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

Kiran Singh

April 1, 2025

Revision History

Date	Version	Notes
2025-02-27	1.0	Initial Release

Contents

List of Tables

List of Figures

1 Symbols, Abbreviations, and Acronyms

symbol	description
CAS	Computing and Software
CI	Continuous Integration
DE	Domain Expert
d_{total}	bin size, or number of binary descriptions
D_{bin}	1D array of 32 byte binary strings
FOV	field-of-view
I	2D array of pixel values for a greyscale image
I'	transformed 2D array of pixel values
IFC	Image Feature Correspondences
k	number of total keypoints
LD	Lead Developer
m	horizontal image dimension
MG	Module Guide
MIS	Module Interface Specification
n	vertical image dimension
ORB	Oriented FAST and Rotated BRIEF
PEP8	Python Enhancement Proposal-8
PS	Project Supervisor
RGB	Red-Green-Blue coloured imagery
s	patch size
SDI	Software Design Instructor
SLAM	Simultaneous Localization and Mapping
SRS	Software Requirements Specification
t	intensity threshold
VnV	Verification and Validation
σ	standard deviation

The intent of this document is to define the verification and validation (VnV) processes that will be used to assess Image Feature Correspondences for Camera Calibration, (IFC) software. Specifically, this document will be used to characterize the behaviour and performance of this software. The remaining sections of this document outline a detailed summary of the specific objectives of the VnV campaign. This includes considerations for the scope of the VnV efforts with respect to contraints that stem from the CAS 741 development schedule, as well as anticipated activities beyond the scope of CAS 741. Verification procedures are present for select deliverables and milestone, and include an overview of anticipated cases for both system tests unit tests.

2 General Information

2.1 Summary

The Image Feature Correspondences (IFC) software 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 configuration parameters and greyscale imagery data at different poses to identify common features amongst 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. The performance of this system can vary significantly as it is influenced by factors such as overlap in camera fields-of-view (FOV), the observed contrast between objects in an image, and variance in scale, rotation, and ambient illumination conditions. Furthermore, as there is no common baseline to compare this software to as an oracle, the VnV campaign for the IFC software is intended to characterize the performance of the integrated image processing functions against a set of test datasets. Key objectives of this process are defined below.

- assess the reliability in the feature matching pipeline
- assess how users perceive their interactions with the IFC software

- characterize the performances of the IFC software with metrics such as processing time and memory usage
- cross-validate the performance of the IFC software with accepted benchmarked datasets

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

2.3 Challenge Level and Extras

The proposed software will serve as the front end of a robust optimization framework for extrinsic camera calibration. The IFC software must be capable of handling large volumes of high-resolution imagery while ensuring reliability through consistent and repeatable outputs. The software will be designed for robustness in data processing, accommodating a wide range of input conditions. Additionally, it will maintain a streamlined user experience, allowing users to override select default parameters while ensuring that the learning curve remains minimal for efficient adoption.

As the long-term goal for this software is to absorb it into a larger calibration pipeline for research in the domain of mobile robotic, this project is defined as an advanced level of challenge. In addition to the standard codebase, verification report, and documentation for the CAS 741 deliverables, a User Manual shall also be submitted to support future development of the IFC software and its integration into the full camera calibration pipeline.

2.4 Relevant Documentation

Relevant documentation has been hyperlinked throughout the length of the document. This enables the reader to access each resource within the context of the section that the reference is invoked.

3 Plan

This section outlines how each aspect of the VnV effort will be performed. This may be handled by milestone or by general practices, such as continuous integration and linters.

3.1 Verification and Validation Team

The VnV team consists of four members, each of whom play a distinct role in the verification process. These roles and responsibilities are outlined in Table 1.

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 DE and SDI 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 PS to walk through the first revision of the **SRS** document. In this meeting, the PS will offer feedback and recommendations for candidate revisions to the outlined models and requirements. The Lead Developer will then create issues in Github to address each proposed revision.

Name	Role	Description
Kiran Singh	Lead Developer and Test Designer (LD)	Responsibilities include the identification of critical business cases for integrated tests, assessment of schedule and scope considerations, design and implementation of unit tests, system tests, and documentation of test results.
Matthew Giamou	Project Supervisor (PS)	Lead consultant on integrated performance needs and decomposition for software modules. Responsibilities include review of proposed VnV scope, as proposed by Kiran S., and provision of feedback to the general scope of the VnV campaign, and approval of the individual test cases that are proposed by the LD.
Aliyah Jimoh	Domain Expert (DE)	Responsibilities include provision of feedback on proposed test cases for the scope of test cases for both the functional and non-functional requirements. They will also provide feedback on the code walkthrough per the final CAS 741 presentation. This statisfies the need for a reviewer that is removed from the design effort yet still holds sufficient domain knowledge to provide feedback on the overall design and associated VnV efforts.
Spencer Smith	Software Development Instructor (SDI)	Responsibilities include provision of feedback on proposed test cases for the scope of test cases for both the functional and non-functional requirements. They will also provide feedback on the code walkthrough per the final CAS 741 presentation. This reviewer provides an essential stream of feedback as they are fully removed from the application domain and may willing to question assumptions and address biases that may be implicit amongst other members of the VnV team.

Table 1: Roles and Responsibilities of the Verification and Validation Team

3.3 Design Verification Plan

The DE and SDI will review the Module Guide **(MG)** and the Module Interface Specification **(MIS)** against the **MG** and **MIS** checklists. The objective of this review cycle is to ensure that the design of the system:

1. is unambiguous
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 VnV Plan will be verified via inspection by the DE 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 tests outlined in Section 5, which are expected to be completed as part of a Rev 2.0 release of the VnV Plan, as part of the initial release (Rev 1.0) of the (MG) and (MIS).

3.5 Implementation Verification Plan

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 DE and SDI will have the opportunity to observe 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

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 execute 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

The IFC software will be compared against benchmark data from one or more of the following SLAM datasets that feature camera imagery data. These datasets provide a much more challenging scene to identify and match features than generated binary markers. Imagery from the selected datasets will be preprocessed from RGB data to greyscale data, and will undergo a testing procedure similar to that of the systems tests outlined in Section 4.1. These tests would be inclusive of the general form of timing and memory usage metrics outlined in Section 4.2.4. These tests may exceed the constraints of the CAS 741 schedule and will be continued as part of future work.

4 System Tests

This section outlines the general roadmap for the required integrated system tests.

4.1 Tests for Functional Requirements

This section outlines the system tests that verify the requirements outlined in Section 5 of the **SRS**.

4.1.1 Feature Detection

This test section is intended to assess that the system can accept new parameters from users and covers requirements R1 through R7 and R9 through R11, in Section 5.1 of the **SRS**.

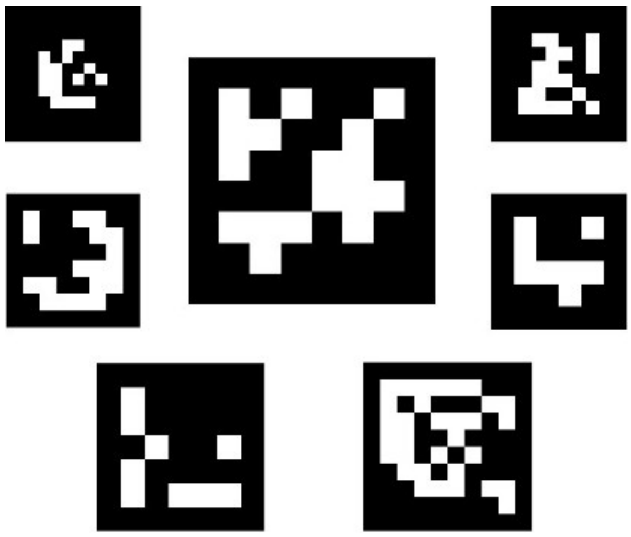

Figure 1: An example of a generated ArUco pattern. Image taken from OpenCV (2025)

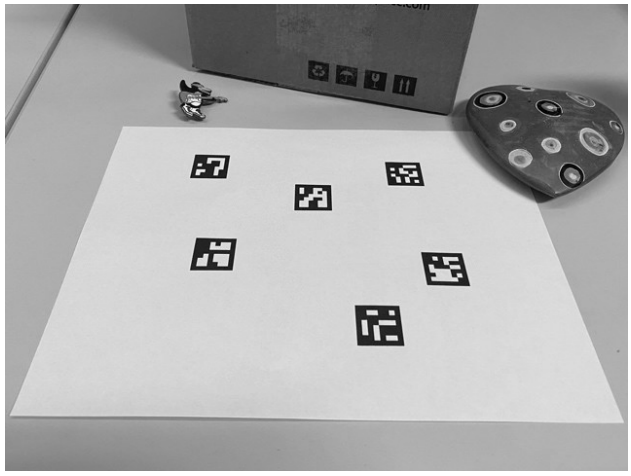

Figure 2: An example of ArUco patterns within the scene of a captured greyscale image. Image taken from OpenCV (2025)

Image Smoother

1. **STFR-IS-01**

Requirements Addressed: R1, R5, R9

Control: Automated

Initial State: Uninitialized

Input: Generated binary patterned fiducial markers, named ArUco markers, will be used as simplified targets for corner detection. A depiction of these markers is shown in Figure 1. A collection of 20 images that contain ArUco markers within the image scene will be provided as test inputs, as shown in Figure 2. Each image I will be defined as 2D array of a user-specified resolution. The user will also assign the image intensity standard deviation σ to define the size of the Gaussian Kernel.

Output: 20 output images as a 2D array, each of which have a descriptive name that clearly identifies the input image from which it originates. The I' array should be of equal size and the same data type as its corresponding image I. All images should be saved under its own uniquely named folder to prevent mixing of input and output imagery.

How test will be performed: Automated Tools (i.e. Pytest)

Keypoint Detector

1. **STFR-KP-01**
   Requirements Addressed: R2, R6, R10

   Control: Automatic

   Initial State: Uninitialized

   Input: A collection of 20 smoothed images I', each defined as 2D array of specified resolution, that contain a binary patterned fiducial markers as outlined in System Test STFR-IS-01. A permissible image intensity threshold, t, shall also be introduced, where t is defined as a non-zero positive integer between 1 and 255.

   Output: A flag which indicates that corner detection is active. A 2D array of size $k \times 2$, where k is the quantity of identified keypoints, where the first and second columns are populated with the horizontal and vertical coordinates for each keypoint. Each entry in the array should be an positive integer value. Each array should have a unique identifier that clearly defines the original image from which its keypoints were

identified. All output arrays should be saved to a unique folder to separate them from the input images.

How test will be performed: Automated Tools (i.e. Pytest)

Feature Definition

1. **STFR-FD-01**
   Requirements Addressed: R3, R4, R7, R11

   Control: Automatic

   Initial State: Uninitialized

   Input: A collection of 20 images, each defined as a 2D array of integers, and a collection of 20 $k \times 2$ 2D arrays, where k is the number of keypoints within each image. A patch size, s, will also be input as a scalar integer to define the the region for which to search for descriptors. A target number of descriptors titled d_{total}, also known as the bin size, will also be provided as a non-negative integer between 1 and 1023.

   Output: D_{bin}, a 1D array of length d_{total} where all entries of the array are 32 byte binary strings.

   Test Case Derivation: ORB: An efficient alternative to SIFT or SURF. This publication outlines methods to develop and the corresponding assessment of binary feature descriptors.

   How test will be performed: Automated Tools (i.e. Pytest)

4.1.2 Feature Comparison

This test section is intended to assess that the system can accept new parameters from users. It addresses requirements R8 and R12 through R15 in Section 5.1 of the **SRS**.

Descriptor Comparison

1. **STFR-DC-01**
   Requirements Addressed: R8, R12, R13, R14, R15

   Control: Automated

   Initial State: Uninitialized

Input: 20 1D arrays, each which have a unique length, D_{bin}, where each element of the array is a 32 byte binary string known as a binary descriptor. Each array should be labeled with a descriptive name that clearly references the instance of both the camera and pose at the time of image capture.

Output: A flag that indicates that Hamming Distances were compared. A dynamically-sized array, stored as a Pandas dataframe, where the length of the array is the number of matched features. The columns of the array will include the properties as follows.

- Binary descriptors of both features as 32 byte binary strings
- Corresponding image IDs for each feature
- Matching scores between both features

The output dataframe should verify that each image ID should be a non-negative integer.

How test will be performed: Automated Tools (i.e. Pytest)

4.2 Tests for Nonfunctional Requirements

4.2.1 Reliability

The IFC software is envisioned to be used as part of the front end of a larger camera calibration pipeline, where the outputs of the IFC software need to produce consistent results for use in a back-end optimization module. Therefore, the IFC software needs to undergo verification to ensure that it can match the same features between two images. The following system test satisfies **NFR1** of the **SRS**.

Repeatability

1. **STNFR-RE-1**
   Type: Dynamic

   Initial State: Uninitialized

   Input/Condition: Two datasets of imagery data. These datasets contain the same collection of images, but are arranged in a different order.

Output/Result: The feature matching pipeline will be use to identify features and compare them amongst images for both datasets. For both datasets, a dynamically-sized array will be output. These arrays will contain the feature descriptors and the image IDs of origin, as specified in as specified in STFR-DC-01. One array will be reordered to reflect the order of the other array. Once the second array has been reordered, the arrays will be compared in terms of their length and contents. The test will be considered a pass if the data in each element of the array can be matched to a corresponding element of the other array.

How test will be performed: PyTest

4.2.2 Usability

As part of a larger pipeline, the IFC software needs to be simple for its users to integrate into the calibration system. This may be reflected in the perception of how simple it is for the user to implement its on their own system and to run the program as its own module. This usability criterion is reflected in **NFR2** of the **SRS**.

User Experience: End-to-End Feature Matching

1. **STNFR-UX-1**
   Type: Dynamic

   Initial State: Uninitialized

   Output/Result: A dataframe of identified features, as outlined in STFR-DC-01. Once the test has been completed, the user will be asked to express their opinion of their experience of the system, and to remark on the following topics.

   - the degree of perceived difficulty in preparing the input files
   - the degree of perceived difficulty in assignment of the camera properties
   - the user's overall satisfaction with using the program

   How test will be performed: Dynamic Test by User

The user will be provided a collection of images from different cameras. Each image will be labelled with a descriptive title that clearly defines what

camera the image originates from. The user shall be responsible to insert all images into a common folder.

The user shall execute the following procedure in the following order.

1. The user shall open the directory to the IFC configuration file.
2. In the IFC program configuration file, the user shall:
   (a) Update the input location as the folder of all the input images.
   (b) Update the system directory with the desired location of the matched feature array.
   (c) Assign a directory for altered imagery, if desired.
   (d) Input the number of cameras per the sized camera data.
   (e) Provide the following details for each camera:
      i. A descriptive name of the camera that matches the prefix of the name of its imagery data.
      ii. The resolution of each camera.
3. The user shall close the IFC configuration file.
4. The user initiates execution of the feature comparison pipeline.
5. The user waits for the image comparison process to complete.
6. After completion, the user opens the output array directory.
7. The user reviews the output array to verify that there are no errant characters.
8. The user closes the output array.

Though not within the current scope of the VnV plan, a separate test may be performed where the user performs the same steps as STNFR-UX-1, and additionally modifies the Gaussian Kernel standard distribution, intensity threshold, and patch size parameters to values within the allowable operational boundaries.

4.2.3 Maintainability

Verification of **NFR3**, outlined in **SRS**, has been delegated as future work in favour of other verification procedures outlined within this document. This tradeoff was made to satisfy the schedule constraints of CAS 741. It would however, be simple to evaluate if a new detection was to be implement. Once the default feature detection method has successfully been completed, the total number of hours spent to design the feature detection module should be summed together. A factor of 0.3 should be applied to the summed effort to define the maximum target effort to develop a new method of feature detection. Any time that is dedicated to implement a new method should be added to the running total. A scenario where the total effort exceeds the allowable target may suggest that the current IFC software is difficult to maintain.

4.2.4 Performance

One of the primary objectives of the VnV campaign is to characterize what system resources are required to execute the IFC software, as outline in **NFR4** and **NFR5**. In the case of a simple configuration with only two cameras at two different poses, the process is trivial. However, for configurations that consist of as many as ten cameras across dozens of robot poses, the problem may scale significantly. Therefore, as a general practice, timing and memory usage metrics should be recorded throughout the duration of any of the primary operations of the IFC software. This includes the following system tests in Section 4.1.

1. STFR-IS-01
2. STFR-KP-01
3. STFR-FD-01
4. STFR-DC-01

Timing Metrics
The timestamp will be recorded at the start of each of the major operations (i.e. Image Smoothing, Keypoint Detection, Assignment of Feature Descriptors, and Feature Comparison). A subsequent timestamp will be recorded after the termination of each operation. The difference between the start

and termination timestamps will be saved and stored in a structured format (e.g., JSON or CSV) within a designated log directory.

Memory Usage Metrics
The memory usage of the IFC software should be logged with a timestamp and operation identifier and stored in a structured format (e.g., JSON or CSV) within a designated log directory. This data may also be augmented with a separate file that outlines the average memory usage, and the peak memory usage with associated timestamps.

4.3 Traceability Between Test Cases and Requirements

The traceability between each scoped test case and its respective requirements is outlined by 2.

	STFR-IS-01	**STFR-KP-01**	**STFR-FD-01**	**STFR-DC-01**	**STNFR-RE-1**	**STNFR-UX-1**
R1	x					
R2		x				
R3			x			
R4			x			
R5	x					
R6		x				
R7			x			
R8				x		
R9	x					
R10		x				
R11			x			
R12				x		
R13				x		
R14				x		
R15				x		
NFR1					x	
NFR2						x
NFR3*	-	-	-	-	-	-
NFR4	o	o	o	o		
NFR5	o	o	o	o		

Table 2: Traceability Matrix Showing the Connections Between Requirements and Instance Models

- 'x' indicates a direct method of verification
- 'o' indicates an indirect method of verification

- * Verification of NFR3 is defined as outside of scope per Section 4.2.3

5 Unit Test Description

This section has been tabled as future work. It will be revised in support of the initial release of the Module Guide and Module Interface Specification.

[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 3

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

OpenCV. Detection of aruco markers, 2025. URL https://docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html. Accessed: 2025-02-25.

6 Appendix

Not Applicable.

6.1 Symbolic Parameters

No symbolic constants have been identified in this document.