VTOL Emergency Landing Test

Project	VTOL Flight Control Systems (FCS)
Module	Emergency Return Function
Version	1.0
Author	Dmytro Kicha
Date	18 Feb 2026

1. Introduction

This document outlines the test strategy for the automatic emergency return functionality of a VTOL aircraft. The objective is to verify the correct activation of the return logic under critical conditions and to ensure the aircraft does not initiate a return in normal or boundary situations.

2. Scope

In Scope:

- Logic for triggering the "Emergency Return" mode
- Battery State of Charge (SoC) sensor reading and interpretation
- Calculation of the current distance from the "Home" point
- Wind speed sensor reading and interpretation
- Mode persistence after emergency trigger
- Sensor update order independence

Out of Scope:

- The physical flight trajectory (PID controllers, stabilization algorithms)
- Functionality of GPS and other navigation sensors (mocked)
- Ground Control Station (GCS) GUI
- Hardware-specific timing issues

3. Test Strategy

- **Test Level:** Test Level: Unit and Integration testing
- **Test Type:** Functional, Negative testing, Boundary Value Analysis
- **Approach:** Grey-box testing. Internal logic verified through external sensor interfaces
- **Tools:** Pytest (for detailed unit tests), Robot Framework (for high-level business scenarios)

4. Acceptance Criteria

1. **Entry Criteria:**

- Sensor mocks (battery, GPS, wind) are available
- Controller API endpoints (/status, /reset, /sensor/update) are accessible
- Go controller binary (FCS) compiles successfully
- Python dependencies are installed

2. **Exit Criteria:**

- All test cases (core logic, boundaries, persistence, order independence) pass
- No unresolved Critical/High defects related to emergency return logic

5. Risks

- Inaccurate sensor mocking could mask real-world issues
- Timing dependencies between sensor updates might not reflect real asynchronous behavior

6. Test Cases

6.1 Core Emergency Logic

ID	Name	Battery (%)	Distance (km)	Wind (km/h)	Expected Mode
1	Normal flight	50.0	1.5	20.0	NORMAL
2	Low battery alone	19.0	1.5	20.0	NORMAL
3	Distance trigger	19.0	5.0	20.0	EMERGENCY
4	Wind trigger	19.0	1.5	40.0	EMERGENCY
5	Both triggers	19.0	5.0	40.0	EMERGENCY
6	Normal battery + exceeded conditions	21.0	5.0	40.0	NORMAL

6.2 Boundary Conditions

ID	Name	Battery (%)	Distance (km)	Wind (km/h)	Expected Mode
7	Battery exactly 20%	20.0	3.0	40.0	NORMAL
8	Battery slightly below 20%	19.9	3.0	40.0	EMERGENCY
9	Distance exactly 2.0 km	19.0	2.0	20.0	NORMAL
10	Distance slightly above 2.0 km	19.0	2.1	20.0	EMERGENCY
11	Wind exactly 35 km/h	35.0	1.5	35.0	NORMAL
12	Wind slightly above 35 km/h	19.0	1.5	35.1	EMERGENCY

6.3 Mode Persistence

ID	Name	Description	Expected
13	Mode persists after trigger	Trigger emergency, then send safe conditions	Mode remains EMERGENCY

6.4 Sensor Update Order Independence

ID	Name	Update order	Expected
14	GPS first	GPS → Wind → Battery	EMERGENCY
15	Wind first	Wind → GPS → Battery	EMERGENCY
16	Battery first	Battery → GPS → Wind	EMERGENCY

7. Environment

OS: Ubuntu (CI/CD), macOS/Linux (development)
Controller: Go binary compiled from /go/cmd/controller
Sensor Mocks: Python classes in /python/mocks/
Test Frameworks:

- Pytest: Detailed unit and integration tests
- Robot Framework: High-level business scenario tests
- Dependencies: Listed in /python/requirements.txt

8. CI/CD integration

Tests run automatically on every pull request to main/develop branches:

- Compile Go controller binary
- Setup Python 3.13 with dependencies
- Run test suite (configurable between Pytest and Robot Framework)
- Upload test reports as artifacts