Chapter Outline

Virtual-Real Safety Reinforcement and Fault Data Methods

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

7.3.1 Virtual-Real Safety Reinforcement and Simulation Nodes
Textbook: Sec. 7.1.4, 7.1.5.1

7.3.2 Triggers and Test Case Design
Textbook: Sec. 7.1.5.2

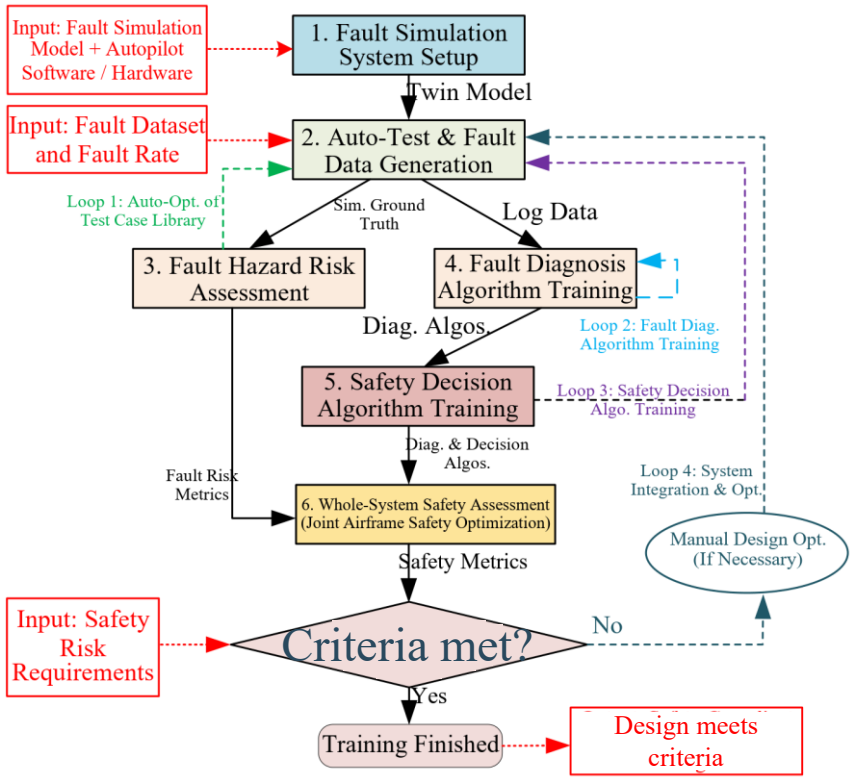
7.3.3 Automated Testing and Event Evidence
Textbook: Sec. 7.1.5.2

7.3.4 Log Synchronization and Labeling
Textbook: Sec. 7.1.5.4 (fault data labeling)

7.3.5 Data Splits and Quality
Textbook: Sec. 7.1.5.4

7.3.1 Virtual-Real Safety Reinforcement and Simulation Nodes: Framework

Six Modules Connect Data, Decisions, Actions, and Design



Accumulation (modules 1, 2): build simulations and run tests to collect operating-condition and fault data.

Diagnosis (module 4): train on logs to identify fault locations and causes.

Decision-making (module 5): train and select responses based on the mission and available capabilities.

Assessment (modules 3, 6): assess hazards and the complete system; test and update the design.

Fig. 7.6 Safety Reinforcement Training Framework

7.3.1 Virtual-Real Safety Reinforcement and Simulation Nodes: Data Flow

Ground Truth Assesses Consequences; Logs Support Diagnosis

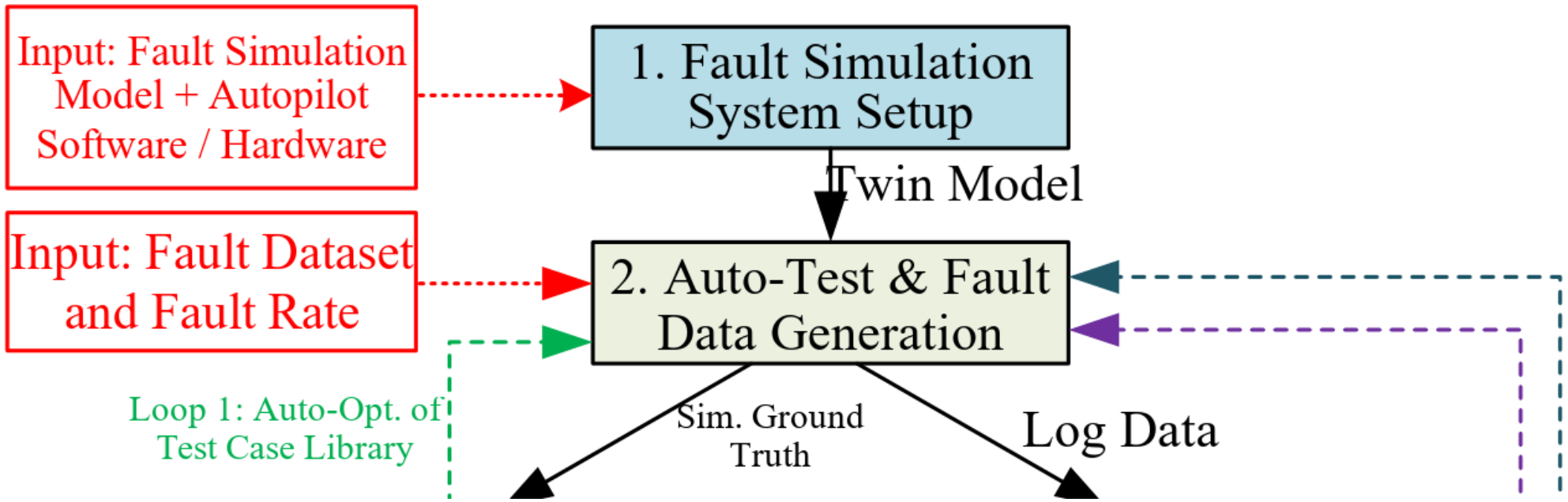


Fig. 7.6 (top) Simulation, Test Data, and Test Case Library Optimization

Loop 1 adjusts test coverage. Fault occurrence rates require reliability evidence; injection proportions organize test samples.

7.3 Safety Reinforcement and Fault Data

7.3.1 Virtual-Real Safety Reinforcement and Simulation Nodes: Feedback

Train Diagnosis and Responses Separately, Then Assess Them Together

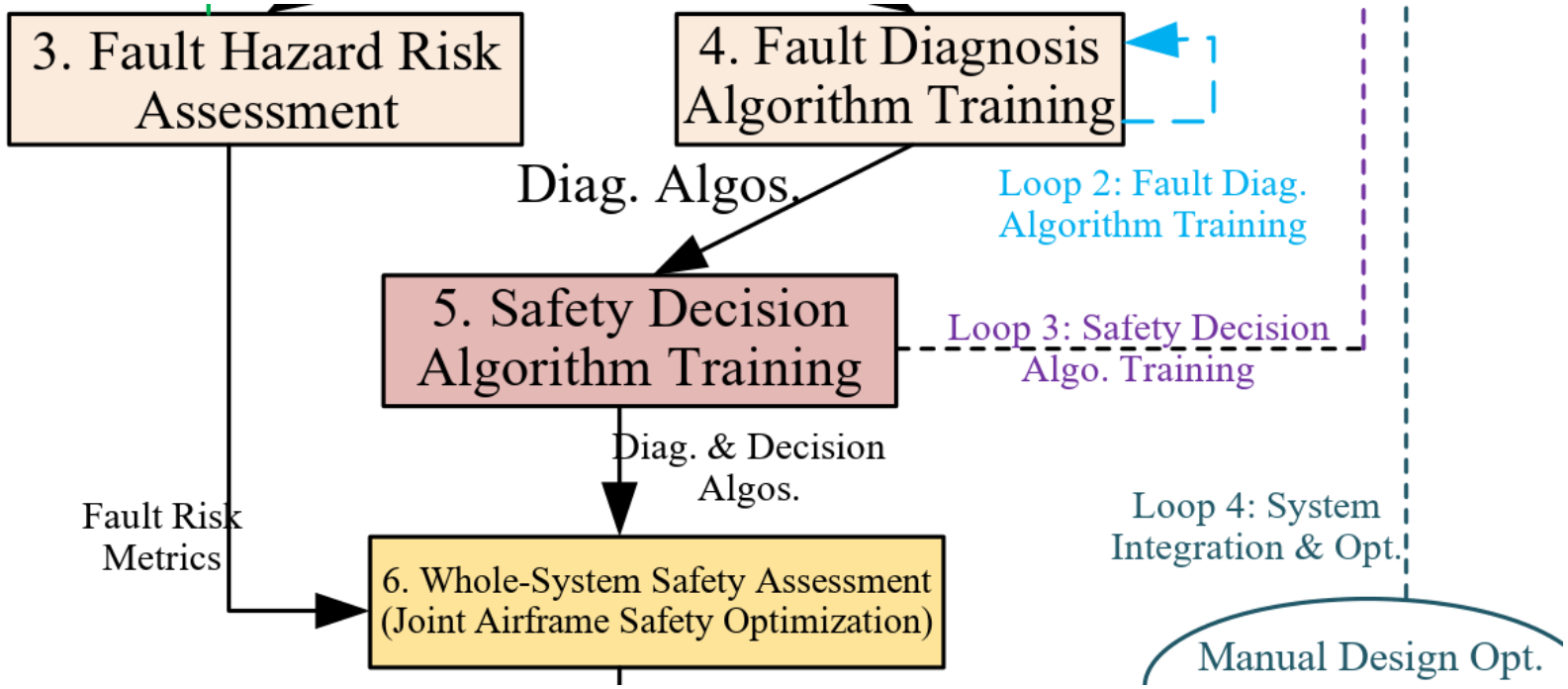


Fig. 7.6 (middle) Inputs to Diagnosis, Decisions, and System Assessment

Loop 2: improve diagnosis; reduce false alarms and misses.

Loop 3: improve actions to reduce harm after a fault.

Joint assessment: check identification and action outcomes.

7.3.1 Virtual-Real Safety Reinforcement and Simulation Nodes: Node Types

The Three Node Types Validate Different Aspects

Node	Autopilot Location	Vehicle and Sensor Sources
SITL Software-in-the-loop	Autopilot software on a computer	Motion model and simulated sensors
HITL Hardware-in-the-loop	Physical flight controller	Motion model and simulated sensors
Real-vehicle experiment	Physical flight controller	Physical vehicle, actuators, and sensors

SITL (textbook SIL): repeatable batch tests. HITL (textbook HIL): board execution. Real-vehicle tests add physical evidence.

7.3 Safety Reinforcement and Fault Data

7.3.1 Virtual-Real Safety Reinforcement and Simulation Nodes: Platform

Control and Feedback Close the Loop at All Three Nodes

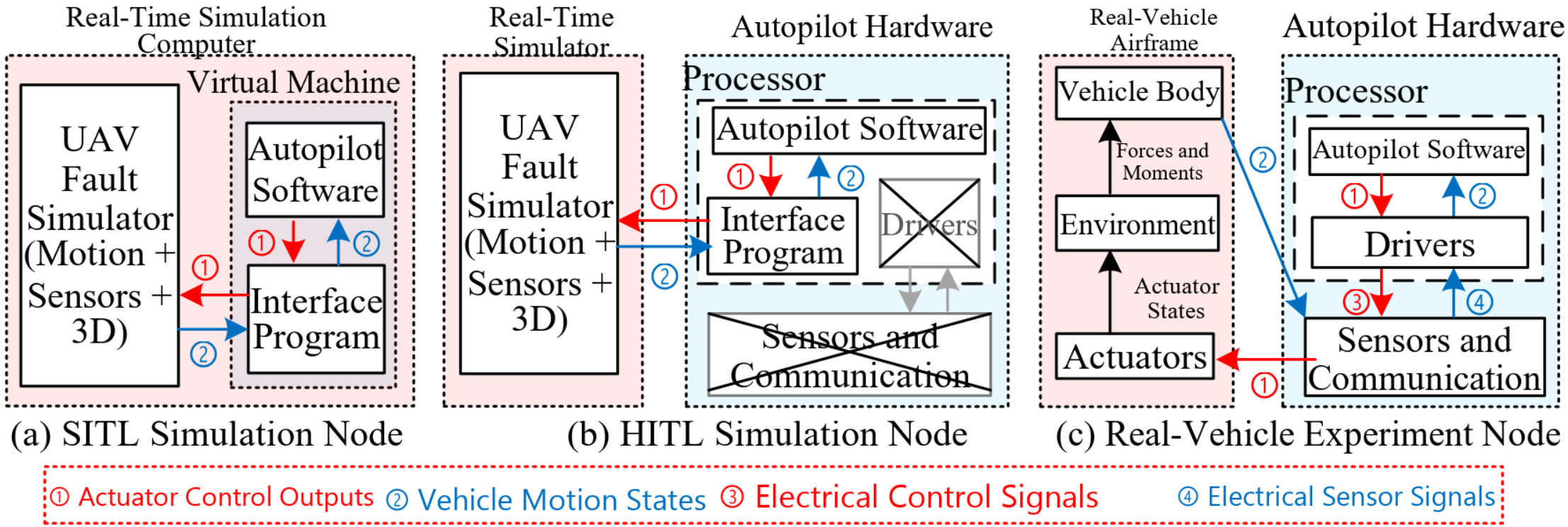
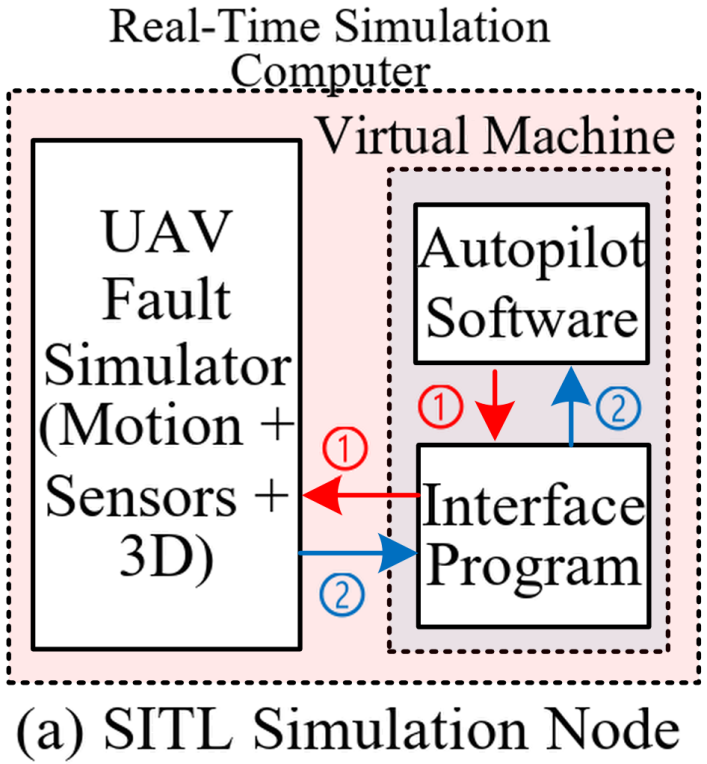


Fig. 7.7 Software Simulation, HITL Simulation, and Real-Vehicle Experiments

7.3.1 Virtual-Real Safety Reinforcement and Simulation Nodes: SITL

SITL Exchanges Model Data through an Interface Program



(a) SITL Simulation Node

Control path: autopilot software issues actuator commands; the simulator updates vehicle motion.

Feedback path: simulated sensors produce measurements; the autopilot estimates the state and closes the control loop.

Injection location: motor effectiveness changes in the model propulsion path; the flight controller still handles the resulting motion.

Fig. 7.7(a) Pure Software Simulation Node

7.3.1 Virtual-Real Safety Reinforcement and Simulation Nodes: HITL

HITL Still Relies on Vehicle and Sensor Models

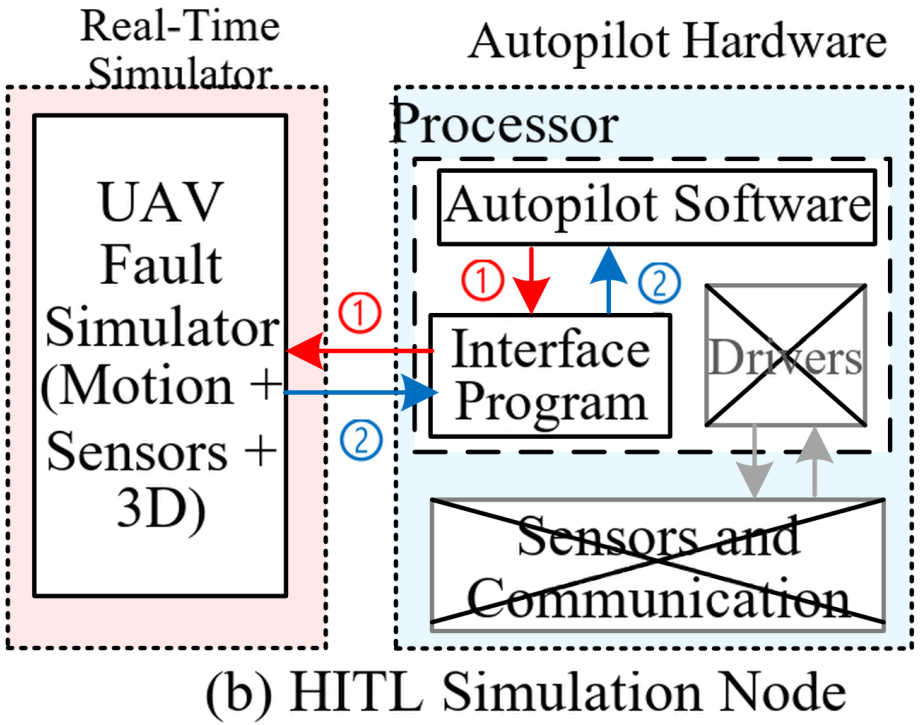


Fig. 7.7(b) HITL Simulation Node

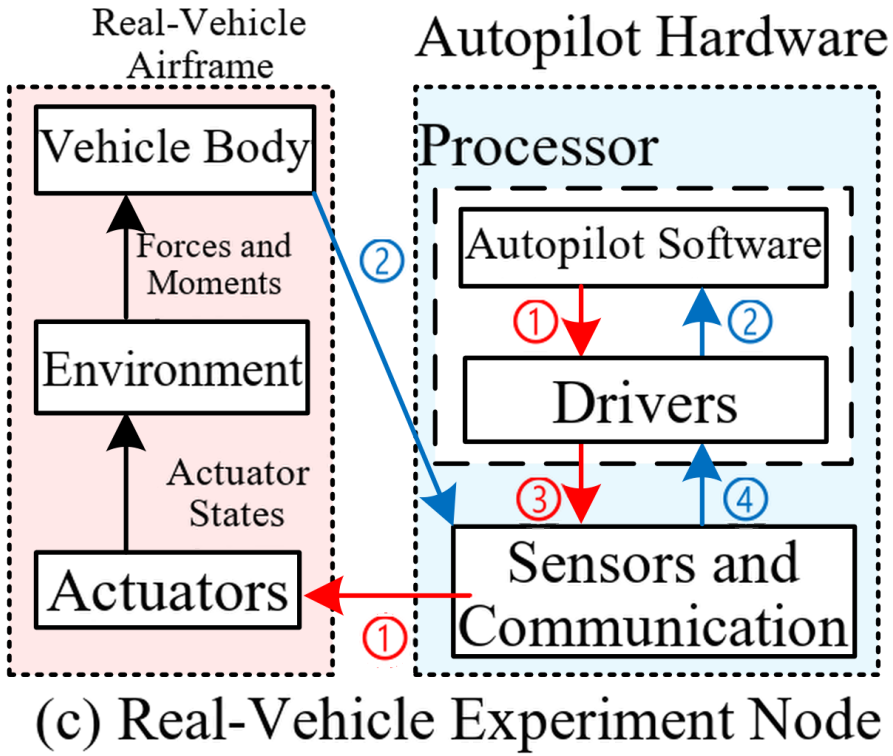
Physical: flight-controller processor, firmware execution, interfaces, and communication delays.

Simulated: vehicle motion, motor dynamics, and sensor measurements.

Model faults: the simulator's fault dynamic-link library (DLL) receives and applies them (Section 4.9).

7.3.1 Virtual-Real Safety Reinforcement and Simulation Nodes: Real Vehicle

Real-Vehicle Tests Include Drivers, Actuators, and Sensors



① and ②: actuator control outputs and vehicle motion states.

③ and ④: electrical signals carrying control values and sensor data.

Validation scope: firmware, drivers, physical response, and the environment jointly determine the result.

Fig. 7.7(c) Real-Vehicle Experimental Node

7.3 Safety Reinforcement and Fault Data

7.3.1 Virtual-Real Safety Reinforcement and Simulation Nodes: Cross-Validation

Run the Same Mission at Each Node; Calibrate with Real Data

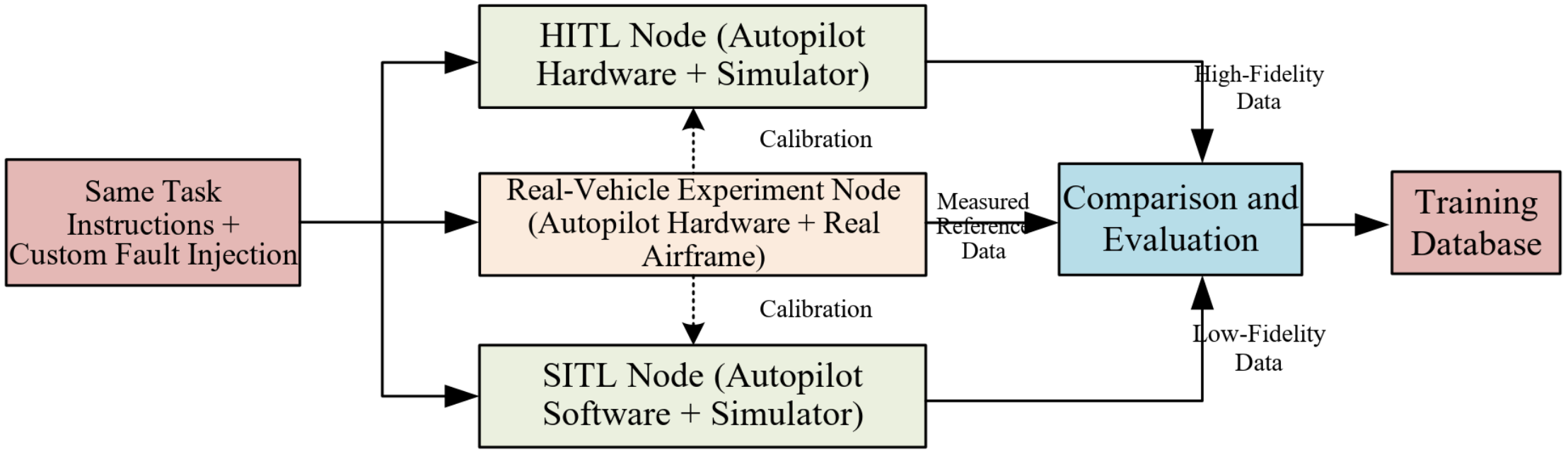


Fig. 7.8 Parallel Virtual and Physical Experiments and Cross-Validation

HITL includes a physical flight controller; SITL provides batch coverage; real-vehicle tests calibrate the target conditions.

7.3.1 Virtual-Real Safety Reinforcement and Simulation Nodes: Comparability

Align Inputs, Frames, Events, and Versions before Comparing

In Fig. 7.8, real-vehicle reference data calibrates HITL and SITL; all three responses enter comparative assessment. Align their conditions first.

Inputs and Conditions	Variables and Events	Versions and Errors
Commands: same mission/faults Initial: same state, payload Wind: matched	Frame: same axes and units Time: same time base Onset: verify Delay: keep; do not shift	Versions: firmware, controller, model, sensors Errors: parameters, timing, conditions

Equal log lengths do not prove synchronization. Event delay may be the difference under test; do not shift it away.

Chapter Outline

Virtual-Real Safety Reinforcement and Fault Data Methods

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

7.3.1 Virtual-Real Safety Reinforcement and Simulation Nodes
Textbook: Sec. 7.1.4, 7.1.5.1

7.3.2 Triggers and Test Case Design
Textbook: Sec. 7.1.5.2

7.3.3 Automated Testing and Event Evidence
Textbook: Sec. 7.1.5.2

7.3.4 Log Synchronization and Labeling
Textbook: Sec. 7.1.5.4 (fault data labeling)

7.3.5 Data Splits and Quality
Textbook: Sec. 7.1.5.4

7.3.2 Triggers and Test Case Design: Injection Methods

Three Injection Methods Require Different Test Conditions

Method	Where the Anomaly Is Applied	Applicable Conditions
Model injection	DLL propulsion, sensors, environment; noise drift or frozen data	SITL/HITL; load the matching fault model
Source code injection	Autopilot drivers, scheduling, control; satellite loss, freezes or stalls	SITL/HITL/real vehicle; one RflySim-recommended method
Physical injection	Physical actuator/sensor/vehicle; relay power cut or blade damage to a propeller	Dedicated real-vehicle tests; not used in the classroom workflow

Injection location sets observation order; injection method sets test conditions.

7.3.2 Triggers and Test Case Design: Cases and Run Records

A Case Defines Conditions; a Run Record Provides Evidence

Test case Conditions, criteria	→	Execution Trigger, reset	→	Run record Logs and events	→	Data sample Features, labels

Case ID: identifies the mission, initial values, fault settings, and criteria. The same case can run repeatedly.

Run ID: identifies one execution and links its logs, events, versions, and execution status.

Healthy control: keep the same mission and environment with faults disabled to distinguish normal maneuvers from abnormal responses.

Compare under controlled conditions; support results with traceable records.

7.3.2 Triggers and Test Case Design: Triggering Methods

Fixed Triggers Aid Comparison; Random Triggers Broaden Coverage

Deterministic Triggering

Condition: preset time (e.g., 15 s), mission phase, or state threshold.

Record: condition expression, time met, and actual fault onset.

Use: functional verification, boundary testing, and regression testing.

Stochastic Triggering

Condition: trigger with a set probability, or sample time and parameters from a distribution.

Record: distribution, range, seed (repeats the sample sequence), and each sampled value.

Use: rare, high-risk events; Monte Carlo simulation broadens coverage.

Trigger sampling sets coverage; real fault probabilities need separate evidence.

7.3.2 Triggers and Test Case Design: Effect Evidence and Records

Each Injection Method Needs Its Own Evidence and Records

Definitions: Unit 7.3.2, “Injection Methods.” Below: effect evidence and records.

Method	Evidence of Effect	What to Record
Model injection	Model parameter/state switches; ground truth changes with it	Fault ID, parameters, command time, simulation ground truth
Source code injection (one recommended method)	Firmware state: lower positioning accuracy or fewer satellites	Firmware version, fault ID, parameters, related log topics
Physical injection	Physical component change, independently measured	Test conditions, action time, independent measurement data

Cases here use model injection. State the target layer, fault ID, and parameters in the configuration.

7.3.2 Triggers and Test Case Design: Controlled Comparison

Change Only the Factor under Study between the Two Groups

Condition	Healthy Case N01	Fault Case F01
Shared conditions	Same vehicle, model, and controller; hover at 15 m	Same as healthy group, including environment and random seed
Fault setting	Fault disabled; motor effectiveness coefficient = 1	One motor: effectiveness coefficient = 0.6
Timing	Arm at 0 s; take off at 5 s; end at 25 s	Same mission; trigger the selected fault at 15 s
Records and criteria	Height, velocity, attitude, outputs, events; hover error, termination reason	Same variables; add activation evidence and post-fault response

Effectiveness: unitless, 1=healthy. Check channels, initial values, and stop conditions.

Chapter Outline

Virtual-Real Safety Reinforcement and Fault Data Methods

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

7.3.1 Virtual-Real Safety Reinforcement and Simulation Nodes
Textbook: Sec. 7.1.4, 7.1.5.1

7.3.2 Triggers and Test Case Design
Textbook: Sec. 7.1.5.2

7.3.3 Automated Testing and Event Evidence
Textbook: Sec. 7.1.5.2

7.3.4 Log Synchronization and Labeling
Textbook: Sec. 7.1.5.4 (fault data labeling)

7.3.5 Data Splits and Quality
Textbook: Sec. 7.1.5.4

7.3.3 Automated Testing and Event Evidence: Data Acquisition

The Test Program Connects Missions, Data, and Assessment

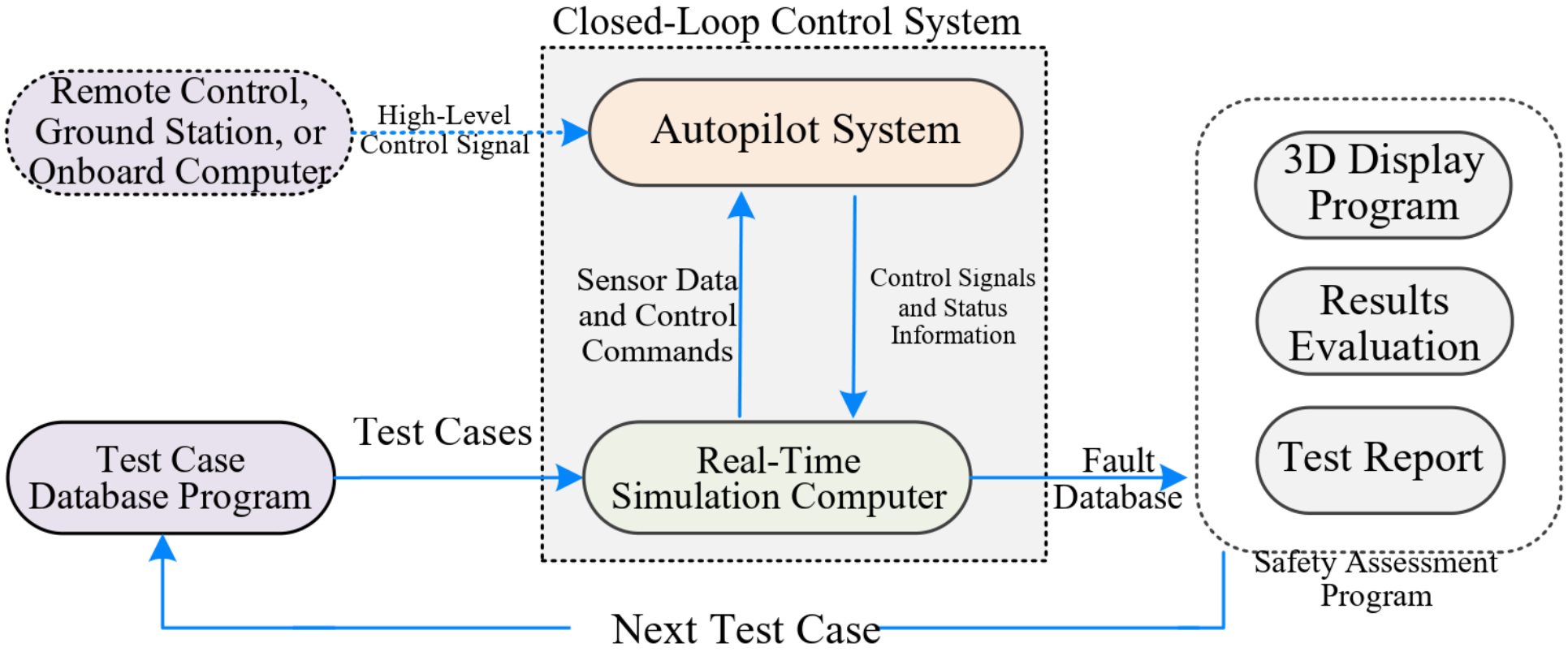
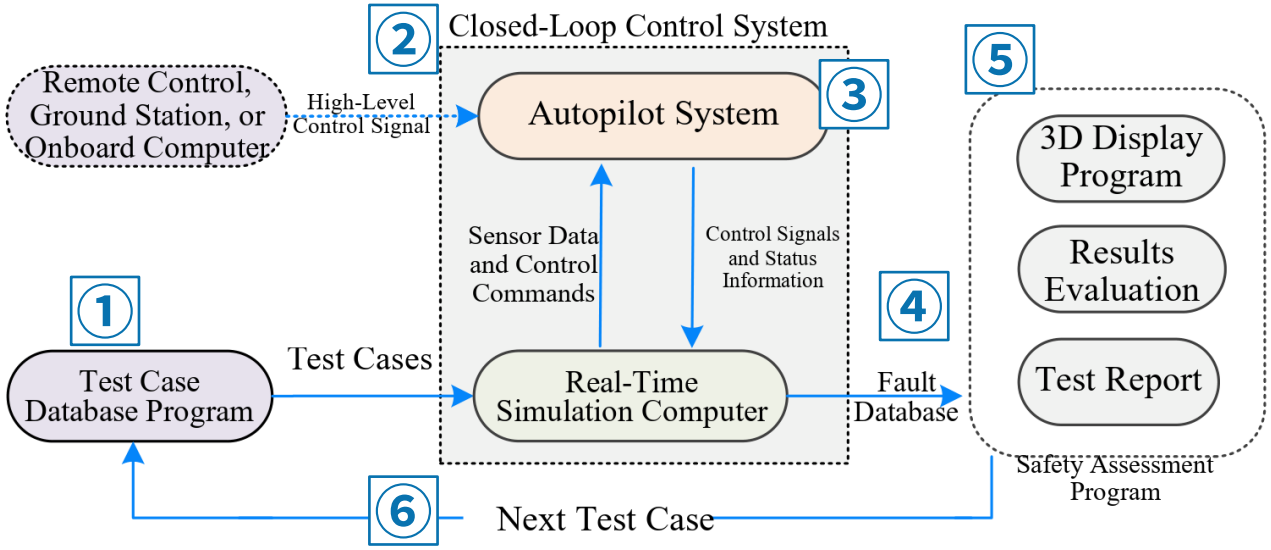


Fig. 7.9 Automated Testing and Training Framework

7.3.3 Automated Testing and Event Evidence: Six Steps

Fig. 7.9: Run Six Steps, Assess, Then Request the Next Case



① Send cases one by one
② Auto-initialize; run the mission
③ Trigger fault; log its time
④ Log terminal state; collect logs
⑤ Compare with mission expectations
⑥ After assessment, get the next case

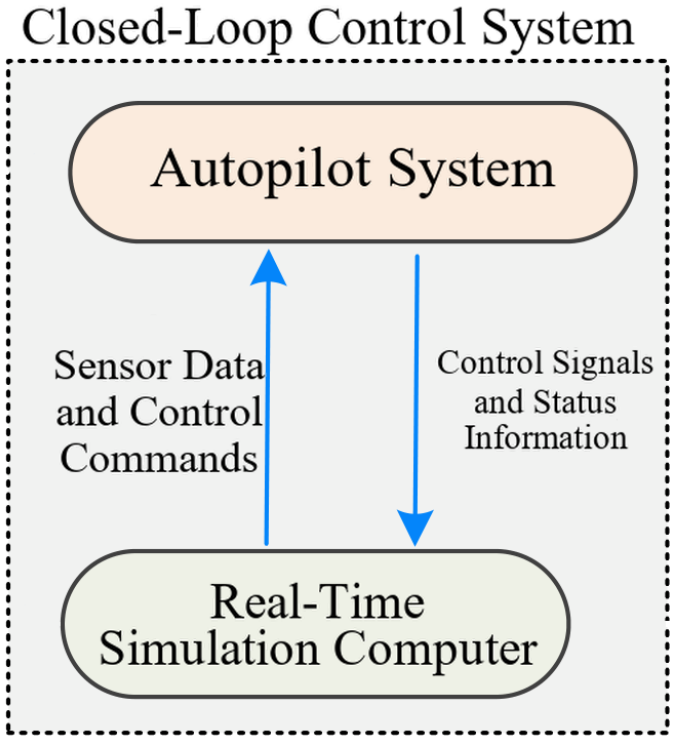
Fig. 7.9 (overview); numbers match the six steps at right

Case: start simulation; run mission; inject faults; collect parameters; assess safety.

Assess: accurate diagnosis? Correctly triggered safety response? Expected motion after the fault? Report quantitative metrics.

7.3.3 Automated Testing and Event Evidence: Data Sources

Motion Ground Truth and Flight-Controller Estimates Differ



Model ground truth: position, velocity, and attitude updated by the simulation computer for checking results.

Flight-controller estimate: state estimated by the autopilot from sensor data for control and diagnosis.

Control output: a command from the autopilot to an actuator; it is not a direct measurement of motor speed.

Fig. 7.9 (middle) Autopilot and Simulation Closed Loop

7.3.3 Automated Testing and Event Evidence: Run Sequence

Reset Initial Values; Archive Each Run at Termination

Reset initial state → **Execute mission** → **Trigger fault** → **End and archive**

Checkpoint	Restore or Confirm	Evidence to Keep
Before starting	Model fault state, flight mode, motion initial values, log buffer	Run ID, initial values, model and firmware versions
At the trigger	Time/phase condition and injection reception status	Command, reception or effect marker, actual parameters
At termination	Time limit, boundary violation, ground impact, or data interruption	Termination reason, complete logs, usability status

7.3.3 Automated Testing and Event Evidence: Published Example

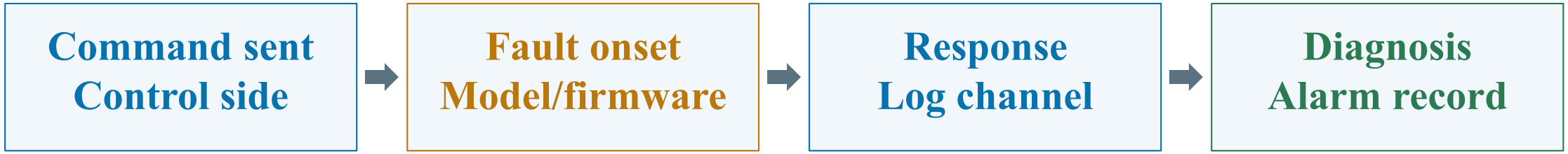
Paper Table 1 Defines an Example by Timing and Criteria

Paper Table 1: Time	Action	Records and Example Criteria
5 s	Arm	Record position, velocity, attitude, and angular rate
10 s	Take off to 10 m and hover	Hover; record the baseline before the fault
20 s	Stop motor 2	Attitude fluctuation ≤10°; max. impact speed ≤5 m/s
25 s	Stop simulation and record	Analyze this test against the stated criteria

Dai et al. (2021), Aerosp. Sci. Technol.; field-programmable gate array (FPGA)-supported HITL.
DOI: 10.1016/j.ast.2021.106727 (link in Section Resources)
Installed copy: [installation directory]/RflySimAPIs/PaperRes/0.RflySimToolChain/

7.3.3 Automated Testing and Event Evidence: Event Markers

Mark Command, Fault Onset, and Detection Separately



e18 logs the injection command in the flight controller via SendFaultEvent (command=31000).

Event Fields	Meaning or Example
run_id; event_type	Execution ID; command_sent / fault_effective / alarm
timestamp; fault_id	Time and unit; target layer and fault ID
parameters; version	Injection parameters; model/firmware/acquisition versions

Chapter Outline

Virtual-Real Safety Reinforcement and Fault Data Methods

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

7.3.1 Virtual-Real Safety Reinforcement and Simulation Nodes
Textbook: Sec. 7.1.4, 7.1.5.1

7.3.2 Triggers and Test Case Design
Textbook: Sec. 7.1.5.2

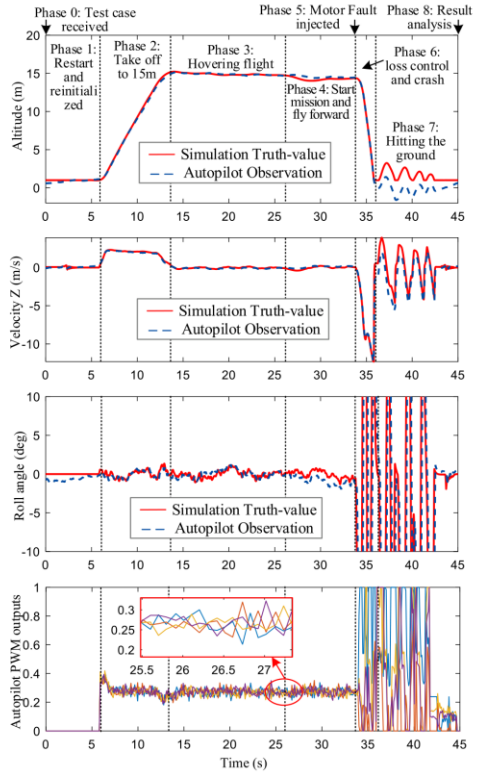
7.3.3 Automated Testing and Event Evidence
Textbook: Sec. 7.1.5.2

7.3.4 Log Synchronization and Labeling
Textbook: Sec. 7.1.5.4 (fault data labeling)

7.3.5 Data Splits and Quality
Textbook: Sec. 7.1.5.4

7.3.4 Log Synchronization and Labeling: Published Comparison

Read the Motor-Stall Test in Paper Fig. 19 by Events

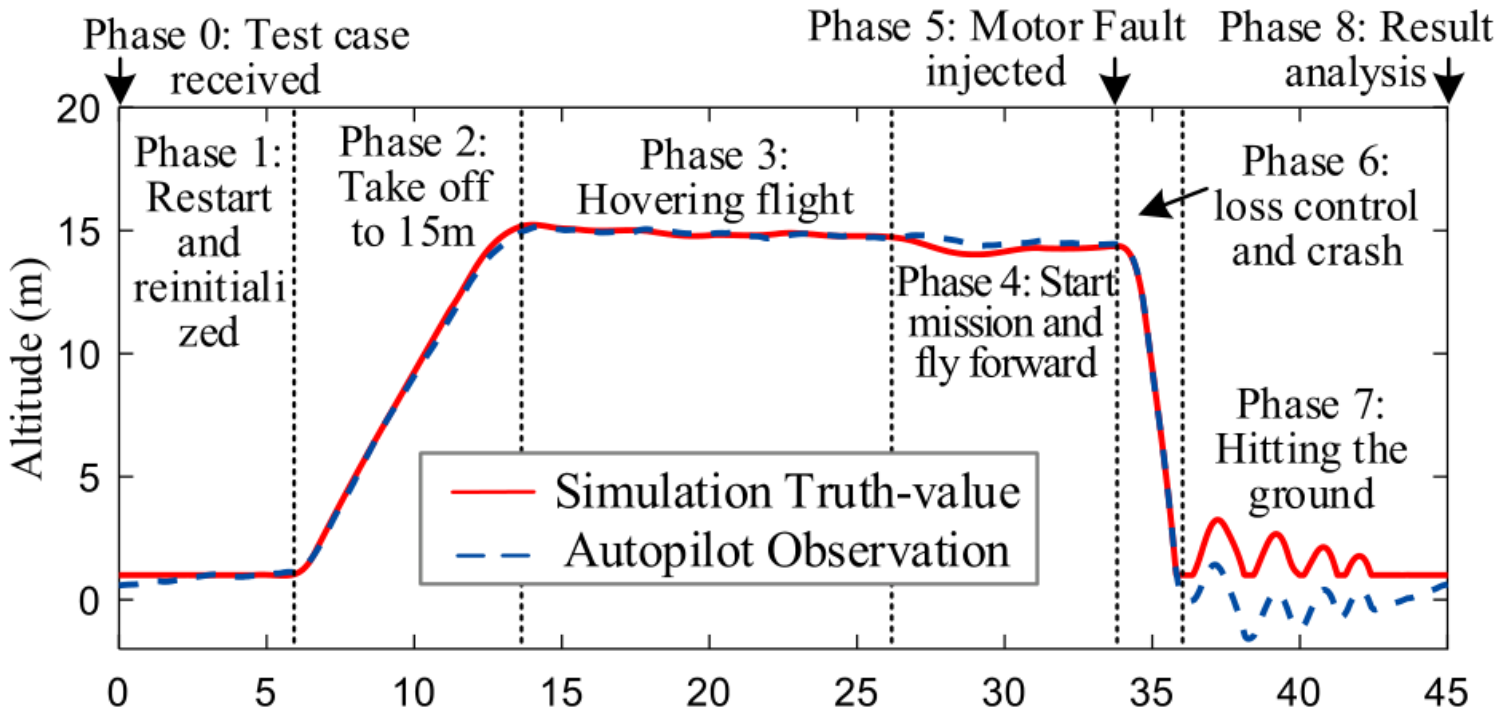


Channel	Normal and Fault Phases
Height (m)	Maintained hover; rapid height loss and ground impact after the stall
Vertical velocity (m/s)	Normal mission response; rapid descent and oscillations after impact
Roll angle (°)	Small attitude changes; violent roll after the stall
PWM outputs (pulse-width modulation)	Normal regulation; abrupt changes and endpoint values during loss of control

Dai et al. (2021), Fig. 19: normal and fault phases of one HITL motor-stall test

7.3.4 Log Synchronization and Labeling: Height and Phases

Events Separate Normal Hover from Height Loss after the Stall



Paper Fig. 19 (same source): height and phase detail

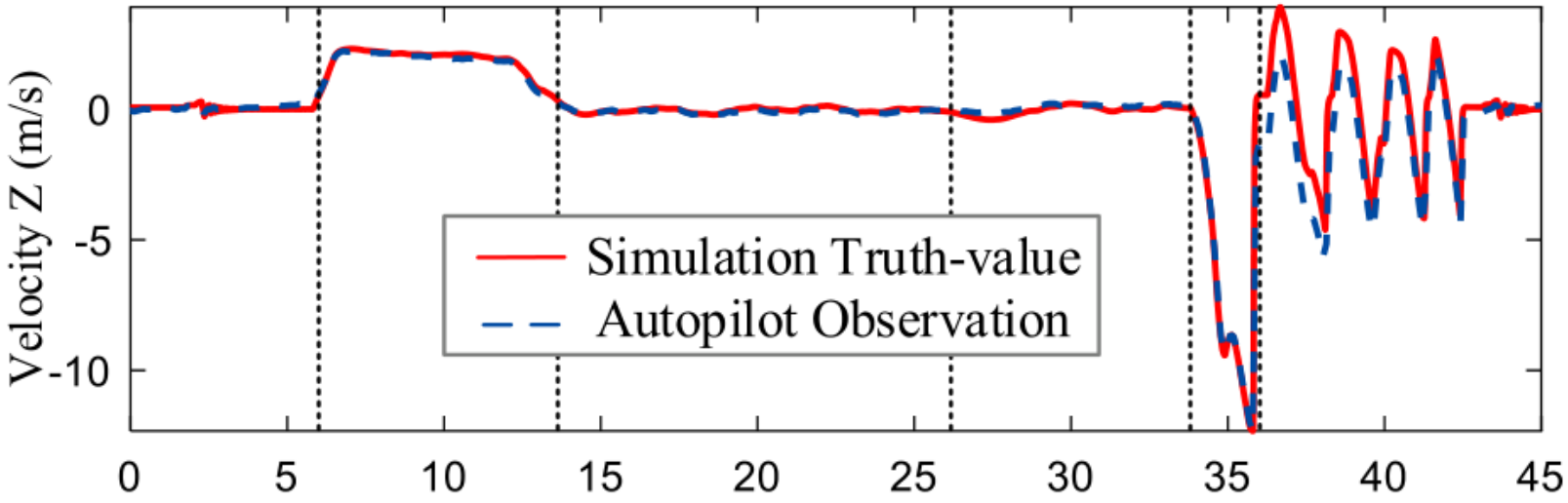
Normal phase: hover near 15 m.

Event: motor stall near 34 s.

Response: height loss, then impact.

7.3.4 Log Synchronization and Labeling: Vertical Velocity

Check Velocity and State Sources before and after Impact



Solid red
Simulation ground truth

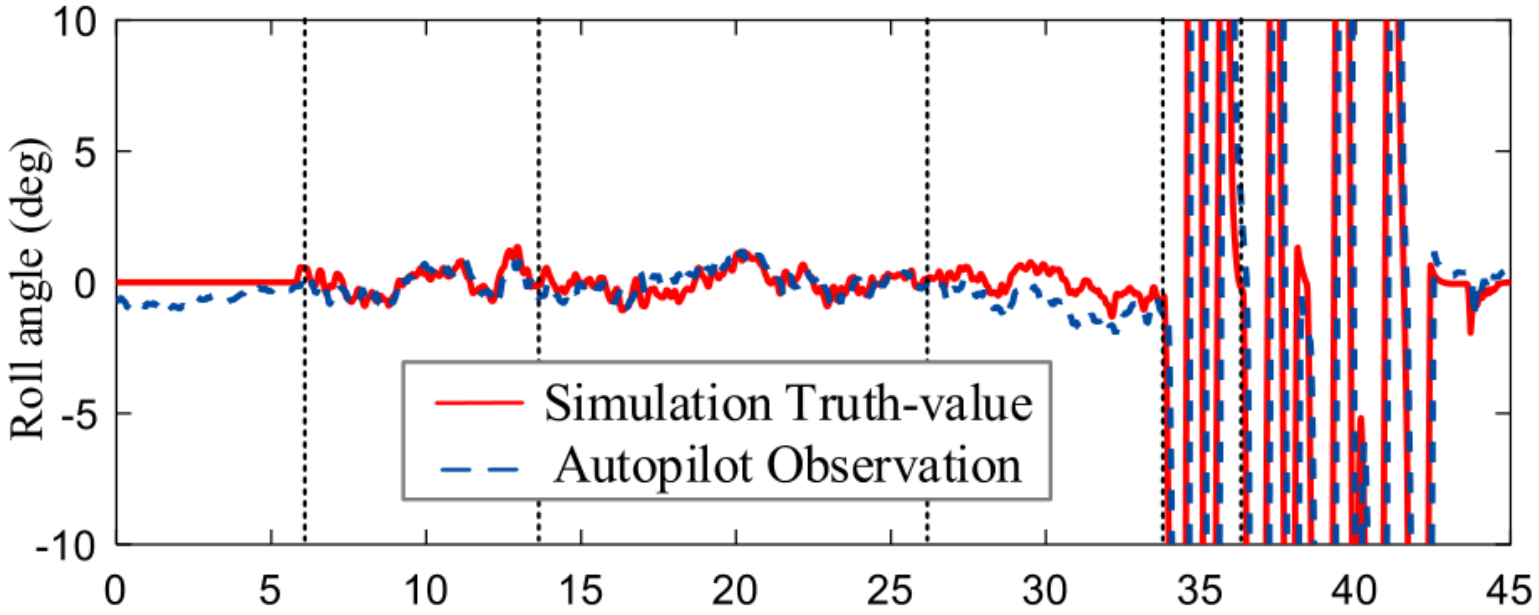
Dashed blue
Autopilot estimate

Paper Fig. 19 (same source): vertical velocity detail

Signs: the figure uses z positive up; north-east-down (NED) uses positive down. Reverse the sign first.
Read descent and post-impact oscillations separately; verify peaks from numerical records.

7.3.4 Log Synchronization and Labeling: Attitude Response

Violent Roll Shows That the Attitude Limit Was Exceeded



Paper Fig. 19 (same source): roll detail (vertical axis ±10°)

Events define the phases; attitude, height, and velocity together explain the motion consequences.

Normal: small roll. **Stall:** violent roll. Axis limit ≠ peak.

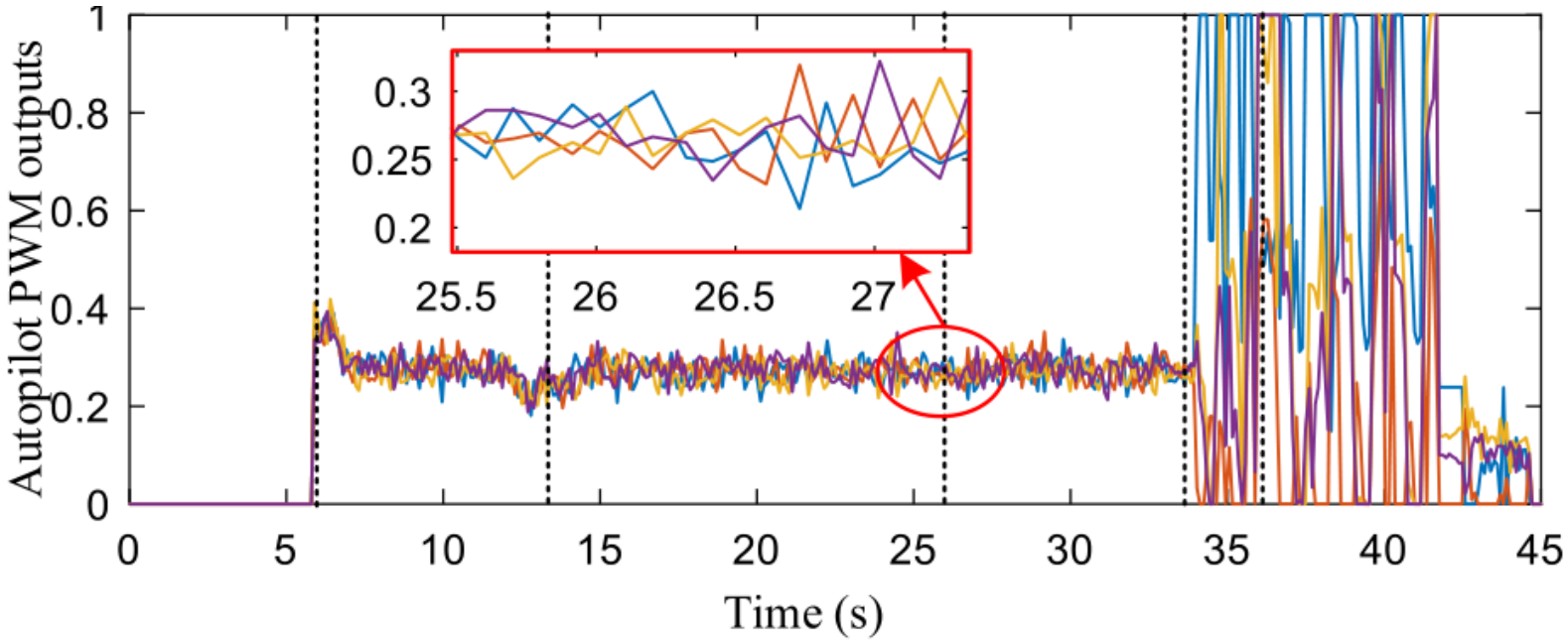
7.3.4 Log Synchronization and Labeling: Actuator Outputs

Output Endpoints Still Require a Check of Actual Motion

PWM:
pulse-width
modulation outputs.

Endpoints:
0 or 1.

Consequence:
loss of motion
control.



Paper Fig. 19: four motor PWM outputs; red box enlarges 25.5–27 s

Stall undetected and unhandled; result limited to this vehicle, model, and control policy.

7.3.4 Log Synchronization and Labeling: Offline Data

e10 Retains a ULog, Four CSV Tables, and Processing Results

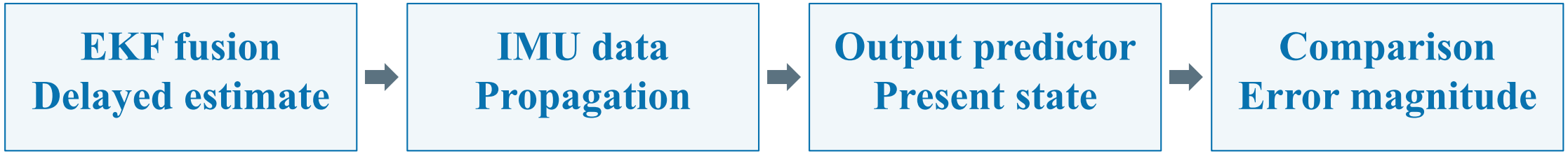
Directory: [installation directory]/RflySimAPIs/7.RflySimPHM/0.ApiExps/e10_data_process/

Actual Files under data/	Contents
09_23_59.ulg	Raw binary log; acquisition context
09_23_59_actuator_outputs_0.csv	4 commands; not measured speed
09_23_59_estimator_status_0.csv	Output-predictor errors: attitude/velocity/position (rad, m/s, m)
09_23_59_sensor_combined_0.csv	3-axis angular rate and acceleration
09_23_59_vehicle_magnetometer_0.csv	Three-axis magnetic field

CSV = comma-separated values. Align by timestamp: dataProcess.py → processed_data.csv.

7.3.4 Log Synchronization and Labeling: Output-Predictor Error

The Output Predictor Propagates Delayed EKF Estimates to Now



Output predictor: uses inertial measurement unit (IMU) data to propagate the extended Kalman filter (EKF) estimate from its delayed fusion time to the present.

Tracking errors: magnitudes of differences between output-predictor states and EKF estimates; attitude (rad), velocity (m/s), and position (m), in that order.

Distinction: these errors reflect consistency within the estimator, not controller tracking of a target trajectory. The three output_tracking_error entries are not x, y, z components.

7.3.4 Log Synchronization and Labeling: Time Synchronization

Unify Timestamps and Units, Then Interpolate to Common Times

Channel A Samples	Channel B Target Times	Aligned Values (Example)
32.0 s → 0	32.1 s	A interpolates to 1 at this time
32.2 s → 2	32.2 s	Keep the original A sample, 2

x is the continuous variable being aligned; both endpoint readings use the same unit.

$$x(32.1) = 0 + \frac{32.1 - 32.0}{32.2 - 32.0}(2 - 0) = 1$$

Interpolate continuous data; mark events by interval; first calibrate clocks with shared events.

PX4 channel timestamps are in microseconds; time in seconds is $t_s = t_{us}/10^6$.

7.3.4 Log Synchronization and Labeling: e10 Alignment

e10 Uses the Timestamp Union, with Uneven Sampling Intervals

Channel in the Open Interval 32–36 s	Raw Samples	Median Interval
Actuator outputs	40	100 ms
Estimator status	467	8 ms
Inertial sensors	658	4 ms
Magnetic field	59	70 ms

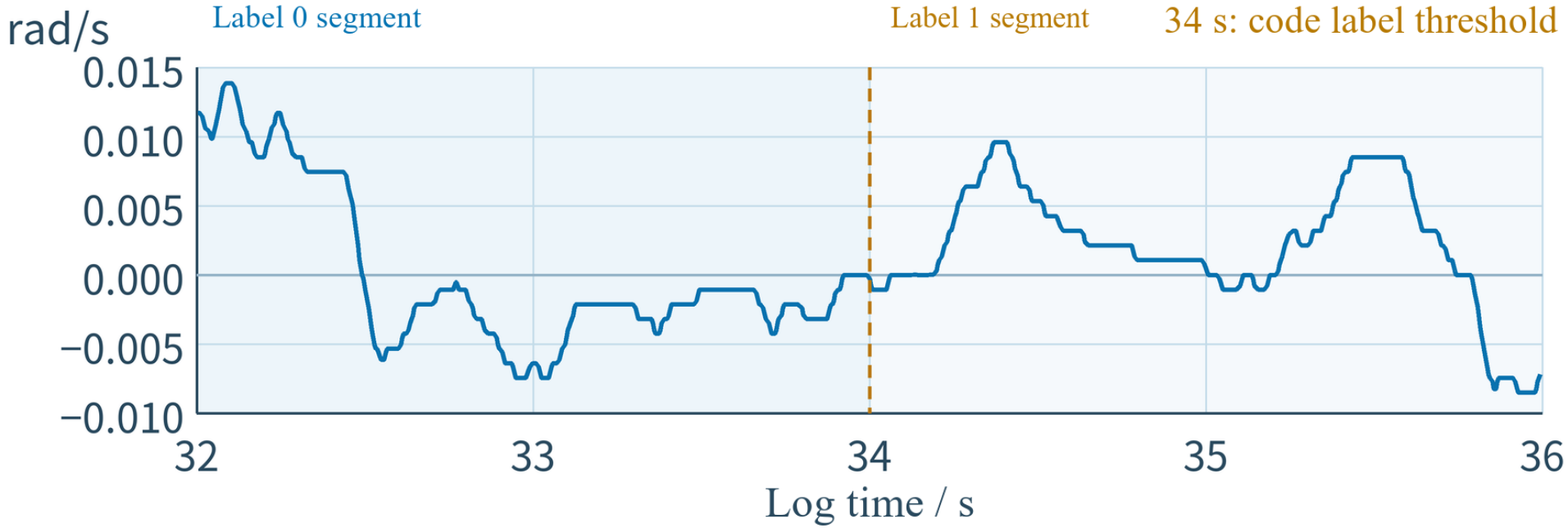
Merged times: 712; adjacent intervals range from 3 to 13 ms, with a median of 4 ms.

np.interp extends endpoint values. Check gaps and the valid time range before use.

7.3.4 Log Synchronization and Labeling: Thresholds and Curves

Code Defines the Label Boundary; Injection Records Define Events

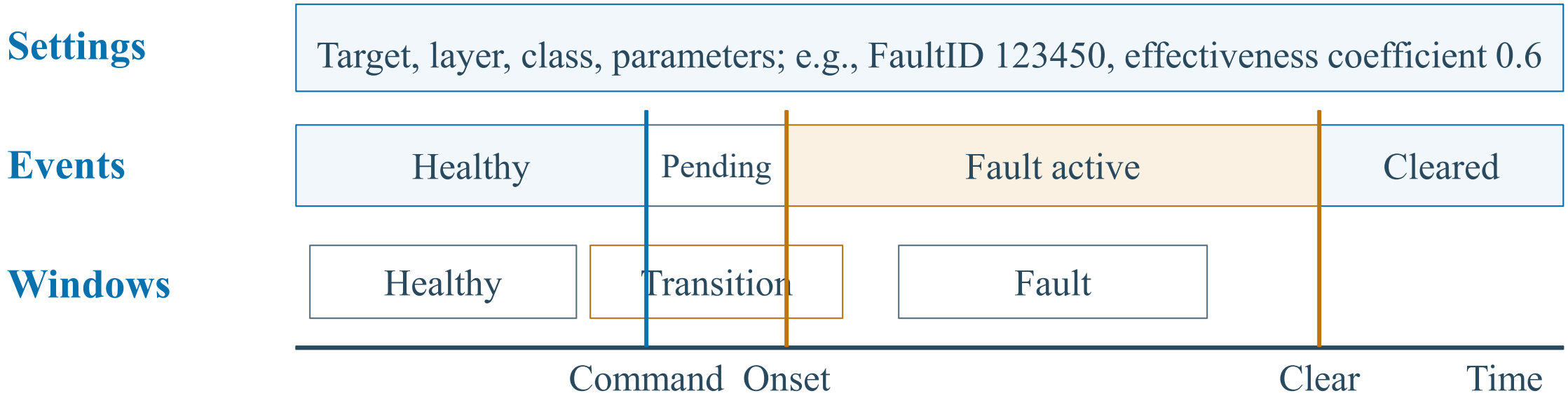
Body x-axis angular rate. The dashed line is the code threshold; this log has no injection event record (command=31000).



Retained e10 sensor_combined data: body x-axis angular rate, 32–36 s

7.3.4 Log Synchronization and Labeling: Fault Data Labels

Labels Combine Fault Class, Parameters, and Time Interval



e10 column 13, flag, uses only the 34 s threshold: 343 rows are 0, 369 are 1; the first 1 is at 34.004 s. No synchronization flag is logged; events must confirm the class.

Automatic labels: log a synchronization flag at injection; derive labels from the injection settings without manual annotation.

7.3.4 Log Synchronization and Labeling: Statistical Features

Rolling Features Capture Local Level and Variation

e10: 16 raw variables × (raw value, rolling mean, sample standard deviation) + 1 label = 49 columns.

One calculation: angular-rate samples 0, 0.01, 0.02 rad/s; use a three-row window.

$$\bar{x} = \frac{0 + 0.01 + 0.02}{3} = 0.01\,\text{rad/s} \qquad s = \sqrt{\frac{\sum_{i=1}^{3}(x_i - \bar{x})^2}{3-1}} = 0.01\,\text{rad/s}$$

Actual loop: e10 uses a 10-row rolling window. Uneven row intervals also make window duration variable.

Recalculate with the same window; results should match processed_data.csv (712×49).

Further processing: reduce dimensionality if needed; record transition windows, short initial windows, and constant channels separately.

Chapter Outline

Virtual-Real Safety Reinforcement and Fault Data Methods

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

7.3.1 Virtual-Real Safety Reinforcement and Simulation Nodes
Textbook: Sec. 7.1.4, 7.1.5.1

7.3.2 Triggers and Test Case Design
Textbook: Sec. 7.1.5.2

7.3.3 Automated Testing and Event Evidence
Textbook: Sec. 7.1.5.2

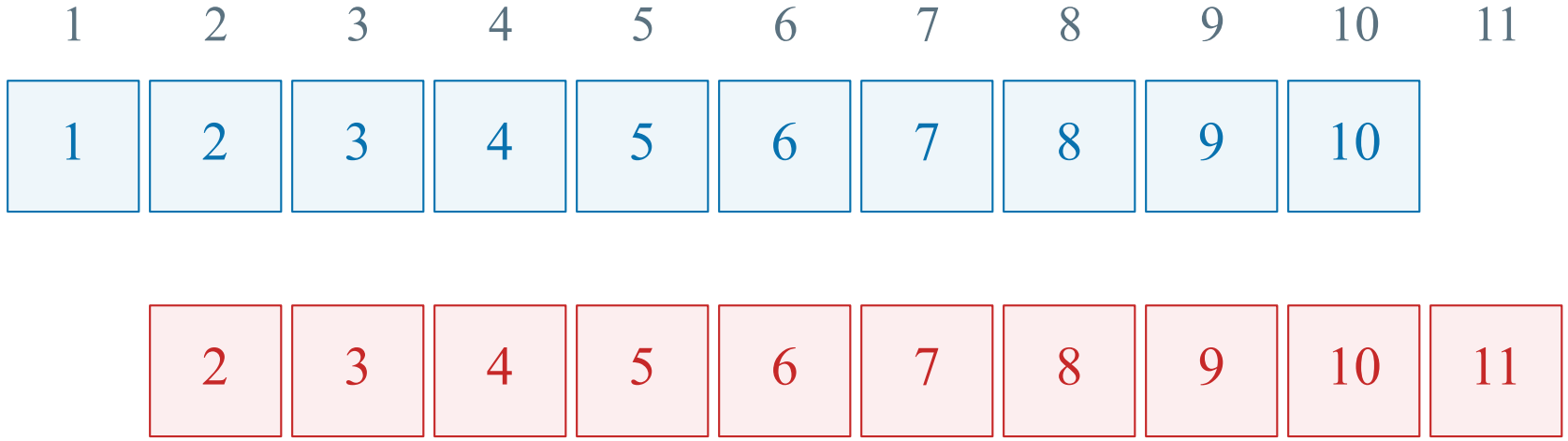
7.3.4 Log Synchronization and Labeling
Textbook: Sec. 7.1.5.4 (fault data labeling)

7.3.5 Data Splits and Quality
Textbook: Sec. 7.1.5.4

7.3.5 Data Splits and Quality: Sliding-Window Leakage

Adjacent Windows Share Nine Rows across a Random Split

Data leakage means test information enters training. Example: length 10 rows, stride 1; assign windows randomly to training and test sets.



Shared raw samples: rows 2–10; nine rows, or 90% of each window.

Window splits may test memory; use independent flights to test generalization.

7.3.5 Data Splits and Quality: Training, Validation, and Test

Split Complete Flights before Creating Windows within Sets

List complete flights → **Group by flight/condition** → **Window within each set**

Set (Example)	Complete Flights	Decisions Supported
Training	All records from F01 and F02	Learn the model; fit normalization mean and scale
Validation	All records from F03	Choose model, threshold, and hyperparameters (set before training)
Test	All records from F04	Final report; freeze model and threshold before evaluation

Keep each flight, adjacent windows, and paired healthy/fault segments in the same set.

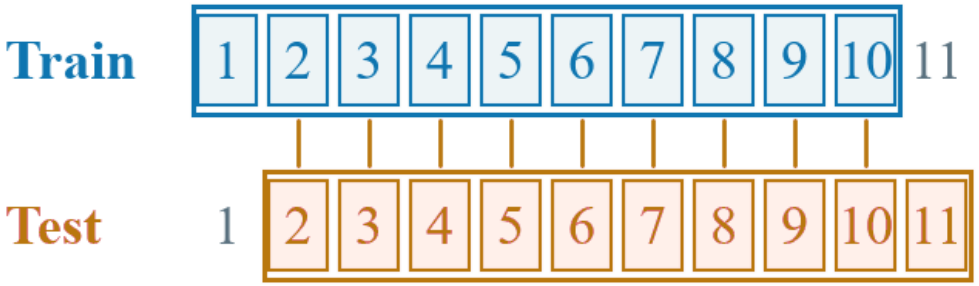
7.3.5 Data Splits and Quality: Moving Windows

Moving Windows Share Samples; Complete Flights Separate Sets

Length 10 rows, stride 1: split complete flights first, then generate windows within each set.

Window first, then split

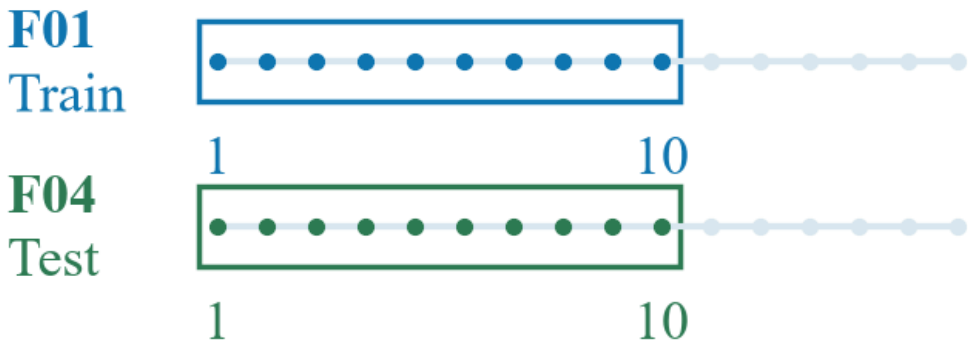
Raw row index F01



Shared: 9 / 10

Split flights, then window

F01 / F02 → Train F04 → Test



Shared across groups: 0

Overlap within a flight is allowed; keep entire flights in one group

Row-index and window illustration; fit normalization and feature selection on training data only.

7.3.5 Data Splits and Quality: Quality Checks

Establish Data Usability before Scoring Diagnostic Models

Check	Basis for Judgment	Action and Record
Time and sampling	Order, duplicates, interval distribution, gaps	Limit interpolation gaps; exclude abnormal sampling rates
Values and meaning	Units, frames, missing values, constants, saturation	e10: x-axis acceleration is all zero here; check the channel
Events and labels	Class, parameters, onset markers, transition windows	Trace original events; record labeling rules
Classes and splits	Independent flights per class, conditions, and sets	Group first; use batch simulation to prevent class imbalance

Report valid flights, exclusions, and processing rules to define where diagnostic results apply.

Section Resources

Original Sources for Data and Methods

Existing fault data: RflyMAD authors' project
https://rfly-openha.github.io/documents/4_resources/dataset.html

Log structure and timestamps: PX4 ULog format
https://docs.px4.io/v1.13/en/dev_log/ulog_file_format

Data leakage and preprocessing: official scikit-learn documentation
https://scikit-learn.org/stable/common_pitfalls.html

Original automated testing paper: Dai et al. (2021), Table 1 / Fig. 19
https://doi.org/10.1016/j.ast.2021.106727

Exercises and Discussion

Apply These Methods to New Data and Test Cases

Question 1: What records are needed to name e10 flag=1 as a fault class? What does the platform use to write the event?

Question 2: With window length 20 and stride 5, how many rows do adjacent windows share, and what fraction of each window? How should the data be split?

Question 3: Design paired healthy/fault cases for gyroscope noise. Specify the trigger and event fields.

Show the supporting records or calculations and state testable processing rules.

Reference Points

Answer with Specific Records, Values, or Case Fields

1. Label meaning: flag=1 means only after 34 s. Add FaultID, parameters, injection events, and clock mapping. e18 SendFaultEvent writes command=31000 records.

2. Overlap: 20−5=15 shared rows, or 75% per window. Split complete flights first, then window within sets; fit preprocessing on training data only.

3. Case design: same mission and environment. Inject FaultID 123543 (gyroscope noise) at 15 s in the fault group; no injection in the healthy group. Record run_id, FaultID, noise gain, command time, and onset time.

Each sample must be traceable to its raw record, event, processing rules, and data set.

Section Summary

Trustworthy Data Needs Comparability, Traceability, and Independent Tests

Cases: mission, initial values, fault, trigger, termination, records, and criteria together define a test design.

Evidence: mark command (e18 logs command=31000), onset, observation, and diagnosis separately; check both ground truth and flight-controller estimates.

Data: unify time and channel meanings before segmentation and feature extraction; retain the physical meaning and source of labels.

Evaluation: split complete flights before windowing; check events, gaps, invalid values, classes, and duplicates across sets.

Pitfalls: case count is not run count; the 34 s threshold is not a confirmed fault; command time is not onset time; a motor stall is not gradual degradation.