System Verification and Validation Plan for Inverted Pendulum Control Systems (IPCS)

Morteza Mirzaei

April 16, 2024

Revision History

Date	Version	Notes
2024-02-18	1.0	Initial Release.
2024-04-13	2.0	Address the comments.

Contents

List of Tables

List of Figures

1 Symbols, Abbreviations, and Acronyms

For complete symbols used within the system, please refer the section 1 in SRS document. Table 1 defines some abbreviations and acronyms used in this document.

Table 1: Table of Abbreviations and Acronyms.

symbol	**description**
SRS	Software Requirement Specification
VnV	Verification and Validation
MIS	Module Interface Specification
IPCS	Inverted Pendulum Control Systems
TC	Test Case

This document outlines the framework for Verification and Validation (VnV), delineating the objectives, scope, and methodologies for confirming the correctness and suitability of the software solution. Verification entails scrutinizing whether the software aligns with predetermined requirements and design parameters, whereas validation ensures it meets user expectations and demands effectively. Through this VnV framework, the team can uphold the integrity, dependability, and utility of the software, while also identifying and rectifying any flaws or discrepancies before deployment or release.

2 General Information

This section will provide the background and objectives for this document.

2.1 Summary

This document presents a verification and validation plan for Inverted Pendulum Control Systems, which aims to simulate, control, and visualize an inverted pendulum on a cart. For complete information about the software, please refer to the Software Requirement Specification document.

2.2 Objectives

The objectives outlined in this Verification and Validation plan are as follows:

- Verify the software's capability to precisely simulate an inverted pendulum.
- Verify the proper functioning of the control system to sustain the pendulum in an upright position.
- Confirm the correctness of the visualization.
- Confirm the userfriendliness and ease of understanding of the visualization.

Performance testing and stress testing are outside of the scope of this project.

2.3 Relevant Documentation

Refer to the Software Requirements Specification for comprehensive information regarding the objectives, requirements, assumptions, and underlying principles of the software. You could also refer to the Module Guide and Module Interface Specification for detailed information on the software's design and architecture. The final report of running the test which described in this document, will be shown in VnV Report.

3 Plan

This section will outline the strategy for validating the documentation and software for Inverted Pendulum Control Systems. The main components to be validated include the Software Requirements Specification (SRS), design, Verification and Validation (VnV) plan, and implementation.

3.1 Verification and Validation Team

The team members and their responsibilities are specified in Table 2.

Table 2: Verification and Validation Team

Name	Document	Role	Description
Morteza Mirzaei	All	Author	Create and manage all the documents, create the VnV plan, perform the VnV testing, verify the implementation.
Dr. Spencer Smith	All	Instructor/Reviewer	Review the documents, design and documentation style.
Seyed Ali Mousavi	All	Domain Expert Reviewer	Review all the documents.
Al Jubair Hossain	SRS	Secondary Reviewer	Review the SRS document
Hunter Ceranic	VnV Plan	Secondary Reviewer	Review the VnV plan.
Adrian Sochaniwsky	MG + MIS	Secondary Reviewer	Review the MG and MIS document.

3.2 SRS Verification Plan

The SRS document will be verified in the following way:

1. Initial review will be performed by the assigned members (Dr. Spencer Smith, Seyed Ali Mousavi, and Al Jubair Hossain). For this, a manual review will be conducted using the provided SRS Checklist and following additional checklist gather from here.

    - □ Are all requirements actually requirements, not design or implementation solutions?
    - □ Is each requirement uniquely and correctly identified?
    - □ Is each requirement verifiable by testing, demonstration, review, or analysis?

- □ Do any requirements conflict with or duplicate other requirements?
- □ Is each requirement in scope for the project?
- □ Are all requirements written at a consistent and appropriate level of detail?
- □ Do the requirements provide an adequate basis for design?

2. The reviewers will provide feedback to the author by creating an issue on GitHub, referencing the checklist mentioned above.
3. Morteza Mirzaei, the author, is responsible for addressing the issues.

3.3 Design Verification Plan

Initial review will be performed by the assigned members (Dr. Spencer Smith, Seyed Ali Mousavi, and Adrian Sochaniwsky). For this, a manual review will be conducted using the following checklist.

- □ Does the design meet all specified requirements outlined in the design specification document?
- □ Are all interfaces between design components specified clearly?
- □ Are all components of the design testable separately?
- □ Is the design scalable to accommodate future growth or changes in requirements?
- □ Are the design components separated as much as possible?
- □ Does each design component have a logical task?

3.4 Verification and Validation Plan Verification Plan

The designated members (Dr. Spencer Smith, Seyed Ali Mousavi, and Hunter Ceranic) will conduct the initial review. This review will entail a manual assessment utilizing the following checklist.

- □ Does the VnV plan verify all the functional and non-functional requirements?

- □ Are all input and output pairs for test cases correct?
- □ Are roles and responsibilities clearly defined for individuals or teams involved in verification and validation activities?

3.5 Implementation Verification Plan

Inverted Pendulum Control Systems will be developed using Python programming language. For static testing, we will use Flake8. Additionally, we conduct code walkthroughs with domain expert (Ali Mousavi). For dynamic testing, we employ system and unit level testing, as explained in detail in sections Section 4 and Section 5.

3.6 Automated Testing and Verification Tools

- Pytest: A unit testing framework for Python that allows for easy and scalable testing of Python code.
- Flake8: A static code analysis tool for Python that checks code against coding style (PEP 8), detects syntax errors (missing indent error), or problematic constructs (cyclomatic complexity).
- Docker: A mechanism used to containerize applications, ensuring their installability and understandability throughout the project.
- GitHub: This platform will be used for version control and collaboration.
- GitHub CI workflow: This will automate regression tests and checks that Inverted Pendulum Control Systems builds are passing before code is merged into a protected branch. Every push or pull request on the main branch triggers the test. Additionally, after every update of the main branch, the CI workflow publishes the new version of the Docker container to Docker Hub.

3.7 Software Validation Plan

No plan for software validation is currently in place. Software validation plan is beyond the scope for Inverted Pendulum Control Systems as we do not have enough experimental data for the validation of the system behavior.

4 System Test Description

This section will define the tests to ensure IP Simulator meets the functional requirements seen in section 5 of the SRS document for simulating inverted pendulum system. The subsections combine several requirements that are separated based on common ideas.

4.1 Tests for Functional Requirements

This section contains the system test cases for the functional requirements which are described in the SRS.

4.1.1 Inputs and Outputs

This part verifies the input and output restrictions outlined in R1 for Inverted Pendulum Control Systems. To review these constraints, refer to the "Input Data Constraints" section in the SRS. Tests are designed for both normal cases and edge and boundary cases, as outlined in Table 3.

Table 3: Input validation test cases

Input						Output	
F	x	θ	m	M	l	Valid	Description
0	0	π	16	80	1	Yes	-
'abc'	0	π	16	80	1	No	invalid input for F
0	'abc'	π	16	80	1	No	invalid input for x
0	0	'abc'	16	80	1	No	invalid input for θ
0	0	π	'abc'	80	1	No	invalid input for m
0	0	π	16	'abc'	1	No	invalid input for M
0	0	π	16	80	'abc'	No	invalid input for l
0	0	π	-16	80	1	No	negative input for m
0	0	π	16	-80	1	No	negative input for M
0	0	π	16	80	-1	No	negative input for l
0	0	π	100	80	1	No	exceed max value for m
0	0	π	16	100	1	No	exceed max value for M
0	0	π	16	80	50	No	exceed max value for l

Inputs and Outputs test

1. test-inout-1

   Control: Automatic

   Initial State: N/A.

   Input: Specified in the Table 3.

   Output: Specified in the Table 3.

   Test Case Derivation: Directly from the requirement.

   How test will be performed: Automatic test using Pytest.

4.1.2 Simulator

This part tests the world simulator responsible for executing the inverted pendulum physics. It assesses the requirements R2 and R3 for Inverted Pendulum Control Systems. Essentially, it verifies that given the initial position of the pendulum and a constant force acting on the cart, the location of the pendulum is determined after a specified duration of time. Tests are designed for both normal cases and edge and boundary cases, as outlined in Table 4. We simply compare the results of the two methods at the end of the interval. This approach makes documenting the results easier and enhances the clarity of understanding the error. Additionally, since the error accumulates over time, we do not sacrifice much accuracy by adopting this method.

Simulator

1. test-sim-1

   Control: Automatic

   Initial State: Specified in the Table 4.

   Input: Specified in the Table 4.

   Output: Specified in the Table 4.

   Test Case Derivation: Solving the ODEs provided in the SRS with numerical analysis tool, SciPy.

   How test will be performed: Automatic test using Pytest.

Table 4: Output Validation Test Cases. The time step (dt) is 0.1 seconds. The results are calculated after 5 seconds.

Input						Output	
F	x	θ	m	M	l	x	θ
0.0	0.0	-90.0	16.0	80.0	1.0	5.0	0.25
0.0	-5.0	-90.0	16.0	80.0	1.0	5.0	-4.75
0.0	+5.0	-90.0	16.0	80.0	1.0	5.0	5.25
0.0	0.0	0.0	16.0	80.0	1.0	6.0	0.08
0.0	0.0	-45.0	16.0	80.0	1.0	2.5	-0.3
0.0	0.0	-90.0	30.0	80.0	1.0	1.5	0.8
0.0	0.0	-90.0	16.0	50.0	1.0	1.45	1.3
0.0	0.0	-90.0	16.0	80.0	20.0	5.0	0.25
+5.0	0.0	-90.0	16.0	80.0	1.0	2.7	0.65
-5.0	0.0	-90.0	16.0	80.0	1.0	5.5	0.0

4.1.3 Control System

This part tests the control system responsible for keeping the inverted pendulum upright. It assesses the requirements (R4) for Inverted Pendulum Control Systems. Essentially, it verifies that after a reasonable amount of time, the pendulum's angular position remains relatively stable in an upward position.

Control System

1. test-control-1

   Control: Automatic

   Initial State: Specified in the Table 4.

   Input: Specified in the Table 4.

   Output: θ should be between $-\frac{\pi}{6}$ and $\frac{\pi}{6}$ after 30 seconds.

   Test Case Derivation: Directly from the requirement.

   How test will be performed: Automatic test using Pytest.

4.1.4 Visualization

This part verifies Requirement 5 (R5), ensuring that the visualization and graphical user interface (GUI) of the software meet the necessary standards.

Table 5: Input test cases

Input	
x	θ
0	0
0	π
-5	π
-5	$\frac{\pi}{2}$

Visualization

1. test-vis-1

   Control: Manual

   Initial State: Specified in the Table 5.

   Input: Specified in the Table 5.

   Output: Visualization of the pendulum at different states.

   Test Case Derivation: Directly from the requirement.

   How test will be performed: First, Capture screenshots of the visualization at different states of the pendulum. Then, compare each screenshot with the expected state of the pendulum.

2. test-vis-2

   Type: Manual

   Initial State: N/A

   Input: Default initial values for the pendulum and cart.

   Output: Video of visualization of the pendulum at different states.

   Test Case Derivation: Directly from the requirement.

How test will be performed: After playing around with the visualization, users will be asked to fill out the following survey.

- □ On a scale of 1 to 5, how clear is the visualization to you?
- □ On a scale of 1 to 5, how clear is the location of the cart to you?
- □ On a scale of 1 to 5, how clear is the angular position of the pendulum to you?
- □ On a scale of 1 to 5, how smooth is the visualization to you? (If it's under 30 frames per second, human eyes can detect the lag.)

At the end of the survey, the average score for each question will be calculated. It should be more than 3 for the visualization to be considered user-friendly. At least one experts will be asked to fill out the survey.

4.2 Tests for Nonfunctional Requirements

4.2.1 Accuracy

Accuracy

1. test-acc-1

   Type: Automatic

   Initial State: Specified in the Table 4.

   Input: Specified in the Table 4.

   Output: Specified in the Table 4.

   How test will be performed: Automatic test using Pytest. The relative error between software simulation and specified values should be less than 5%.

4.2.2 Usability

Usability

1. test-useable-1

   Type: Manual with group of people

   Initial State: None

   Input: None

   Output: Survey can help to know about the user perspective towards the system.

   How test will be performed: The domain expert will be asked to fill out the survey specified in Table 6. This test will be done after the implementation of the software. More than 3 questions should be answered with yes to pass the test.

Table 6: Usability test survey.

No.	Question
1.	Could you run the software on your operating system?
2.	Is system running smoothly on your computer?
3.	Is invalid input's message clear?
4.	Is text easy to read?

4.2.3 Maintainability

Maintainability

1. test-maintain-1

   Type: Manual

   Initial State: N/A

   Input/Condition: N/A

   Output/Result: A score from experts on the maintainability of the software.

   How test will be performed: We use survey in the Section 3.3 as a measure of maintainability. The design and survey will be given to the experts and they will provide feedback on the maintainability of the software. This will be done after the implementation of the software. Author and domain expert will do the survey.

4.2.4 Portability

Portability

1. test-port-1

   Type: Manual

   Initial State: N/A

   Input/Condition: test-sim-1 and est-vis-2.

   Output/Result: Successful test implies portability of the software.

   How test will be performed: The test will be performed manually by executing the software and tests on the Linux and macOS virtual machines.

2. test-port-2

   Type: Manual

   Initial State: N/A

   Input/Condition: test-sim-1.

   Output/Result: Successful test implies portability of the software.

   How test will be performed: The test will be performed manually by executing the test on docker container version of the software.

4.3 Traceability Between Test Cases and Requirements

5 Unit Test Description

This section will outline the unit testing strategy for Inverted Pendulum Control Systems.

5.1 Unit Testing Scope

The scope of the unit tests will not include testing the GUI and visualization part of the Inverted Pendulum Control Systems.

Table 7: Relation of Test Cases and Requirements.

	R1	R2	R3	R4	R5	NFR1	NFR2	NFR3	NFR4
test-inout-1	X								
test-sim-1		X	X			X			
test-control-1				X					
test-vis-1					X				
test-vis-2					X				
test-acc-1		X	X			X			
test-useable-1					X		X		
test-maintain-1								X	
test-port-1									X
test-port-2									X

5.2 Tests for Functional Requirements

Details of the tests for every module will be provided here, with each test being incorporated into an automated unit test suite.

5.2.1 World Checker Module

This test implemented in the `test_checker.py` file. It makes sure World Checker module raise proper exceptions when the input is not valid. It tests negetive and zero values for masses. It also tests the invalid values (like strings) for input parameters.

UT1: **test conf**

5.2.2 Config Reader Module

This test is implemented in the `test_conf.py` file. It tests the Config Reader module, raising proper exceptions when the test file is not found or there are no inputs.

UT2: **test file not found**

UT3: **test no input exception**

5.2.3 Fuzzy Controller Module

This test is implemented in the `test_fuzzy.py` file. It tests the Fuzzy Controller module logic by inputting different values for the pendulum and cart and checking the force output.

UT4: **test fuzzy**

5.2.4 Simulation Module

This test is implemented in the `test_simulator.py` file. It tests the Simulation module by inputting different values for the pendulum and cart and checking the output of the simulation after one step of dt.

UT5: **test exception**

UT6: **test tick**

5.3 Traceability Between Test Cases and Modules

Traceability between test cases and modules is shown in Table 8.

Table 8: Traceability Between Test Cases and Modules

Module	Tests
World Checker	UT1
Config Reade	UT2, UT3
Fuzzy Controller	UT4
Simulation	UT5, UT6