Platform, Interfaces, and Verification

- **Tool roles:** Environments host programs; the motion model, autopilot, and 3D scene calculate vehicle state, control, and observations, respectively.
- **Configuration:** Match the simulation mode, tools, and target hardware to the task so that models, code, and interfaces agree.
- **Verification:** Hold comparison conditions fixed; inspect inputs, setpoints, and responses, and use raw data to explain the effect of a change.
- **Common pitfalls:** TestWSL checks only WSL; verify the first run with flight records. Read back mode to confirm a mode change. --load imports only to the local image store.
- **Next:** Chapter 3 covers 3D environment modeling and sensor generation, connecting scenes and observations to the loop.

Configure the platform for the task; closed data loops and repeatable results support further development.