Verification and Validation Report: Image Feature Correspondences for Camera Calibration

Kiran Singh

April 14, 2025

1 Revision History

Date	Version	Notes
2025-04-16	Rev 1.0	Initial Release

2 Symbols, Abbreviations and Acronyms

symbol	**description**
CSV	comma separated value
FAST	Features from Accelerated Segment Test
IFCS	Image Feature Correspondences for Camera Calibration software
T	Test

Contents

1 Revision History **i**

2 Symbols, Abbreviations and Acronyms **ii**

3 Functional Requirements Evaluation **1**
3.1 Feature Detection 1
3.2 Feature Comparison 4

4 Nonfunctional Requirements Evaluation **6**
4.1 Reliability 6
4.2 Usability 6
4.3 Maintainability 7
4.4 Performance 7

5 Comparison to Existing Implementation **8**

6 Unit Testing **8**

7 Changes Due to Testing **8**

8 Automated Testing **8**

9 Trace to Requirements **9**

10 Trace to Modules **9**

11 Code Coverage Metrics **9**

List of Tables

List of Figures

1 Outputs of the image smoothing procedure on an untransformed ArUco marker 2
2 Greyscale and noise-reduced images generated from the building dataset 3

3 Generated images of the keypoint detection test with mapped keypoints 4
4 Generated images of the keypoint detection test with mapped keypoints 5
5 Optimal candidate matches between ArUco_000 and ArUco_001 6
6 Outline of Automated Unit Tests 9

This document outlines the results of the system and unit tests for the Image Feature Correspondences for Camera Calibration software. The detaileds of the associated tests are outlined in the **VnV Plan**.

3 Functional Requirements Evaluation

3.1 Feature Detection

Image Smoother

1. **STFR-IS-01**

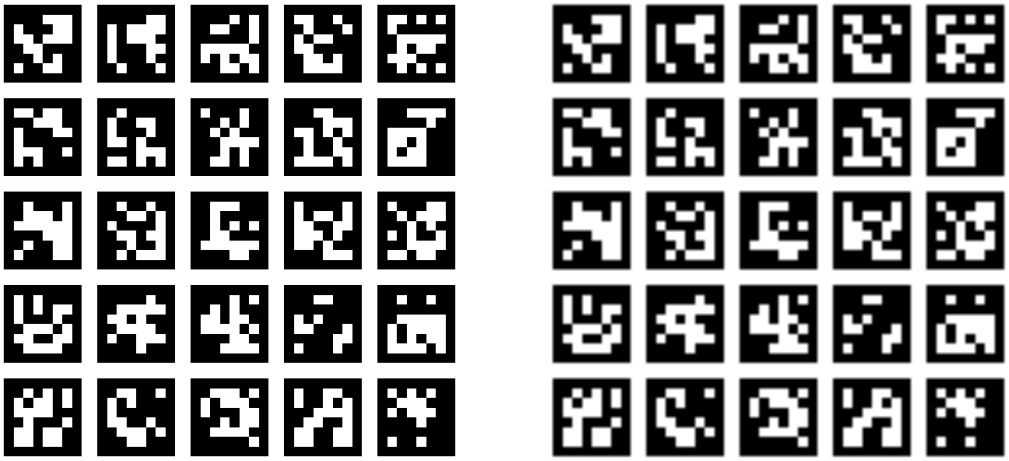
   This test evaluates the capacity of the system to import generated ArUco marker imagery, convert the imagery to a greyscale format, perform Gaussian smoothing, and save both the greyscale and smoothed images using a kernel size of 5 and a standard deviation of 1.0. This test was manually initiated and executed via Pytest using the test_STFR-IS-01.py program. The test passed all checks and provides a summary.txt of the results under its timestamped STFR-IS-01 output folder.

   Requirements Addressed:

   - R1 (Update Image Noise)
   - R5 (Default Noise Suppression)
   - R9 (Perform Noise Reduction)

   Figure 1 shows an example of the generated greyscale and smoothed imagery of the ArUco markers. All generated output imagery can be reviewed in the STFR-IS-01 output folder.

   As the results from processing a binary ArUco marker do not produce a distinct change in colour to the human eye, a second test was executed on the test imagery provided in the testImages\building folder.



(a) Converted Greyscale Image (b) Gaussian-Smoothed Image

Figure 1: Outputs of the image smoothing procedure on an untransformed ArUco marker

Keypoint Detection

1. **STFR-KP-01**
   This test evaluates the capacity of the system to import smoothed ArUco marker imagery and identify keypoints through use of rotated-FAST methods. A pixel intensity threshold of 60 was set for this test. This test was manually initiated and executed via Pytest using the test_STFR-KP-01.py program. The test successfully passed all checks and provides a summary.txt of the results under its timestamped STFR-KP-01 output folder.

   Requirements Addressed:

   - R2 (Update Pixel Intensity Threshold)
   - R6 (Perform Corner Detection)
   - R10 (Identify Keypoints)

   Figure 3 shows an example of the generated greyscale and smoothed imagery of the ArUco markers. All generated output imagery and corresponding CSV files can be reviewed in the STFR-KP-01 output folder.

(a) Building - RGB Input Image

(b) Building - Greyscale Image

(c) Building - Smoothed Image

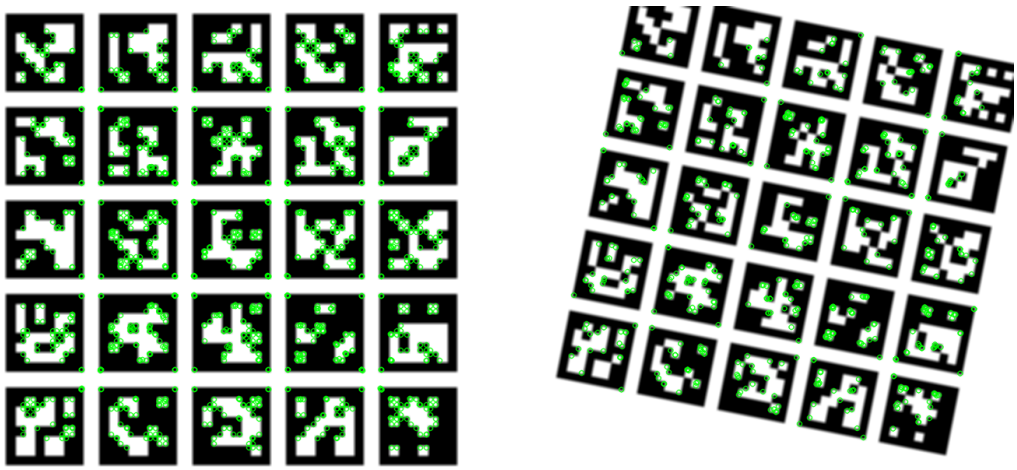
Figure 2: Greyscale and noise-reduced images generated from the building dataset

Feature Description

1. **STFR-FD-01**
   This test evaluates the capacity of the system to create oriented-BRIEF descriptors using identified keypoints from smoothed greyscale imagery. A target of 100 was set with a search patch size of 31. This test was manually initiated and executed via Pytest using the test_STFR-FD-01.py program. The test successfully passed all checks and provides a summary.txt of the results under its timestamped STFR-FD-01 output folder.

   Requirements Addressed:

   - R3 (Update Patch Size)



(a) ArUco_000 with keypoints (b) ArUco_001 with keypoints

Figure 3: Generated images of the keypoint detection test with mapped keypoints

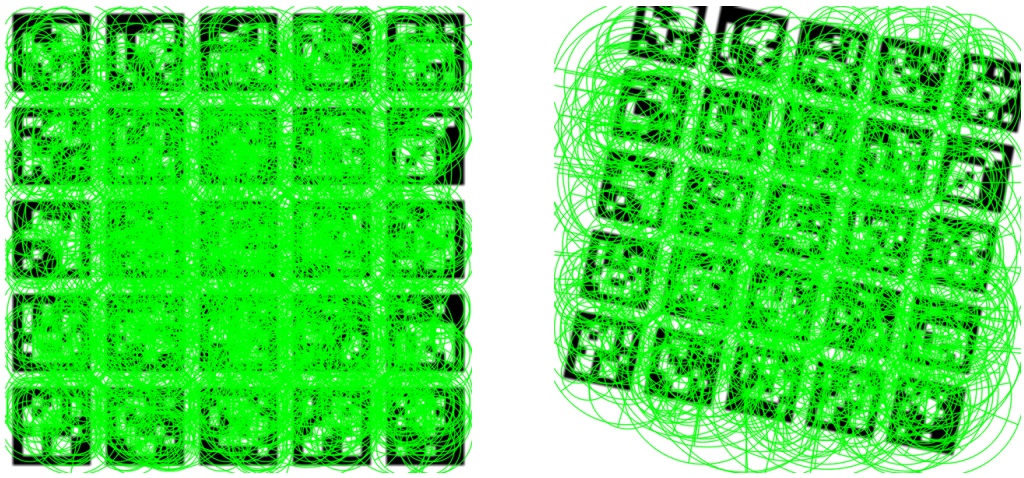
- R4 (Update Descriptor Bin Size)
- R7 (Binary Descriptors)
- R11 (Define Descriptors)

Figure 4 shows an example of two ArUco markers with keypoints scaled per their feature descriptors. All generated output imagery and corresponding CSV files can be reviewed in the STFR-FD-01 output folder.

3.2 Feature Comparison

Descriptor Comparison

1. **STFR-FM-01**
   This test evaluates the capacity of the system to compare two sets of predefined feature descriptor between two similar images. Using brute-force matching, a maximum Hamming distance of 25 was set for displayed images. Additionally, a maximum of 30 match candidates were permitted to be displayed in the generated imagery. This test was



(a) ArUco_000 with scaled keypoints (b) ArUco_001 with scaled keypoints

Figure 4: Generated images of the keypoint detection test with mapped keypoints

manually initiated and executed via Pytest using the test_STFR-FM-01.py program. The test successfully passed all checks and provides a summary.txt of the results under its timestamped STFR-FM-01 output folder.

Requirements Addressed:

- R8 (Descriptor Comparison - Hamming Distance)
- R12 (Search for Matches Candidates)
- R13 (Verify Match Candidates)
- R14 (Confirm Pose Identifiers)
- R15 (Report Match Candidates)

Figure 5 shows an example of thirty candidate feature matches between markers ArUco_000 and ArUco_001. All generated output imagery and corresponding CSV files can be reviewed in the STFR-FM-01 output folder.

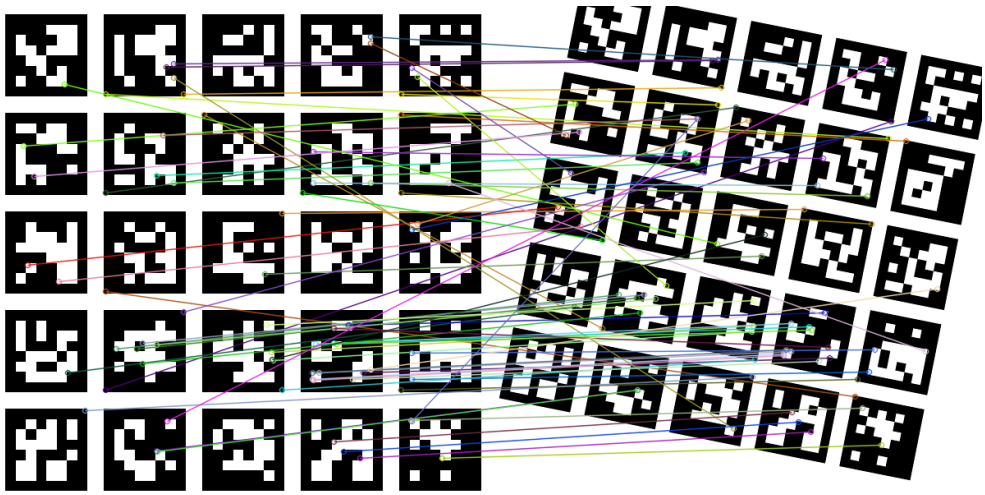


Figure 5: Optimal candidate matches between ArUco_000 and ArUco_001

4 Nonfunctional Requirements Evaluation

4.1 Reliability

- NFR1 (Invariance to the Order of Images)

A test script named **test_STNFR-RE-01.py** was prepared. In this script, two images from the **Lego** image set are renamed and reordered, as outlined in the **Lego_Swap** folder. Both of these datasets are processed to identify candidate matches. The resulting match candidates are compared between each data set to confirm that each descriptor and corresponding coordinates from the Lego dataset is matched to the equivalent descriptor in the transformed Lego_Swap dataset. This test passed with zero errors, and the results are summarized in the **STNFR-RE-01**.

4.2 Usability

Ten (10) individuals were given a demonstration of the IFCS software, which included a detailed walkthough of the download process, installation processes, and setup of the virtual environment. Audience members were shown how to add new images from the provided library to the inputs folder, adjust the methods and parameters of image processing, and initiate the IFCS

pipeline. Once completed, audience members were shown the following outputs.

- generated greyscale imagery
- generated smoothed imagery
- generated imagery with keypoints
- generated imagery with descriptors
- generated imagery that compare images with candidate descriptor matches
- CSV files of keypoints, descriptors and candidate matches

Following the demonstration, a show-of-hands identified that seven (7) of ten audience members stated that they have a favourable opinion of the software and its simplicity of use. At a success rate of 70%, this fall below the target rate of 80%. However, this may be improved with the implementation of any one of a user manual, user walkthrough video, or a one-on-one training session with one of the IFCS developers.

4.3 Maintainability

Requirements Addressed:

- NFR3 (Allocation of Developer Resources for New Features)

Per the **VnV Plan**, this requirement has been identified as out of scope for the Rev 1.0 release.

4.4 Performance

Requirements Addressed:

- NFR4 (Timing Metrics)
- NFR5 (Memory Usage Metrics)

Timing metrics and memory usage were identified as test features of interest for the lifespan of the IFCS software. These tools would be appended to compare the relative performance of different methods such as FAST and Harris scores for keypoint detection. However, as the scope of the Winter 2025 development cycle narrowed to prioritize robust performance of ORB feature detection and brute-force matching, the implementation of timing metrics will be deferred to the development cycle of Summer 2025. These metric wil be assessed as part of the systems tests as follows through the Pytest Monitor plugin, which has the capacity to assess both timing and memory metrics and is suitable for integration with GitHub Actions.

5 Comparison to Existing Implementation

This section is **not applicable**.

6 Unit Testing

7 Changes Due to Testing

[This section should highlight how feedback from the users and from the supervisor (when one exists) shaped the final product. In particular the feedback from the Rev 0 demo to the supervisor (or to potential users) should be highlighted. —SS]

8 Automated Testing

All unit tests are automated to run via a pull request and Pytest Github Actions. These tests can be found in the **test**. Each test is initiated by the **run_unit_checks.py** program, as shown in Figure 6.

```
14      # List of test files to run
15      test_files = [
16          "test_specParams.py",
17          "test_config.py",
18          "test_imagesmooth.py",
19          "test_kpdetect.py",
20          "test_featdesc.py",
21          "test_featmatches.py",
22          "test_imagePlot.py",
23          "test_outputFormat.py",
24          "test_verifyOutput.py",
25          "test_main.py",
26      ]
```

Figure 6: Outline of Automated Unit Tests

9 Trace to Requirements

10 Trace to Modules

11 Code Coverage Metrics

Appendix — Reflection

The information in this section will be used to evaluate the team members on the graduate attribute of Reflection.

The purpose of reflection questions is to give you a chance to assess your own learning and that of your group as a whole, and to find ways to improve in the future. Reflection is an important part of the learning process. Reflection is also an essential component of a successful software development process.

Reflections are most interesting and useful when they're honest, even if the stories they tell are imperfect. You will be marked based on your depth of thought and analysis, and not based on the content of the reflections themselves. Thus, for full marks we encourage you to answer openly and honestly and to avoid simply writing "what you think the evaluator wants to hear."

Please answer the following questions. Some questions can be answered on the team level, but where appropriate, each team member should write their own response:

1. What went well while writing this deliverable?

2. What pain points did you experience during this deliverable, and how did you resolve them?

3. Which parts of this document stemmed from speaking to your client(s) or a proxy (e.g. your peers)? Which ones were not, and why?

4. In what ways was the Verification and Validation (VnV) Plan different from the activities that were actually conducted for VnV? If there were differences, what changes required the modification in the plan? Why did these changes occur? Would you be able to anticipate these changes in future projects? If there weren't any differences, how was your team able to clearly predict a feasible amount of effort and the right tasks needed to build the evidence that demonstrates the required quality? (It is expected that most teams will have had to deviate from their original VnV Plan.)