Verification and Validation Report: RwaveDetection

Junwei Lin

May 8, 2025

1 Revision History

Date	Version	Notes
April 11, 2025	1.0	Creation

2 Symbols, Abbreviations and Acronyms

symbol	description
T	Test

Contents

List of Tables

List of Figures

This document presents the Verification and Validation (VnV) report for the project RwaveDetection, detailing the results of the system's verification against functional and nonfunctional requirements, as well as the validation of its correctness and quality.

3 Functional Requirements Evaluation

All functional requirements have been verified through unit testing using the GoogleTest framework. Each requirement has corresponding test cases designed to validate correct behavior under both expected and edge-case conditions. The completeness of this coverage is documented in the traceability matrix provided in Section 8, demonstrating that all functional requirements are adequately tested.

4 Nonfunctional Requirements Evaluation

4.1 Usability

A comprehensive guide for compilation and usage is provided in the project's README.md file, ensuring ease of use for developers and users.

4.2 Maintainability

To ensure maintainability, `cpplint` and `cppcheck` were used for static code analysis. `cpplint` checks for style and formatting issues, ensuring adherence to a consistent coding standard. `cppcheck` detects potential bugs and performance issues.

4.3 Portability

The system has been tested and verified on Ubuntu (including Docker environments), Debian, and Windows 10. It works consistently across these platforms, ensuring portability and compatibility.

4.4 Reusability

Through code reviews, it was confirmed that each module is designed with reusability in mind, ensuring that components can be easily reused in future projects.

5 Unit Testing

A comprehensive unit testing process was conducted to validate the functionality, robustness, and correctness of the system's core components. A total of 89 unit test cases were executed, covering modules including R-wave detection, I/O processing, general digital filtering, root mean square error (RMSE) evaluation, mathematical operations, and data structures.

- **Total Test Cases:** 89
- **Passed:** 89
- **Failed:** 0
- **Execution Time:** Approximately 2.18 seconds (with the longest individual test taking about 1.89 seconds)

All unit tests passed successfully, demonstrating high reliability and stability of the implemented modules.

6 Changes Due to Testing

Testing revealed the need for more accessible runtime information, leading to the addition of a logger module. To facilitate testing and debugging, the logger was configured to output messages to a file instead of standard output.

7 Automated Testing

Automated testing is integrated into the development workflow via GitHub Actions. The workflow is triggered on each push or pull request and performs the following checks:

- **cpplint:** Checks adherence to the Google C++ Style Guide.
- **cppcheck:** Performs static analysis to detect potential bugs and code quality issues.
- **Google Test:** Executes the full suite of unit tests described previously.
- **LCOV:** Generates code coverage reports to monitor test completeness.

This automated setup ensures consistent code quality, early bug detection, and continuous test coverage monitoring throughout the development lifecycle.

8 Trace to Requirements

	FR1	FR2	FR3	FR4	FR5	FR6	FR7	FR8	NFR1	NFR2	NFR3	NFR4
TC_IIR_FIR	X											
TC_BUT_CHEB		X	X									
TC_SQUARING				X								
TC_THRESHOLDING					X							
TC_RMSE						X						
TC_ALG							X	X				
TC_USABILITY									X			
TC_MAINTAINABILITY										X		
TC_PORTABILITY											X	
TC_REUSABILITY												X

Table 1: Traceability Matrix Showing the Connections Between Requirements and System Test Cases

9 Trace to Modules

	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10
RegularizeTest		X								
LfilterTestSuite		X								
ButterTest			X							
RWaveDetectionTest					X		X			
calRMSESuite						X				
RWaveDetectionFilterTest					X		X			
calSquareSuite				X						
read	X							X		
write	X							X		
ErrTest									X	
outputTest										X

Table 2: Traceability Matrix Showing the Connections Between Unit Tests and Modules

10 Code Coverage Metrics

Code coverage for this project is evaluated using the `lcov` tool. Automated coverage reports are generated and uploaded through GitHub Actions to Codecov. The coverage dashboard can be accessed at:

https://app.codecov.io/gh/Lychee-acaca/CAS741

As of the current assessment, the overall test coverage is **95.01%**. The coverage statistics for each major source file are summarized below:

File	Lines Covered / Total	Coverage (%)
`RwaveDetect.cpp`	15 / 15	100.00%
`annRMSE.cpp`	18 / 19	94.74%
`annRMSE.hpp`	1 / 1	100.00%
`dataStructure.hpp`	91 / 94	96.81%
`generalDigitalFilter.cpp`	24 / 24	100.00%
`generalDigitalFilter.hpp`	10 / 10	100.00%
`io_processing.cpp`	49 / 55	89.09%
`io_processing.hpp`	3 / 3	100.00%
`logger.cpp`	33 / 37	89.19%
`logger.hpp`	12 / 12	100.00%
`mmath.cpp`	50 / 52	96.15%
`pantomp.cpp`	127 / 134	94.78%
`pantomp.hpp`	5 / 5	100.00%

This high level of coverage demonstrates that most of the critical logic in the project has been thoroughly tested.