Chapter Outline

Find and Call the Right Interface for a Task

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

4 Units in This Section

2.4.1 Tasks and Interface Selection
Textbook: Sec. 2.2.2.4–2.2.2.6, 2.2.3

2.4.2 Control Interfaces and Calling Conventions
Textbook: Sec. 2.2.3.2

2.4.3 Configuration Outputs and Vision Calls
Textbook: Sec. 2.2.2.7–2.2.2.8, 2.2.3.3

2.4.4 Modeling and Extension Interfaces
Textbook: Sec. 2.2.3.1–2.2.3.2

2.4.1 Tasks and Interface Selection: Concepts

Which Interfaces Does a Task Need?

- **API and SDK:** An application programming interface (API) defines how to make calls; a software development kit (SDK) supplies implementations, documentation, and tools.
- **Task roles:** Use control interfaces to read states and send targets, vision interfaces to acquire images, and 3D interfaces to change maps or viewpoints.
- **Configurations and scripts:** Configurations store objects and parameters; scripts start components and organize calls. Load an existing configuration without reopening the configuration tool.
- **General method:** Identity, units, coordinates, and timing apply across vehicle types; this lesson uses calls for a PX4 UAV simulation.

Interfaces define interactions, configurations describe objects, and programs organize tasks.

2.4.1 Tasks and Interface Selection: Module Index

Find Six SDK Module Groups by Task

Folder	Task and Representative File	Input or Output
comm	Network communication: NetSimAPIV4	Nodes, links, or network parameters
ctrl	Control and state: PX4MavCtrlV4	Motion targets, mode commands, state feedback
phm	Health assessment: AutoMavCtrl	Test actions, faults, and assessment procedures
swarm	Multi-vehicle interaction: distSimCtrlAPI	Multi-vehicle states and coordinated control
ue	3D control: UE4CtrlAPI	Map, object, and viewpoint commands
vision	Sensor reception: VisionCaptureApi	Sensor configurations, images, point clouds

Table 2.3 Python Modules: Six Folders and Representative Tasks

2.4.1 Tasks and Interface Selection: Local Access

Three Ways to Open the Same Local SDK Documentation

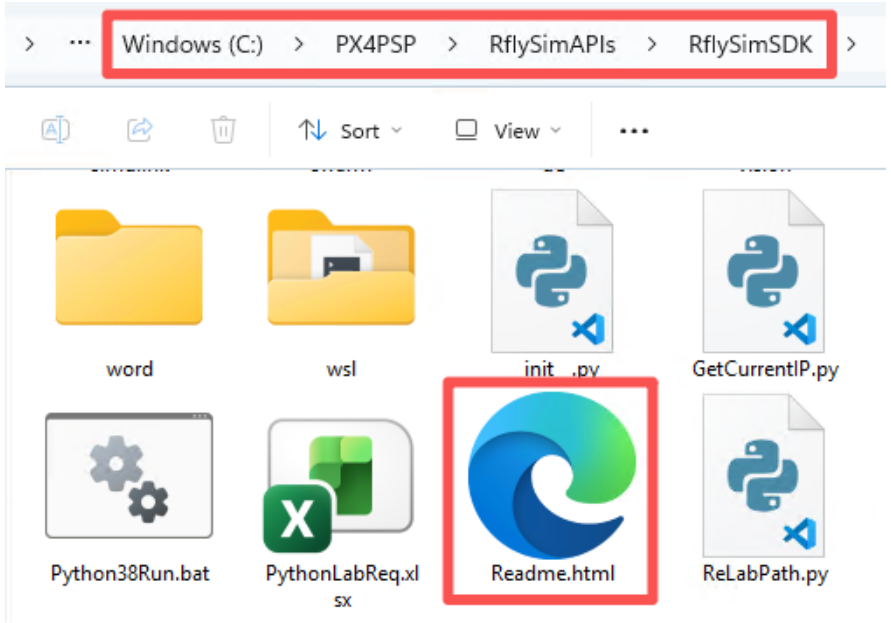


Fig. 2.14(a) Installation Folder

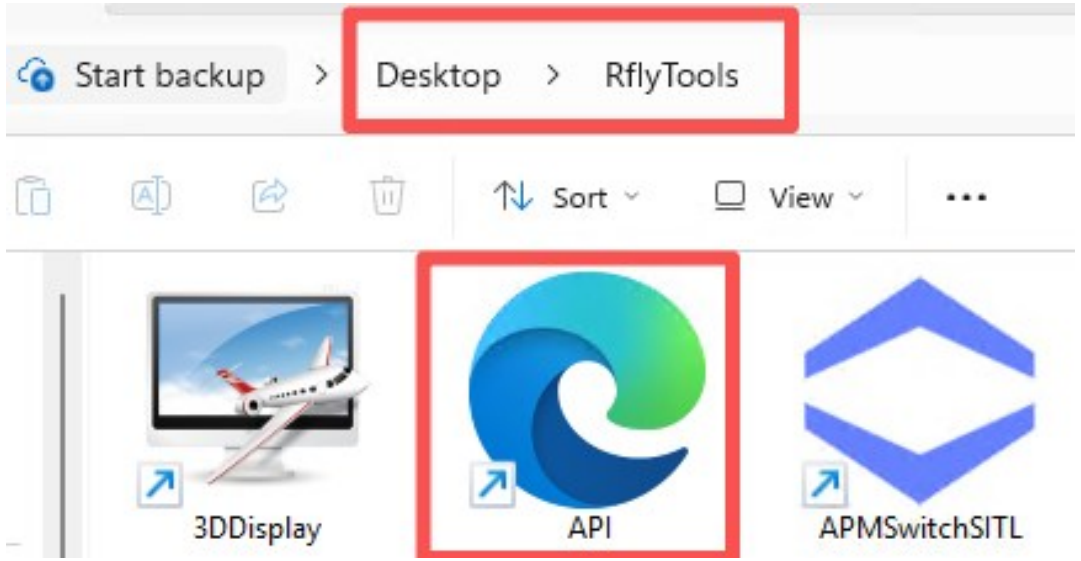


Fig. 2.14(b) Desktop Shortcut

RflySimSDK contains module folders such as comm and ctrl; Readme.html opens the API docs.

The desktop RflyTools\API shortcut and the launcher's Secondary Development entry open the same documentation, matching the local SDK version.

2.4 API and SDK Interfaces

2.4.1 Tasks and Interface Selection: Reading the Documentation

Find Functions and Examples in the Documentation

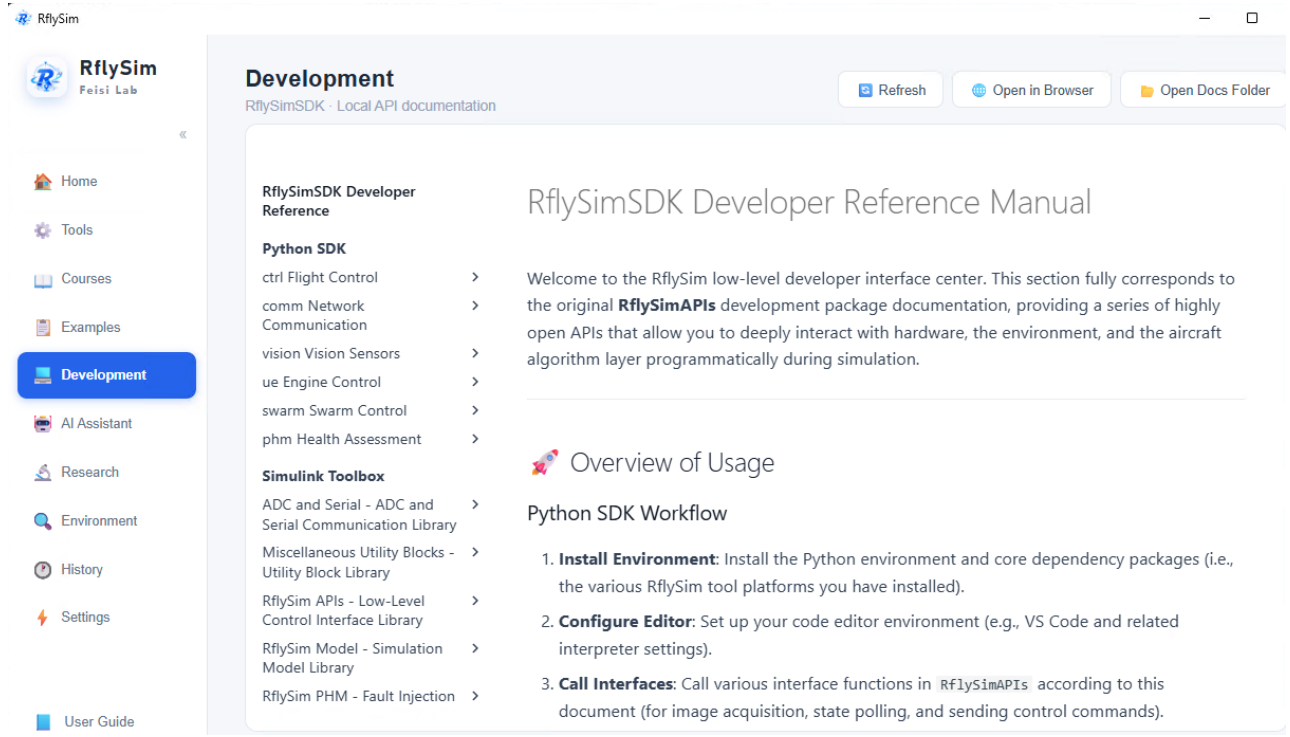


Figure: SDK Developer Reference in the Launcher

- **Locate:** ctrl → PX4MavCtrlV4; vision → VisionCaptureApi.
- **Read the signature:** Check argument order, frames, and units; success usually means only that a request was sent.
- **Read the context:** Initialize communication, keep updating targets,
check feedback to assess results, then shut down.

Parameters define data meaning; source threads, state changes, and packet sending determine call effects.

2.4.1 Tasks and Interface Selection: Execution Environment

Where Does the Calling Program Run?

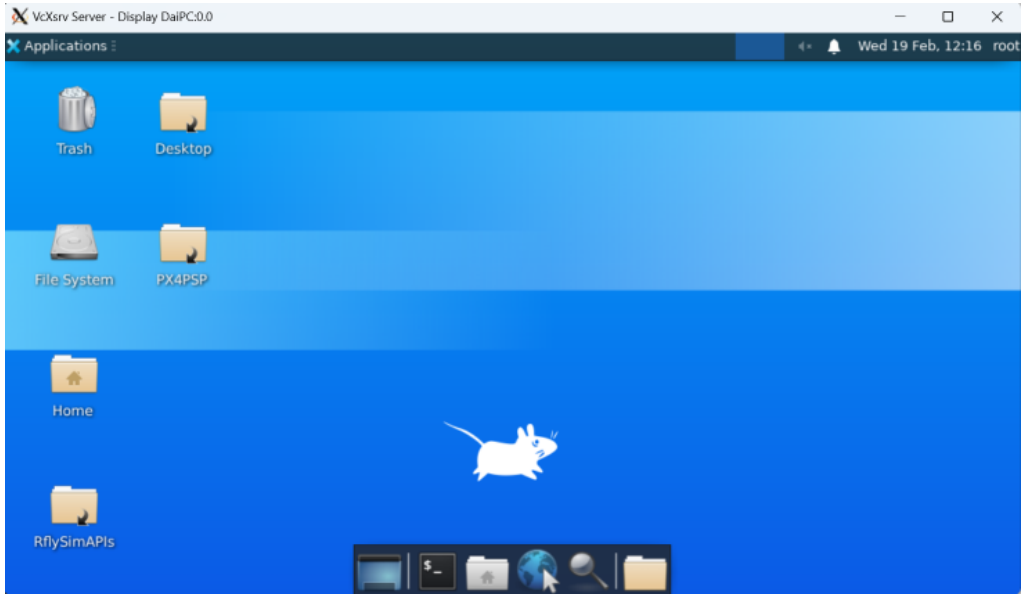


Fig. 2.11(d) WinWSL/WslGUI

Installation and first launch: see 2.1 and 2.2.

- **Windows:** The bundled interpreter is Python38\python.exe under the installation folder. The SDK is registered for control and vision calls.
- **WSL (Windows Subsystem for Linux):** WinWSL provides Linux with ROS 1 and ROS 2 (Robot Operating System) preinstalled for Linux or ROS examples.
- **Editor:** Install the Python extension in VS Code and select this python.exe to run with its SDK and dependencies.

The editor edits code; the interpreter determines the environment that runs it.

2.4.1 Tasks and Interface Selection: Files and Programs

How Do Configurations, Scripts, and Programs Work Together?

File or Object	Role	Typical Use
Config.json	Stores sensor identity, pose, and output method	jsonLoad reads VisionSensors
BAT startup script	Sets component parameters and starts processes	Starts simulation, ROS, or the receiver
Python program	Calls interfaces, updates targets, and uses feedback	Control loops, image acquisition, and processing
Graphical configuration tool	Edits parameters and saves reusable configurations	VisCreate, SimCreate

Reuse an existing JSON file or startup script; see 2.2 for a complete first run.

Chapter Outline

From Parameter Conventions to Control Feedback

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

4 Units in This Section

2.4.1 Tasks and Interface Selection
Textbook: Sec. 2.2.2.4–2.2.2.6, 2.2.3

2.4.2 Control Interfaces and Calling Conventions
Textbook: Sec. 2.2.3.2

2.4.3 Configuration Outputs and Vision Calls
Textbook: Sec. 2.2.2.7–2.2.2.8, 2.2.3.3

2.4.4 Modeling and Extension Interfaces
Textbook: Sec. 2.2.3.1–2.2.3.2

2.4.2 Control Interfaces and Calling Conventions: Object Identity

Vehicle IDs, Sensor IDs, and List Indices

Identifier	Identified Object	Example in This Lesson
CopterID	CopterSim or 3D vehicle identity	Vehicle 1
MAVLink SysID	Autopilot communication system ID	Set by firmware and connection configuration
SeqID	Vision sensor identity	SeqID = 1 in the configuration
List index i	Element position in a receive array	Img[0] for the first sensor

Map vehicle → sensor → array element before reading data or sending control targets.

Vehicle IDs identify motion objects; SeqID identifies a sensor; an index selects a receive-list element.

2.4.2 Control Interfaces and Calling Conventions: Communication Endpoints

Port Conventions Depend on Communication Mode

```
PX4MavCtrler(ID=1, ip='127.0.0.1',
        Com='udp', port=0)  # In serial mode, port is the baud rate
```

Default Local UDP (User Datagram Protocol)	Send to CopterSim	Receive Returned Data
Vehicle 1	20100	20101
Vehicle 2	20102	20103

- **Port rule:** Transmit on 20100+2(ID−1), receive on the next port; vehicle 3 uses 20104/20105.
- **Two-way communication:** Destination IP and port locate the sending endpoint; return address and listening port route feedback to the program.

Default UDP: send/receive ports follow vehicle IDs; destination/return settings complete the round trip.

2.4.2 Control Interfaces and Calling Conventions: Coordinates and Units

A Velocity Value Needs a Frame and Units

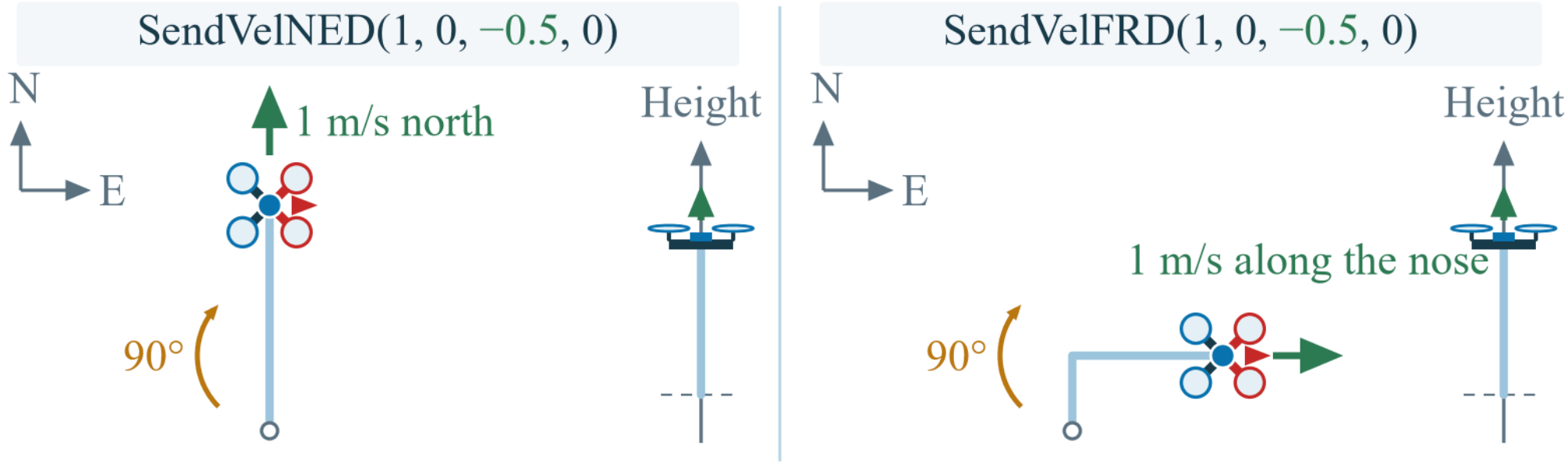
```
mav.SendVelNED(1, 0, 0, 0)
mav.SendVelFRD(1, 0, 0, 0)
```

- **NED (north–east–down):** The first line sets a northward target of 1 m/s, independent of heading.
- **FRD (forward–right–down) interface:** In this PX4 example, heading rotates horizontal velocity; the second line requests 1 m/s horizontally toward the nose.
The third component is positive downward.
- **Fourth argument:** yawrate is yaw angular velocity in rad/s; the first three arguments are in m/s.
- **Defaults:** Omitted arguments are NaN, not zero velocity. Both FRD horizontal components must be valid; this example supplies all four values.

Northward velocity stays northward; FRD horizontal velocity follows heading. Upward velocity is negative.

2.4.2 Control Interfaces and Calling Conventions: Reading Directions

How Do NED and FRD Velocity Directions Differ?



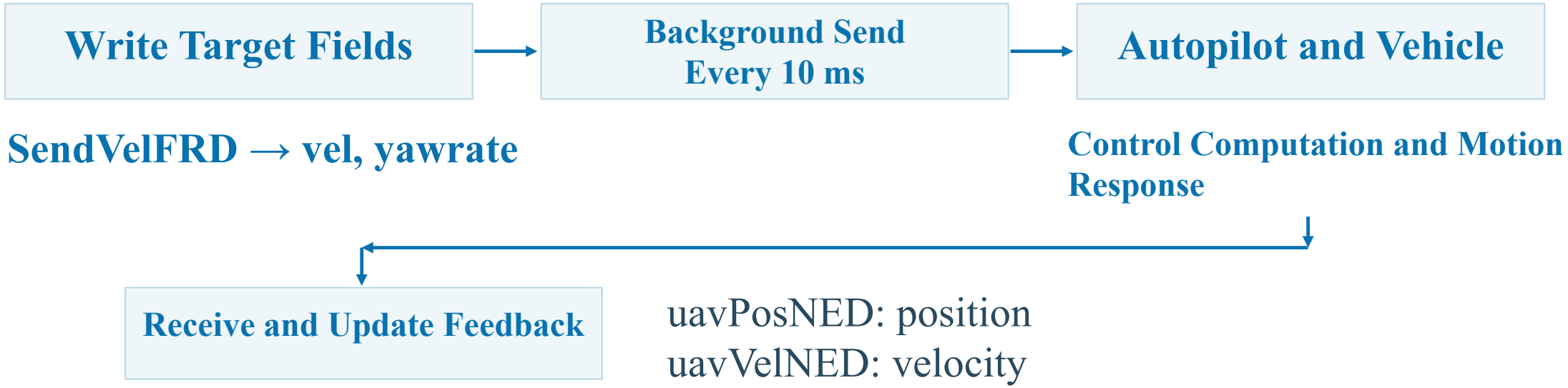
NED north; FRD follows heading; vz < 0: climb

Figure: The Same Velocity Values in NED and FRD (Conceptual Illustration)

Northward velocity stays northward; FRD horizontal velocity follows heading. Upward velocity is negative.

2.4.2 Control Interfaces and Calling Conventions: Targets and Feedback

Setting a Target and Reading State Are Different



Thread: a loop alongside the main program. Cache: the object's latest target or feedback values.

Arrival requires fresh feedback and both position error and actual velocity within the task thresholds.

Target cache → background send → autopilot motion control from estimates; receive thread → feedback.

2.4.2 Control Interfaces and Calling Conventions: Complete Call Sequence

Control Calls from Initialization to Normal Shutdown

Stage	Representative Call or State	Result
Establish connection	Construct object; InitMavLoop()	Start reception; wait for valid state
Enter control	Confirm readiness; initOffboard()	Start target stream; confirm mode
Execute task	SendVel…; read position and velocity	Continuous targets and motion feedback
Finish task	Land and confirm disarming	Landed and disarmed
End communication	endOffboard(); stopRun()	Loops stopped; connection closed

Confirm communication and control state, then keep updating targets until the task is complete.

2.4.2 Control Interfaces and Calling Conventions: PX4 Software-in-the-Loop (SITL)

Initialization and Shutdown Also Change Vehicle State

Function	Threads and Connection	Vehicle State or Parameters
InitMavLoop	Starts reception; creates send-thread object	Also sends a disarm command
initOffboard	Starts continuous target transmission	Sends parameter settings; requests arming and mode change
endOffboard	Stops and joins the send thread	May request exit; does not land or restore parameters
stopRun	Stops reception; closes connection	Sends a forced-disarm command if still armed

Software-in-the-loop (SITL; SIL in the textbook): initialization requests disarming or mode changes; stopRun also requests forced disarming if still armed.

2.4.2 Control Interfaces and Calling Conventions: Reading the Code

Read the Key Lines of a Minimal Control Call Sequence

```
import time, PX4MavCtrlV4 as PX4MavCtrl
mav = PX4MavCtrl.PX4MavCtrler(1)
mav.InitMavLoop()
while not mav.isPX4Ekf3DFixed: time.sleep(0.5)
mav.initOffboard()
mav.SendPosNED(0, 0, -1.7, 0)
position = mav.uavPosNED
velocity = mav.uavVelNED
```

- **Connection object:** ID=1 selects vehicle 1 on ports 20100/20101.
- **Readiness:** Wait for isPX4Ekf3DFixed to become True before requesting external control.
- **Target and result:** z=−1.7 means 1.7 m above the origin; the fourth argument is yaw angle in rad.
- **Check feedback:** Read position and velocity. Arrival requires fresh feedback with error and velocity within the task thresholds.

Adapted from Chapter 1 MavTest.py; production needs a wait timeout. Shutdown: Complete Call Sequence table.

2.4.2 Control Interfaces and Calling Conventions: State Checks

When to Continue and When to Finish

Stage	Condition for Continuing	If Not Met
Reception	Correct instance ID; uavTimeStmp increasing	Check address, mode, and message source
External control	isPX4Ekf3DFixed and isArmed true; QGC shows Offboard	Timeout on ground: do not start task; in flight: proceed to landing
Target execution	Fresh feedback: error and velocity within thresholds	Keep updating or enter the task's response branch
Normal shutdown	Landing complete; vehicle confirmed disarmed	Land and confirm state before stopping loops

Fresh feedback, valid estimates, and correct mode support control; land and disarm before disconnecting.

2.4.2 Control Interfaces and Calling Conventions: Troubleshooting

What to Check When a Command Has No Effect

- **Wrong object:** Constructor ID, destination address, and current vehicle do not match; the command may have reached another instance.

- **Wrong mode:** The current mode does not accept the position or velocity target; the value exists but is not used in control computation.

- **Target not updated:** Without the background loop, targets are not sent continuously. If the algorithm stops updating while the loop runs, the old cached target is sent repeatedly.

- **Stale feedback:** The program reads cached old values. Check message time, mode, and state together, rather than relying only on the 3D view.

Chapter Outline

From Configuration Outputs to Vision Data

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

4 Units in This Section

2.4.1 Tasks and Interface Selection
Textbook: Sec. 2.2.2.4–2.2.2.6, 2.2.3

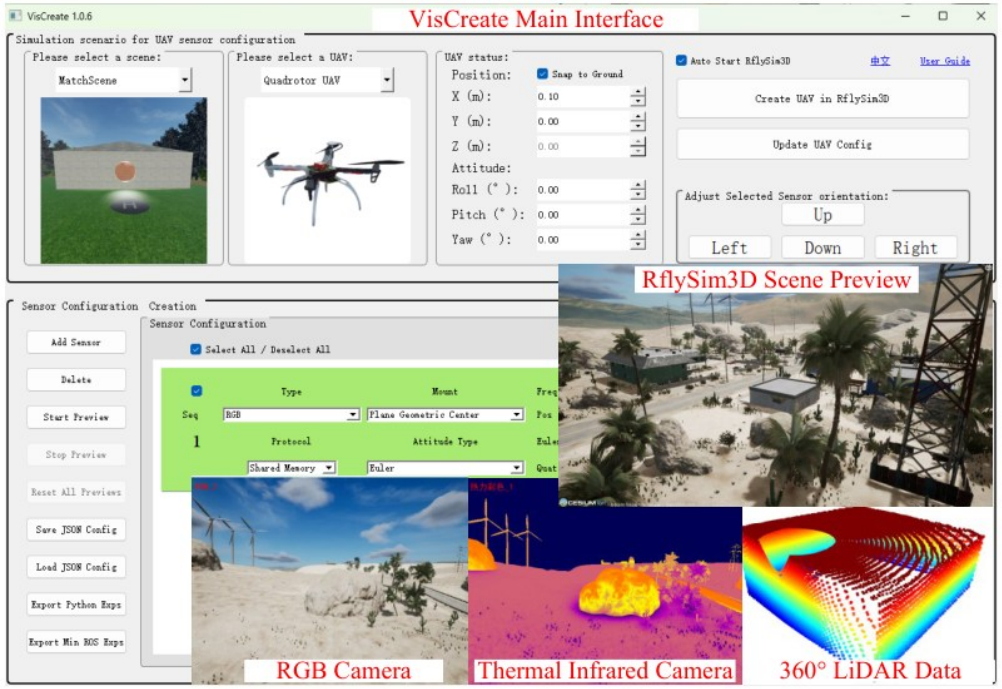
2.4.2 Control Interfaces and Calling Conventions
Textbook: Sec. 2.2.3.2

2.4.3 Configuration Outputs and Vision Calls
Textbook: Sec. 2.2.2.7–2.2.2.8, 2.2.3.3

2.4.4 Modeling and Extension Interfaces
Textbook: Sec. 2.2.3.1–2.2.3.2

2.4.3 Configuration Outputs and Vision Calls: Configuration Overview

VisCreate Configures Observations; the 3D Engine Generates Data



- VisCreate (observations): Set the scene, vehicle, and sensors; save Config.json or export Python/ROS examples.
- A scene preview is not camera output; visible light, thermal infrared, and point clouds serve different observation tasks.
- SimCreate (experiments) organizes vehicle types and experiments; 3D commands (interaction) control scene objects.

Fig. 2.12 VisCreate Interface and Visual Sensor Function Preview

VisCreate stores sensor parameters; the 3D engine renders from them; the receiver acquires observations.

2.4.3 Configuration Outputs and Vision Calls: Camera Fields

What Must Be Configured for a Virtual Camera?

Field	Configured Value	Meaning
SeqID / TargetCopter	1 / 1	Sensor identity / attached vehicle
TypeID / TargetMountType	1 / 0	RGB / rigid mount at vehicle center
DataWidth / DataHeight	640 / 480	Output width/height (pixels)
CameraFOV / SensorPosXYZ	90 / [0.3, 0, 0]	Field of view (°) / mount offset (m)
DataCheckFreq / SendProtocol	30 / [1, 0, …]	Update rate (Hz) / 1 selects UDP

Identity, mounting pose, imaging parameters, and transmission method together define an observation.

2.4.3 Configuration Outputs and Vision Calls: Multimodal Outputs

One Scene Can Produce Different Observations



Fig. 2.12 Visible-Light Output

Color and texture pixels
for appearance recognition and localization

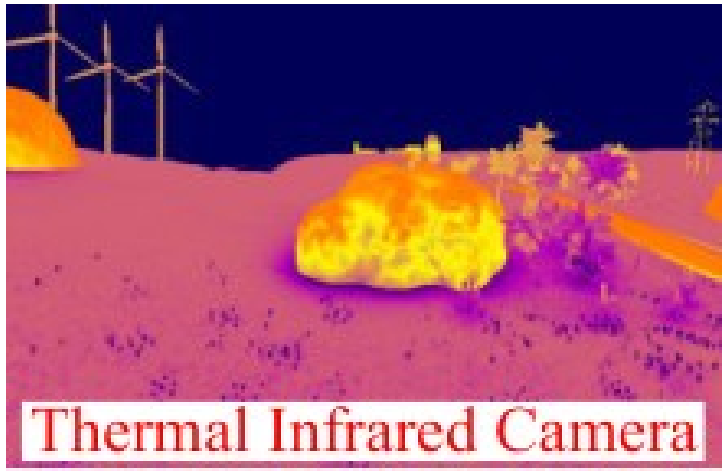


Fig. 2.12 Thermal-Infrared Output

Simulated thermal-radiation differences
for thermal-target perception

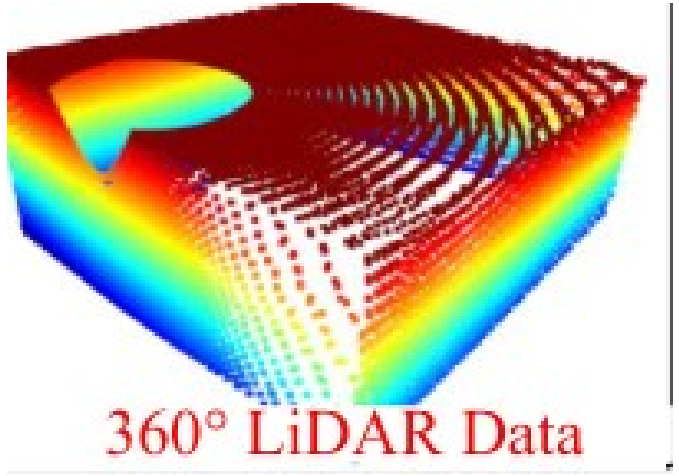


Fig. 2.12 Point-Cloud Output

3D coordinates of spatial points
for obstacle detection and mapping

The observation type changes what the data mean; the algorithm must use the corresponding input.

2.4.3 Configuration Outputs and Vision Calls: Call Sequence

Three Steps from JSON to Image Reception

```
import VisionCaptureApi
vis = VisionCaptureApi.VisionCaptureApi()
vis.jsonLoad(jsonPath="Config.json")
```

```
ok = vis.sendReqToUE4(windID=0)
```

```
if ok:
    vis.startImgCap()
```

- **Parse:** jsonLoad appends configurations; relative paths start at the entry script's folder.
- **Request:** An empty list returns False. True means the request was sent, not that an image arrived.
- **Receive:** startImgCap starts threads; new frames update images, flags, and timestamps.

2.4.3 Configuration Outputs and Vision Calls: Detecting New Frames

Timestamp Deduplication and Image Pairing

```
i = 0
if vis.hasData[i]:
    stamp = vis.timeStmp[i]
    if stamp != last_stamp:
        frame = vis.Img[i].copy()
        last_stamp = stamp
```

Deduplication snippet: Reception is running.
Initialize last_stamp to None; retain it across iterations.

- **i = 0:** First configuration-list entry, not CopterID or SeqID.
- **hasData:** Reading does not clear it; the same frame may be read again.
- **timeStmp:** Compare against the previous value; reset the baseline when capture restarts.

The receive thread writes images/timestamps separately; strict pairing must enqueue both as one item.

Timestamps avoid repeated processing; strict image/timestamp pairing requires receiver synchronization.

2.4.3 Configuration Outputs and Vision Calls: Deduplication and Pairing

Deduplication Does Not Ensure Image/Timestamp Pairing

hasData does not clear automatically; old frames may be read repeatedly.

Comparing timeStmp deduplicates; same-frame pairing is not ensured.

UDP writes image/timestamp separately; enqueue them together for strict pairing.

Times shown are illustrative, not measured.

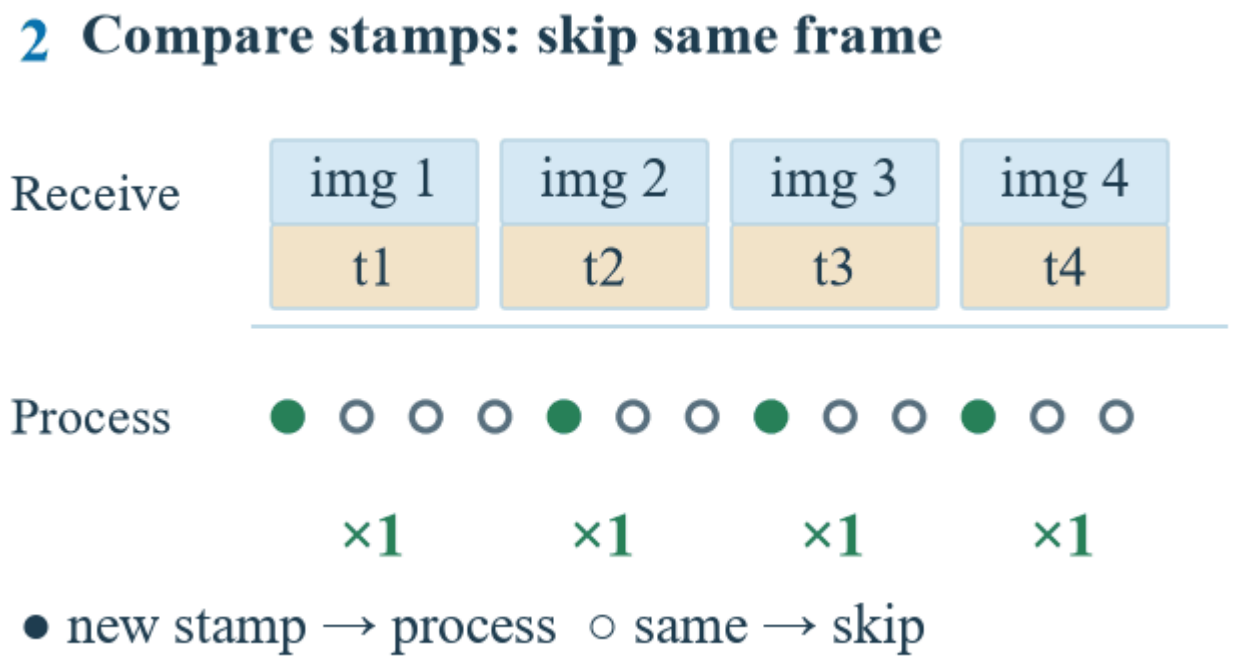


Figure: Timestamp Deduplication / Paired Queuing (Conceptual)

Timestamp deduplication may read mismatched images; the receiver must preserve same-frame pairing.

2.4.3 Configuration Outputs and Vision Calls: Using Saved Parameters

How Do Saved Parameters Enter the Program?

```
"DataWidth": 640,
"DataHeight": 480,
"CameraFOV": 90.0,
"SensorPosXYZ": [0.3, 0, 0]
```

Field of view (FOV): 90° → 60°
Pixels: 640×480 (unchanged)

Fix pose; compare the target's share of the image.

- **File to object:** After saving, restart the receiver and load once into a new object to avoid duplicate list entries.
- **Object to request:** Request and start reception with the new configuration; the 3D engine generates observations and the program reads updated frames.
- **Change one variable:** Change only the field of view (FOV) from 90° to 60° and compare the visible area.

After saving the file, load it and send a request for changed parameters to affect observations.

2.4.3 Configuration Outputs and Vision Calls: Changing the Field of View

How Does the Image Change as the Field of View Narrows?

Fix 640×480 pixels and camera pose; reduce only horizontal FOV from 90° to 60°.
Top-view/right image: target width grows 64 → ~111 px; both side reference panels leave view.

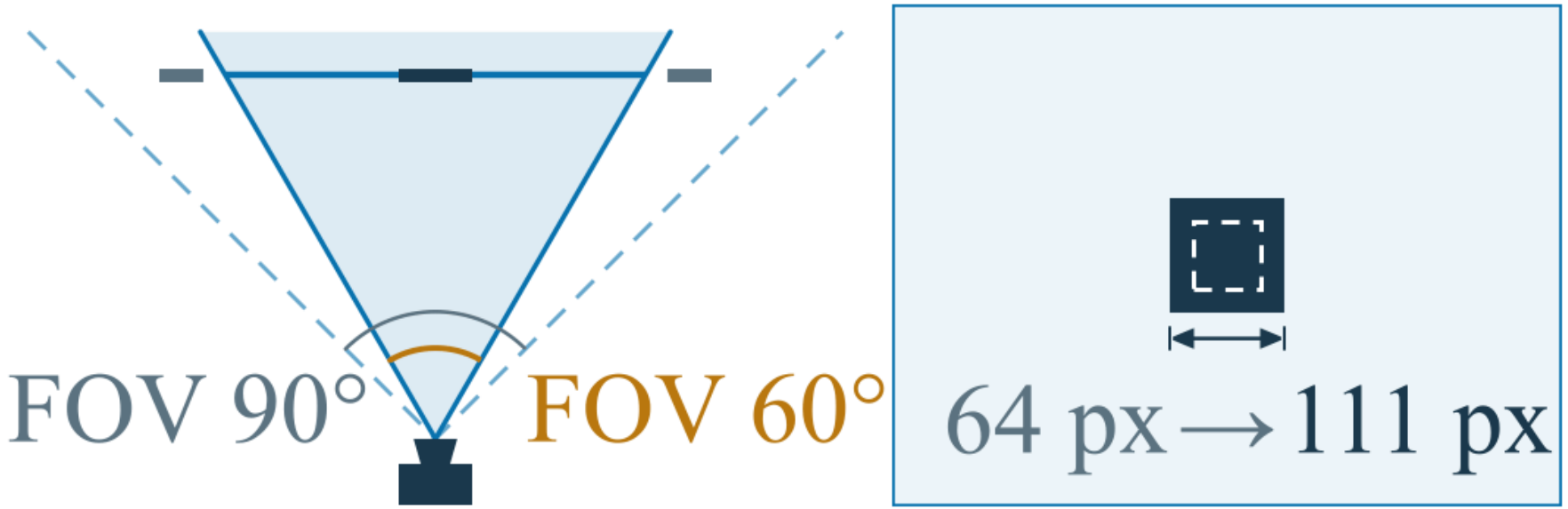
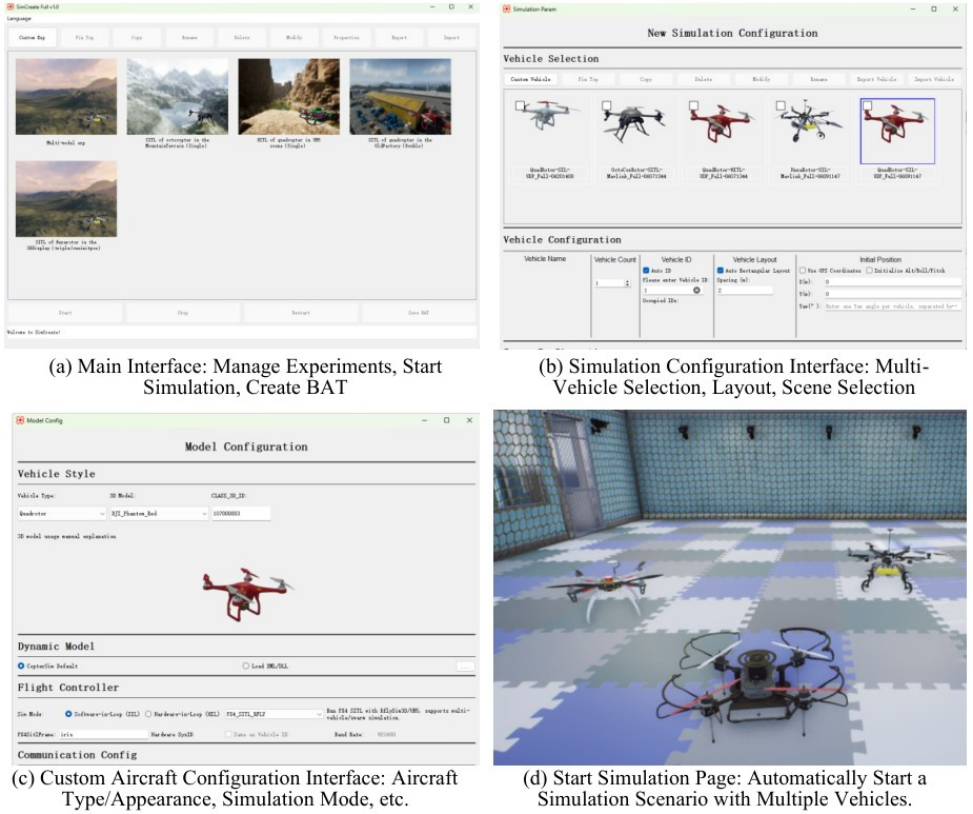


Figure: Narrower Horizontal Field of View at Fixed Pixel Count (Geometric Illustration)

Fixed pixels and pose: a narrower field of view reduces coverage and increases target width in pixels.

2.4.3 Configuration Outputs and Vision Calls: Organizing Experiments

SimCreate Organizes Experiments, Configuration, and Execution



(a) Main Interface: Manage Experiments, Start Simulation, Create BAT

(b) Simulation Configuration Interface: Multi-Vehicle Selection, Layout, Scene Selection

(c) Custom Aircraft Configuration Interface: Aircraft Type/Appearance, Simulation Mode, etc.

(d) Start Simulation Page: Automatically Start a Simulation Scenario with Multiple Vehicles.

- **(a) Experiment management:** Create, save, and launch experiments.
- **(b) Simulation configuration:** Select vehicles, layout, and scene.
- **(c) Vehicle-type configuration:** Set appearance, model, and mode.
- **(d) Running result:** Show the vehicles and scene actually launched.

Fig. 2.13 SimCreate's Three-Level Interface and Running Result

2.4.3 Configuration Outputs and Vision Calls: Experiment Configuration

Which Objects Does an Experiment Configuration Record?

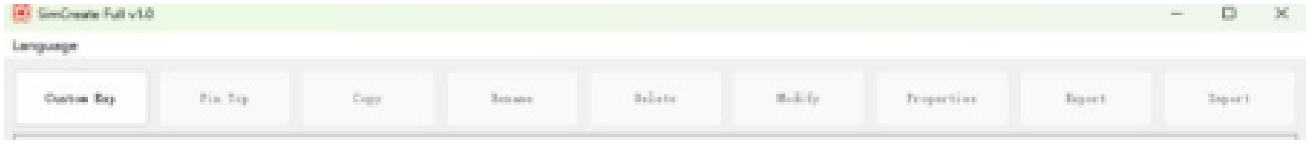


Fig. 2.13(a) Experiment Management Toolbar (Detail)

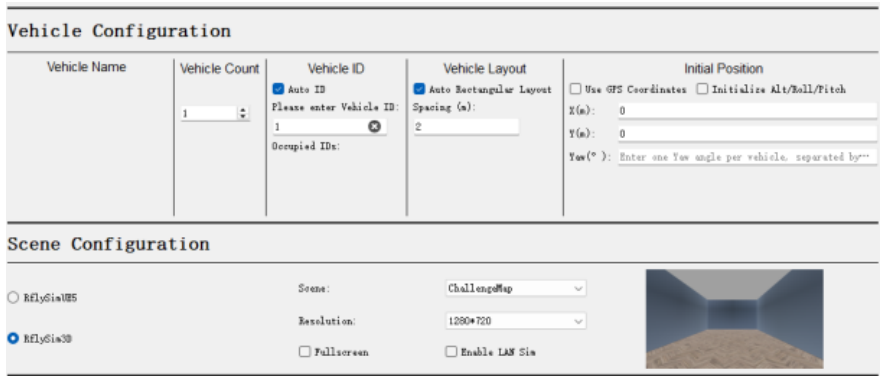


Fig. 2.13(b) Vehicle and Scene Configuration (Detail)

- **Experiment management:** Save, launch the experiment from the main UI, then reopen it to check that settings are restored.
- **Vehicles:** With starting ID 2 and count 3, automatic IDs are 2, 3, and 4; initial states must match each vehicle.
- **Scene:** Environment, resolution, and startup options are saved with the experiment.

Saving must preserve parameters so vehicle IDs and initial layout can be reproduced.

2.4.3 Configuration Outputs and Vision Calls: Vehicle Types and Scripts

How Does Vehicle-Type Configuration Enter a Run?

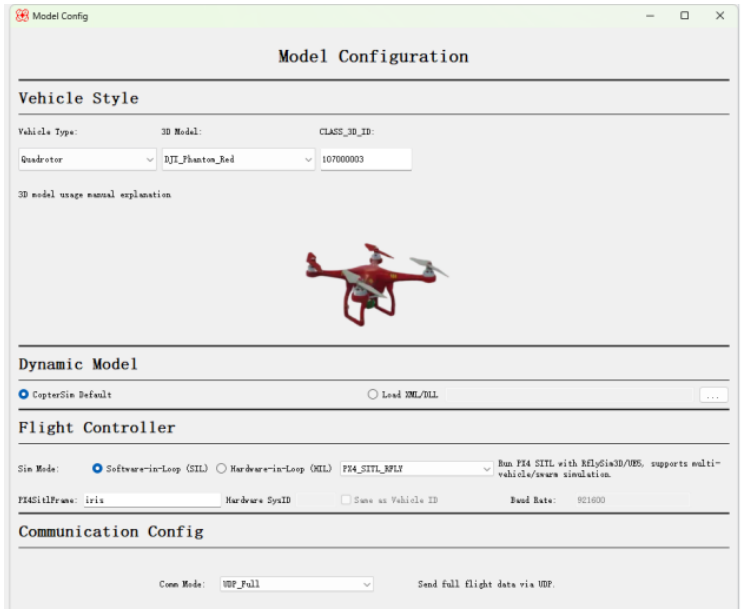
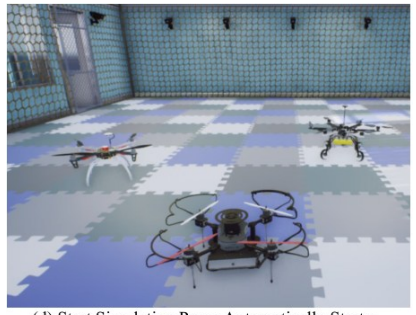


Fig. 2.13(c) Vehicle-Type Configuration

```
SET /a CLASS_3D_ID=105000003      → Appearance
SET DLLModel=0                    → Motion model
SET SimMode=PX4_HITL              → Simulation mode
SET UE4_MAP=Quarry                → Map
```



(d) Start Simulation Page: Automatically Start a Simulation Scenario with Multiple Vehicles.

Fig. 2.13(d) Run Result

Another experiment script exported by SimCreate; hardware-in-the-loop (HITL; HIL in the textbook) requires a physical autopilot.

Configure appearance, motion model, and simulation mode separately, then launch them through a script.

2.4.3 Configuration Outputs and Vision Calls: UI Effects

What Do 3D UI Operations Change?

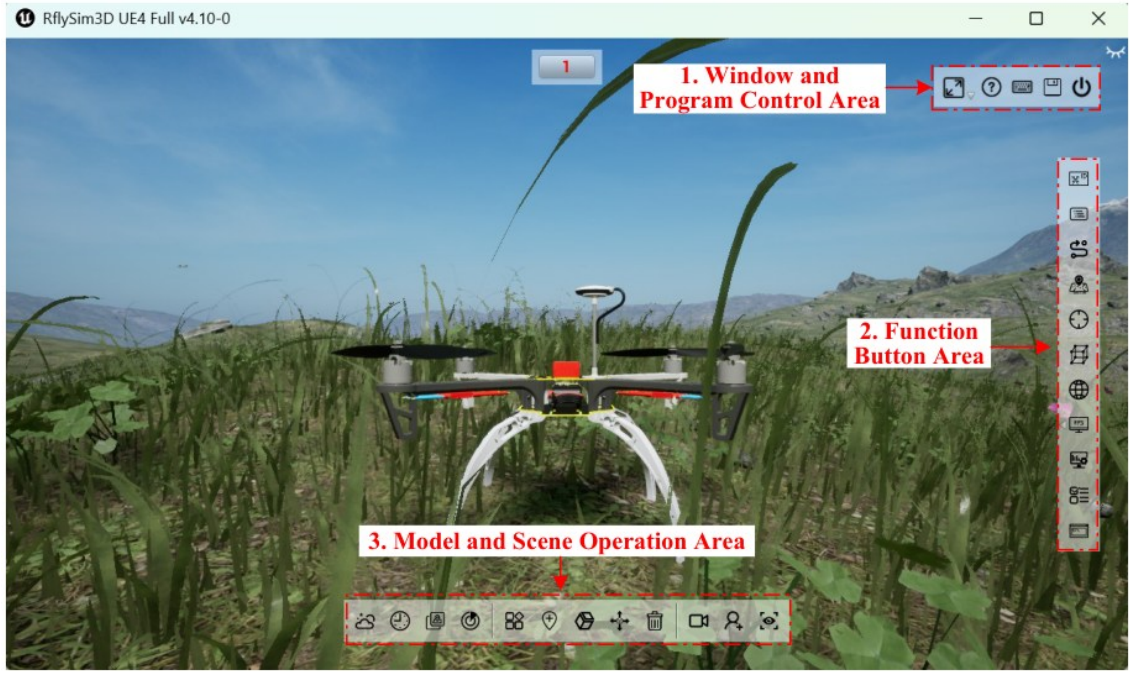


Fig. 2.16 RflySim3D Configuration Function Keys

- **Area 1:** Window size, settings, help, and program exit.
- **Area 2:** Mini-map, frame rate, vehicle IDs, and other information; also opens command entry.
- **Area 3:** Weather, time, and maps change the observation environment; vehicle focus and viewpoint switches change only the displayed view.

Viewpoints change views; weather/maps set the environment; run controls change the simulation process.

2.4.3 Configuration Outputs and Vision Calls: Shortcuts and Console

Use Keys for Focus, Views, and Maps; Enter Commands in the Console

Key	Action	Notes
F1	Open help menu	Lists keyboard shortcuts
B+number	Focus on vehicle with that ID	Changes only observation focus
V+number	Switch to viewpoint N	0: follow; 1: front, etc.; display only
M+number	Switch to map N	CopterSim may switch it back
` (backtick)	Open console	Enter equivalent shortcut commands

RflySim3D shortcut and console-command index: https://rflysim.com/doc/zh/4.DevRef/3d-keys.html

Shortcuts and console commands access the same 3D functions; focus and viewpoint changes affect only the displayed view.

2.4.3 Configuration Outputs and Vision Calls: Command Targets

Express a UI Operation as an Explicit Command

```
import UE4CtrlAPI
ue = UE4CtrlAPI.UE4CtrlAPI()
ue.sendUE4Cmd('RflyChangeViewKeyCmd B 1', windowID=0)
```

- **Command:** Equivalent to B+1 in the 3D window; focus on the vehicle with CopterID 1.
- **Receiving window:** windowID=0 sends only to window 0, not vehicle 0; omitting it sends to windows 0–2.
- **Effect:** Changing focus changes only the view. Control interfaces send motion targets for the autopilot to execute.

The window ID selects the receiver; the ID inside the command selects the target object.

Chapter Outline

Understand Graphical Blocks and Extension Interfaces

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

4 Units in This Section

2.4.1 Tasks and Interface Selection
Textbook: Sec. 2.2.2.4–2.2.2.6, 2.2.3

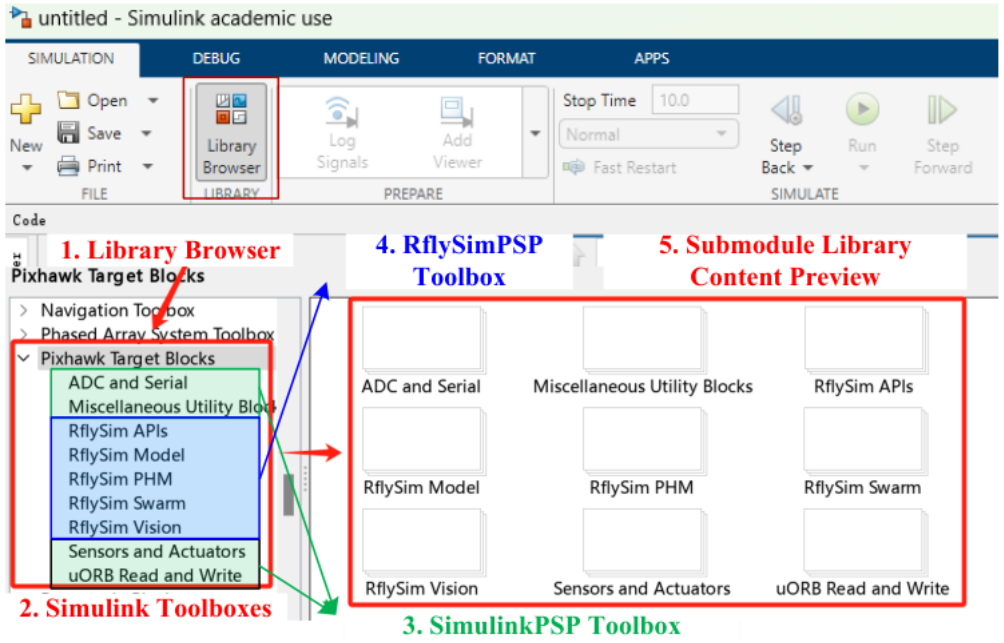
2.4.2 Control Interfaces and Calling Conventions
Textbook: Sec. 2.2.3.2

2.4.3 Configuration Outputs and Vision Calls
Textbook: Sec. 2.2.2.7–2.2.2.8, 2.2.3.3

2.4.4 Modeling and Extension Interfaces
Textbook: Sec. 2.2.3.1–2.2.3.2

2.4.4 Modeling and Extension Interfaces: Block Overview

Simulink Blocks Also Have Input/Output Conventions



- **1–2 Access the libraries:** Library Browser and toolbox hierarchy.
- **3 SimulinkPSP:** Official library for hardware I/O and PX4 internal messages.
- **4 RflySimPSP:** Motion models, control, health, swarm, and vision.
- **5 Individual blocks:** Ports, parameters, and sampling conventions.

Fig. 2.15 RflySim Toolchain Simulink Module Library

Ports carry algorithm signals; parameters set interface behavior; sample time sets the call rate.

2.4.4 Modeling and Extension Interfaces: Library Roles

What Do the SimulinkPSP Base and RflySimPSP Extension Libraries Provide?

Library	Actual Sublibrary Name	Typical Task
Base	ADC and Serial	Analog-to-digital conversion and serial I/O
Base	Miscellaneous Utility Blocks	Logging, printing, parameter passing
Base	Sensors and Actuators	Sensor acquisition and actuator interfaces
Base	uORB Read and Write	Internal message read/write
Extension	RflySim APIs / Model	State/control interfaces / motion models
Extension	RflySim PHM / Swarm / Vision	Health assessment / swarm / vision

2.4.4 Modeling and Extension Interfaces: Internal Messages

How Do uORB Messages Become Algorithm Inputs?

PX4 Internal Message → **uORB Read Block** → **Algorithm Input Signal**

vehicle_local_position

- **Publish/subscribe:** The micro Object Request Broker (uORB) carries sensor, estimation, and control messages within PX4.
- **Select fields:** x, y, z are north, east, and down positions (m); xy_valid and z_valid indicate horizontal and vertical position validity.
- **Conditions:** Also check that timestamps advance. Do not feed invalid or stale estimates directly to control. MAVLink communicates externally; uORB connects internal modules.

uORB carries PX4 internal state to algorithms; field units, frames, and validity flags determine its use.

2.4.4 Modeling and Extension Interfaces: Ports and Sampling

After Selecting a Block, Read Its Ports and Sampling

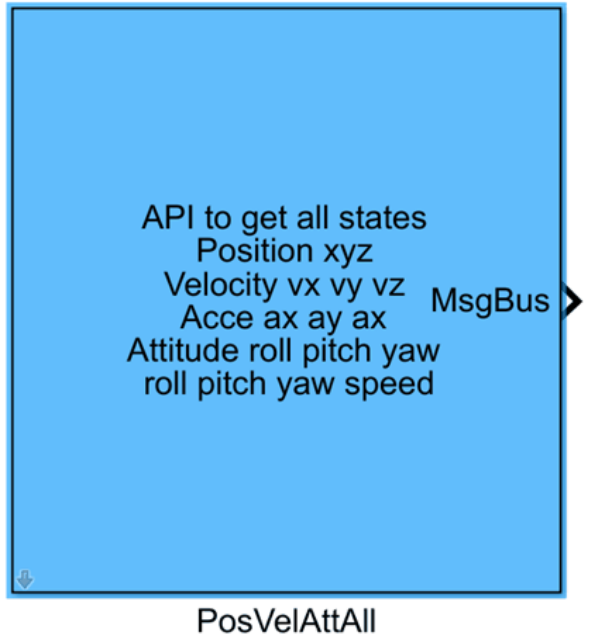


Figure: PosVelAttAll Block

- **MsgBus:** An output bus bundles signals by field name, with pos_vel_acc and att_rate branches.
- **Field sources:** The former comes from vehicle_local_position; the latter organizes attitude, quaternions, and angular rates.
- **Parameters and sampling:** SamTime defaults to 1/250 s. Reading faster than publication returns the previous message again.
- **Algorithm connection:** Select fields by name with Bus Selector; use the original message definition for units and validity.

A bus carries named state fields; reading every 4 ms still returns messages at the publisher's update rate.

2.4.4 Modeling and Extension Interfaces: Build Commands

Setup, Build, and Upload Use Different Commands

MATLAB Command	Action	Target or Condition
PX4CMD	Sets build target; keeps source version	For example, px4_fmu-v6c_default
PX4Build	Builds firmware	Firmware for the selected target
PX4Upload	Uploads firmware already built	Matching autopilot hardware model
PX4AppName	Sets generated module name	Automatically generated control code
PX4SitlSet	Connects generated controller to SITL	Personal Edition or higher; code already generated
PX4SitlRec	Restores standard SITL control path	Personal Edition or higher; environment restoration

Target settings determine what is built; the build output then enters simulation or hardware.

2.4.4 Modeling and Extension Interfaces: Environment Setup

How Can a Custom Python Environment Find the SDK?

- **Interpreter:** Use the intended environment in both editor and terminal; run import sys; print(sys.executable) in Python to check the actual path.

- **Search paths:** After import finds the SDK, check module.__file__ for the loaded location. If a module is missing, check SDK path registration first.

- **Third-party dependencies:** The SDK also imports packages such as NumPy. Check the named missing package in the current environment; a missing package is not a reason to reinstall the entire platform.
- **Registration script:** On Windows outside venv, RflySimSDK\ReLabPath.py can write rflysim.pth.
Register venv paths in its own site-packages; the Linux/WSL branch changes PYTHONPATH.

2.4.4 Modeling and Extension Interfaces: ROS Adaptation

ROS Programs Must Preserve Interface Conventions

Example File	Task	Conventions to Preserve
Config.json	Defines sensors and observation parameters	IDs, mounting poses, types
sensor_ros_pub.py	Requests/publishes observations; includes takeoff	Message content, coordinates, time
sensor_ros_sub.py	Receives; fix ROS 2 subscription type first	Topic, data type, updates
vision_ros1/2.rviz	Observation settings for the matching branch	Reference frame and displayed objects

Files: e13_VisAPIPyCpp\2.rosPythonExps. The publisher includes vehicle control; run it in SITL.

An adaptation layer changes transport form, not the physical meaning of observations and targets.

2.4.4 Modeling and Extension Interfaces: Deliverables

What Should a Minimal Interface Experiment Deliver?

Task	Program and Configuration	Results and Explanation
Control calls	Object ID, connection mode; target sequence, exit conditions	Position/velocity feedback, mode, time
Vision calls	JSON, receive method; sensor order, consumer code	Images or point clouds, timestamps, parameters
Common requirements	Retain changes and corresponding versions	State targets, conditions, and actual observations

Organize interfaces using the workflow in 2.3; choose a complete example to reproduce in 2.5.

Configurations, programs, and results must correspond so others can explain and reproduce the calls.

2.4 API and SDK Interfaces: Class Exercise

What Is Missing from These Three Call Sequences?

Snippet or Symptom	Missing or Confused Element	Correction
Construct object, then call SendVelFRD only	Receive/send loops; control conditions	Initialize; check readiness and mode
Keep processing while hasData=True	Cached data do not imply a new frame	Deduplicate by timestamp; enqueue pairs at receiver
Treat windowID as CopterID	Window ID confused with vehicle ID	Separate window parameter from command target

Discuss: Does a completed call mean task completion? Check feedback and task conditions.

Call placement, data state, and identifier meaning determine the correction.

2.4 API and SDK Interfaces: Lesson Summary

From Task to Interface to Explainable Results

- **Locate the task:** Control, communication, vision, 3D, health, and swarm interfaces each have distinct roles.
- **Read the conventions:** Identity, endpoints, frames, units, defaults, and running state jointly constrain a call.
- **Connect the outputs:** Configurations, startup scripts, programs, and graphical blocks together organize an experiment.
- **Explain results:** Judge motion from fresh feedback and error; identify new data by timestamp. Land and disarm before closing the control interface.

Visual-sensor fields, output types, and request examples
https://rflysim.com/doc/zh/4.DevRef/vis-cfg.html