Chapter Outline

Fault Injection Interfaces and Algorithm Development Framework

Chapter 7 Health Management and Safety Assessment

7.1 Health Management Fundamentals

7.2 Unified Fault Modeling

7.3 Safety Reinforcement and Fault Data

7.4 Fault Hazards and Safety Assessment

7.5 Diagnosis and Prognostics

7.6 Safety Decisions and Improvement

7.7 Fault Injection Interfaces

7.8 Experiments and Validation

7.9 Exercises and Classroom Discussion

5 Units in This Section

7.7.1 Fault Injection Layers and Framework

Textbook: Sec. 7.2.1

7.7.2 Assessment and Diagnosis Development Framework

Textbook: Sec. 7.2.2

7.7.3 Fault IDs and Parameter Interfaces

Textbook: Sec. 7.2.1, 7.2.3

7.7.4 Injection, Recovery, and Reset Rules

Textbook: Sec. 7.1.5.2, 7.2.3

7.7.5 Observation Interfaces and Effect Verification

Textbook: Sec. 7.1.2.3, 7.1.5.1, 7.2.3

7.7 Fault Injection Interfaces

7.7.1 Fault Injection Layers and Framework: Overview

Every Fault Acts at a Specific Layer

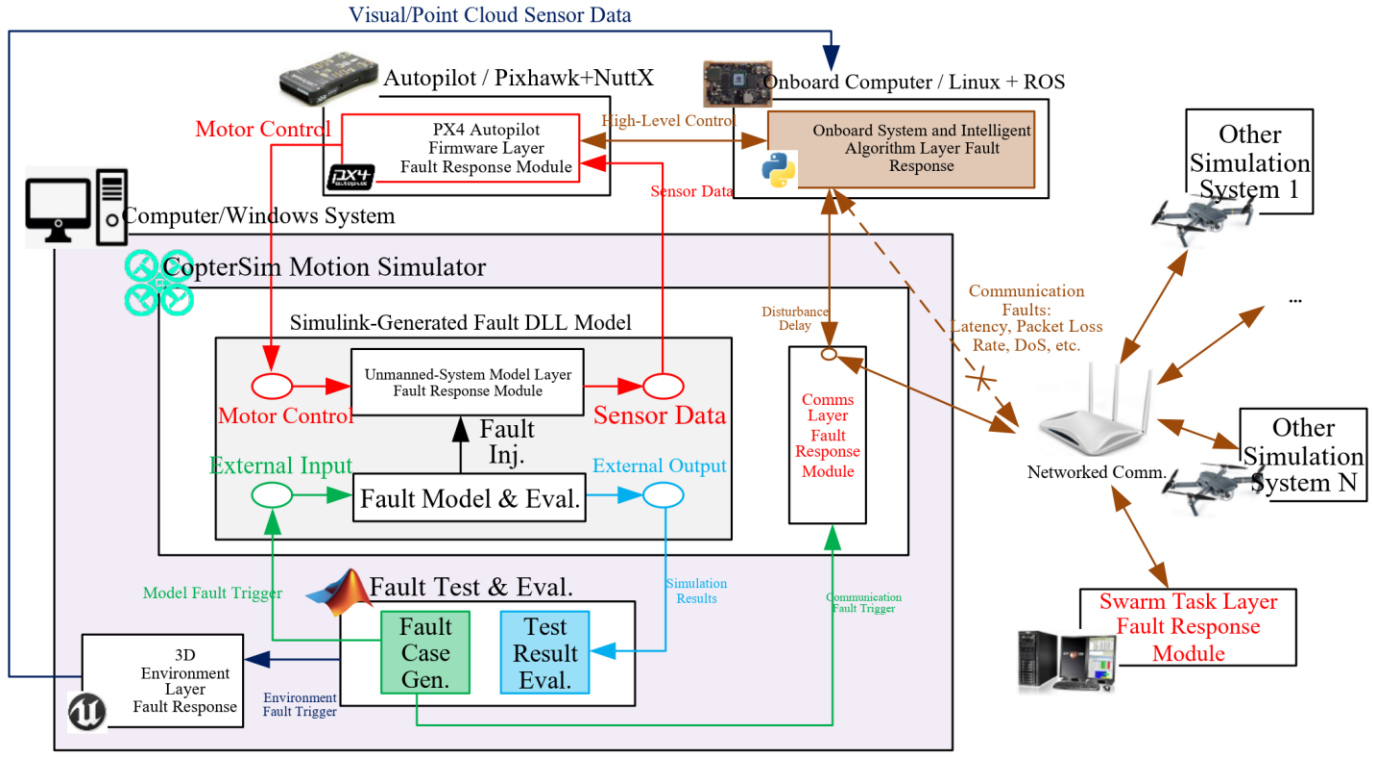


Fig. 7.13 Fault Modeling and Injection Framework

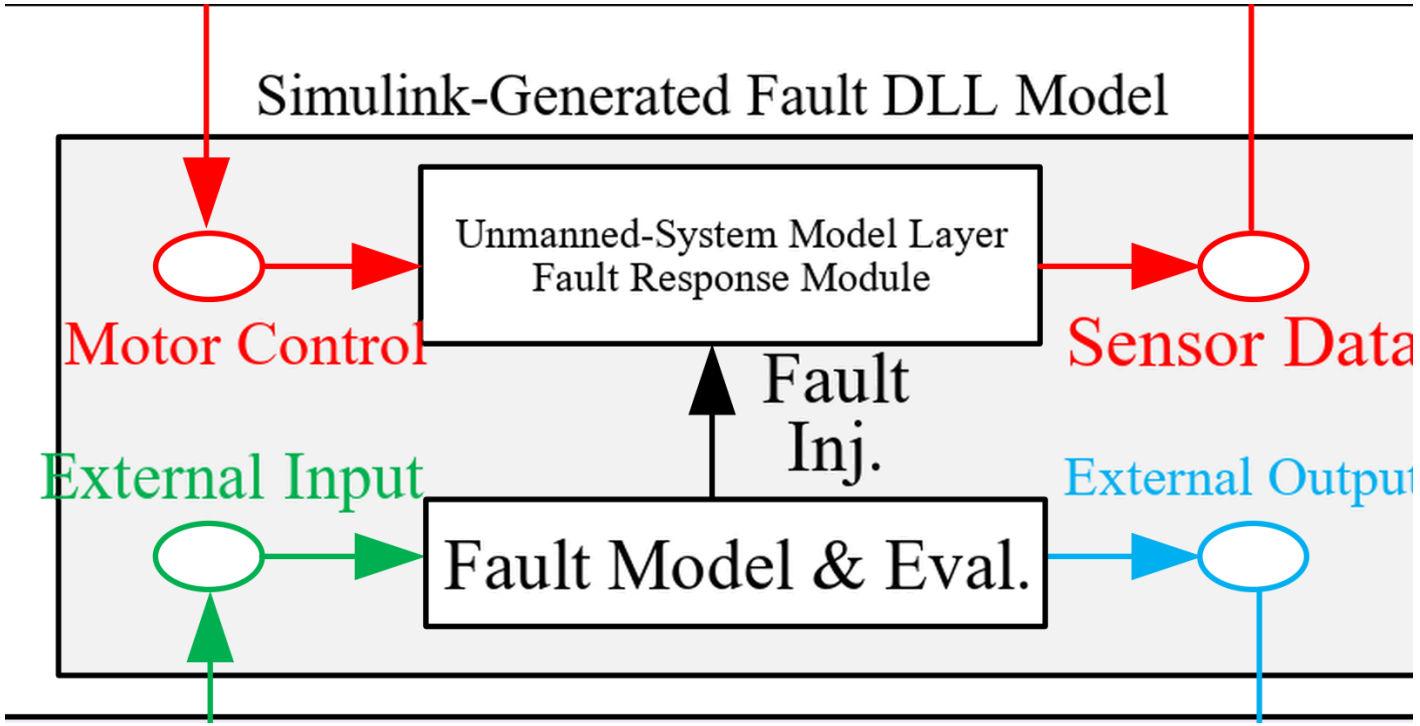
Model: change propulsion or simulated measurements. DLL = dynamic-link library.

Firmware: change drivers or control logic.

High-level/external: environment, communication, algorithms, coordination. ROS: Robot Operating System; DoS: denial of service.

7.7.1 Fault Injection Layers and Framework: Model Layer

Model Triggers Are Separate from Flight-Control Channels



Red: control outputs and simulated sensor data.

Green/black: parameters enter the fault module, then act on the model.

Cyan: simulation results for checking.

Fig. 7.13 (model layer): fault triggering and control feedback.
SITL: software-in-the-loop (textbook SIL); HITL: hardware-in-the-loop (textbook HIL).

7.7 Fault Injection Interfaces

7.7.1 Fault Injection Layers and Framework: Firmware and Algorithms

Firmware and Intelligent Algorithms Affect Different Computations



Fig. 7.13: model, flight controller, and onboard computing

Firmware: alter drivers, estimation, or control inside the flight controller. NuttX is its real-time operating system (Section 5.5).
Intelligent algorithms: corrupt onboard recognition, positioning, or decisions; high-level control carries the effect to the flight controller. ROS organizes onboard communication (Section 6.1).
Data: onboard algorithms also receive vision/point-cloud data from the 3D environment.

7.7.1 Fault Injection Layers and Framework: Environment Layer

Match Scene Disturbances to Their Dynamic Effects

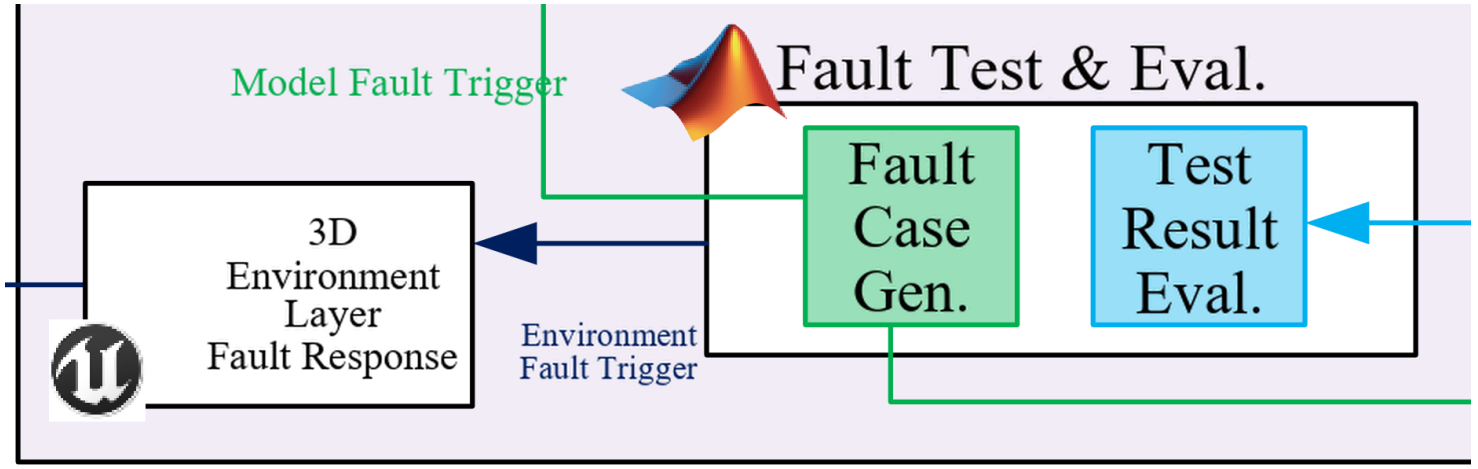


Fig. 7.13 (environment and cases): scene and model triggers

Visual change: the 3D scene changes camera or point-cloud observations.

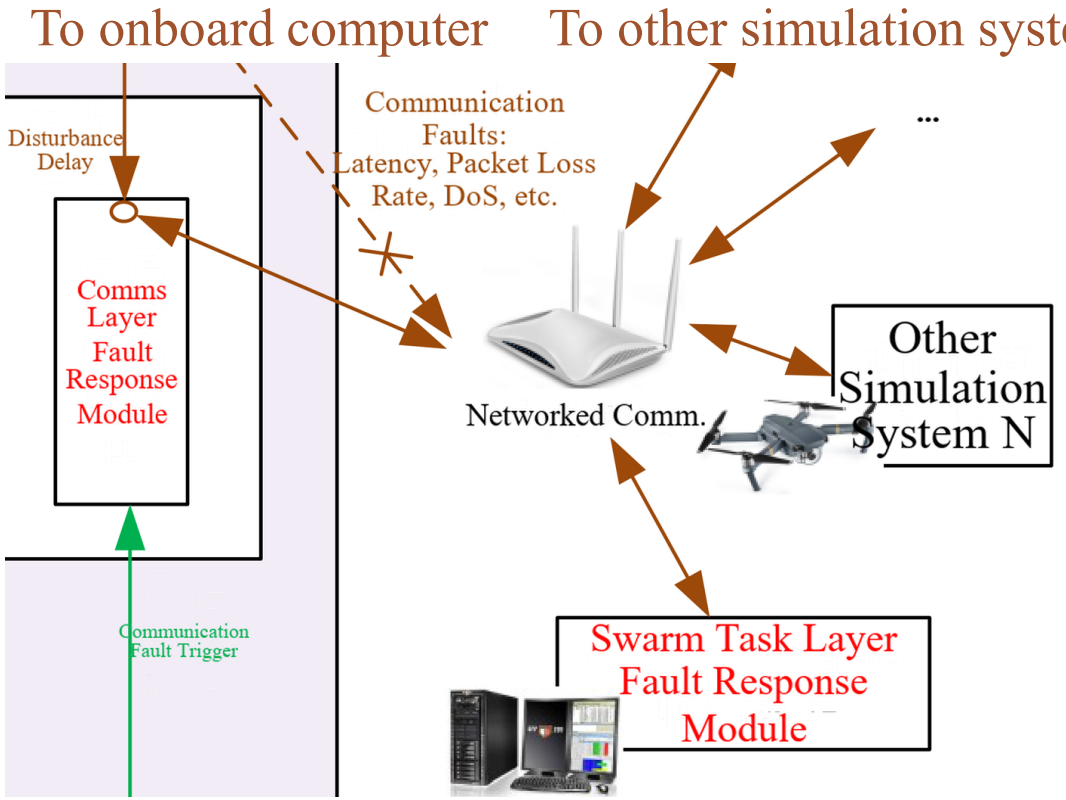
Dynamic change: disturbance inputs change modeled forces and motion.

Interpretation: check parameters and timing in both channels for the same case.

7.7 Fault Injection Interfaces

7.7.1 Fault Injection Layers and Framework: Communication and Swarm Tasks

Communication and Swarm Faults Affect Messages and Coordination



Communication: add delay, packet loss, or interruption. Denial of service (DoS) affects communication availability.

Check communication: send/receive times, sequence numbers, and losses.

Swarm tasks: observe allocation, vehicle states, and execution outcomes.

Scope: motor parameter arrays do not control these two layers.

Fig. 7.13 (right) Communication and Swarm-Task Layers

Chapter Outline

Fault Injection Interfaces and Algorithm Development Framework

Chapter 7 Health Management and Safety Assessment

7.1 Health Management Fundamentals

7.2 Unified Fault Modeling

7.3 Safety Reinforcement and Fault Data

7.4 Fault Hazards and Safety Assessment

7.5 Diagnosis and Prognostics

7.6 Safety Decisions and Improvement

7.7 Fault Injection Interfaces

7.8 Experiments and Validation

7.9 Exercises and Classroom Discussion

5 Units in This Section

7.7.1 Fault Injection Layers and Framework

Textbook: Sec. 7.2.1

7.7.2 Assessment and Diagnosis Development Framework

Textbook: Sec. 7.2.2

7.7.3 Fault IDs and Parameter Interfaces

Textbook: Sec. 7.2.1, 7.2.3

7.7.4 Injection, Recovery, and Reset Rules

Textbook: Sec. 7.1.5.2, 7.2.3

7.7.5 Observation Interfaces and Effect Verification

Textbook: Sec. 7.1.2.3, 7.1.5.1, 7.2.3

7.7.2 Assessment and Diagnosis Development Framework: Overview

Diagnosis Is Part of the Testing and Assessment Data Chain

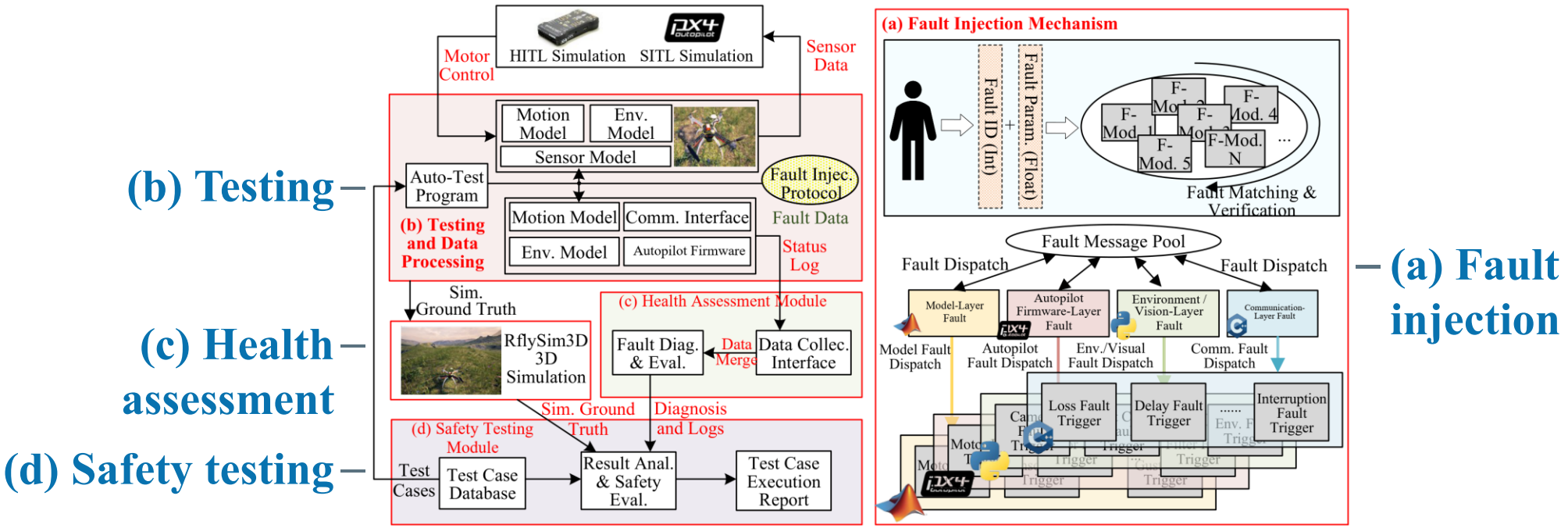


Fig. 7.14 Assessment and Diagnosis Algorithm Development Framework

Set fault conditions; diagnose from logs; assess with ground truth and mission responses.

7.7.2 Assessment and Diagnosis Development Framework: IDs and Labels

Match Fault Identifiers, Parameters, and Target Layers

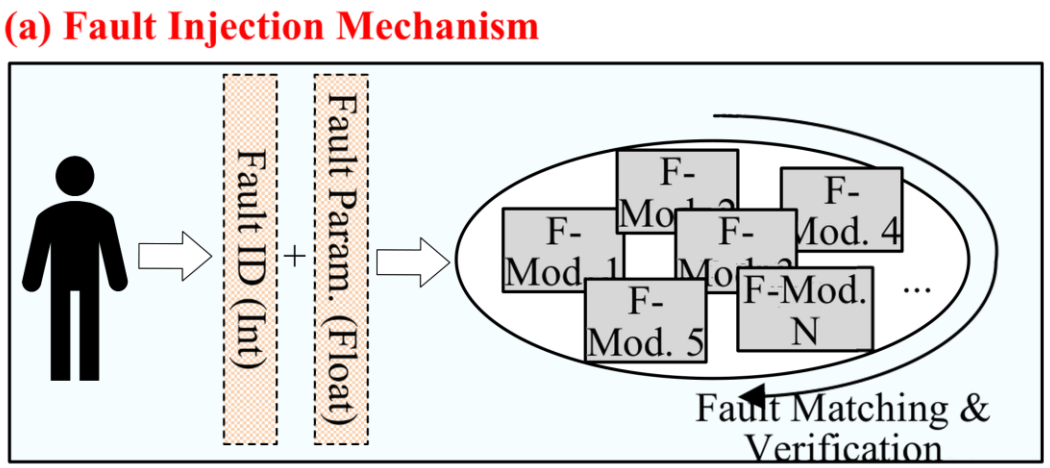


Fig. 7.14(a), top: matching IDs and parameters

Fault ID (integer) + parameters (floats): match the model, then validate parameters.

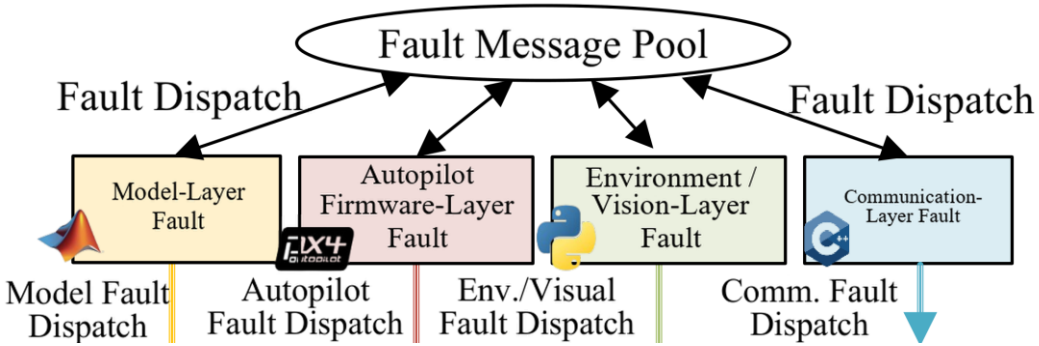


Fig. 7.14(a), middle: target-layer selection

Separate channels: model, firmware; environment/vision; communication. Include the layer in labels.

Record class, target, parameters, effective time, and window.

ID format: Unit 7.7.3, External Injection Interface.

7.7.2 Assessment and Diagnosis Development Framework: Testing and Data

Retain Inputs, Responses, and Timing in the Same Test Record

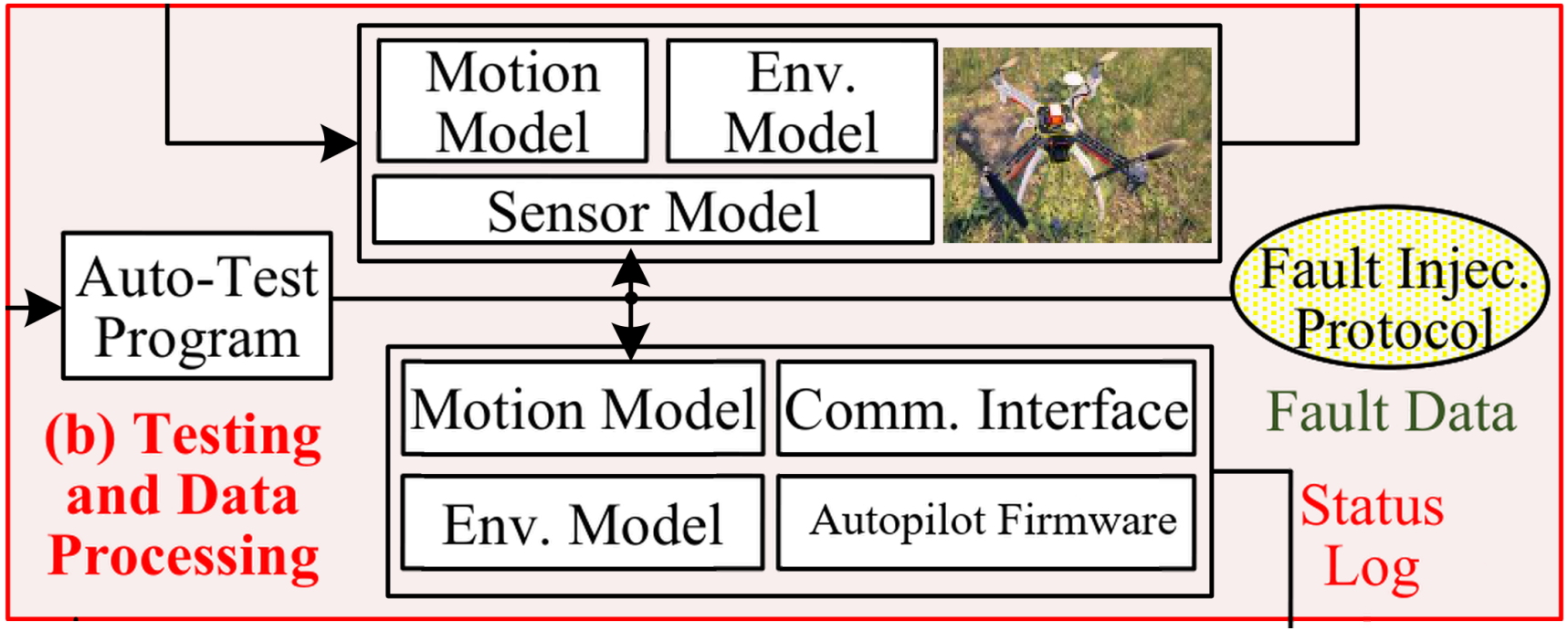


Fig. 7.14(b) Test Scheduling and Data Acquisition

Upper: 3D models; lower: in-the-loop models with autopilot firmware. Timestamp all logs.

7.7 Fault Injection Interfaces

7.7.2 Assessment and Diagnosis Development Framework: Diagnosis and Safety

Diagnosis Reads Logs; Assessment Also Uses Truth and Mission Response

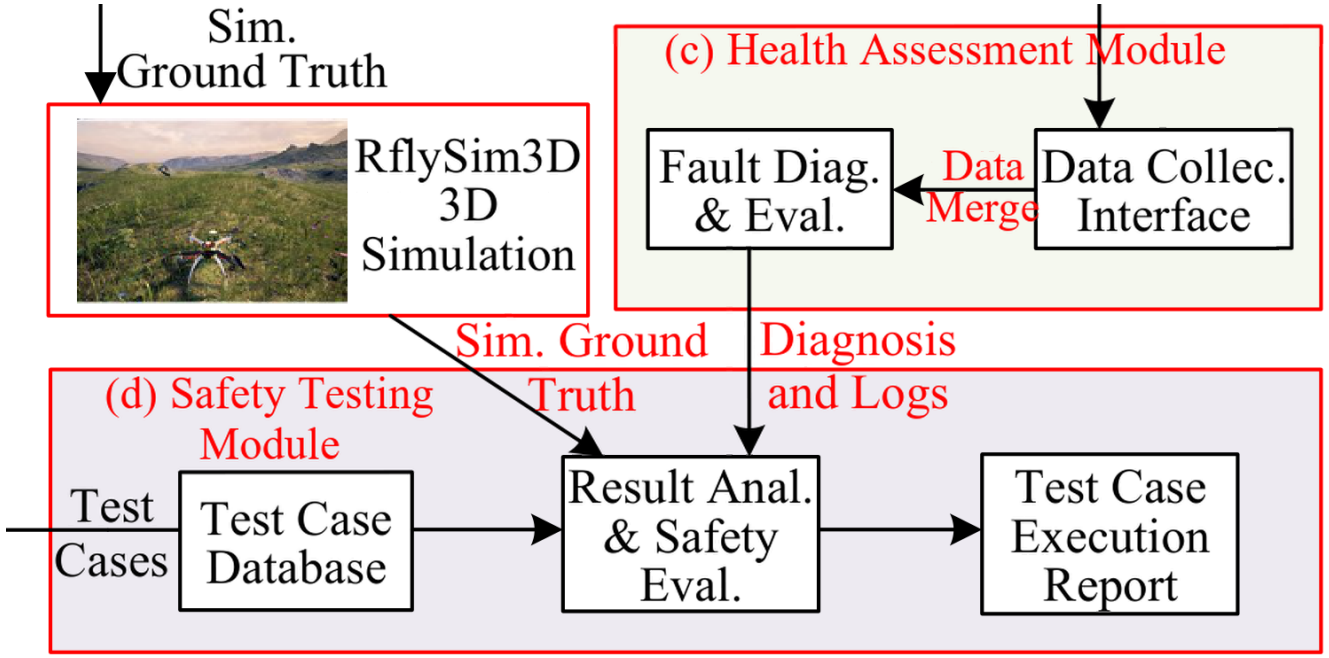


Fig. 7.14(c,d) Diagnosis and Assessment Branches

Diagnosis input: sensor logs and states collected through acquisition interfaces.

Assessment reference: test labels and simulation ground truth, used only for evaluation.

Mission consequences: report control response, mission error, and safety margins.

Deployed inputs lack true classes; test ground truth checks diagnostic correctness.

Chapter Outline

Fault Injection Interfaces and Algorithm Development Framework

Chapter 7 Health Management and Safety Assessment

7.1 Health Management Fundamentals

7.2 Unified Fault Modeling

7.3 Safety Reinforcement and Fault Data

7.4 Fault Hazards and Safety Assessment

7.5 Diagnosis and Prognostics

7.6 Safety Decisions and Improvement

7.7 Fault Injection Interfaces

7.8 Experiments and Validation

7.9 Exercises and Classroom Discussion

5 Units in This Section

7.7.1 Fault Injection Layers and Framework

Textbook: Sec. 7.2.1

7.7.2 Assessment and Diagnosis Development Framework

Textbook: Sec. 7.2.2

7.7.3 Fault IDs and Parameter Interfaces

Textbook: Sec. 7.2.1, 7.2.3

7.7.4 Injection, Recovery, and Reset Rules

Textbook: Sec. 7.1.5.2, 7.2.3

7.7.5 Observation Interfaces and Effect Verification

Textbook: Sec. 7.1.2.3, 7.1.5.1, 7.2.3

7.7.3 Fault IDs and Parameter Interfaces: External Injection Interface

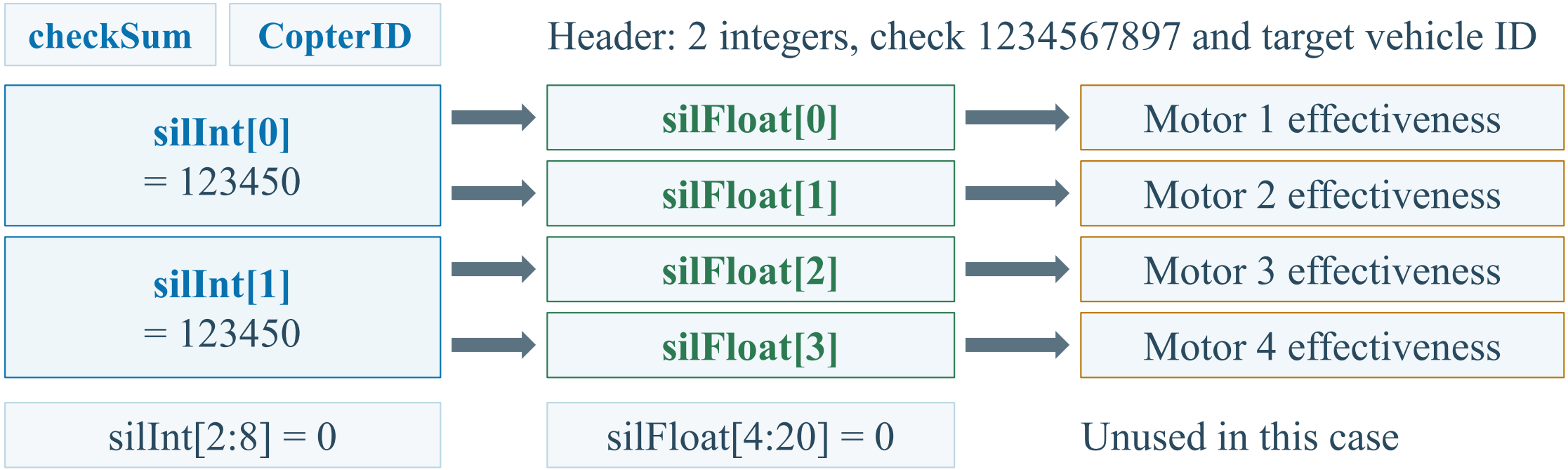
Fault Parameters Use Eight Integer and Twenty Float Slots

Field	Length / Type	Use in This Implementation
checkSum **CopterID**	2 integers	Check identifier 1234567897; target vehicle number
inSILInts	8 × int32	Match fault types and locate parameter slots
inSILFLoats	20 × float32	Fault parameters; receiver-defined extension slots
Target model port	$30100 + 2(\mathrm{ID} - 1)$	DLL: vehicle 1→30100; vehicle 2→30102 Flight controller: 20100 series

Call: mav1.sendSILIntFloat(silInt, silFloat, copterID=1)

7.7.3 Fault IDs and Parameter Interfaces: Parameter Extraction

Each Integer Slot Selects Two Float Parameter Slots



Motor FaultID 123450 appears twice; two parameters per slot supply four motors.
Motor numbering follows receiver connections, not image positions.

7.7.3 Fault IDs and Parameter Interfaces: Motor Effectiveness Model

This Motor Module Scales Normalized Control Inputs

```
if hasFault
    y(i) = inPWMs(i) * FaultParams(i);
end
```

$$u_{i,\text{out}} = h_i u_{i,\text{in}} = 0.6 \times 0.5 = 0.3$$

$u_{i,\text{in}}$, $u_{i,\text{out}}$: dimensionless normalized motor controls; PWM = pulse-width modulation.
h_i: control effectiveness; 1 preserves output; 0 zeros this channel.
Control scaling does not guarantee proportional thrust reduction.

7.7.3 Fault IDs and Parameter Interfaces: Model versus Firmware

Model and Firmware Fault IDs Have Different Receiver Rules

Item	Model Motor Case	Paired Firmware Injection
Receiver	Fault DLL loaded by CopterSim	Modified PX4 driver or control module
Send interface	sendSILIntFloat	e3_PX4FailureGenerator SendHILCtrlMsg (advanced)
Same ID	123456: payload drift	v1.12.3: GPS input modification GPS: Global Positioning System
Reproduction	Model contains receiving logic; arrays match the module	Match firmware, message layout, and hardware link.

7.7.3 Fault IDs and Parameter Interfaces: Firmware Extension

Official Firmware Injection Also Depends on Component Support

```
failure gps off
failure gps ok
```

Prerequisites: enable SYS_FAILURE_EN. v1.12.3 already includes the failure command.

Meaning: off stops GPS-data publication; ok restores normal publication.

Check command replies, restored GPS data, and flight-controller state; consult support tables for the selected version.

Components, fault types, and support: PX4 official main-branch documentation
https://docs.px4.io/main/en/debug/failure_injection

Chapter Outline

Fault Injection Interfaces and Algorithm Development Framework

Chapter 7 Health Management and Safety Assessment

7.1 Health Management Fundamentals

7.2 Unified Fault Modeling

7.3 Safety Reinforcement and Fault Data

7.4 Fault Hazards and Safety Assessment

7.5 Diagnosis and Prognostics

7.6 Safety Decisions and Improvement

7.7 Fault Injection Interfaces

7.8 Experiments and Validation

7.9 Exercises and Classroom Discussion

5 Units in This Section

7.7.1 Fault Injection Layers and Framework

Textbook: Sec. 7.2.1

7.7.2 Assessment and Diagnosis Development Framework

Textbook: Sec. 7.2.2

7.7.3 Fault IDs and Parameter Interfaces

Textbook: Sec. 7.2.1, 7.2.3

7.7.4 Injection, Recovery, and Reset Rules

Textbook: Sec. 7.1.5.2, 7.2.3

7.7.5 Observation Interfaces and Effect Verification

Textbook: Sec. 7.1.2.3, 7.1.5.1, 7.2.3

7.7.4 Injection, Recovery, and Reset Rules: One Injection

Reduce One Motor's Effectiveness; Keep the Others at 1

```
import PX4MavCtrlV4 as PX4MavCtrl
mav1 = PX4MavCtrl.PX4MavCtrler(1)
silInt = [123450, 123450] + [0] * 6
silFloat = [0.6, 1.0, 1.0, 1.0] + [0.0] * 16
mav1.sendSILIntFloat(silInt, silFloat, copterID=1)
```

Model Motor Number	1	2	3	4
Effectiveness	0.6	1.0	1.0	1.0

Configure the software development kit (SDK) module path, then import. One-motor setting: other coefficients are healthy (1).

7.7.4 Injection, Recovery, and Reset Rules: Triggering and Timing

Trigger Conditions Separate Healthy and Fault Phases



Original condition: 20 s host elapsed time and flag==1; set flag=2 after sending, so it sends once.

State-trigger design: same mission state, valid estimates, specified duration. Store conditions and thresholds in the case.

Timing evidence: record host send and reception/effect times separately; map clocks before comparing.

7.7.4 Injection, Recovery, and Reset Rules: Undoing Effectiveness Loss

Restore Output by Resending Healthy Parameters with the Same ID

```
silInt = [123450, 123450] + [0] * 6
silFloat = [1.0, 1.0, 1.0, 1.0] + [0.0] * 16
mav1.sendSILIntFloat(silInt, silFloat, copterID=1)
```

Parameter recovery: all four modeled coefficients return to 1, restoring the output/input ratio.

Check response: compare commands, attitude, and velocity before/after; inertia and controller state persist.

Parameter recovery differs from flight-state recovery.

7.7.4 Injection, Recovery, and Reset Rules: Reset Mechanism

All-Zero IDs Do Not Clear This Model's Latched State

```
persistent hFault;
persistent fParam;
if hFaultTmp
    hFault = hFaultTmp;
    fParam = fParamTmp;
end
hasFault = hFault;
```

Only matched IDs update the latch. All-zero IDs fail matching; the old state remains. Independent next run: reinitialize; check coefficients, vehicle initial state, and a healthy baseline.

7.7.4 Injection, Recovery, and Reset Rules: Latching and Continuous Motion

Packets Update the Latch; Vehicle Motion Remains Continuous

Two 123450 IDs match; all-zero IDs retain the latch; healthy values restore output ratio.

Matching IDs update memory; motion stays continuous $t = 10.0$ s

2 s Valid ID: inject **4 s Zero IDs: hold** **6 s Same ID: healthy**

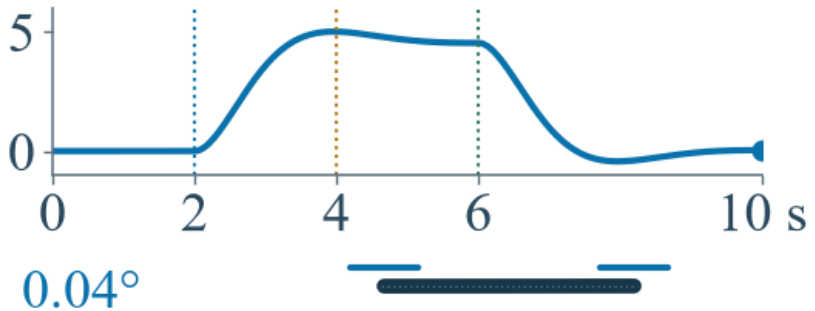
Packet: ID / pair **Persistent values**

1: 123450 [1.0, 1.0]	[1.0, 1.0]
2: 123450 [1.0, 1.0]	[1.0, 1.0]

Other 6 IDs = 0 hasFault = 1

Match: write PWM: 0.50 × 1.0 = 0.50

Angle example / deg



0.04°

Parameter recovery is not a state reset; hasFault stays latched at 1

Static receiver check; angle is an independent second-order example, not SDK flight data.

7.7.4 Injection, Recovery, and Reset Rules: Ending and Repeating Runs

Cleanup Does Not Complete Experiment Reset

End the case: stop mission updates; restore pure-effectiveness parameters; record final vehicle and flight-controller states.

Cleanup: endMavLoop() may request forced disarming and end Offboard; EndTrueDataLoop() stops reception. Neither resets latched faults.

New case: reinitialize the model; restore vehicle/control initial states, mission flag, and time base.

Before rerunning: healthy coefficients, matching input/output, valid estimates, fresh logs; then trigger the fault.

Chapter Outline

Fault Injection Interfaces and Algorithm Development Framework

Chapter 7 Health Management and Safety Assessment

7.1 Health Management Fundamentals

7.2 Unified Fault Modeling

7.3 Safety Reinforcement and Fault Data

7.4 Fault Hazards and Safety Assessment

7.5 Diagnosis and Prognostics

7.6 Safety Decisions and Improvement

7.7 Fault Injection Interfaces

7.8 Experiments and Validation

7.9 Exercises and Classroom Discussion

5 Units in This Section

7.7.1 Fault Injection Layers and Framework

Textbook: Sec. 7.2.1

7.7.2 Assessment and Diagnosis Development Framework

Textbook: Sec. 7.2.2

7.7.3 Fault IDs and Parameter Interfaces

Textbook: Sec. 7.2.1, 7.2.3

7.7.4 Injection, Recovery, and Reset Rules

Textbook: Sec. 7.1.5.2, 7.2.3

7.7.5 Observation Interfaces and Effect Verification

Textbook: Sec. 7.1.2.3, 7.1.5.1, 7.2.3

7.7.5 Observation Interfaces and Effect Verification: Evidence Layers

Record Commands, Model State, Observations, and Estimates Separately

Record Layer	Minimum Contents	What It Establishes
Command	Vehicle ID, FaultID, parameters, send time	The anomaly requested by the sender
Effective model state	Match state, effectiveness, model input/output	The anomaly actually used by the receiver
Observation and control	Sensor data, actuator values, flight mode	How the closed loop responds
Estimate and ground truth	Flight-controller estimate; DLL motion reference	Consistency of estimated and modeled motion

7.7.5 Observation Interfaces and Effect Verification: Coordinate Checks

Use the Same Reference Origin before Comparing Position

SDK Field	Frame / Unit	Reference Origin
uavPosNED	North-east-down (NED); m	Flight-controller estimator's local origin
truePosNED	NED motion ground truth; m	CopterSim / 3D map center

$$p_{\text{true,local}}(t) = p_{\text{true,map}}(t) - p_{\text{origin,map}}$$

origin: estimator's local origin in map coordinates; it may differ from takeoff.
North example: origin 10 m, takeoff 12 m, truth 13 m → local position 3 m.
Subtracting takeoff gives 1 m: the 2 m origin offset is not estimation error.

7.7.5 Observation Interfaces and Effect Verification: Received Signals

Check Data Freshness and Time Bases with Printed States

```
mav1.InitMavLoop()
mav1.InitTrueDataLoop()
print(mav1.uavPosNED, mav1.truePosNED)
```

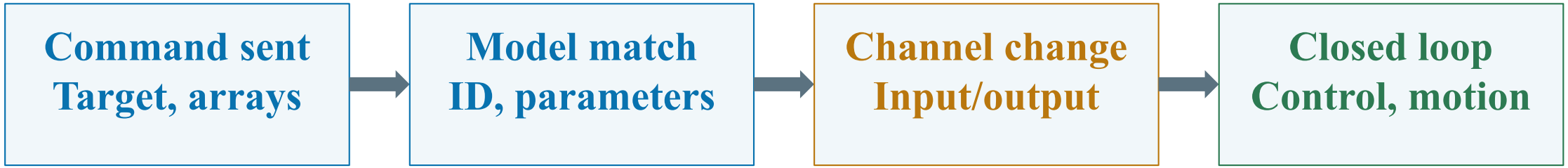
Updates: timestamps increase and reception flags are valid. Never judge from default startup zeros.

Time alignment: check source, units, and origin; map events or known offsets to a common timeline.

Port condition: InitTrueDataLoop and InitDllLogLoop use the same 30101-series ports; choose one.

7.7.5 Observation Interfaces and Effect Verification: Activation Criteria

Separate Default Flight Evidence from Internal Model Evidence



Default: printed uavPosNED and truePosNED; PX4 actuator outputs, attitude, position.

Advanced: log effectiveness and PWMIn/PWMOut in source; build the matching DLL; compare at the same model time.

First compare other motors' compensation/saturation, then attitude/position. 3D text reflects only the sender.

7.7.5 Observation Interfaces and Effect Verification: Common Failures

Trace the Message Path When Sending Produces No Change

Observation	Check First	Resolving Evidence
Sent; parameters unchanged	Vehicle ID, model port, loaded fault DLL	Received fields, ID match, effective parameters
Parameters change; channel unchanged	Motor order, model connections, actual parameter values	Channel input/output, actual module connections
Channel changes; hover continues	Control compensation, limits; remaining control capability	Healthy/fault control values under matching conditions
Abnormal start on run 2	Latch, vehicle/controller state, mission flag	New instance; healthy baseline; fresh log start

7.7.5 Observation Interfaces and Effect Verification: Minimal Validation

Use One Checklist for Healthy, Fault, and Recovery Phases

Phase	Check Coefficients	Signals / Relationships
Healthy control	1, 1, 1, 1	Estimates, truth, actuator outputs; advanced: coefficients, motor input/output
One motor fault	0.6, 1, 1, 1	Motor 1 scaling; compensation; motion response
Parameter recovery	1, 1, 1, 1	Output/input ratio restored; observe subsequent motion
Independent case	Latch reset; same initial state	New healthy baseline, time origin, and mission flags

7.7.5 Observation Interfaces and Effect Verification: Versions and Extensions

Bind Reproduction Records to Model and Interface Snapshots

Documentation: [installation directory]/RflySimAPIs/RflySimSDK/html/index.html Python control interfaces and Simulink prognostics and health management (PHM) modules.

Interface checks: sendSILIntFloat signature, message type, receiver ID, parameter indices, and model connections.

Pair source and DLL: record SDK, library source, model, and DLL snapshots. Confirm source rules against the matching DLL through build records or signals.

Method review: Section 7.3 organizes triggers and records; Section 7.5 infers faults from observations.

Exercises and Discussion

Build Complete Evidence for One Motor Injection

Conditions: vehicle ID=2; motor 3 effectiveness 0.4, others 1; motor 3 normalized input 0.55.

Question 1: Write the eight-integer and twenty-float arrays. Which float slot holds motor 3's parameter?

Question 2: Calculate motor 3's fault output. Can you conclude that thrust is also only 40%?

Question 3: How do you restore output and fully reset? Why are all-zero IDs insufficient?

Question 4: No program error and a displayed fault message: what evidence is still needed to confirm activation?

Reference Points

Parameters, Recovery, and Activation Need Separate Answers

```
silInt = [123450, 123450] + [0] * 6
silFloat = [1.0, 1.0, 0.4, 1.0] + [0.0] * 16
```

$$u_{3,\text{out}} = 0.4 \times 0.55 = 0.22$$

Motor 3: silFloat[2] via silInt[1]; vehicle 2 port 30102.
0.22 is control output; thrust ratio depends on motor/rotor models.

Same IDs + four 1s restore output; all-zero IDs fail matching and retain the old fault.
Independent case: reinitialize the model and restore the same initial state.

Activation evidence: correct target ID, updated effective parameters, changed target channel, recorded closed-loop response.

Section Summary

Reproducible Injection Includes Reception, Recovery, and Reset

Nodes/layers: SITL, HITL, and real vehicles cover different links; model and firmware protocols are not interchangeable.

Interface: eight integer slots and twenty float slots. Each 123450 carries two motor parameters; repeat it twice for a quadrotor.

Trigger/end: record state conditions and times separately. Healthy parameters restore output; a new instance resets the latch.

Pitfalls: sending is not activation; zero coefficients zero the output. Stopping reception or sending all-zero IDs does not clear the latch.