System Verification and Validation Plan for PD Controller

Naveen Ganesh Muralidharan

December 14, 2020

1 Revision History

Date	**Version**	**Notes**
28-Oct-20 1	1.0	The first draft of the VnV plan
14-Dec-20 1	2.0	This versions contains the following changes, • Changed scope to PD controller. • Incorporated Dr.Smith's and reviewers feedback. • Added more test cases.

Contents

List of Tables

2 Symbols, Abbreviations, and Acronyms

symbol	description
API	Application Program Interface
DCCC	Data and Control Coupling
N	No
PDF	Portable Document Format
PEP8	Python Enhancement Proposal 8
stdin	Standard input stream
stderr	Standard error stream
stdout	Standard output stream
T	Test
VnV	Verification and Validation
Y	Yes

All the units, abbreviations, and symbols recorded in the Software Requirement Specification [8] apply to this document as well.

This document encompasses the Verification and Validation (VnV) plan for the PD Controller software.

Section 3 of this document sets the context for the VnV plan. Section 4 provides a high-level overview of the VnV plan. Sections 5 and 6 contain the systems and unit test cases respectively.

3 General Information

3.1 Summary

The software under test is the simulation of a PD control loop. The functions of the PD control loop are,

- Calculating the Error Signal. Error Signal is the difference between the User Input (Set-Point) and the output of the Power Plant (Process-Variable).
- Computing the output of the PD Controller.
- Computing the response of the Power-Plant.

3.2 Objectives

The objectives of the Verification and Validation procedures are to,

- Establish confidence in software correctness.
- Ensuring that the software meets the expected quality standards.

3.3 Relevant Documentation

The requirements for the PD Controller software are captured in the Software Requirements Specification [8]

The software design information are captured in the Module Guide [5] and Module Interface Specification [6] documents respectively.

Team Member	Role
Naveen Ganesh Muralidharan	Author
Dr. Spencer Smith	Course Instructor and Domain Expert
Ting-Yu Wu	Domain Expert
Siddharth (Sid) Shinde	Secondary reviewer - SRS
Gabriela Sánchez Díaz	Secondary reviewer - VnV Plan

Table 1: Verification and Validation Team

4 Plan

4.1 Verification and Validation Team

The members of the VnV team for this project are listed in Table-1.

4.2 SRS Verification Plan

The SRS will be independently peer-reviewed by members of the VnV team, specifically, Dr. Spencer Smith, Ting-Yu Wu, and Siddharth (Sid) Shinde.

Any issues identified during the review are tracked and verified in Github [4].

4.3 Design Verification Plan

The software for this project is auto-generated by the Drasil Software [1]. Therefore manual verification of the design is not required.

4.4 Implementation Verification Plan

The implemented software is tested as follows,

- Automated systems testing, where the corresponding test cases are listed in section 5.
- Automated unit testing, where the corresponding test cases are listed in section 6.
- Statement coverage check.

- Static code analysis.

4.5 Automated Testing and Verification Tools

The tools utilized for verification are listed below,

- Systems Testing - Pytest [2] will be used for automated systems testing. Since this is a blackbox test, Pytest will be used at the stdin, stdout, and stderr levels.
- Unit Testing - Pytest [2] will be used for automated unit testing. Since this is a whitebox test, Pytest will be used at the API level.
- Statement coverage - PyTest-Cov [10]. PyTest-Cov is used along with Pytest to obtain the statement coverage.
- Memory leaks - Valgrind [11] will be used for memory leak analyses.
- Linting - Flake8 [9]. Linting tool that checks for the coding style against the PEP8 standard.

4.6 Software Validation Plan

There are no plans for the Validation of the PD Controller software.

5 System Test Description

5.1 Tests for Functional Requirements

This section contains the systems test cases for the functional requirements in the SRS [8]. The test cases are organized into two categories, the input-output tests and simulation parameters test.

5.1.1 Input Output tests

This section verifies section 4.2.6, and the functional requirements of section 5.1 in the SRS [8]. Various inputs are provided to the Software Under Test, and the output is verified.

	Input					Output	
ID	r_{t}	K_{p}	K_{d}	t_{step}	t_{sim}	y_{t}	Error Message
TC-PD-1-1	1	20	1	0.01	10	0.9524	N/A
TC-PD-1-2	-0.0001	20	1	0.01	10	N/A	InputError
TC-PD-1-3	1	-0.0001	1	0.01	10	N/A	InputError
TC-PD-1-4	1	20	-0.0001	0.01	10	N/A	InputError
TC-PD-1-5	1	20	1	0.0009	10	N/A	InputError
TC-PD-1-6	1	20	1	10	10	N/A	InputError
TC-PD-1-7	1	20	1	0.01	0.9999	N/A	InputError
TC-PD-1-8	1	20	1	0.01	60.0001	N/A	InputError

Table 2: TC-PD-1 - Input constraints tests

Input Constraints test

- TC-PD-1
  - **Control:** Automatic
  - **Initial State:** None
  - **Input:** Set the inputs to the values specified in the 'Input' columns in Table-2.
  - **Output:** Verify that the outputs of the software match the values specified in the 'Output' columns of Table-2 within the allowable margin of error.
  - **Requirement ID(s):** FR: Input-Values, FR: Verify-Input-Values, FR: Calculate-Values, FR: Output-Values.
  - **Test Case Derivation:** This test case is to test the behavior of the system when the system is supplied with inputs that are outside the physical constraints, as specified in Table-4 in the SRS [8]. In the test cases TC-PD-1-2 to TC-PD-1-7, the system should produce an InputError, as the values supplied are beyond the physical constraints of the input signals.

    The output, y_{t} refers to the last value of the output list.

	Input					Output
ID	r_t	K_p	K_d	t_step	t_sim	y_t
TC-PD-2-1	20	10	1	0.01	10	18.18
TC-PD-2-2	20	5	1	0.01	10	16.67
TC-PD-2-3	20	10	15	0.01	10	18.19
TC-PD-2-4	20	10	1	0.01	5	18.17

Table 3: TC-PD-2 - Output values

The output specified in TC-PD-1-1 has been independently verified using a Simulink Model ([3], [7]). A relative error of 5% is applied to accommodate rounding off errors, and floating point representation errors between the two software.

- **How test will be performed:** The test will be automated with Pytest as mentioned in section 4.5.

Output test

- TC-PD-2
  - **Control:** Automatic
  - **Initial State:** None
  - **Input:** Set the inputs to the values specified in the 'Input' columns in Table-3.
  - **Output:** Verify that the outputs of the software matches the value specified in the 'Output' column of Table-3 within the allowable margin of error.
  - **Requirement ID(s):** FR: Input-Values, FR: Verify-Input-Values, FR: Calculate-Values, FR: Output-Values.
  - **Test Case Derivation:** This test case is to prove that each input to the software uniquely affects the output of the software.

    The output, y_t refers to the last value of the output list.

The outputs have been independently verified using a Simulink Model ([3], [7]). A relative error of 5% is applied to accommodate rounding off errors, and floating point representation errors between the two software.

  - **How test will be performed:** The test will be automated with Pytest as mentioned in section 4.5.

...

5.2 Tests for Nonfunctional Requirements

This section contains the test cases for the non-functional requirements (section 5.2) of the SRS [8].

5.2.1 Portability Test

- TC-PD-3

  - **Type:** Semi-Automated, Dynamic.
  - **Initial State:** None
  - **Requirement ID(s):** NFR: Portable
  - **Input/Condition:** Execute TC-PD-1 and TC-PD-2 in each of the the following operating systems,
    1. Windows 10.
    2. Bodhi Linux 5.1.0.
  - **Output/Result:** Verify that on each of the operating systems the test case TC-PD-1 and TC-PD-2 passes.
  - **How test will be performed:** On each of the Operating systems, execute the functional test suite using Pytest [2].

5.2.2 Maintainability tests

Modularity Test

- TC-PD-4

- **Type:** Manual, Inspection.
- **Initial State:** None
- **Requirement ID(s):** NFR: Maintainable
- **Input/Condition:** N/A
- **Output/Result:** Verify that the source code is modular.
- **How test will be performed:** Review the source code files. Ensure that each functionality of the software is handled in its own module.

Linting

- TC-PD-5

  - **Type:** Static, Automated.
  - **Initial State:** None
  - **Requirement ID(s):** NFR: Maintainable
  - **Input/Condition:** The source code files.
  - **Output/Result:** Verify that the source code does not contain any PEP-8 violations.
  - **How test will be performed:** This test will be automated using the Flake8 [9] tool.

Documented

- TC-PD-6

  - **Type:** Inspection
  - **Initial State:** None
  - **Requirement ID(s):** NFR: Maintainable
  - **Input/Condition:** The PDF API reference generated with Doxygen.
  - **Output/Result:** Verify that all the classes, methods and modules in the source code are documented in the API reference.

- **How test will be performed:**
    1. Run ‘make doc‘ to generate the Doxygen Latex files.
    2. Navigate to the Latex directory and run ‘make’ to generate the Doxygen PDF file.
    3. Inspect the PDF file, ensure all the functions in the source code are adequately documented.

5.2.3 Security Tests

Memory leak check

- TC-PD-7

    - **Type:** Automated, Dynamic.
    - **Initial State:** None
    - **Requirement ID(s):** NFR: Secure
    - **Input/Condition:** Execute the test case of TC-PD-1-1.
    - **Output/Result:** Verify that the test case TC-PD-1-1 passes and there are no memory leaks identified in the report generated by Valgrind.
    - **How test will be performed:** Valgrind [11] is used to fork the python script. After the execution, a report is generated by Valgrind.

Divide by-zero check

- TC-PD-8

    - **Type:** Static, Inspection.
    - **Initial State:** None
    - **Requirement ID(s):** NFR: Secure
    - **Input/Condition:** Source code of the PD controller.
    - **Output/Result:** For every division in the source code, verify that the denominator is tested for a non zero value.

  - **How test will be performed:** Manual inspection of the source code.

—

Negative square root check

- TC-PD-9

  - **Type:** Static analysis and Inspection
  - **Initial State:** None
  - **Requirement ID(s):** NFR: Secure
  - **Input/Condition:** Source code of the PD controller.
  - **Output/Result:** For every square root function in the source code, verify that the operands are tested for negative values before the function call.
  - **How test will be performed:** Manual inspection of the source code.

—-

5.2.4 Verifiability test

- TC-PD-10

  - **Type:** Inspection
  - **Initial State:** None
  - **Requirement ID(s):** NFR: Verifiable
  - **Input/Condition:** VnV report and the SRS of the PD Controller.
  - **Output/Result:** Verify that for each requirement in the SRS, there exists at-least one test case in the VnV report.
  - **How test will be performed:** Manual inspection of the documents.

Statement Coverage Test

- TC-PD-11
  - **Type:** Dynamic, Automated.
  - **Initial State:** None
  - **Requirement ID(s):** NFR: Verifiable
  - **Input/Condition:** Execute test cases TC-PD-1 and TC-PD-2.
  - **Output/Result:** Verify that the statement coverage is 100%.
  - **How test will be performed:** This test will be executed with Pytest [2] and the Pytest-Cov [10]. The tests should be executed with the '–cov' command line option.

Data Coupling and Control Coupling (DCCC) Analysis The following tests the Data and Control Coupling between the modules.

Data Coupling

- TC-PD-12
  - **Type:** Manual, Inspection.
  - **Initial State:** None
  - **Requirement ID(s):** NFR: Verifiable
  - **Input/Condition:** Execute TC-PD-1.
  - **Output/Result:** Analyze the Data Coupling is achieved between the modules.
  - **How test will be performed:** TC-PD-1 is executed with Pytest. The generated log file (log.txt) is examined for Data coupling.

Control Coupling

- TC-PD-13

	Input-Values	Verify-Input-Values	Calculate-Values	Output-Values	Portable	Secure	Maintainable	Verifiable
TC-PD-1	X	X	X	X				
TC-PD-2	X	X	X	X				
TC-PD-3					X			
TC-PD-4							X	
TC-PD-5							X	
TC-PD-6							X	
TC-PD-7						X		
TC-PD-8						X		
TC-PD-9						X		
TC-PD-10								X
TC-PD-11								X
TC-PD-12								X
TC-PD-13								X

Table 4: Requirements vs Test Cases Trace Matrix

- **Type:** Manual, Inspection.
- **Initial State:** None
- **Requirement ID(s):** NFR: Verifiable
- **Input/Condition:** Execute TC-PD-1.
- **Output/Result:** Analyze the Couple Coupling is achieved between the modules.
- **How test will be performed:** TC-PD-1 is executed with Pytest. The generated log file (log.txt) is examined for Coupling coupling.

5.3 Traceability Between Test Cases and Requirements

Table-4 contains the mapping of requirements to test cases.

6 Unit Test Description

The source code for the PD Controller has been auto-generated by the Drasil software ([1]). The generated software contains the following modules,

- Calculations.py - Provides functions for calculating the outputs.
- Constants.py - Provides the structure for holding constant values.

- Control.py - Controls the flow of the program.
- InputParameters.py - Provides the function for reading inputs and the function for checking the physical constraints on the input.
- OutputFormat.py - Provides the function for writing outputs.

Automated unit-testing will be performed on select modules.

6.1 Unit Testing Scope

The following areas were tested as part of the Systems testing,

- Software correctness and physical constraints were tested in test cases TC-PD-1 and TC-PD-2.
- Statement coverage of 100% was tested in test cases TC-PD-11.
- Coupling was tested in TC-PD-12 and TC-PD-13.

Therefore the only additional testing that is necessary for the analysis of the Step Time (t_{step}), and Simulation Time (t_{sim}). For this, the Calculations.py module will be unit tested.

6.2 Tests for Functional Requirements

6.2.1 Calculations.py

This section tests for the Step Time (t_{step}) through automated unit testing.

- TC-PD-11
  - **Type:** Automated, Black Box.
  - **Initial State:** None
  - **Requirement ID(s):** FR: Calculate-Values
  - **Input/Condition:** Set the input as follows,
    - $r_{\text{t}} = 1$
    - $K_{\text{p}} = 10$

	Input-Values	Verify-Input-Values	Calculate-Values	Output-Values	Portable	Secure	Maintainable	Verifiable
Calculations.py			X					

Table 5: Requirements vs Modules Trace Matrix

  * $K_\text{d} = 1$
  * $t_\text{step} = 0.01$
  * $t_\text{sim} = 1$
- **Output/Result:** Verify that the length of the list y_t is 101.
- **Test Case Derivation:** The ODE is integrated every t_step seconds until t_sim seconds. Therefore the no of samples will be $((t_\text{sim}/t_\text{step}) + 1)$, where 1 is for the initial value.
- **How test will be performed:** This test will be executed with Pytest [2].

6.3 Traceability Between Test Cases and Modules

Table-5 contains the mapping of requirements to modules.

References

[1] J. Carette and S. Smith. Drasil, 2020. URL https://jacquescarette.github.io/Drasil/.

[2] H. Krekel. Pytest, 2004. URL https://docs.pytest.org/en/stable/.

[3] Mathworks. Simulink, version 10.2, r2020b, 2020. URL https://www.mathworks.com/products/simulink.html.

[4] Microsoft. Github, 2020. URL https://github.com/muralidn/CAS741-Fall20.

[5] N. G. Muralidharan. Module guide for pid controller. https://github.com/muralidn/CAS741-Fall20/blob/master/docs/Design/MG/MG.pdf, 2020.

[6] N. G. Muralidharan. Module interface specification for pid controller. https://github.com/muralidn/CAS741-Fall20/blob/master/docs/Design/MIS/MIS.pdf, 2020.

[7] N. G. Muralidharan. PD_Simulation.slx, version 10.2, r2020b, 2020. URL https://github.com/muralidn/CAS741-Fall20/blob/master/test/data/pseudo-oracle/PD_Simulation.slx.

[8] N. G. Muralidharan. System requirements specification for pd controller. https://github.com/muralidn/CAS741-Fall20/blob/master/docs/SRS/SRS.pdf, 2020.

[9] I. C. Tarek Ziadé. Flake8. URL https://gitlab.com/pycqa/flake8/.

[10] P.-C. team. Pytest-cov. URL https://github.com/pytest-dev/pytest-cov.

[11] V. D. Team. Valgrind. URL www.valgrind.org.