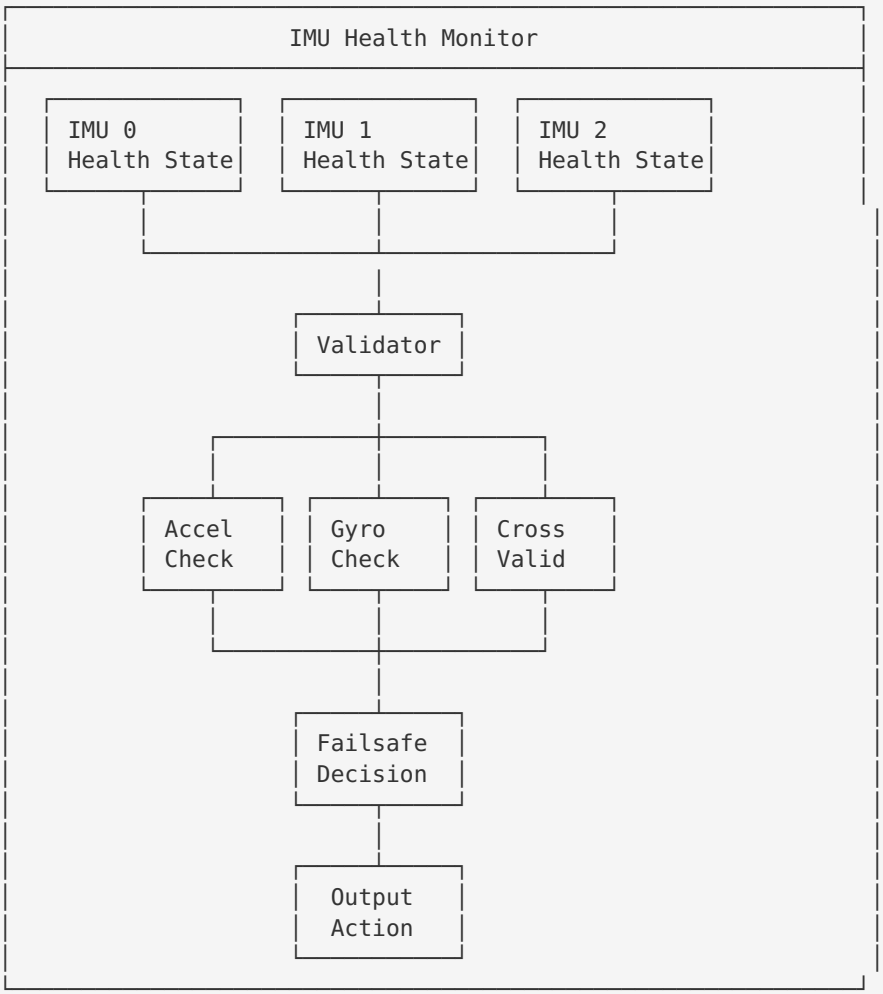
IMU Health Monitor - Software Design Document

Document Information

Field	Value
Document ID	SDD-IMU-001
Version	1.0.0
Standard	DO-178C DAL A
Date	2026-01-31

1. Architecture Overview

1.1 Module Structure



1.2 Data Flow

1. IMU readings received from HAL layer
2. Individual sensor validation (accel + gyro)
3. Cross-validation between sensors
4. Health history update
5. Status determination
6. Primary IMU selection
7. Failsafe action determination
8. Return action to flight controller

2. Data Structures

2.1 Monitor State

```
typedef struct {
    IMU_Health_State imu_health[3];  // Per-IMU health tracking
    uint8_t primary_imu;              // Currently selected IMU
    uint8_t active_imu_count;         // Number of connected IMUs
    bool system_healthy;              // Overall health flag
    bool failsafe_active;             // Failsafe engaged
    Failsafe_Action pending_action;   // Current action
    uint32_t total_cycles;            // Cycle counter
    uint32_t error_count;             // Error counter
} Monitor_State;
```

2.2 IMU Health State

```
typedef struct {
    IMU_Status status;                // Current status
    int32_t accel_variance;           // Accel noise metric
    int32_t gyro_variance;            // Gyro noise metric
    uint8_t consecutive_failures;     // Failure counter
    uint8_t health_history[8];        // Circular buffer
    uint8_t history_index;            // Buffer index
    uint32_t last_update_us;          // Last update time
} IMU_Health_State;
```

3. Algorithm Design

3.1 Accelerometer Validation

```
Algorithm: validate_accel_reading(reading)
1. If reading.valid == false:
   Return FALSE

2. mag_squared = x² + y² + z²  (using 64-bit intermediate)
3. expected = GRAVITY² / 1000
4. diff = |mag_squared - expected|
5. tolerance = TOLERANCE² / 1000
6. Return (diff <= tolerance)
```

Rationale: Uses squared magnitude to avoid expensive sqrt operation. 64-bit intermediate prevents overflow.

3.2 Gyroscope Validation

```
Algorithm: validate_gyro_reading(reading)
1. If reading.valid == false:
   Return FALSE

2. mag_squared = x² + y² + z²
3. max_stationary = THRESHOLD × 100
4. Return (mag_squared <= max_stationary)
```

Rationale: At rest, gyroscope should read near-zero. Large values indicate failure.

3.3 Cross-Validation

```
Algorithm: cross_validate_imus(reading1, reading2)
1. If either invalid:
   Return INT32_MAX

2. diff = |r1.x - r2.x| + |r1.y - r2.y| + |r1.z - r2.z|
3. Return diff
```

Rationale: L1 norm is computationally efficient and sufficient for failure detection.

3.4 Health History

Circular buffer of 8 samples tracks recent health:
- Each sample: 1 = healthy, 0 = unhealthy
- Status determined by count of healthy samples:
- ≥6 healthy → HEALTHY
- ≥3 healthy → DEGRADED
- <3 healthy → FAILED

3.5 Failsafe Decision

```
Algorithm: determine_failsafe_action(state)
1. Count healthy and degraded IMUs
2. If healthy >= 2: Return NONE
3. If healthy == 1: Return WARN
4. If degraded >= 1: Return LAND_IMMEDIATELY
5. Return TERMINATE
```

4. Timing Analysis

4.1 Call Graph (Critical Path)

```
imu_monitor_update [85 µs]
├── validate_accel_reading ×3 [24 µs]
│   └── compute_magnitude_squared ×3 [15 µs]
├── validate_gyro_reading ×3 [18 µs]
│   └── compute_magnitude_squared ×3 [15 µs]
├── update_health_history ×3 [12 µs]
├── determine_imu_status ×3 [24 µs]
│   └── count_healthy_samples ×3 [15 µs]
├── cross_validate_imus [12 µs]
├── select_primary_imu [10 µs]
└── determine_failsafe_action [6 µs]
```

4.2 Loop Bounds

Loop	Max Iterations	Location
IMU iteration	3	imu_monitor_update
History iteration	8	count_healthy_samples
Nested init	3 × 8 = 24	imu_monitor_init

5. Memory Map

5.1 Stack Frame Analysis

Function	Local Variables	Max Stack
imu_monitor_update	32 bytes	64 bytes
validate_accel_reading	24 bytes	48 bytes
compute_magnitude_squared	32 bytes	32 bytes
Total (worst case)		~150 bytes

5.2 Static Data

All data is passed via parameters. No static/global variables used.

6. Error Handling

6.1 Invalid Input

- NULL pointers: Undefined (precondition violation)
- Invalid IMU count: Clamped to MAX_IMU_COUNT
- Invalid readings: Flagged via `valid` field

6.2 Sensor Failures

Failure Mode	Detection	Response
Stuck sensor	Cross-validation	Mark DEGRADED
Noise spike	History filtering	Ignore transient
Complete failure	All checks fail	Mark FAILED
Multiple failures	< 2 healthy	LAND_IMMEDIATELY
Total loss	0 healthy	TERMINATE

7. Verification Approach

7.1 Unit Testing

- Test each function in isolation
- Cover all code paths
- Boundary value analysis
- MC/DC coverage required

7.2 Integration Testing

- Test with simulated IMU data
- Inject failures and verify response
- Timing verification

7.3 Hardware-in-Loop Testing

- Test with actual flight controller
- Verify real-time performance
- EMI/environmental testing