SIL setup

Objective:

Validate emergency return logic in a fully simulated environment before hardware integration

Components:

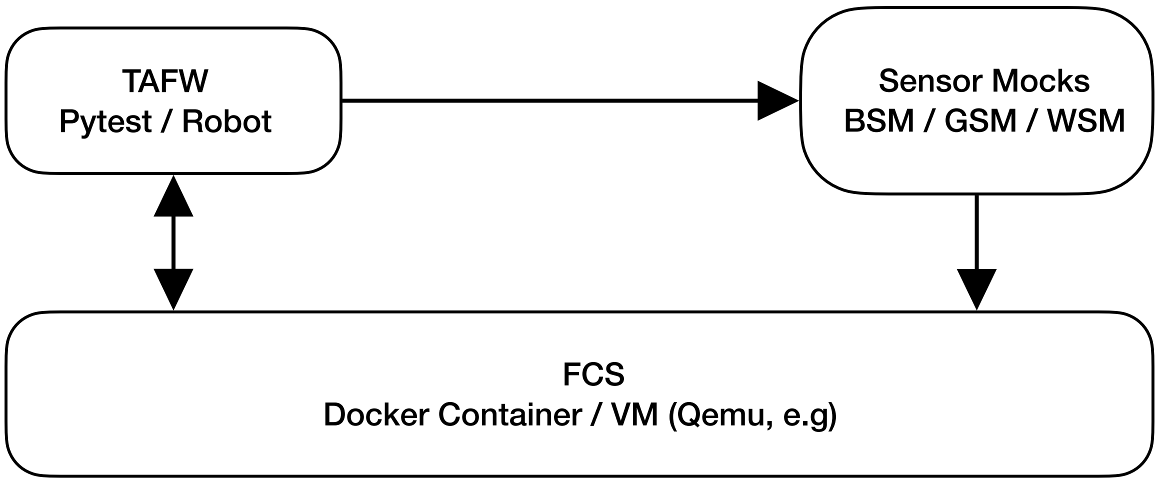
1. Flight Control Software **(FCS)** (Docker Container, VM, etc)
2. Mocking Sensor Simulation:
   - Battery Simulation mock **(BSM)**
   - GPS Simulation mock **(GSM)**
   - Wind Simulation mock **(WSM)**

3. Test automation Framework **(TAFW)**

Execution Environment:

1. Linux VM
2. Dockerized controller
3. CI integration (GitHub Actions, GitLab, Jenkins, e.g.)

Component Testing Schema:



Execution Flow:

1. Controller starts in NORMAL mode
2. Test injects simulated sensor data
3. Controller evaluates decision logic
4. Test queries status endpoint
5. Assertion validation

Usage:

1. Fast regression testing
2. Boundary testing
3. CI validation
4. Failure scenario injection

HIL setup

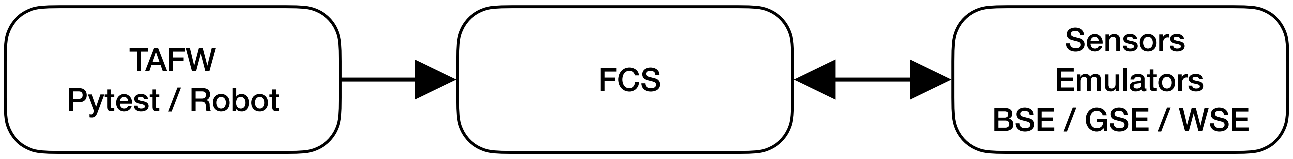
Objective:

Validate emergency return logic on real flight hardware with simulated environmental inputs

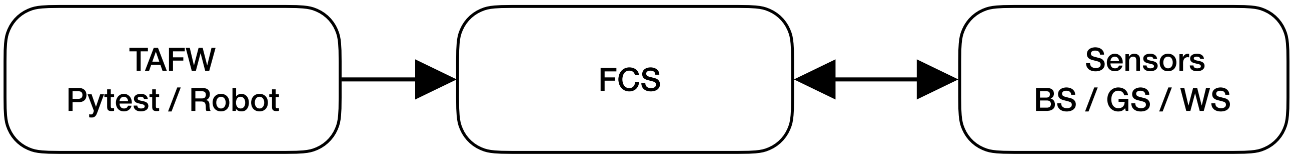
Components:

1. Flight Control Software **(FCS)** (Real device / Raspberry Pi / …)
2. Sensors:
   - Battery Sensor **(BS)**
   - GPS Sensor **(GS)**
   - Wind Sensor **(WS)**
3. Sensors Emulators:
   - Battery Sensor Emulator **(BSE)** (Analog battery voltage generator)
   - GPS Sensor Emulator **(GSE)** (UART / NMEA)
   - Wind Sensor Emulator **(WSE)** (CAN / I2C)
4. Test automation Framework **(TAFW)**

Component Testing Schema:



Integration Testing Schema:



Execution flow:

1. Test PC commands sensor emulator to inject values
2. Real controller processes sensor data
3. Mode transition is transmitted via telemetry
4. Test framework verifies correct emergency state

Usage:

1. Validates real firmware behavior
2. Detects timing issues
3. Verifies hardware abstraction layer
4. Detects integration-level failures

Safety Considerations

Emergency landing logic is treated as safety-critical functionality

Verification includes:

- Boundary strictness validation
- Mode latching behavior
- No unintended reversion to NORMAL mode
- Validation under asynchronous sensor updates
- Failure mode robustness

Continuous Verification Strategy

SIL:

- Executed on every commit
- Used for regression and coverage tracking

HIL:

- Nightly automated runs
- Pre-release validation
- Hardware regression suite

Test artifacts stored for traceability