System Verification and Validation Plan for PID Controller

Naveen Ganesh Muralidharan

November 27, 2020

1 Revision History

Date	Version	Notes
28-Oct-20 1	1.0	The first draft of the VnV plan

Contents

List of Tables

List of Figures

You can remove this if you don't need it

2 Symbols, Abbreviations, and Acronyms

symbol	description
API	Application Program Interface
N	No
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 PID 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 PID control loop. The functions of the PID 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 PID 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.
- Ensuring that the software is robust.

3.3 Relevant Documentation

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

The software design information are captured in the Module Guide [6] and Module Interface Specification [7] documents respectively.

4 Plan

4.1 Verification and Validation Team

The members of the VnV team for this project are,

- Naveen Ganesh Muralidharan
- Dr. Spencer Smith
- Ting-Yu Wu
- Siddharth (Sid) Shinde
- Gabriela Sánchez Díaz
- Seyed Parsa Tayefeh (Parsa) Morsal

You could summarize this information in a table and give information on each person's specific role.

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 [5].

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 [4] 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 [4] will be used for automated unit testing. Since this is a whitebox test, Pytest will be used at the API level.
- Statement coverage-Pycharm [3]. Pycharm automatically tracks the statement coverage upon the execution of systems test scripts.
- Memory leaks - Valgrind [9].
- Static Analysis - PyCharm [3]. Static analysis includes checks for linting, coding standards, etc.

4.6 Software Validation Plan

The pseudo-oracle method is used for the validation of the software requirements. The PID gains are calibrated with a free simulation model [2] of the PID controller. The calibrated gains are then supplied as inputs to the Software Under Test, and the outputs of the two software are compared to validate the requirements.

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 requirements R1 and R2 of section 5.1 in the SRS [8]. Various inputs are provided to the Software Under Test, and the output is verified.

Nice table ✓

Table 1: TC-PID-1

	Input						Output		
ID	SP	K_p	K_i	K_d	T_{sim} (s)	t_{step} (s)	$y(t)$	$T_{\text{elapsed}}(s)$	Error message
TC-PID-1-1	5	1	1	0	10	0.01	4.1	10	N
TC-PID-1-2	0	1	1	0	10	0.01	None	None	Y
TC-PID-1-3	30.1	1	1	0	10	0.01	None	None	Y
TC-PID-1-4	5	-0.01	1	0	10	0.01	None	None	Y
TC-PID-1-5	5	10.1	1	0	10	0.01	None	None	Y
TC-PID-1-6	5	1	-0.01	0	10	0.01	None	None	Y
TC-PID-1-7	5	1	10.1	0	10	0.01	None	None	Y
TC-PID-1-8	5	1	1	-0.01	10	0.01	None	None	Y
TC-PID-1-9	5	1	1	10.1	10	0.01	None	None	Y
TC-PID-1-10	5	1	1	0	0	0.01	None	None	Y
TC-PID-1-11	5	1	1	0	120.1	0.01	None	None	Y
TC-PID-1-12	5	1	1	0	10	0	None	None	Y
TC-PID-1-13	5	1	1	0	10	1.1	None	None	Y

Input Constraints test

- TC-PID-1
  - **Control:** Automatic
  - **Initial State:** None
  - **Input:** Set all inputs of the software to the values specified in the 'Input' columns in Table-1.
  - **Output:** Verify that all the outputs of the software match the values specified in the 'Output' columns of Table-1.
  - **Requirement ID(s):** R1, R2, R3, R4, R5.
  - **Test Case Derivation:** This test case is to test the behavior of the system when the system is supplied with inputs that are out

In your implementation you can use exceptions so that you can test whether the exception is generated when it should be.

of range. The system should either produce an output or an error message.

For test cases with output, a relative error of 10% is applied to accommodate rounding off errors, and floating point representation errors.

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

Output test

- TC-PID-2

  - **Control:** Automatic
  - **Initial State:** None
  - **Requirement ID(s):** R1, R2, R3, R4, R5.
  - **Input:** Set the inputs to the software as follows,
    $SP = 5$
    $K_p = 1$
    $K_i = 2$
    $K_d = 0$
    $T_{\text{sim}} = 30$ s
    $t_{\text{step}} = 0.01$ s
  - **Output:** Verify that the value in the last index of the following lists are,
    $y(t) = 5$ +/- 0.01
    $T_{\text{elapsed}} = 30$ +/- 0.01 s
  - **Test Case Derivation:** The inputs of this test case have been calibrated with a free PID simulation model [2]. Therefore in this test case, at the end of the simulation time, the Process Variable ($y(t)$) will be equal to the Set-Point.
    A relative error of 10% is applied to the process variable to accommodate rounding off errors and floating point representation errors. Similarly, the simulation should end in the specified time, with a relative error of 10% to accommodate rounding off errors, floating point representation errors, and operating system timer resolution.

Rather than a pass/fail specify the error. You could consider a positive description of the error.

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

5.1.2 Simulation parameters test

This section tests the simulation parameters, namely, the step time and the simulation time.

Simulation step time test

- TC-PID-4

  – **Control:** Automatic

  – **Initial State:** None

  – **Requirement ID(s):** R1, R2, R3, R4, R5.

  – **Input:** Set the input to the software as follows,
  $SP = 5$
  $K_p = 1$
  $K_i = 1$
  $K_d = 0$
  $T_{\text{sim}} = 10$ s
  $t_{\text{step}} = 0.5$ s

  – **Output:** Verify that the difference between the subsequent values of the T_{elapsed} is 0.5 s.

  – **Test Case Derivation:** t_{step} is the time for one iteration of the of the control loop; all computations are t_{step} apart.
  A relative error of 10% is applied to the process variable for loss of precision for rounding off errors and floating point representation errors. Similarly, the simulation should end in the specified time, with a relative error of 10% for rounding off errors, floating point representation errors, and operating system timer resolution.

  – **How test will be performed:** The test will be automated using Pytest as specified 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 Software Quality Tests

Statement Coverage Test

- TC-PID-5
  - **Type:** Dynamic, Automated.
  - **Initial State:** None
  - **Requirement ID(s):** NFR-6
  - **Input/Condition:** Execute test cases TC-PID-1 to TC-PID-5.
  - **Output/Result:** Verify that the statement coverage is 100%.
  - **How test will be performed:** This test will be executed with Pytest [4] in the Pycharm [3] IDE. The tests should be executed with the code coverage option.

Static code analysis

- TC-PID-6
  - **Type:** Static, Automated.
  - **Initial State:** None
  - **Requirement ID(s):** NFR-6
  - **Input/Condition:** Execute TC-PID-2
  - **Output/Result:** Verify that the inspection report does not contain any warnings.
  - **How test will be performed:** This test will be executed in PyCharm [3]. The default profile for verification should be selected.

5.2.2 Portability Test

- TC-PID-7

  - **Type:** Manual, Dynamic.
  - **Initial State:** None
  - **Requirement ID(s):** NFR-1
  - **Input/Condition:** Execute TC-PID-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-PID-2 passes.
  - **How test will be performed:** Manual execution of the test script on each of the operating systems.

5.2.3 Modularity Test

- TC-PID-8

  - **Type:** Manual.
  - **Initial State:** None
  - **Requirement ID(s):** NFR-2
  - **Input/Condition:** Execute TC-PID-2.
  - **Output/Result:** Verify that the test case TC-PID-2 passes.
  - **How test will be performed:**
    1. Create a new python script. This script may be one of the systems tests listed in section 5.
    2. Import the PID controller software as a module in the created script.
    3. Feed the same inputs as specified in TC-PID-2.

5.2.4 Security Tests

Memory leak check

- TC-PID-9
  - **Type:** Automated, Dynamic.
  - **Initial State:** None
  - **Requirement ID(s):** NFR-3
  - **Input/Condition:** Execute the test script of TC-PID-2.
  - **Output/Result:** Verify that the test case TC-PID-2 passes and there are no memory leaks identified in the report generated by Valgrind.
  - **How test will be performed:** Valgrind [9] is used to fork the python script. After the execution, a report is generated by Valgrind.

Divide by-zero check

- TC-PID-10
  - **Type:** Static, Inspection.
  - **Initial State:** None
  - **Requirement ID(s):** NFR-3
  - **Input/Condition:** Source code of the PID 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-PID-11

Good → This is a specific a thing to look for. Were you thinking to prove that divide by zero isn't possible in the code? This might be difficult to prove in general.

- **Type:** Static analysis and Inspection
- **Initial State:** None
- **Requirement ID(s):** NFR-3
- **Input/Condition:** Source code of the PID 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.

5.2.5 Maintainability test

- TC-PID-11

  - **Type:** Inspection
  - **Initial State:** None
  - **Requirement ID(s):** NFR-4
  - **Input/Condition:** The source code, and the User's Guide of the PID controller.
  - **Output/Result:** Verify that all the classes, methods and modules in the source code are documented in the User's Guide.
  - **How test will be performed:** Manual inspection of the source code and the User's guide.

Are you writing a user's guide? This is a great idea, but you might run out of time

5.2.6 Verifiability test

- TC-PID-13

  - **Type:** Inspection
  - **Initial State:** None
  - **Requirement ID(s):** NFR-5
  - **Input/Condition:** V&V report and the SRS of the PID Controller.

Good work!

Table 2: Requirements vs Test Cases Trace Matrix

	R1	R2	R3	R4	R5	NFR1	NFR2	NFR3	NFR4	NFR5	NFR6
TC-PID-1	X	X	X	X	X						
TC-PID-2	X	X	X	X	X						
TC-PID-4	X	X	X	X	X						
TC-PID-5	X	X	X	X	X						
TC-PID-6											X
TC-PID-7						X					
TC-PID-8							X				
TC-PID-9								X			
TC-PID-10								X			
TC-PID-11								X			
TC-PID-12									X		
TC-PID-13										X	

- **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.

5.3 Traceability Between Test Cases and Requirements

Table-2 contains the mapping of requirements to test cases.

6 Unit Test Description

6.1 Unit Testing Scope

6.2 Tests for Functional Requirements

6.2.1 Module 1

1. test-id1

Type:

(a) **Initial State:**
(b) **Input:**
(c) **Output:**
(d) **Test Case Derivation:**
(e) **How test will be performed:**

2. test-id2

   Type:

3. **Initial State:**
4. **Input:**
5. **Output:**
6. **Test Case Derivation:**
7. **How test will be performed:**
8. ...

6.2.2 Module 2

...

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:**

1. 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

[1] J. Carette and S. Smith. Drasil, 2020. URL https://jacquescarette.github.io/Drasil/. [Online; accessed 28-Oct-2020].

[2] http://grauonline.de/. Free pid simulation, 2020. URL http://grauonline.de/alexwww/ardumower/pid/pid.html. [Online; accessed 28-Oct-2020].

[3] JetBrains. Pycharm, 2010. URL https://www.jetbrains.com/pycharm/. [Online; accessed 28-Oct-2020].

[4] H. Krekel. Pytest, 2004. URL https://docs.pytest.org/en/stable/. [Online; accessed 28-Oct-2020].

[5] Microsoft. Github, 2020. URL https://github.com/muralidn/CAS741-Fall20. [Online; accessed 28-Oct-2020].

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

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

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

[9] V. D. Team. Valgrind, 2020. URL www.valgrind.org. [Online; accessed 28-Oct-2020].