System Verification and Validation Plan for Image Feature Correspondences for Camera Calibration

Kiran Singh

February 22, 2025

Revision History

Date	Version	Notes
2025-02-24	1.0	Initial Release

[The intention of the VnV plan is to increase confidence in the software. However, this does not mean listing every verification and validation technique that has ever been devised. The VnV plan should also be a **feasible** plan. Execution of the plan should be possible with the time and team available. If the full plan cannot be completed during the time available, it can either be modified to "fake it", or a better solution is to add a section describing what work has been completed and what work is still planned for the future. —SS]

[The VnV plan is typically started after the requirements stage, but before the design stage. This means that the sections related to unit testing cannot initially be completed. The sections will be filled in after the design stage is complete. the final version of the VnV plan should have all sections filled in. —SS]

Contents

List of Tables

[Remove this section if it isn't needed —SS]

List of Figures

[Remove this section if it isn't needed —SS]

1 Symbols, Abbreviations, and Acronyms

symbol	description
FOV	Field-of-view
SRS	Software Requirements Specification
VnV	Verification and Validation

[symbols, abbreviations, or acronyms — you can simply reference the SRS (Singh, 2019) tables, if appropriate —SS]

[Remove this section if it isn't needed —SS]

The intent of this document is to define the verification and validation process that will be used to assess the feature correspondence software from camera imagery. Specifically, this document will be used to characterize the behaviour and performance of this software. The remaining section of this document outline a high level summary of the general system system and specific objects of the VnV process. It also defines specific strategies for the individual verification plans, an overview of anticipated system tests, and an outline of specific unit test cases.

2 General Information

2.1 Summary

Image Feature Correspondences (IFC) is a feature comparison algorithm that is intended to be used as part of a pipeline to perform extrinsic camera calibration for applications in mobile robotics. It accepts camera intrinsics and imagery data at different poses to identify common features across collected images.

2.2 Objectives

The VnV process is intended to characterize how well the for the IFC software performs in its intended capacity to identify features amongst collected imagery. This can vary significantly as it is influenced by factors such as overlap in camera fields-of-view (FOV), contrast between objects in an image, and variance in either scale or rotation. Furthermore, as there is no common baseline to compare this software to as an oracle model, the intent of the VnV for the IFC software is to characterize the performance of the integrated image processing functions against a set of selected datasets. Key objectives of this process are defined below.

- correctness of feature extraction
- correctness of feature comparison
- assessment of scale variance between features
- assessment of rotation variance between features

- benchmarking of processing time and memory usage for large image strategies
- cross-validation of system performance with accepted benchmarked datasets

Characterization of the individual functions within the OpenCV library themselves remains outside of the scope of this project, as we can assume that the library has been verified by uts own implementation team.

[State what is intended to be accomplished. The objective will be around the qualities that are most important for your project. You might have something like: "build confidence in the software correctness," "demonstrate adequate usability." etc. You won't list all of the qualities, just those that are most important. —SS]

[You should also list the objectives that are out of scope. You don't have the resources to do everything, so what will you be leaving out. For instance, if you are not going to verify the quality of usability, state this. It is also worthwhile to justify why the objectives are left out. —SS]

[The objectives are important because they highlight that you are aware of limitations in your resources for verification and validation. You can't do everything, so what are you going to prioritize? As an example, if your system depends on an external library, you can explicitly state that you will assume that external library has already been verified by its implementation team. —SS]

2.3 Challenge Level and Extras

[State the challenge level (advanced, general, basic) for your project. Your challenge level should exactly match what is included in your problem statement. This should be the challenge level agreed on between you and the course instructor. You can use a pull request to update your challenge level (in TeamComposition.csv or Repos.csv) if your plan changes as a result of the VnV planning exercise. —SS]

[Summarize the extras (if any) that were tackled by this project. Extras can include usability testing, code walkthroughs, user documentation, formal proof, GenderMag personas, Design Thinking, etc. Extras should have

already been approved by the course instructor as included in your problem statement. You can use a pull request to update your extras (in TeamComposition.csv or Repos.csv) if your plan changes as a result of the VnV planning exercise. —SS]

2.4 Relevant Documentation

[Reference relevant documentation. This will definitely include your SRS and your other project documents (design documents, like MG, MIS, etc). You can include these even before they are written, since by the time the project is done, they will be written. You can create BibTeX entries for your documents and within those entries include a hyperlink to the documents. —SS]

Singh (2019)

[Don't just list the other documents. You should explain why they are relevant and how they relate to your VnV efforts. —SS]

3 Plan

[Introduce this section. You can provide a roadmap of the sections to come. —SS]

3.1 Verification and Validation Team

The VnV team consists of four members, each of whom play a distinct role in the verification process.

Name	Role	Description
Kiran Singh	Lead Developer and Test Designer	Responsibilities include identification of key business cases for integrated tests, design and implementation of module tests, unit tests, and documentation of test results.
Matthew Giamou	Project Supervisor	Lead consultant on integrated performance needs and decomposition for software modules. Responsibilities include review of proposed VnV scope, approval of test cases as proposed by Kiran S., and provision of feedback as to whether the test cases need to be redefined within the scope of the full implementation of the IFC.
Aliyah Jimoh	Peer Reviewer	Responsibilities include provision of feedback on proposed test cases for the scope of test cases for both the functional and non-functional requirements as an reviewer that is less familiar with the overall implementation of the system that its primary developer
Spencer Smith	Software Development Instructor	Responsibilities include provision of feedback on proposed test cases for the scope of test cases for both the functional and non-functional requirements as an reviewer that is less familiar with the application domain of the system than all other reviewers

3.2 SRS Verification Plan

The **SRS** shall be reviewed by each member of the reviewer team to form consensus that the SRS has been correctly decomposed into sufficient requirements.

- the models are deemed to be comprehensible
- the models are deemed to be correct

- the associated requirements are traced correctly with respect to the models and project scope
- the requirements are decomposed in a manner that facilitates verification

Feedback on the SRS from the Peer Reviewer and Instructor will be captured through the use of Github Issues. Specifically, both reviewers will use the **SRS Checklist**. The lead developer will respond in turn to each issue and if reserves the right to reject a proposed change as needed.

The Lead Developer will schedule a meeting with the Project Supervisor to walk through the first revision of the SRS document. In this meeting, the Project Supervisor will offer feedback and recommendations for candidate revisions to the outlined models and requirements. The Lead Developer will then prepare issues in Github to address each proposed revision.

3.3 Design Verification Plan

[Plans for design verification —SS] [The review will include reviews by your classmates —SS] [Create a checklists? —SS]

The Peer Reviewer and Course Instructor will review the Module Guide **(MG)** and the Module Interface Specification **(MIS)** against the **MG** and **MIS** checlists. The intent of this process is to ensure that the design of of the system is:

1. umabiguous
2. adheres to best-practices of module design
3. aligns with the requirements as identified in the **SRS**

3.4 Verification and Validation Plan Verification Plan

[The verification and validation plan is an artifact that should also be verified. Techniques for this include review and mutation testing. —SS]

The VnV Plan will be verified via inspection by the Peer Reviewer and the Software Development Instructor. The **VnV Plan Checklist** will be

used as the assessment criteria for the inspection. Feedback will provided as Github issues and will be handled in the same manner as feedback for the SRS.

Mutation testing will be performed against the test outlined in Section 5.

[The review will include reviews by your classmates —SS] [Create a checklists? —SS]

3.5 Implementation Verification Plan

[You should at least point to the tests listed in this document and the unit testing plan. —SS] [In this section you would also give any details of any plans for static verification of the implementation. Potential techniques include code walkthroughs, code inspection, static analyzers, etc. —SS] [The final class presentation in CAS 741 could be used as a code walkthrough. There is also a possibility of using the final presentation (in CAS741) for a partial usability survey. —SS]

The IFC software shall be verified against the test procedures outlined in Sections 4.1 and 4.2. Static verification of the IFC software will consist of a code walkthrough. This will take place during the CAS 741 final presentation, where the Peer Reviewer and Course Instructor in the will have the opportunity to observed the code and raise issues following the presentation via GitHub.

Dynamic verification of the IFC software will consist of system and unit tests via PyTest. System tests are outlined in Section 4. Unit tests will be outlined in Rev 2 of the VnV Plan in Section 5.

3.6 Automated Testing and Verification Tools

[What tools are you using for automated testing. Likely a unit testing framework and maybe a profiling tool, like ValGrind. Other possible tools include a static analyzer, make, continuous integration tools, test coverage tools, etc. Explain your plans for summarizing code coverage metrics. Linters are another important class of tools. For the programming language you select, you should look at the available linters. There may also be tools that verify that coding standards have been respected, like flake9 for Python. —SS] Several tools will be used to support automated testing and verification. They

include:

- Continuous Integration (CI) will be facilitated via GitHub Actions. A pull request will be used to run automated tests.
- Pytest will be used to perform system tests, unit tests, and to assess code coverage.
- flake8 will be used as a linter to ensure adherence to PEP8 standards.
- PyLint, as an alternative linter to flake8.

3.7 Software Validation Plan

[If there is any external data that can be used for validation, you should point to it here. If there are no plans for validation, you should state that here. —SS] [You might want to use review sessions with the stakeholder to check that the requirements document captures the right requirements. Maybe task based inspection? —SS] The IFC software will be compared against benchmark data from **[INSERT DATASET HERE]** to characterize its performance.

4 System Tests

[There should be text between all headings, even if it is just a roadmap of the contents of the subsections. —SS]

4.1 Tests for Functional Requirements

[Subsets of the tests may be in related, so this section is divided into different areas. If there are no identifiable subsets for the tests, this level of document structure can be removed. —SS]

[Include a blurb here to explain why the subsections below cover the requirements. References to the SRS would be good here. —SS]

4.1.1 Area of Testing1

[It would be nice to have a blurb here to explain why the subsections below cover the requirements. References to the SRS would be good here. If

a section covers tests for input constraints, you should reference the data constraints table in the SRS. —SS]

Title for Test

1. test-id1

   Control: Manual versus Automatic

   Initial State:

   Input:

   Output: [The expected result for the given inputs. Output is not how you are going to return the results of the test. The output is the expected result. —SS]

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

   How test will be performed:

2. test-id2

   Control: Manual versus Automatic

   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:

4.1.2 Area of Testing2

...

4.2 Tests for Nonfunctional Requirements

[The nonfunctional requirements for accuracy will likely just reference the appropriate functional tests from above. The test cases should mention reporting the relative error for these tests. Not all projects will necessarily have nonfunctional requirements related to accuracy. —SS]

[For some nonfunctional tests, you won't be setting a target threshold for passing the test, but rather describing the experiment you will do to measure the quality for different inputs. For instance, you could measure speed versus the problem size. The output of the test isn't pass/fail, but rather a summary table or graph. —SS]

[Tests related to usability could include conducting a usability test and survey. The survey will be in the Appendix. —SS]

[Static tests, review, inspections, and walkthroughs, will not follow the format for the tests given below. —SS]

[If you introduce static tests in your plan, you need to provide details. How will they be done? In cases like code (or document) walkthroughs, who will be involved? Be specific. —SS]

4.2.1 Area of Testing1

Title for Test

1. test-id1

   Type: Functional, Dynamic, Manual, Static etc.

   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:

4.2.2 Area of Testing2

...

4.3 Traceability Between Test Cases and Requirements

[Provide a table that shows which test cases are supporting which requirements. —SS]

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

Kiran Singh. System requirements specification. https://github.com/..., 2019. URL https://github.com/KiranSingh15/CAS-741-Image-Correspondences/blob/main/docs/SRS/SRS.pdf.

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?

[This is a section that would be appropriate for some projects. —SS]

End of document.