Chapter Outline

Platform Architecture and Communication Interfaces

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

6.7.1 Platform Architecture and Responsibilities
Textbook: Sec. 6.2.1

6.7.2 Communication Links and Protocols
Textbook: Secs. 6.1.1.5 and 6.2.2.1

6.7.3 Minimal Control and State Acquisition
Textbook: Secs. 6.2.2.2 and 6.2.2.4 (control flow)

6.7.4 Extended Interfaces and Algorithm Tools
Textbook: Secs. 6.2.2.2–6.2.2.3, 6.2.2.5, and 6.2.3

6.7.5 Communication Optimization and Acceptance
Textbook: Secs. 6.2.2.4 (send rate) and 6.2.3

6.7.1 Platform Architecture and Responsibilities: Core Components

Who Computes Setpoints, Responses, and Images?

Component	Inputs and Responsibilities	Main Outputs
External program	Mission, state, environment → compute references	Position, velocity, and other setpoints
PX4 flight controller	Estimated state → low-level control	Actuator commands, telemetry
CopterSim	Compute motion; relay state and commands	Sensor data, motion state
RflySim3D	Pose → scene display and imaging	Images, visible motion
QGroundControl (QGC)	Configuration, monitoring, missions, manual operation	Parameter, mode, and mission requests

6.7.1 Platform Architecture and Responsibilities: Communication Overview

Follow the Arrows: Commands, State, and Images

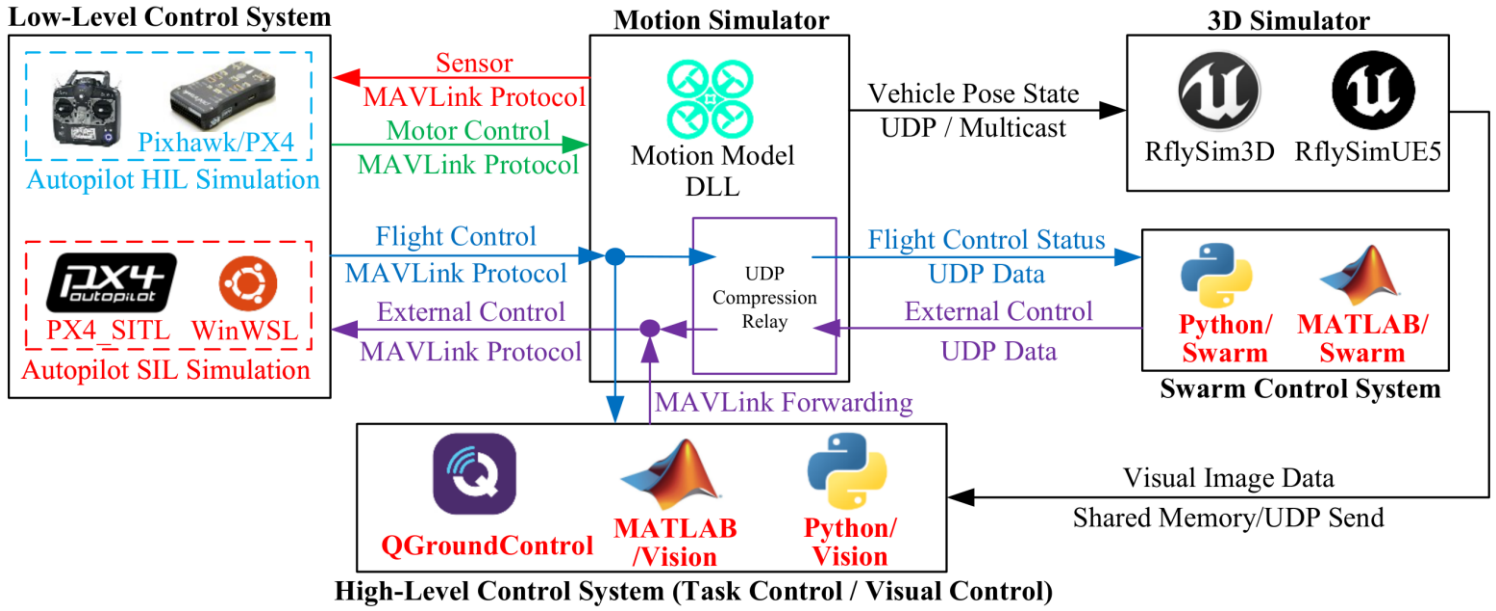


Fig. 6.8 Communication Architecture of External Control Programs

Control inputs change model or vehicle behavior.

State: distinguish estimates from model ground truth.

Images come from scene rendering for vision processing.

6.7.1 Platform Architecture and Responsibilities: Sensors and Actuators

The Motion Model and Flight Controller Close the Simulation Loop

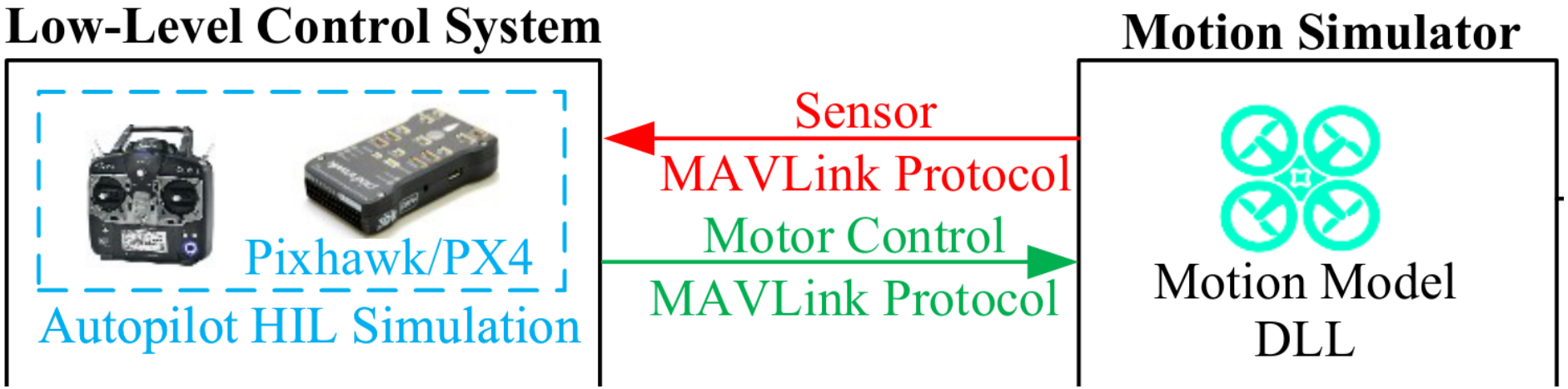


Fig. 6.8 (Upper Left) Sensor Inputs and Motor Control Outputs

DLL: dynamic-link library containing an executable motion model.

Closed loop: the model generates sensor data; the flight controller outputs commands; the model computes the next motion.

6.7.1 Platform Architecture and Responsibilities: State and Command Relay

Position Setpoints Go In; Estimated State Comes Back

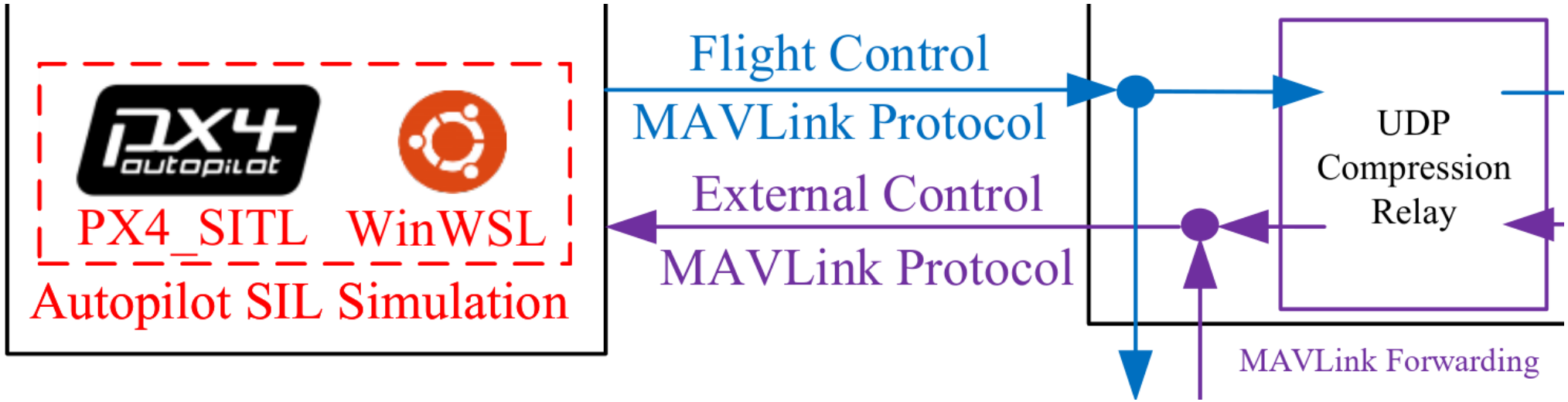


Fig. 6.8 (Center Left) State Feedback and External Control Commands

MAVLink defines message fields, IDs, and encoding for state and commands.

UDP relay: both ends must share data definitions for the full or simplified format.

6.7.1 Platform Architecture and Responsibilities: Display and Perception

3D Display Needs Pose; Vision Control Also Needs Images

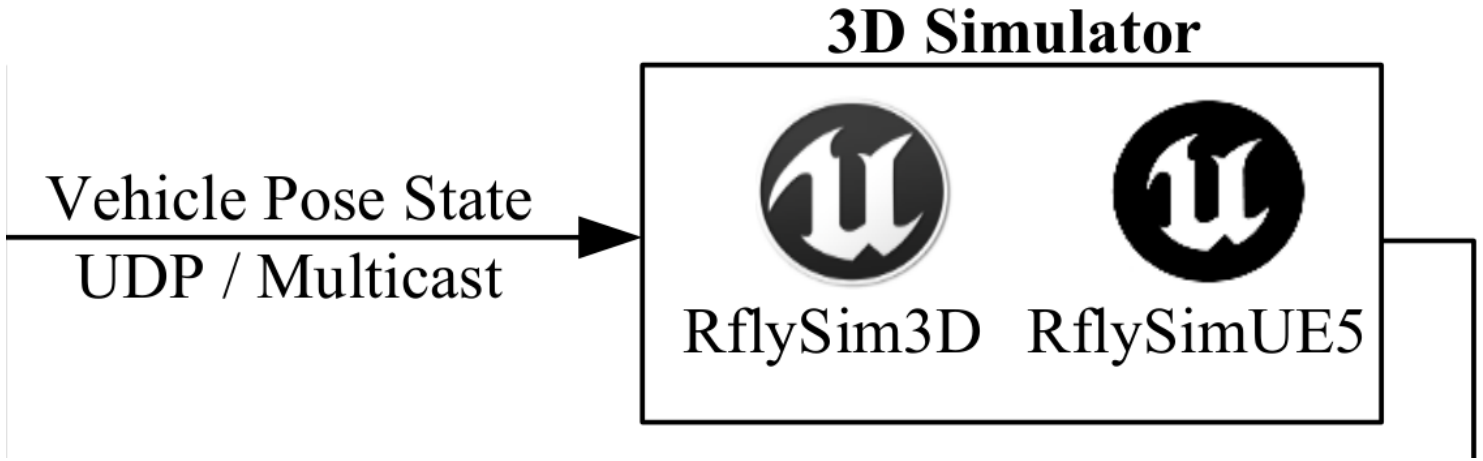


Fig. 6.8 (Upper Right) Pose Drives 3D Display

Pose channel: position and attitude determine the vehicle and camera state in the scene.

Image channel: camera images feed vision algorithms; usually much more data than a position message.

6.7.1 Platform Architecture and Responsibilities: Swarm Branch

Organize Swarm State and Setpoints by Vehicle Identity

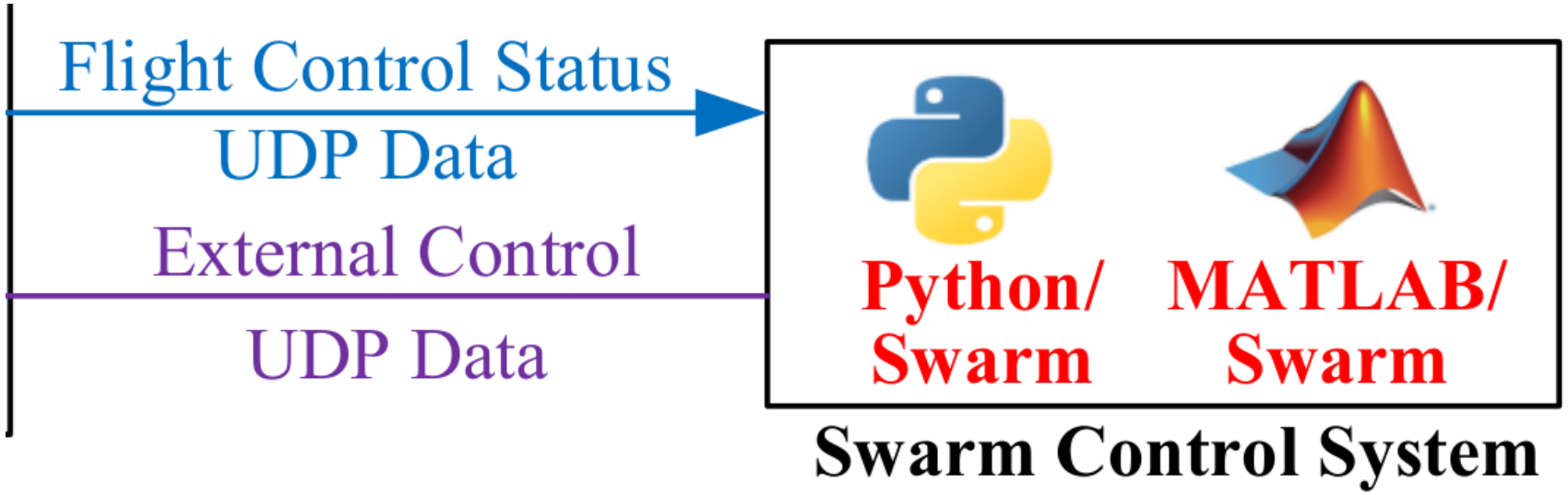


Fig. 6.8 (Center Right) Swarm State and Command Channels

Per-vehicle identity: match each ID, destination port, and received state.

Reduced data: retain required fields; record timestamps, coordinate frames, and units.

6.7.1 Platform Architecture and Responsibilities: Mission and Vision Control

High-Level Programs Turn State, Images, and Missions into Setpoints

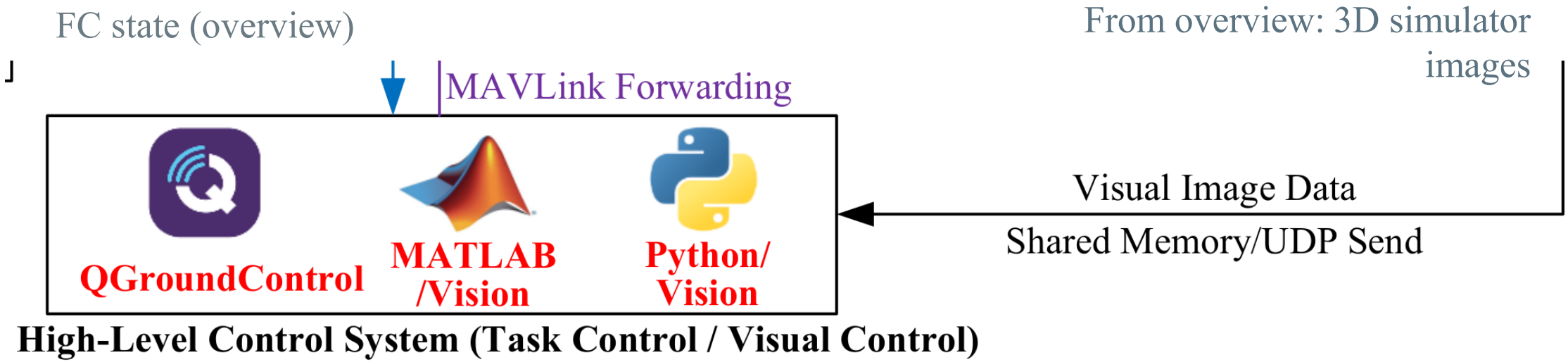


Fig. 6.8 (Bottom) Mission, Vision, and Flight-Controller Communication; FC: flight controller

A mission or vision program computes a target; an interface sends it to the flight controller.

To hardware: reuse message semantics; recheck ports, frame origins, state estimation, timing.

Chapter Outline

Platform Architecture and Communication Interfaces

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

6.7.1 Platform Architecture and Responsibilities
Textbook: Sec. 6.2.1

6.7.2 Communication Links and Protocols
Textbook: Secs. 6.1.1.5 and 6.2.2.1

6.7.3 Minimal Control and State Acquisition
Textbook: Secs. 6.2.2.2 and 6.2.2.4 (control flow)

6.7.4 Extended Interfaces and Algorithm Tools
Textbook: Secs. 6.2.2.2–6.2.2.3, 6.2.2.5, and 6.2.3

6.7.5 Communication Optimization and Acceptance
Textbook: Secs. 6.2.2.4 (send rate) and 6.2.3

6.7.2 Communication Links and Protocols: Links and Software Layers

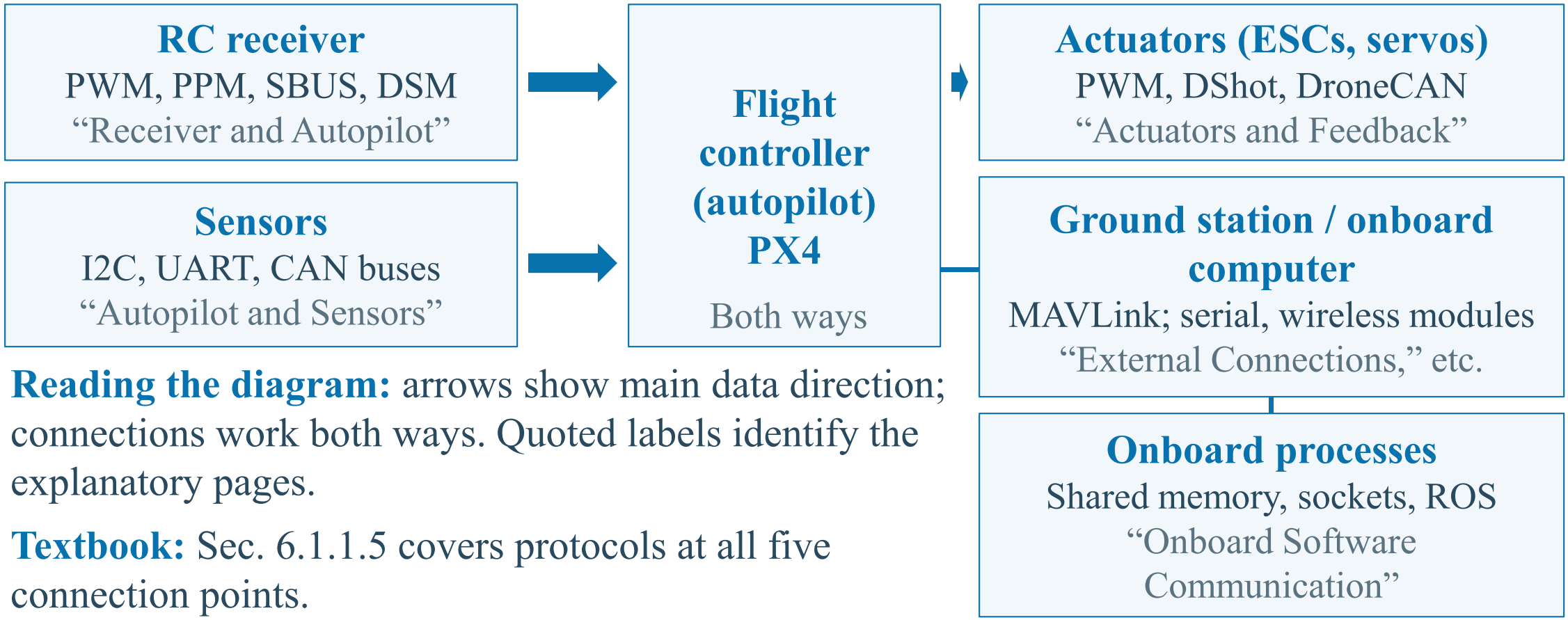
One Link Can Carry Multiple Message Protocols

Layer	Question It Answers	Examples Here
Application / SDK	SDK (software development kit): mission → function calls	SendPosNED; MAVSDK
Message protocol	How are fields, units, and identities interpreted?	MAVLink; custom platform data
Transport	How do message bytes reach a process?	UDP datagrams; TCP byte streams
Connection link	Which devices and channels are used?	Serial, telemetry radio, Wi-Fi, cellular

Serial electrical interfaces: TTL, RS232, RS485; match voltage levels, wiring, and converters.

6.7.2 Communication Links and Protocols: Connection Overview

Five External Connection Types Use Different Interfaces



Reading the diagram: arrows show main data direction; connections work both ways. Quoted labels identify the explanatory pages.

Textbook: Sec. 6.1.1.5 covers protocols at all five connection points.

6.7.2 Communication Links and Protocols: External Connections

TELEM1 and TELEM2 Are Configurable Serial Ports

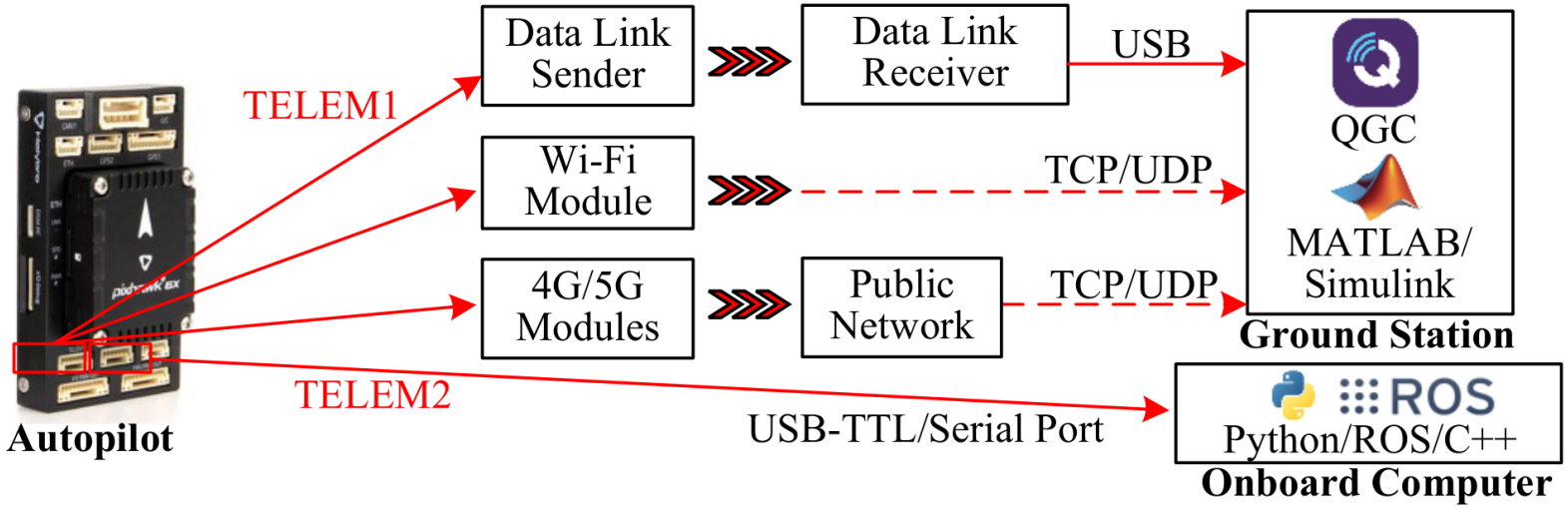


Fig. 6.9 Autopilot External Communication Framework

TELEM1: often connects telemetry radios, Wi-Fi, or cellular modules for telemetry and ground-station use.

TELEM2: often connects an onboard computer through USB-TTL or serial.

Configuration: flight-controller parameters assign port roles; external modules provide wireless connectivity.

6.7.2 Communication Links and Protocols: Module Selection

Choose Communication Modules by Range, Load, and Jitter

Module	Typical Tasks	Check Before Selection
USB-TTL / serial	Onboard direct link, hardware debugging	Voltage levels, wiring, baud rate
Telemetry radio	Remote telemetry and motion commands	Band, throughput, packet loss, coverage
Wi-Fi	LAN control, larger data transfers	Congestion, interference, reconnection
4G / 5G	Ground-service access across regions	Public IP addressing, latency jitter, coverage

Entry: [Installation directory]/RflySimAPIs/6.RflySimExtCtrl/0.ApiExps/e0_ComConnDemo/Readme.pdf

6.7.2 Communication Links and Protocols: Receiver and Autopilot

RC Protocols Carry Channel Values, Not Motor Outputs

Format	Channel Representation	Properties and Checks
PWM	Pulse width represents each channel value	Operator input; interpreted by mode
PPM	One pulse train multiplexes several channels	Less wiring; less interference-resistant than SBUS
SBUS	Serial digital frames carry multiple channels	Strong interference resistance; check signal inversion
DSM / DSMX	Spektrum radio-control system	Check receiver and protocol generation

Scope: SBUS is a serial protocol; the upstream wireless link is separate. Wireless interference resistance also depends on the transmitter/receiver system.

The same signal format has different meanings at RC inputs and actuator outputs.

6.7.2 Communication Links and Protocols: Autopilot and Sensors

Identify the Bus, Then Its Data Format

Interface / Format	Layer	Typical Use and Checks
IIC (I^2C)	Board-level bus	Few pins, low-speed use; address and pull-ups
UART	Asynchronous serial interface	Positioning, ranging; baud rate and framing
CAN	Multi-node bus	High reliability, redundant networks possible; termination
UAVCAN / DroneCAN	CAN message and node protocol	Version, node ID, message type
NMEA / UBX	GNSS navigation-data formats	Readable NMEA; efficient, lower-bandwidth UBX

6.7.2 Communication Links and Protocols: Actuators and Feedback

Actuator Interfaces Deliver Commands to Servos and ESCs

Format	Command Encoding	Feedback and Conditions
PWM	Pulse width	Conventional PWM carries no state feedback
OneShot / MultiShot	Shorter pulse timing	Still pulse-based; check ESC support
DShot / bidirectional DShot	Digital commands with checksum	eRPM feedback, etc.; implementation-dependent
DroneCAN	Bus control messages	Check speed, temperature, and other fields by device

Serial telemetry returns state; BLHeli32 is a firmware family, not a universal serial protocol.

6.7.2 Communication Links and Protocols: Onboard Software Communication

Processes Exchange Data; Algorithms Interpret It

Mechanism	Exchange Method	Rules to Define
Shared memory	Processes access the same data region	Synchronization, ownership, update flags
Pipes / message queues	Byte streams or message boundaries	Backlog, blocking, message order
Sockets	Local or network process communication	Addresses, ports, framing, timeouts
ROS topics / services	Publish/subscribe; request/reply	Types, timestamps, frames, QoS

DDS: Data Distribution Service. QoS: quality-of-service policies for reliability, queues, and other communication behavior.

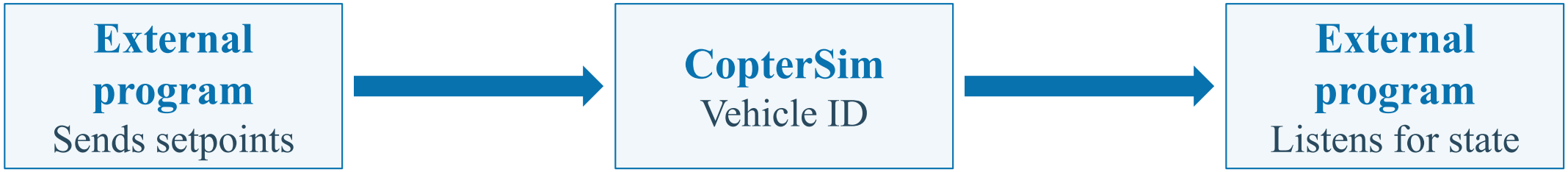
6.7.2 Communication Links and Protocols: Simulation Ports

Identify the Sender and Receiver for Every Port

$$p_{\text{cmd}} = 20100 + 2(\text{ID} - 1), \quad p_{\text{state}} = p_{\text{cmd}} + 1$$

Command destination port
20100+2(ID−1)

State destination port
Command port+1



Examples: ID=1 → 20100/20101; ID=2 → 20102/20103; ID=3 → 20104/20105.

30100 series: model ground truth and integrated-model interfaces; check format and purpose for the selected mode.

Settings: UDP needs IP addresses and ports; serial also needs device name and baud rate.

Chapter Outline

Platform Architecture and Communication Interfaces

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

6.7.1 Platform Architecture and Responsibilities
Textbook: Sec. 6.2.1

6.7.2 Communication Links and Protocols
Textbook: Secs. 6.1.1.5 and 6.2.2.1

6.7.3 Minimal Control and State Acquisition
Textbook: Secs. 6.2.2.2 and 6.2.2.4 (control flow)

6.7.4 Extended Interfaces and Algorithm Tools
Textbook: Secs. 6.2.2.2–6.2.2.3, 6.2.2.5, and 6.2.3

6.7.5 Communication Optimization and Acceptance
Textbook: Secs. 6.2.2.4 (send rate) and 6.2.3

6.7.3 Minimal Control and State Acquisition: External Control Workflow

"End" in the Diagram Ends Initialization, Not the Mission

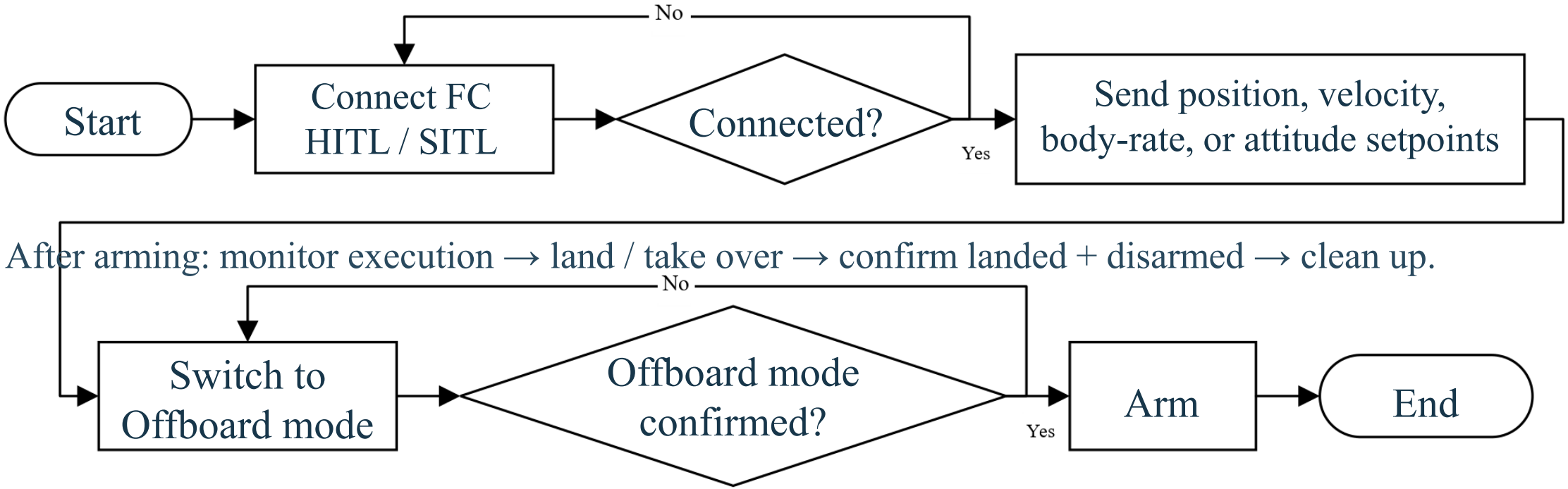


Fig. 6.10 External Control Program Development (HITL: hardware-in-the-loop; SITL: software-in-the-loop; textbook HIL/SIL); FC: flight controller

6.7.3 Minimal Control and State Acquisition: Complete State Workflow

Advance Each Stage Only When Received State Allows It

1 Prepare	2 Enter	3 Execute	4 Finish
Connection, heartbeat Position and health	Valid setpoint stream Mode confirmation, arming	Update reference Feedback, error, timeouts	Land or take over Confirm landing, disarm

Advance using returned state: HEARTBEAT, mode, position, and final state.

Exception path: stale state, lost mode, or excessive error triggers the agreed landing / manual-takeover procedure.

6.7.3 Minimal Control and State Acquisition: Example Entry

Start from a Released Example with a Defined Simulation State

[Installation directory]/RflySimAPIs/6.RflySimExtCtrl/0.ApiExps/e1_PX4MavCtrlAPITest/

File	Purpose	Check Before Use
Readme.pdf	Experiment aims, setup, steps	Simulation conditions match the installed version
PX4MavCtrlAPITest.bat	Start simulation, QGC, and scene	Administrator privileges; close same-name processes
Python38Run.bat	Open the platform Python environment	SDK imports directly
PX4MavCtrlAPITest.py	Entry for control and scene APIs	Distinguish control, display, and cleanup functions

Simulation only: e1 auto-enters via initOffboard; here, send explicitly and verify state.

6.7.3 Minimal Control and State Acquisition: Running the Example and Checking Results

Run e1 in Four Steps: Start, Ready, Run, Exit

1 Start: run the .bat as administrator to open QGC, CopterSim, and two RflySim3D windows.

2 Ready: wait for GPS 3D fixed & EKF initialization finished in the CopterSim log.

3 Run: double-click Python38Run.bat; enter python PX4MavCtrlAPITest.py in that window.

4 Exit: press any key in the batch window to end all same-name CopterSim, QGC, and RflySim3D processes. Startup also ends running CopterSim processes.

Expected: front-camera and global 3D views, a human model, and takeoff. Source target: 1.7 m above the origin; after 10 s, descend at 0.2 m/s.

Troubleshoot: missing module → use Python38Run.bat; no takeoff / arming fails → wait for initialization and check QGC; bad 3D display → check graphics drivers.

6.7.3 Minimal Control and State Acquisition: SDK Side Effects

Initialization, Exit, and Cleanup Functions Also Issue Control Actions

Function	Installed Implementation	Boundary Used Here
InitMavLoop	Starts reception; also sends a disarm command	Simulation starts grounded and disarmed
initOffboard	Starts sending, auto-arms; changes parameters below	Disables loss failsafes; simulation only. Use explicit steps here.
endOffboard	Stops sending; no LAND or parameter restoration	Cannot prove landing is complete
stopRun	Forces disarming if armed	Call only after confirmed landing and disarming

initOffboard: NAV_RCL_ACT = 0; NAV_DLL_ACT = 0; COM_RCL_EXCEPT = 4.

6.7.3 Minimal Control and State Acquisition: SDK Parameters and Restoration

Restore the Loss Parameters Changed by initOffboard Before Real Flight

[Installation directory]/RflySimAPIs/RflySimSDK/ctrl/PX4MavCtrlV4.py (PX4 MAVLink branch)

initOffboard() writes these values; neither endOffboard() nor stopRun() restores them.

Parameter	Written Value and Effect	Before Real-Vehicle Use
NAV_RCL_ACT	0: no action on RC loss	Restore RC-loss action
NAV_DLL_ACT	0: no action on telemetry loss	Restore data-link-loss action
COM_RCL_EXCEPT	4: Offboard ignores RC loss	Restore RC-loss handling

Restoration: record original values first. Before real-vehicle use, restore each or configure it for the vehicle type.

See this section: "SDK Side Effects" and "Termination and Exceptions."

6.7.3 Minimal Control and State Acquisition: Initialization and Message Hooks

One Receive Loop Collects Messages and Distributes Them to Observers

```
import time
import PX4MavCtrlV4 as PX4MavCtrl
mav = PX4MavCtrl.PX4MavCtrler(1)
mav.InitMavLoop(2)
conn = mav.the_connection
conn.message_hooks.append(on_message)
```

Connection check: receive HEARTBEAT from the target controller; local position keeps updating; mode and arming state are readable.

ID = 1:
Ports for vehicle 1.

Mode = 2:
MAVLink_Full; match the CopterSim mode.

Message hook:
Define on_message first, then register it; one receiver.

One receiver parses messages; multiple observers use the same received data.

6.7.3 Minimal Control and State Acquisition: State Acquisition and Freshness

Position, Source Timestamp, and Receipt Time Must Share One Message

```
def on_message(conn, msg):
    src = (msg.get_srcSystem(), msg.get_srcComponent())
    if src != (1, 1): return
    if msg.get_type() == "LOCAL_POSITION_NED":
        sample = (msg.time_boot_ms,
                  msg.x, msg.y, msg.z,
                  time.monotonic())
        save_sample(sample)
```

time_boot_ms: ms since boot; monotonic(): local seconds.
Never subtract across time bases. NED: north/east/down (m).
uavPosNED: estimate; truePosNED: ground truth; check origins.
uavTimeStmp cannot establish position freshness.

Example source: system 1;
autopilot component 1.
Verify actual HEARTBEAT.

Atomic snapshot: lock reads/writes.
On restart / wrap, exit first; then rebuild the time baseline.

Both ages ≤0.2 s:
Since last receipt;
since last sample advance.

6.7.3 Minimal Control and State Acquisition: Position Setpoints and Transmission

Writing a Target Variable Must Be Followed by Sending the Message

```
mav.SendPosNED(0, 0, -1.7, 0)

def publish():
    mav.sendMavOffboardAPI(
        mav.type_mask, mav.coordinate_frame,
        mav.pos, mav.vel, mav.acc,
        mav.yaw, mav.yawrate)
```

Position:
North 0 m, east 0 m, down −1.7 m.

Yaw:
0 rad; not a full three-axis attitude.

Mask:
Use position and yaw; ignore other fields.

Setting a reference, encoding/sending it, and executing it on the controller are three separately testable steps.

6.7.3 Minimal Control and State Acquisition: Entering Offboard Control

Establish the Setpoint Stream Before Requesting Mode and Arming

```
mav.SendPosNED(*position_now(), yaw_now())
pump(mav, 1.5)  # 1.5 s, 30 Hz
mav.SendSetMode(6)
wait_for(mav, watch.offboard, 5)  # 5 s
mav.SendMavArm(True)
wait_for(mav, watch.armed, 5)  # 5 s
mav.SendPosNED(0, 0, -1.7, 0)
```

v1.12.3: gap >1 s (COM_OF_LOSS_T) means stream loss. After arming, reset the takeoff target; judge arrival by error. Other versions: Section 5.5, “Control Outputs Across Versions.”

Helpers, not SDK functions:
Keep sending at 30 Hz while waiting;
see “Reference Implementation.”

Mode value 6:
PX4 Offboard main mode;
confirm via HEARTBEAT.

SDK flag:
isInOffboard controls sending;
it is not the actual flight mode.

6.7.3 Minimal Control and State Acquisition: Minimal Closed-Loop Acceptance

Arrival Requires Both Measured Error and Hold Time

$$e = \|\mathbf{p}_{\text{ref}} - \mathbf{p}_{\text{meas}}\|_2$$

Reference $\mathbf{p}_{\text{ref}} = (0,0,-1.7)$ m; tabulated feedback $\mathbf{p}_{\text{meas}}$; same local NED origin.
e: 3D point-to-point distance (m).

Measured Position / m	Position Error / m	Arrival Confirmed?
(0, 0, 0)	1.70	No; still at the start
(0.15, −0.20, −1.70)	0.25	No; exceeds 0.20 m
(0.06, −0.10, −1.68)	0.12	Instantaneously within limit; still hold for 2 s

Teaching criterion: error ≤0.20 m continuously for 2 s, with fresh samples and mode still Offboard.

6.7.3 Minimal Control and State Acquisition: Termination and Exceptions

Confirm Landing and Disarming Before Cleaning Up Communication

```
mav.SendSetMode(4, 6)
wait_for(mav, watch.auto_land, 5)
wait_for(mav, watch.on_ground, 60, False)
wait_for(mav, watch.disarmed, 15, False)
mav.stopRun()
```

endOffboard stops sending; it does not send LAND.
stopRun forces disarming if armed.
Final state: landed status returned + HEARTBEAT armed flag cleared.

PX4 mode enums:
Main mode AUTO=4;
submode LAND=6.

False argument:
Stop setpoints after LAND is confirmed;
keep receiving state.

Failure path:
On timeout, take over; never put stopRun in unconditional finally.

6.7.3 Minimal Control and State Acquisition: Reference Implementation (State Storage)

Store Receipt Time and Last Sample Advance Time Under One Lock

```
import threading, time
from pymavlink import mavutil
lock, latest = threading.Lock(), {}
FRESH = dict(POS=.2,ATT=.2,HB=3,EXT=3)
def put(key, item, t, src=None):
    with lock:
        old = latest.get(key)
        new = (src is None or old is None
               or src > old[3])
        if new:
            latest[key] = (item, t, t, src)
        else:
            latest[key] = (old[0], t,
                           old[2], old[3])
def save_sample(s):
    put("POS", s, s[4], s[0])
NAME = {"ATTITUDE":"ATT",
        "HEARTBEAT":"HB",
        "EXTENDED_SYS_STATE":"EXT"}
def on_message(conn, m):
    ids = (m.get_srcSystem(),
           m.get_srcComponent())
    if ids != (1, 1): return
    t, k = time.monotonic(), m.get_type()
    if k == "LOCAL_POSITION_NED":
        save_sample((m.time_boot_ms,
                     m.x, m.y, m.z, t))
    elif k in NAME: put(NAME[k], m, t)
```

6.7.3 Minimal Control and State Acquisition: Reference Implementation (Read, Send, Wait)

Read Fresh State by Threshold and Keep Sending While Waiting

```
def get(key, must=False):  # Fresh content; stale: None, or raise if must=True
    with lock: item, now = latest.get(key), time.monotonic()
    if item and max(now - item[1], now - item[2]) <= FRESH[key]: return item[0]
    if must: raise TimeoutError(key + " stale")
# publish(): current target via mav.sendMavOffboardAPI (“Position Setpoints and Transmission”).
def pump(mav, secs):  # Call publish() at about 30 Hz for secs seconds
    end = time.monotonic() + secs
    while time.monotonic() < end: publish(); time.sleep(1 / 30)
def wait_for(mav, pred, timeout, keep_sending=True):  # Return only when pred() is true
    end = time.monotonic() + timeout
    while not pred():
        if time.monotonic() > end: raise TimeoutError(pred.__name__ + " timeout")
        if keep_sending: publish()
        time.sleep(1 / 30)
```

6.7.3 Minimal Control and State Acquisition: Reference Implementation (Pose and Predicates)

Predicates Read Only Returned Mode, Armed Flag, and Landed State

```
def position_now(): return get("POS", True)[1:4]  # Fresh position (x, y, z) / m
def yaw_now(): return get("ATT", True).yaw  # Fresh yaw / rad
ARM = mavutil.mavlink.MAV_MODE_FLAG_SAFETY_ARMED  # Armed bit in base_mode
def field(key, name):  # Fresh message field; None if missing or stale
    v = get(key); return None if v is None else getattr(v, name)
def px4_mode():  # custom_mode: main bits 16–23; submode bits 24–31 (PX4 v1.12.3)
    cm = field("HB", "custom_mode") or 0; return (cm >> 16) & 0xFF, (cm >> 24) & 0xFF
class watch:  # Predicates use returned state only; stale messages yield False
    def offboard(): return px4_mode()[0] == 6  # Main mode OFFBOARD
    def auto_land(): return px4_mode() == (4, 6)  # Main AUTO, submode LAND
    def armed(): bm = field("HB", "base_mode"); return bm is not None and bool(bm & ARM)
    def disarmed(): bm = field("HB", "base_mode"); return bm is not None and not bm & ARM
    def on_ground(): return field("EXT", "landed_state") == 1  # ON_GROUND=1
```

Order: import → define → connect/register → hold target → enter → validate → land → clean up

Chapter Outline

Platform Architecture and Communication Interfaces

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

6.7.1 Platform Architecture and Responsibilities
Textbook: Sec. 6.2.1

6.7.2 Communication Links and Protocols
Textbook: Secs. 6.1.1.5 and 6.2.2.1

6.7.3 Minimal Control and State Acquisition
Textbook: Secs. 6.2.2.2 and 6.2.2.4 (control flow)

6.7.4 Extended Interfaces and Algorithm Tools
Textbook: Secs. 6.2.2.2–6.2.2.3, 6.2.2.5, and 6.2.3

6.7.5 Communication Optimization and Acceptance
Textbook: Secs. 6.2.2.4 (send rate) and 6.2.3

6.7.4 Extended Interfaces and Algorithm Tools: Development Interface Comparison

Programming Interfaces Differ; Closed-Loop Acceptance Stays the Same

Interface Route	Programming Abstraction	Development Needs
pymavlink / RflySim SDK	Message fields / common control calls	Python teaching, protocol debugging
MAVSDK	Action / Telemetry Offboard control	Mission applications, multilingual clients
MAVROS	ROS topics and services ↔ MAVLink	Existing ROS nodes and perception
uXRCE-DDS / ROS 2 (v1.14 and later)	Selected PX4 uORB message-bus topics ↔ ROS 2	Direct use of PX4 internal message types

Main: Python/RflySim SDK. Alternatives for your setup: MAVSDK, MAVROS, ROS 2.

Checks: valid state/setpoints; confirmed mode; measured motion error; explicit final state.

6.7.4 Extended Interfaces and Algorithm Tools: MAVSDK Example

MAVSDK Uses Async APIs for Setpoints and Telemetry

```
from mavsdk.offboard import PositionNedYaw
await drone.offboard.set_position_ned(
    PositionNedYaw(0, 0, 0, 0))
await drone.offboard.start()
await drone.offboard.set_position_ned(
    PositionNedYaw(5, 0, -5, 90))
```

Directory: 6.RflySimExtCtrl/0.ApiExps/e17_MAVSDKExps/
Entry: 2.OffboardPosNEDExp/offboard_position_ned.py

Target semantics:
North 5 m, east 0 m,
5 m up; east-facing yaw 90°.

Feedback:
Telemetry position and mode;
check in_air and armed.

Add termination steps:
Request action.land();
confirm landing and disarming.

Prerequisites: connected, healthy positioning, and action.arm() complete.
Call within an async function; the original ends with offboard.stop() only.

6.7.4 Extended Interfaces and Algorithm Tools: ROS Integration and Coordinates

MAVROS Positions Usually Use East-North-Up (ENU)

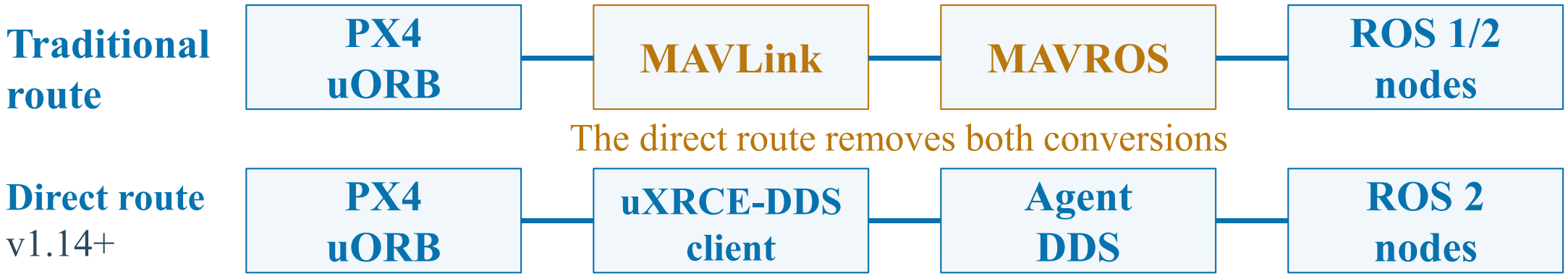
Operation	ROS 1 MAVROS Interface	Checks
Continuous setpoints	/mavros/setpoint_position/local	PoseStamped; ENU coordinates
Position feedback	/mavros/local_position/pose	Time, origin, actual position
Mode and arming	/mavros/state; service requests	After service response, check returned state

$$(x, y, z)_{\mathrm{ENU}} = (3,2,4)\ \mathrm{m} \Rightarrow (x, y, z)_{\mathrm{NED}} = (2,3,-4)\ \mathrm{m}$$

Directory: 6.RflySimExtCtrl/0.ApiExps/e18_MavrosExps/
Entry: mavros_offboard_posctl_test.py; check the ROS environment and Readme.pdf first.

6.7.4 Extended Interfaces and Algorithm Tools: The uXRCE-DDS Route

Direct Route: MAVLink and MAVROS Are Removed; Client and Agent Remain



Match firmware, px4_msgs definitions, dds_topics configuration, namespaces, and QoS.

Extensions: synchronize exported uORB definitions. MAVLink also supports custom messages with matching definitions at both ends.

PX4 v1.14+ required; version check: Section 5.5, "Control Outputs Across Versions." Docs: https://docs.px4.io/v1.14/en/middleware/uxrce_dds

Directory: 6.RflySimExtCtrl/0.ApiExps/e19_uXRCE-DDS_ROS2CtrlExps/

6.7.4 Extended Interfaces and Algorithm Tools: Common Interfaces and Extensions

Choose Python, ROS, Models, or Native Code by Algorithm Location

Location	Development Route	Interface and Checks
External script	Python: PX4MavCtrlV4.py	UDP / serial; signatures and timing
ROS node	ROS: PX4MavCtrlV4ROS.py	Via MAVROS; requires Linux and ROS
External model	Simulink: OffboardAPI.slx	UAV Toolbox blocks; input/output units
Native application	C++ / MAVLink or MAVSDK	Message definitions, threads, timeouts, logging
Integrated-model DLL	Controller and motion model run together	inSIL external input; model boundary and ground truth

Entry: 6.RflySimExtCtrl/1.BasicExps/e4_OffboardCtrl/OffboardAPI.slx
Libraries: RflySimSDK/ctrl/; fields: this chapter's API.pdf and RflySimSDK/html/index.html.

6.7.4 Extended Interfaces and Algorithm Tools: Toolbox Overview

Algorithm Libraries Provide Models, Guidance, and Trajectories

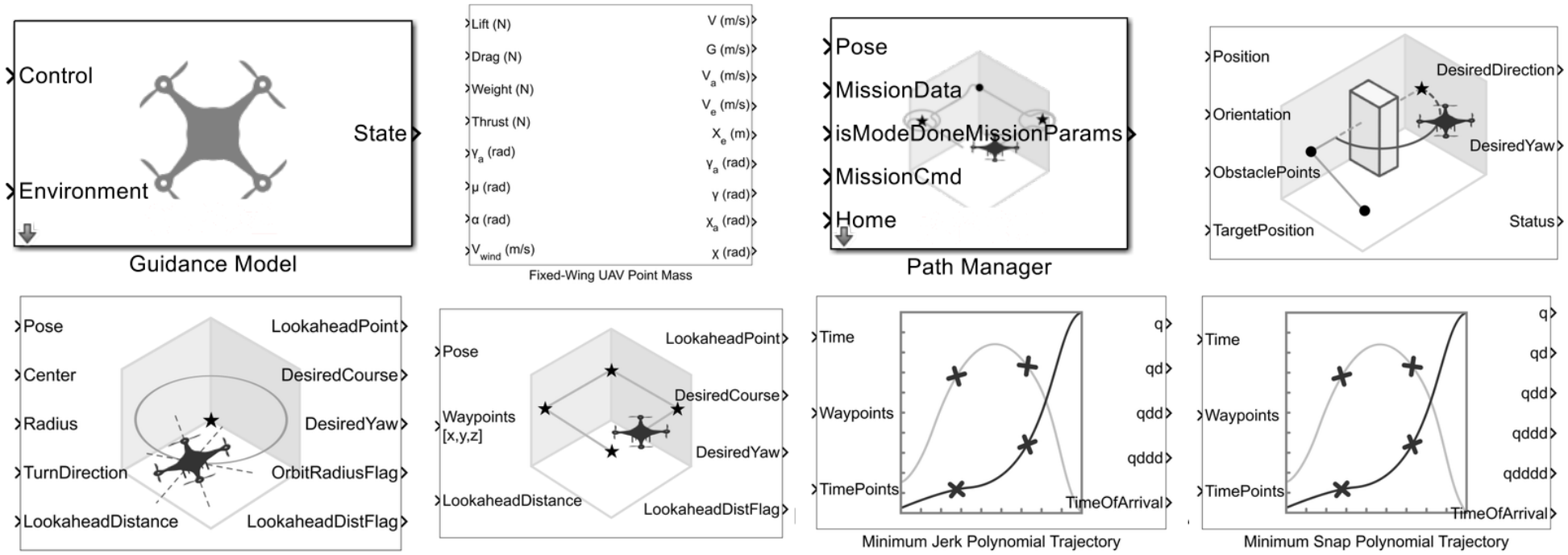


Fig. 6.11 Simulink UAV Algorithm Library

Blocks provide models, guidance, or trajectory references; design the controller separately.

6.7.4 Extended Interfaces and Algorithm Tools: Models and Guidance

Guidance Models and Path Followers Output Different Quantities

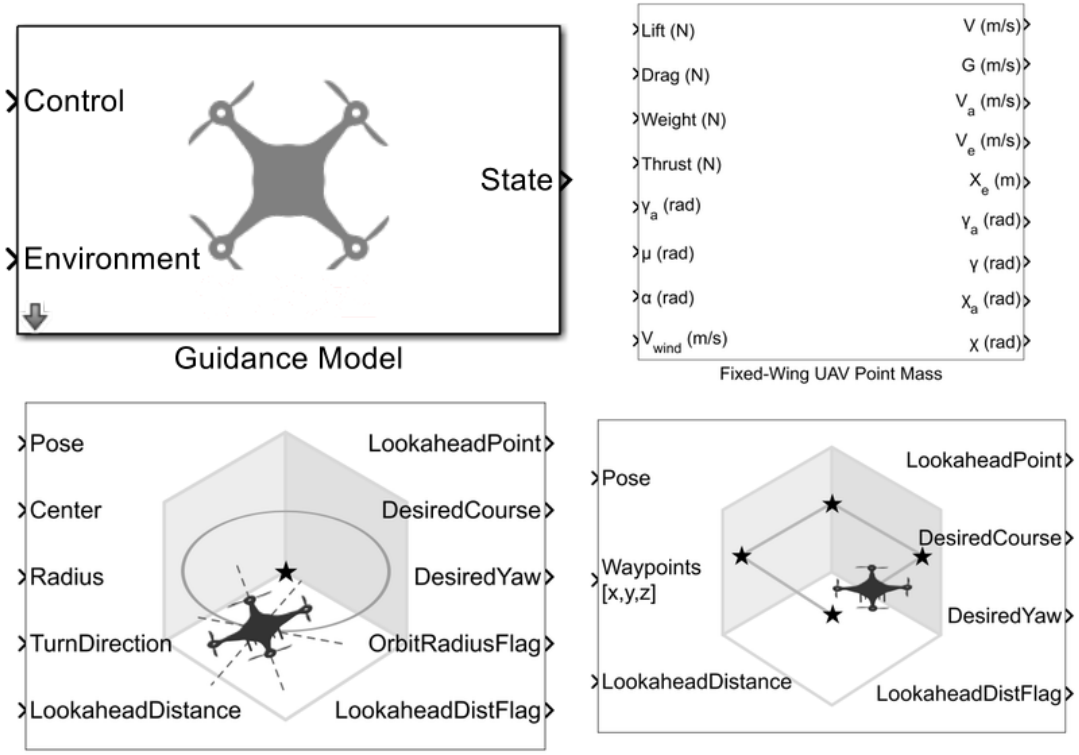


Fig. 6.11 (Left Half) Models and Path-Following Blocks

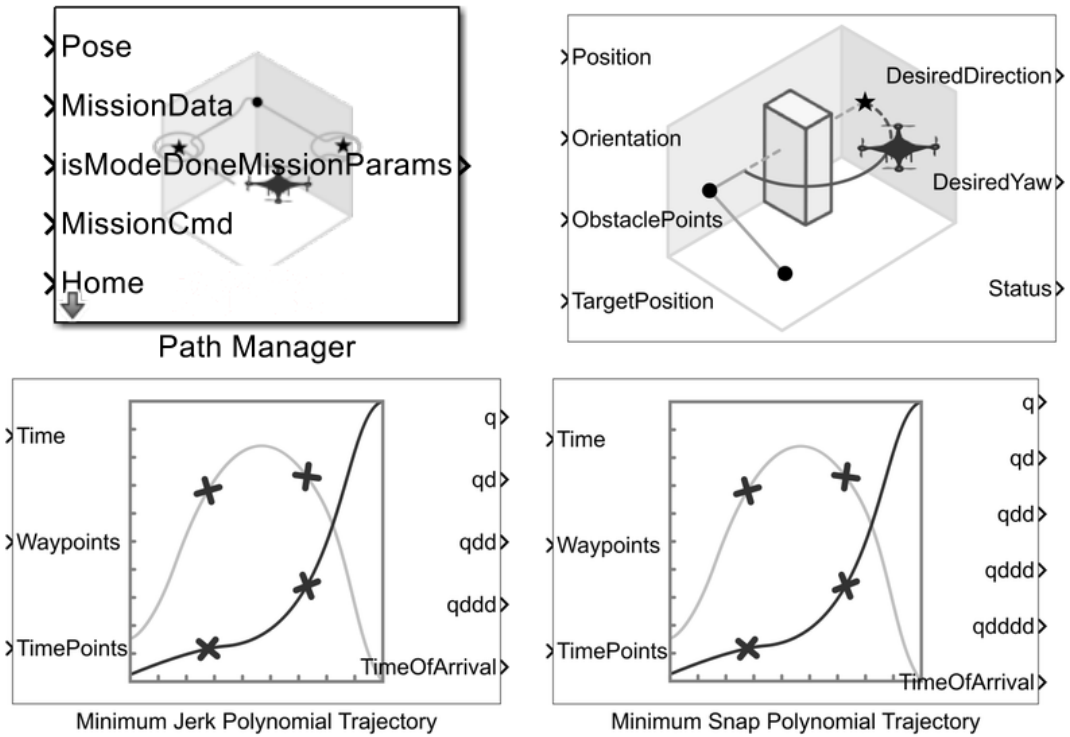
Guidance model: reduced-order kinematics and autopilot response → state.

Fixed-wing point mass: forces and motion parameters → approximate motion.

Waypoint / orbit follower: pose and path → lookahead point and heading references.

6.7.4 Extended Interfaces and Algorithm Tools: Missions and References

Mission and Trajectory Blocks Still Need Downstream Control



Path management: mission sequence and pose → mission parameters.

Avoidance: 3D VFH+ (Vector Field Histogram) gives a direction; check motion constraints.

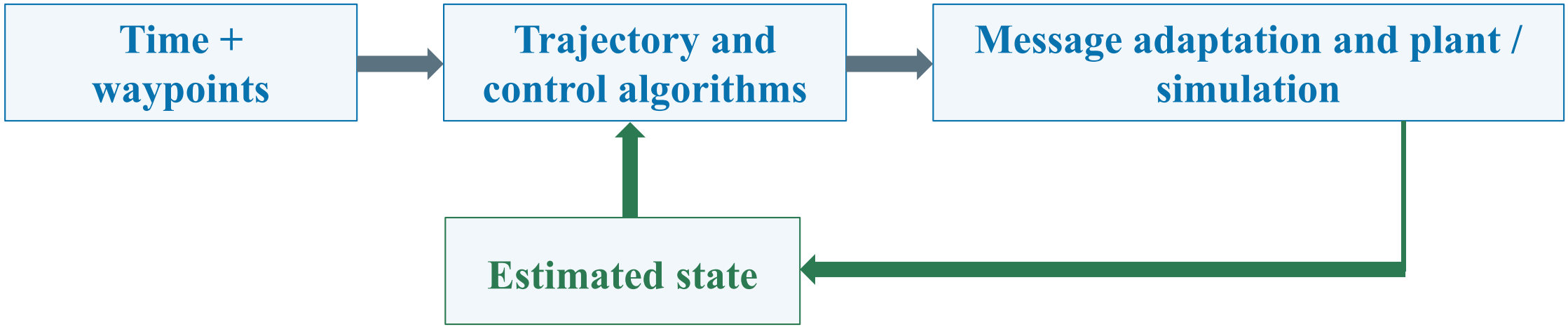
Polynomials: waypoints and time → position, velocity, acceleration (PVA). q, qd, qdd respectively; q is not a quaternion.

Higher derivatives: Jerk also outputs qddd; Snap additionally outputs qddddd.

Fig. 6.11 (Right Half) Mission, Avoidance, and Trajectory Blocks
Top: Path Manager / Obstacle Avoidance; bottom: Minimum Jerk / Minimum Snap.

6.7.4 Extended Interfaces and Algorithm Tools: Algorithm Assembly

Define Reference, Feedback, and Output Interfaces for the Simulink Loop



Interfaces: define frames, units, input masks, sample periods, and timeout handling.

Log reference PVA, feedback state, actual commands, timestamps, and flight mode.

Tools and deployment: UAV, Automated Driving, Reinforcement Learning, and Model Predictive Control (MPC) toolboxes via ROS/UDP; code generation where supported.

Chapter Outline

Platform Architecture and Communication Interfaces

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

6.7.1 Platform Architecture and Responsibilities
Textbook: Sec. 6.2.1

6.7.2 Communication Links and Protocols
Textbook: Secs. 6.1.1.5 and 6.2.2.1

6.7.3 Minimal Control and State Acquisition
Textbook: Secs. 6.2.2.2 and 6.2.2.4 (control flow)

6.7.4 Extended Interfaces and Algorithm Tools
Textbook: Secs. 6.2.2.2–6.2.2.3, 6.2.2.5, and 6.2.3

6.7.5 Communication Optimization and Acceptance
Textbook: Secs. 6.2.2.4 (send rate) and 6.2.3

6.7.5 Communication Optimization and Acceptance: Communication Rates

Reference, Send, Receive, and Inner-Loop Rates Are Different

Reference updates
Illustration: 10 Hz
Determines algorithm target-update times
Log: reference version and time

Setpoint sending
Here: 30 Hz
Determines command-stream continuity
Log: send times and longest gap

State reception
Illustration: 50 Hz
Determines whether feedback updates
Log: receipt times by message type

Inner control loop
Illustration: 400 Hz
Determines low-level control timing
Log: controller settings and internal logs

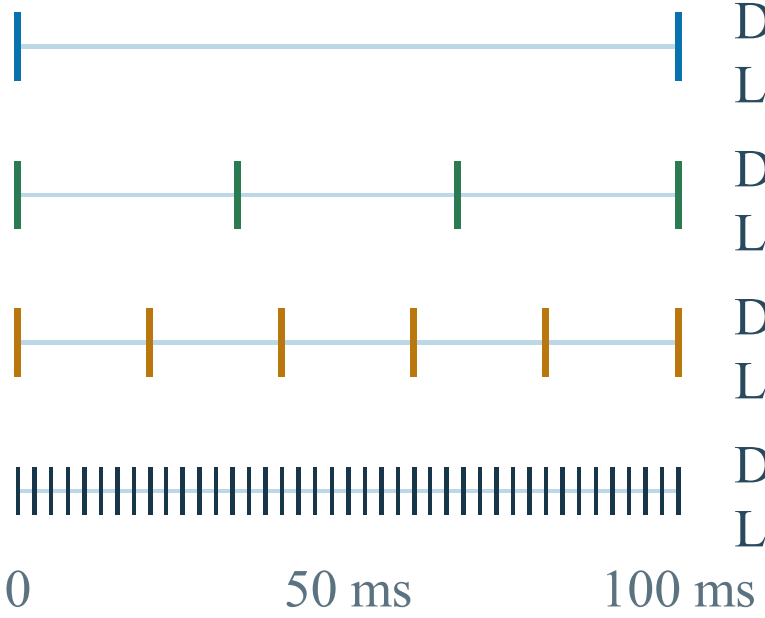


Figure: event times for four processes over 0–100 ms (illustrative, not platform measurements)

Current SDK source: Full send interval 0.01 s; Simple interval 1/30 s.
Explicit sending here uses 30 Hz; log actual intervals and jitter.

6.7.5 Communication Optimization and Acceptance: Data-Age Example

Many Received Frames Can Still Be Repeated Old Samples

Sample Time / ms	Local Receipt Time / s	Meaning
10000	10.000	First position sample
10050	10.050	New position sample
10050	10.100	Duplicate; receive rate ≠ update rate

At local time 10.350 s: receipt age = evaluation time − last frame's local receipt time.

$$A_{\text{recv}} = 10.350 - 10.100 = 0.250\ \text{s}$$

Time since sample advance = 10.350−10.050 = 0.300 s, using local receipt time of the last new sample.

Teaching threshold: 200 ms. Both checks fail; enter exception handling.

6.7.5 Communication Optimization and Acceptance: Duplicate Samples and Freshness

Duplicates Reset Receipt Age While Time Since Sample Advance Grows

Frame 3 repeats; stall exceeds the limit 50 ms earlier

	Sample (ms)	Received (s)	
1	10000	10.000	
2	10050	10.050	
3	10050	10.100	Repeat

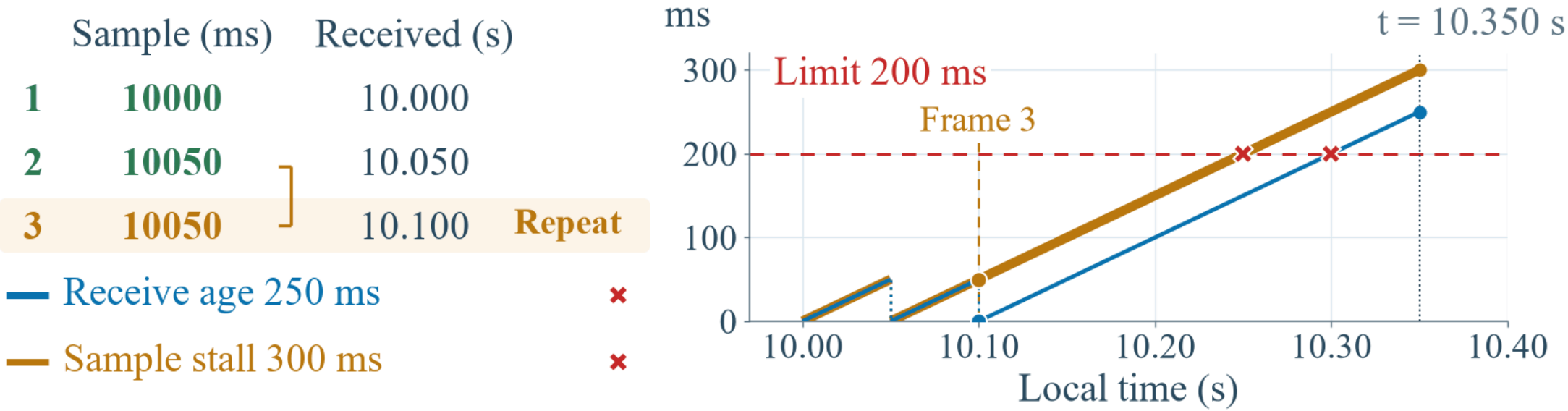


Figure: source/receipt times and both age checks (offline example, not platform measurements)

Ages use local receipt times. Receipt age: since last frame; duplicates reset it.

Time since sample advance: since the last new sample; use controller times only for order.

6.7.5 Communication Optimization and Acceptance: Bandwidth Budget Example

Calculate Link Utilization Before Raising the Send Rate

Teaching assumptions: serial 115200 baud, 8N1; 80 B per frame including protocol overhead.
8N1: 8 data bits, no parity, 1 stop bit; plus 1 start bit → 10 bits per byte.

$$\text{One-way capacity } C = \frac{115200}{10} = 11520 \text{ B/s}, \quad R = 80f$$

Send rate f	One-way data rate R	Capacity Used
30 Hz	2400 B/s	20.8%
100 Hz	8000 B/s	69.4%

Headroom: budget for heartbeats, telemetry, parameters, and traffic bursts.

Optimize: reduce irrelevant messages or use a compatible reduced format; retain the setpoint stream and recheck errors and gaps.

6.7.5 Communication Optimization and Acceptance: Modes and Data Granularity

Simulation Mode and Data Format Are Independent Choices

Simulation Mode	What Runs in the Loop	Main Checks
PX4_HITL	Real flight controller + motion model	Hardware links, firmware, simulated sensors
PX4_SITL	Flight-controller software + motion model	Processes, ports, message directions
PX4_SITL_RFLY	Platform-managed SITL connections	Dedicated port mapping, multi-instance setup
Simulink / DLL_SIL	Controller + motion-model DLL	Model interfaces, parameters, ground truth

Format choices: MAVLink / platform UDP; Full / Simple; sensor-data settings.
Modes depend on the installed version; check current software and interface definitions.

6.7.5 Communication Optimization and Acceptance: Connection and Control Troubleshooting

Diagnose Each Layer with Evidence Closest to the Fault

Symptom	First Evidence to Check	Action
No heartbeat	IP, port, format, source ID	Fix connection settings
Heartbeat, but no new position	Position timestamps and receipt age	Check message stream and estimator
Mode request rejected	COMMAND_ACK (command acknowledgment), status text	Fix setpoint stream and entry conditions
Motion, but abnormal error	Reference/feedback frames, units, timing	Check origin, signs, and motion

Diagnostic log: SDK and firmware versions, connection settings, sample times, command acknowledgments, errors, and final state.

Classroom Exercise

Check Connection, Coordinates, and Time with One Condition Each

Question	Reference Answer	Validation Evidence
Which ports for ID=5?	Command 20108; state 20109	20100+2×(5−1); identify sender/receiver
ENU (−1, 4, 2.5) → NED?	(4, −1, −2.5), in m	Changing signs only gives (−1, 4, −2.5)
New sample 10.2 s; duplicate 10.3 s: At 10.45 s, which age exceeds 0.2 s?	Receipt age 0.15 s: within limit. Time since sample advance 0.25 s: over limit.	Calculate separately, using local times for both.

Three shared requirements: clear vehicle identity, clear coordinates, and evidence from actual state messages.

Classroom Exercise

The Program Says “End”: Which Final-State Evidence Is Missing?

```
send_target()
time.sleep(10)
mav.endOffboard()
# finally: mav.stopRun()
```

The left-hand sequence is incomplete.
endOffboard sends no LAND.
stopRun forces disarming if armed.

Arrival evidence: measured/reference position error remains within the threshold for a hold interval.

Recovery evidence: an explicit landing / takeover request, plus returned mode and landed state.

Cleanup condition: landed and disarmed; if state is unknown, enter takeover.

Section Summary

Connected Interfaces Need a Trajectory Generator for Continuous Setpoints

Responsibilities: external program → setpoints; flight controller → low-level control; model → response; 3D program → display/images.

Layers: links → bytes; protocols → meaning; SDKs → calls. Set ports and coordinates.

Closed loop: valid stream → confirm mode/arming → check error and final state using actual state.

MAVSDK/MAVROS/ROS 2: syntax differs; all require state and final-state checks.

Optimization: check generation/send/receipt times separately; recheck control after reducing load.

Errors: targets as feedback; endOffboard as landing; uavTimeStmp as position freshness; ENU→NED by sign alone, without axis swap; reversed port directions.

Simulation only: restore failsafes disabled by initOffboard before real-vehicle use.

Interfaces transmit targets; trajectory planning defines the target at each time.