IMU Health Monitor - Integration Guide

1. Prerequisites

1.1 Hardware Requirements

- ARM Cortex-M4 or M7 processor (STM32F4/F7/H7 series)
- Minimum 256 KB RAM
- Clock frequency ≥ 168 MHz
- FPU (optional, not required)

1.2 Software Requirements

- ARM GCC toolchain (arm-none-eabi-gcc)
- C11 compliant compiler
- Ardupilot HAL layer (for sensor access)

2. File Integration

2.1 Copy Files

```
# Create directory in Ardupilot libraries
mkdir -p libraries/AP_IMU_Monitor

# Copy generated files
cp generated_c/imu_health_monitor.h libraries/AP_IMU_Monitor/
cp generated_c/imu_health_monitor.c libraries/AP_IMU_Monitor/
```

2.2 Update Build System

Add to `wscript` or `CMakeLists.txt` :

```
# wscript
conf.env.INCLUDES += ['libraries/AP_IMU_Monitor']
```

3. Code Integration

3.1 Header Include

```
#include <AP_IMU_Monitor/imu_health_monitor.h>
```

3.2 State Declaration

```
// In AP_InertialSensor.h or similar
class AP_InertialSensor {
private:
    Monitor_State _imu_monitor_state;
    // ...
};
```

3.3 Initialization

```
void AP_InertialSensor::init()
{
    // Initialize IMU monitor
    int32_t result = imu_monitor_init(&_imu_monitor_state);
    if (result != 0) {
        hal.console->printf("IMU Monitor init failed\n");
    }

    // ... rest of initialization
}
```

3.4 Main Loop Integration

```
void AP_InertialSensor::update()
{
    // Collect IMU readings
    IMU_Reading readings[MAX_IMU_COUNT];
    uint8_t imu_count = 0;

    for (uint8_t i = 0; i < _backend_count && i < MAX_IMU_COUNT; i++) {
        if (_backends[i] != nullptr && _backends[i]->healthy()) {
            Vector3f accel = _backends[i]->get_accel();
            Vector3f gyro = _backends[i]->get_gyro();

            // Convert to fixed-point (mm/s² and mrad/s)
            readings[imu_count].accel.x = (int32_t)(accel.x * 1000.0f);
            readings[imu_count].accel.y = (int32_t)(accel.y * 1000.0f);
            readings[imu_count].accel.z = (int32_t)(accel.z * 1000.0f);

            readings[imu_count].gyro.x = (int32_t)(gyro.x * 1000.0f);
            readings[imu_count].gyro.y = (int32_t)(gyro.y * 1000.0f);
            readings[imu_count].gyro.z = (int32_t)(gyro.z * 1000.0f);

            readings[imu_count].timestamp_us = AP_HAL::micros();
            readings[imu_count].valid = true;
            imu_count++;
        }
    }

    // Run health monitor
    Failsafe_Action action = imu_monitor_update(
        &_imu_monitor_state,
        readings,
        imu_count,
        AP_HAL::micros()
    );

    // Handle failsafe action
    _handle_failsafe_action(action);
}
```

3.5 Failsafe Handler

```
void AP_InertialSensor::_handle_failsafe_action(Failsafe_Action action)
{
    switch (action) {
        case FAILSAFE_ACTION_NONE:
            // Normal operation - nothing to do
            break;

        case FAILSAFE_ACTION_WARN:
            // Log warning, maybe notify GCS
            GCS_SEND_TEXT(MAV_SEVERITY_WARNING,
                "IMU Health: Single IMU remaining");
            break;

        case FAILSAFE_ACTION_SWITCH_IMU:
            // Primary IMU switched - log event
            GCS_SEND_TEXT(MAV_SEVERITY_INFO,
                "IMU Health: Switched to backup IMU %d",
                _imu_monitor_state.primary_imu);
            break;

        case FAILSAFE_ACTION_LAND_IMMEDIATELY:
            // Critical - initiate emergency landing
            GCS_SEND_TEXT(MAV_SEVERITY_CRITICAL,
                "IMU FAILURE: Emergency landing initiated");
            AP::vehicle()->set_mode(Mode::Number::LAND,
                ModeReason::IMU_FAILURE);
            break;

        case FAILSAFE_ACTION_TERMINATE:
            // Catastrophic - all IMUs failed
            GCS_SEND_TEXT(MAV_SEVERITY_EMERGENCY,
                "TOTAL IMU FAILURE: Flight termination");
            AP::vehicle()->arming().disarm(AP_Arming::Method::CRASH);
            break;
    }
}
```

4. Telemetry Integration

4.1 Diagnostic Reporting

```
void AP_InertialSensor::send_imu_health_status(mavlink_channel_t chan)
{
    for (uint8_t i = 0; i < _imu_monitor_state.active_imu_count; i++) {
        Diagnostic_Report report;
        imu_get_diagnostic(&_imu_monitor_state, i, nullptr, &report);

        // Send via MAVLink (custom message or use STATUS_TEXT)
        mavlink_msg_imu_health_send(
            chan,
            report.imu_index,
            report.status,
            report.accel_magnitude,
            report.gyro_magnitude
        );
    }
}
```

5. Testing

5.1 Bench Testing

```c
// Test harness
void test_imu_monitor()
{
    Monitor_State state;
    imu_monitor_init(&state);

    // Create normal readings
    IMU_Reading readings[3];
    for (int i = 0; i < 3; i++) {
        readings[i].accel.x = 0;
        readings[i].accel.y = 0;
        readings[i].accel.z = 9810;  // 1g in mm/s²
        readings[i].gyro.x = 0;
        readings[i].gyro.y = 0;
        readings[i].gyro.z = 0;
        readings[i].timestamp_us = 0;
        readings[i].valid = true;
    }

    // Run multiple cycles
    for (int cycle = 0; cycle < 100; cycle++) {
        Failsafe_Action action = imu_monitor_update(
            &state, readings, 3, cycle * 2500);
        assert(action == FAILSAFE_ACTION_NONE);
    }

    // Inject failure
    readings[0].valid = false;
    for (int cycle = 0; cycle < 10; cycle++) {
        imu_monitor_update(&state, readings, 3, 100000 + cycle * 2500);
    }

    // Verify degraded state
    assert(state.imu_health[0].status == IMU_STATUS_FAILED);
    printf("IMU Monitor tests passed!\n");
}
```

6. Troubleshooting

6.1 Common Issues

Issue	Cause	Solution
False positives	Noisy IMU	Increase thresholds
Missed failures	Thresholds too loose	Decrease thresholds
Timing violations	Slow processor	Optimize or reduce rate
Stack overflow	Too many locals	Check stack size

6.2 Debug Logging

```
#ifdef IMU_MONITOR_DEBUG
    printf("IMU[%d] status=%d accel_var=%d gyro_var=%d\n",
        i,
        state->imu_health[i].status,
        state->imu_health[i].accel_variance,
        state->imu_health[i].gyro_variance);
#endif
```

7. Certification Considerations

7.1 DO-178C Compliance

- This code is generated for DO-178C Level A
- Unit tests with MC/DC coverage required for certification
- Hardware-in-loop testing required
- DER review required before flight

7.2 Modification Restrictions

⚠️ **WARNING**: This code is auto-generated and certified as-is.
Any modifications require:
1. Full regression testing
2. Updated verification documentation
3. Re-certification review