System Verification and Validation Plan for Optimal EM Placement

Hussein Saad

April 15, 2025

Revision History

Date	Version	Notes
April 15, 2025	1.2	Implement instructor suggestions
April 13, 2025	1.1	Implement domain expert suggestions
February 23, 2025	1.0	Initial draft

Contents

1 Symbols, Abbreviations, and Acronyms

symbol	description
T	Test
EM	Electromagnet
VnV	Verification and Validation
SRS	Software Requirements Specification
PV	Positive value required
TL	Input parameter too large
IT	Invalid type

This document provides the road-map of the verification and validation plan for Optimal EM Placement. The plan ensures the requirements and models in the SRS document are correctly implemented. It starts with introducing general information about the program in Section 2, followed by a plan, and then an outline of system tests in Section 4. Finally, a description of unit tests for functional and non-functional requirements is provided in Section 5.

2 General Information

2.1 Summary

This document outlines the verification and validation plan for Optimal EM Placement, a program that calculates the optimal positions of EM actuators that yield the highest system manipulability, as defined in the SRS.

2.2 Objectives

The main purpose of this document is to define how validation will be performed for the requirements outlined in the SRS document. The verification plan includes strategies to ensure the correct execution of system requirements testing. In particular, the VnV document devises procedures to confirm that intermediate data and computations are approximately correct, and conform to the theoretical expectations laid out in the SRS. The optimal EM positions are solved for by the `cvxpy` library — a popular open-source convex optimization library — which is assumed to be reliable and correct for the purposes of this project. In general, we aim to build maximum confidence in the correctness of our program. Additionally, the document outlines methods to verify that the input interface and output format are sufficiently accessible and interpretable by the intended users, as defined in the SRS. However, extensive usability verifications, beyond what is necessary for the outlined user backgrounds, is not within scope of this project.

2.3 Challenge Level and Extras

This project is an advanced project as it is the product of research work that will later be submitted for publication in an academic conference/journal.

2.4 Relevant Documentation

The Problem Statement motivates the program while the SRS provides information about the requirements of the functioning program. The MG and MIS documents contain architecture and design rationale.

3 Plan

This section outlines the testing plan for the Optimal EM Placement program, starting with the team in Section 3.1, followed by the SRS verification plan, design verification plan, VnV verification plan, implementation verification plan, automated testing, ending with verification tools in Section 3.6.

3.1 Verification and Validation Team

Name	Document	Role	Description
Hussein Saad	All	Author	Execution and management of VnV document and tests.
Dr. Spencer Smith	All	Instructor	Review all documents.
Uriel Cruz	All	Domain Expert	Review documents and provide technical feedback.
Alaap Grandhi	VnV	Secondary Reviewer	Review the VnV plan and suggest improvements.
Dr. Matthew Giamou	All	External Reviewer	Review documents and provide technical feedback.

Table 1: Verification Team

3.2 SRS Verification Plan

The Optimal EM Placement SRS document will be verified primarily through ad hoc feedback from reviewers. Document design and alignment with course goals will be reviewed by the instructor (Dr. Spencer Smith), while more technical details will benefit from the comments of the Domain Expert Reviewer (Uriel Cruz) and the External Reviewer (Dr. Matthew Giamou). After the creation/update of the SRS document, reviewers will provide the author with feedback through GitHub issues. The author will implement the suggested changes as necessary. In addition, the SRS checklist created by Dr. Smith will be used to be used to verify that the document adheres to all specified criteria. Consequently, both reviewer feedback and the checklist will ensure that the SRS meets the necessary technical and design standards.

3.3 Design Verification Plan

The design documents, Module Guide (MG) and Module Interface Specification (MIS), will be verified through inspection by reviewers. The MG checklist and MIS checklist by Dr. Smith will serve as a reference for reviewers.

3.4 Verification and Validation Plan Verification Plan

The Verification and Validation Plan will itself be verified primarily through manual reviews by the Instructor, Domain Expert, and Secondary Reviewer. Reviewers will compare the document against the criteria specified by Dr .Smith in the VnV checklist.

3.5 Implementation Verification Plan

The implementation of Optimal EM Placement will be verified primarily in two different ways:

1. **Code Walkthrough (Static):** Walkthroughs will be performed by the author, domain expert, and secondary reviewer. The reviewers will run and interact with the program, checking for bugs and vulnerabilities in the code. Static analysis tools like `Pylint` can be used to examine the code for smells and potential refactoring opportunities. This process is followed by a discussion in which members discuss their findings and suggest enhancements.

2. **Test Cases (Dynamic):** As outlined in Section 4, test cases will be executed to verify the function and non-functional requirements mentioned in the SRS.

3.6 Automated Testing and Verification Tools

Automated testing as outlined in the previous section will be carried out using the `PyTest` library, by comparing expected outputs to calculated outputs given some constant user input.

4 System Tests

4.1 Tests for Functional Requirements

The functional and nonfunctional requirements of the program are given in Sections 5.1 and 5.2 of the SRS. The relationship between test cases and requirements is given in the traceability matrix in Section 4.3.

Error codes in Table 2 are explained in Section 1 of the document.

4.1.1 Input Tests - Invalid Input

Incorrect EM Properties

Var	Inputs					
N	-1	10	8	2^{40}	100	100
I	10	0	15	20	10^6	10
A	6	6	-1	8	8	6
Valid	N	N	N	N	N	Y
Error	PV	PV	PV	TL	TL	N/A
Case #	1	2	3	4	5	6

Table 2: EM properties test cases

1. test-em-props-1. . . 6

   Control: Automatic.

   Initial State: Pending input.

   Input: Set of input values for EM Properties as given in Table 2.

   Output: An appropriate error message, if necessary, as given in Table 2.

   Test Case Derivation: These cases test the behaviour of the system when given invalid system related inputs as described in Table 2 of the SRS.

   How test will be performed: Using `PyTest`.

Invalid Input Type

Var	Inputs				
N	'A'	100	100	100	100
I	10	10	10	10	10
A	4	4	4	4	4
M	100	5.5	100	100	100
K	4	4	[a]	4	4
V	1	1	1	'b'	1
t	0.3	0.3	0.3	0.3	0.3
Valid	N	N	N	N	Y
Error	IT	IT	IT	IT	N/A
Case #	1	2	3	4	5

Table 3: Input type test cases

1. test-inp-type-1...5

   Control: Automatic.

   Initial State: Pending input.

   Input: Set of input values for the system as given in Table 3.

   Output: An appropriate error message, if necessary, as given in Table 3.

   Test Case Derivation: These cases test the behaviour of the system when given invalid input types, based on what is described in the SRS.

   How test will be performed: Using `PyTest`.

System Setup

Var	Inputs					
M	10^4	10	0	10^4	10^4	10^4
K	10^5	0	0	6	8	4
V	1	1	1	0	1	1
t	0.1	0.1	0.1	0.1	0	0.1
Valid	N	N	N	N	N	Y
Error	PV	PV	PV	TL	TL	N/A
Case #	1	2	3	4	5	6

Table 4: System setup test cases

1. test-sys-setup-1...6

   Control: Automatic.

   Initial State: Pending input.

   Input: Set of input values for the system as given in Table 4.

   Output: An appropriate error message, if necessary, as given in Table 4.

   Test Case Derivation: These cases test the behaviour of the system when given invalid EM related inputs as described in Table 2 of the SRS.

   How test will be performed: Using `PyTest`.

4.1.2 Output Tests - Correct Output

Correct Positions Selected

Var	Inputs	
N	100	100
I	10	10
A	6	6
M	10^3	10^5
K	3	6
V	1	1
t	0.1	0.1
Output	$x_{1,8,9} = 1$	$x_{1,4,9,11,20,33} = 1$
Case #	1	2

Table 5: Output correctness test cases

1. test-output-correct-1...2

   Control: Automatic.

   Initial State: Pending input.

   Input: Set of input values for as given in Table 5.

   Output: A binary vector x, such that the values at the indices shown in Table 5 are 1. An value equalling 1 at an index, indicates an EM actuator taking the pose at that index.

   Note: The above must hold for a set of randomly generated poses with a `numpy` random seed of 100.

   Test Case Derivation: These cases ensure the system returns the correct optimal positions based on the inputs it receives.

   How test will be performed: Using `PyTest`.

4.1.3 Intermediate Tests - Intermediate Computations

Singular Values

1. test-intmed-svd-1...2

   Control: Automatic.

Initial State: Pending input.

Input: Set of input values as given in Table 5.

Output: 2.82×10^{-4}, 7.16×10^{-4} for test-intmed-svd-1 and test-intmed-svd-2 respectively. These values correspond to the lowest singular value of the $\mathcal{U}$ matrix described in Section 4.2.4 of the SRS.

Test Case Derivation: These cases ensure the program is correctly constructing the $\mathcal{U}$ matrix and computing the magnetic field and force values.

How test will be performed: Using `PyTest`.

4.2 Tests for Nonfunctional Requirements

The 4 nonfunctional requirements for Optimal EM placement is given in Section 5.2 of the SRS.

4.2.1 Accuracy

1. test-nf-acc

   Type: Manual.

   Initial State: Pending input.

   Input: Various.

   Output: An array of lowest singular values from 100 runs.

   How test will be performed: The author will use various parameters on the OEMP, greedy and random algorithms, and plot a singular value distribution from the 100 runs to compare performance, as explained in NF1 of the SRS. This test related to the Singular Values test in 4.1.3. The average minimum singular value across OEMP runs must be at least 50% greater than that of greedy runs.

4.2.2 Usability

1. test-nf-use

   Type: Manual.

Initial State: None.

Input: None.

Output: None.

How test will be performed: A group of users will be asked to use the system and attempt to get a solution for their inputs. They will then answer the questions provided in the Usability Survey. An average score of 7 or above indicates the user found the program satisfactorily usable.

4.2.3 Maintainability

1. test-nf-mtn

   Type: Static.

   Initial State: None.

   Input: None.

   Output: None.

   How test will be performed: The author will meet with the Domain Expert Reviewer and Secondary Reviewer to present a code walkthrough and discuss details related to software maintainability, in particular: the modularity of the code, the comprehensiveness of the test cases, and the consistency of formatting.

4.2.4 Portability

1. test-nf-prt

   Type: Manual.

   Initial State: None.

   Input: None.

   Output: None.

   How test will be performed: The author will manually run all tests listed in this document on a Windows, MacOS and Linux machine.

4.3 Traceability Between Test Cases and Requirements

Traceability between tests and requirements is visualized below:

	R1	R2	R3	R4	R5	R6	NFR1	NFR2	NFR3	NFR4
4.1.1	X	X								
4.1.2						X				
4.1.3			X	X	X					
4.2.1							X			
4.2.2								X		
4.2.3									X	
4.2.4										X

Table 6: Traceability between tests and requirements

5 Unit Test Description

[This section should not be filled in until after the MIS (detailed design document) has been completed. —SS]

[Reference your MIS (detailed design document) and explain your overall philosophy for test case selection. —SS]

[To save space and time, it may be an option to provide less detail in this section. For the unit tests you can potentially layout your testing strategy here. That is, you can explain how tests will be selected for each module. For instance, your test building approach could be test cases for each access program, including one test for normal behaviour and as many tests as needed for edge cases. Rather than create the details of the input and output here, you could point to the unit testing code. For this to work, you code needs to be well-documented, with meaningful names for all of the tests. —SS]

5.1 Unit Testing Scope

[What modules are outside of the scope. If there are modules that are developed by someone else, then you would say here if you aren't planning on verifying them. There may also be modules that are part of your software, but have a lower priority for verification than others. If this is the case, explain your rationale for the ranking of module importance. —SS]

5.2 Tests for Functional Requirements

[Most of the verification will be through automated unit testing. If appropriate specific modules can be verified by a non-testing based technique. That can also be documented in this section. —SS]

5.2.1 Module 1

[Include a blurb here to explain why the subsections below cover the module. References to the MIS would be good. You will want tests from a black box perspective and from a white box perspective. Explain to the reader how the tests were selected. —SS]

1. test-id1

Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

Initial State:

Input:

Output: [The expected result for the given inputs —SS]

Test Case Derivation: [Justify the expected value given in the Output field —SS]

How test will be performed:

2. test-id2

   Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

   Initial State:

   Input:

   Output: [The expected result for the given inputs —SS]

   Test Case Derivation: [Justify the expected value given in the Output field —SS]

   How test will be performed:

3. ...

5.2.2 Module 2

...

5.3 Tests for Nonfunctional Requirements

[If there is a module that needs to be independently assessed for performance, those test cases can go here. In some projects, planning for nonfunctional tests of units will not be that relevant. —SS]

[These tests may involve collecting performance data from previously mentioned functional tests. —SS]

5.3.1 Module ?

1. test-id1

   Type: [Functional, Dynamic, Manual, Automatic, Static etc. Most will be automatic —SS]

   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:

5.3.2 Module ?

...

5.4 Traceability Between Test Cases and Modules

[Provide evidence that all of the modules have been considered. —SS]

References

6 Appendix

This is where you can place additional information.

6.1 Symbolic Parameters

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

6.2 Usability Survey Questions

Answer the following questions one a scale of 1 to 10 (where 1 is "very poor" and 10 is "excellent"):

- How easy was it to understand and use the program's input fields?
- How useful are the error messages or feedback provided when an incorrect input is given?
- How would you rate the overall flow between input and output interactions during your usage?
- How clear and helpful is the output produced by the program?
- How satisfied are you with the overall interaction experience of the program?