Project Title: System Verification and Validation Plan for Attitude Check

Adrian Sochaniwsky

February 10, 2024

Revision History

Date	Version	Notes
2024/02/13	1.0	Initial draft

Contents

List of Tables

List of Figures

1 Symbols, Abbreviations, and Acronyms

For symbols and units see the Section 1 of the SRS (Sochaniwsky, 2024). Table 1 defines the abbreviations and acronyms used in this document.

Table 1: Table of Abbreviations and Acronyms

symbol	**description**
accel	Accerleromter
gyro	Gyroscope
IMU	Inertial Measurement Unit
mag	Magnetometer
SRS	Software Requirements Specification
VnV	Verification and Validation

2 Introduction

A verification and validation (VnV) plan is a document that describes the objectives, scope, methods, and criteria for verifying and validating a software product. Verification is the process of checking whether the software meets the specified requirements and design specifications. Validation is the process of checking whether the software meets the user's needs and expectations. A VnV plan helps to ensure the quality, reliability, and functionality of the software, as well as to identify and correct any defects or errors before the software is released or deployed.

This document details the plan for verification and validation of Attitude Check. Section 3 provides an overview and objectives of this document. Section 4 discusses the methods of achieving the objectives. Section 5 covers the test descriptions.

3 General Information

3.1 Summary

Attitude Check is an IMU-based attitude estimation algorithm. It consumes sensor data and produces and estimate of the current orientation of the sensor relative to the Earth.

3.2 Objectives

The objectives of this Verification and Validation plan are the following:

- Build confidence in the software correctness.
- Demonstrate the software's ability to accurately estimate orientation.

Objectives that are not included in the scope of this document:

- Demonstration of adequate usability.
- Verification of external libraries.

It will be assumed that the documentation of Attitude Check is adequate to facilitate adequate usability for developers wishing to use it. Furthermore, it is assumed that external libraries have been independently verified and validated.

3.3 Relevant Documentation

See the Software Requirements Specification (Sochaniwsky, 2024) for Attitude Check, it details the goals, requirements, assumptions, and theory of the software.

4 Plan

This section will detail the plan for the verification of the documentation and software for Attitude Check. The primary items that will be verified are: SRS, design, VnV, implementation.

4.1 SRS Verification Plan

The SRS (Sochaniwsky, 2024) will be verified via feedback from a domain expert and secondary reviewers. The following is a checklist for SRS review:

- ☐ Problem Statement: confirm it is a valid statement.
- ☐ Goal Statement: confirm it is a valid goal.
- ☐ Physical system: check for missing descriptions.
- ☐ Assumptions: confirm assumptions are specific and meaningful.
- ☐ Theory: confirm instance models and all required theory is logical and complete enough for development.
- ☐ Requirements: confirm that the requirements are reasonably complete for testers and clients.

4.2 Design Verification Plan

To verify the design, a domain expert and secondary reviewer will use the following checklist:

- ☐ Inspect the design document for the following items:
  -
- ☐ Create a high-level diagram that represents the relationships between modules. This is intended to find gaps in the design.

4.3 Verification and Validation Plan Verification Plan

This VnV plan will be verified via inspection from a domain expert and a secondary review. The inspection will look for the following items:

- ☐ Confirm all sections contain a verification checklist or description.
- ☐ Inspect system-level test cases for coverage of SRS requirements and goal statements.

4.4 Implementation Verification Plan

A combination of static and dynamic tests will be used to verify Attitude Check. Cppcheck (cpp, 2023) is a static code checking tool that will be employed to analyze the code for undefined behavior and poor design constructs.

System and unit level tests will be used to dynamically verify Attitude Check. Section 5 outlines the system-level tests. Section 6 details the unit tests.

4.5 Automated Testing and Verification Tools

- Cmake (cma, 2023) is a build system tool that will be used to simplify building and testing Attitude Check.
- GoogleTest is a unit testing framework that support c++ code. It also includes a mocking framework. See https://github.com/google/googletest.
- GCOV, LCOV are tools for calculating and displaying unit test coverage metrics.
- Valgring is a suite of tools for memory and performance profiling. It will be used to check for memory leaks and efficient memory utilization.
- Uncrustify is a tool that applies code formatting rules based on a configuration file. It will be used before every commit to the repo.
- Github PR checklists will help reviewers/developers see and check that all rules and tests are completed before code enter a protected branch (i.e. main).

- Github CI workflow will automate regression tests and checks that Attitude Check builds are passing before a PR is merged into a protected branch.
- Docker is a mechanism to containerize applications. It will be used to ensure installability and usability of the project.

4.6 Software Validation Plan

For the scope of this project, software validation will consist of benchmarking against existing algorithms. The Root Mean Squared Error (RMSE) of Attitude Check should lie within ϵ of the RMSE of attitude estimators implemented by Laidig (2023). The data will come from the BROAD dataset (Laidig et al., 2021).

5 System Test Description

5.1 Tests for Functional Requirements

5.1.1 Inputs and Outputs

This section covers requirements R1 and R4 of the SRS. This includes the input and output requirements for Attitude Check. For input constraints see Section 4.2.8 of the SRS.

T1: **Input/Output Test**

Control: Automatic

Initial State: Uninitialized

Test Case Derivation: N/A

How test will be performed: At each step, apply the inputs and assert the output.

5.1.2 Calculations

This section covers requirements R2 and R3 of the SRS. This includes the input and output requirements for Attitude Check. For input constraints see Section 4.2.8 of the SRS.

Step	Input	Output
1	Initialize with: $\Delta t = 1.0, {}^{E}\mathbf{b} = [1.0, 1.0, 1.0]$, $\gamma = 0.6, \text{outputType} = \text{Quat}$	Get params method matches the inputs.
2	One set of [accel, gyro, mag] measurements.	Orientation output should match output format from above
3	Initialize with: $\Delta t = 1.0, {}^{E}\mathbf{b} = [1.0, 1.0, 1.0]$, $\gamma = 0.6, \text{outputType} = \text{Rot}$	Get params method matches the inputs.
4	One set of [accel, gyro, mag] measurements.	Orientation output should match output format from above
5	Initialize with: $\Delta t = 1.0, {}^{E}\mathbf{b} = [1.0, 1.0, 1.0]$, $\gamma = 0.6, \text{outputType} = \text{Euler}$	Get params method matches the inputs.
6	One set of [accel, gyro, mag] measurements.	Orientation output should match output format from above

T2: **Orientation with Magnetometer**

Control: Automatic

Initial State: Uninitialized

Step	Input	Output
1	One complete set of valid initialization inputs.	-
2	[accel, gyro, mag] measurements.	Calculate RMSE of output vs. ground truth
3	Repeat step 2 for all measurements in dataset	-
4	Calculate average RMSE	Assert average error is within the tolerance.

Test Case Derivation: Benchmark dataset with labelled ground truth orientation for each set of sensor measurements.

How test will be performed: At each step, apply the inputs and assert the output.

T3: **Orientation without Magnetometer**

Control: Automatic

Initial State: Uninitialized

Step	Input	Output
1	One complete set of valid initialization inputs.	-
2	[accel, gyro, mag] measurements.	-
3	[accel, gyro] measurements.	Calculate RMSE of output vs. ground truth
4	Repeat step 3 for all measurements in dataset	-
5	Calculate average RMSE	Assert average error is within the tolerance.

Test Case Derivation: Benchmark dataset with labelled ground truth orientation for each set of sensor measurements.

How test will be performed: At each step, apply the inputs and assert the output.

Note: Only Pitch and Roll components are evaluated.

5.2 Tests for Nonfunctional Requirements

5.2.1 Accuracy

T4: **test-a1**

Type: Automatic

This test refers to Section 5.1.2.

5.2.2 Usability

T5: **test-u1**

Type: Manual

Initial State: N/A

Input/Condition: N/A

Output/Result: Successful integration with existing code.

How test will be performed: A manual inspection of the code, specifically the header file. Can a developer read the documention/header file and correctly interact with the functions/objects?This test passes if this is true.

5.2.3 Maintainability

T6: **test-m1**

Type: Manual

Initial State: N/A

Input/Condition: N/A

Output/Result: Successful addition of a likely change within the threshold defined in NFR3 of the SRS.

How test will be performed: If a likely change needs to be added, measure the development time and compare it to the total development time. If it is less than the threshold specified in the SRS, this test passes.

5.2.4 Portability

T7: **test-p1**

Type: Manual

Initial State: N/A

Input/Condition: One set of initialization inputs, 1 set of measurements

Output/Result: Result of compilation and execution.

How test will be performed: Build the project, initialize and input 1 set of measurements. Test is successful if the build has no errors and if the code produces the same result on Windows 11, macOS, and Ubuntu.

Table 2: Relation of Test Cases and Requirements.

	R1	R2	R3	R4	NFR1	NFR2	NFR3	NFR4
T1	X			X				
T2		X			X			
T3			X		X			
T4					X			
T5						X		
T6							X	
T7								X

5.3 Traceability Between Test Cases and Requirements

6 Unit Test Description

6.1 Unit Testing Scope

6.2 Tests for Functional Requirements

6.2.1 Module 1

T7: **Inputs: Invalid time**

Initial State: ...

Input: ...

Output: ...

Test Case Derivation: ...

How test will be performed: ...

T7: **Inputs: Invalid mag ref**

Initial State: ...

Input: ...

Output: ...

Test Case Derivation: ...

How test will be performed: ...

T7: **Inputs: Invalid gamma**

Initial State: ...
Input: ...
Output: ...
Test Case Derivation: ...
How test will be performed: ...

T7: **Inputs: Invalid output type**

Initial State: ...
Input: ...
Output: ...
Test Case Derivation: ...
How test will be performed: ...

T7: **Inputs: Valid**

Initial State: ...
Input: ...
Output: ...
Test Case Derivation: ...
How test will be performed: ...

6.3 Tests for Nonfunctional Requirements

6.3.1 Module ?

1. test-id1

   Type:
   Initial State:
   Input/Condition:

Output/Result:

How test will be performed:

2. test-id2

   Type: Functional, Dynamic, Manual, Static etc.

   Initial State:

   Input:

   Output:

   How test will be performed:

6.3.2 Module ?

...

6.4 Traceability Between Test Cases and Modules

References

Cmake. https://cmake.org/, 2023.

Cppcheck. https://cppcheck.sourceforge.io/, 2023.

Daniel Laidig. Berlin Robust Orientation Estimation Assessment Dataset (BROAD). https://github.com/dlaidig/broad/tree/main, 2023.

Daniel Laidig, Marco Caruso, Andrea Cereatti, and Thomas Seel. Broad—a benchmark for robust inertial orientation estimation. *Data*, 6(7), 2021. ISSN 2306-5729. doi: 10.3390/data6070072. URL https://www.mdpi.com/2306-5729/6/7/72.

Adrian Sochaniwsky. System requirements specification for attitude check. https://github.com/adrian-soch/attitude_check/blob/7c8cd7387ab120dd1267b56902499314c696e42b/docs/SRS/SRS.pdf, 2024.

7 Appendix

7.1 Symbolic Parameters

The definition of the test cases will call for SYMBOLIC_CONSTANTS. Their values are defined in this section for easy maintenance.