Object Detection Report

Brooklyn Taylor - 4308085

# 1. Introduction

This report details the development and evaluation of an object detection program using C++ and OpenCV. The project aimed to accurately detect a specific object within a scene using feature detection, descriptor matching, homography computation, and bounding box visualization. The focus was on detection accuracy over real-time performance, with multiple methods tested and prioritized based on accuracy, even at the cost of longer execution times.

# 2. Implementation of Object Detection

This section outlines the implementation of object detection based on the assignment parameters, referred to as the ‘Standard Test.’ It provides an overview of the variables and methods used, with testing results included later to justify design choices.

## 2.1 Feature Detection & Descriptor Extraction

**Method Implemented:**

* After running speed tests and analysing feature matching and inlier counts, SIFT was chosen for its superior accuracy despite the longer execution times.

## 2.2 Descriptor Matching

**Method Implemented:**

* Testing different matching methods with the SIFT detector showed similar feature matching results. However, due to SIFT producing many float-type descriptors, FLANN was selected based on test performance.

## 2.3 Homography Computation

**Method Implemented:**

* RANSAC was applied with cv::findHomography() to compute a robust homography matrix. This step was crucial in filtering outlier matches and ensuring that the detected object’s boundary box is properly localized in the scene.

## 2.4 Bounding Box Visualization

* **Technique:**
  + The object’s corners in the scene were defined by the detector and matcher, and cv::perspectiveTransform() was used to project these points into the scene.
  + The projected points were then connected using cv::line() to draw a bounding box around the detected object in scene.
  + This gave a visual representation of the object detection in the scene, though did not give value representations of the accuracy, this was addressed in later testing. Due to the method of accuracy detection I used, the provided sample could not be used due to a lack of all visible corners, while these could be approximated, I thought it to be more suitable to test with similar objects in environments with more visible corners.

## 2.5 Testing Results and Choice Justification

### Detector

**Approach:**

* Both SIFT and ORB detect KeyPoints and compute descriptors using OpenCV’s detectAndCompute(). The detector will be selected based on the number of inliers it produces from the KeyPoint matches it finds in both images, since this is a static program rather than a real time application accuracy is the determinant.

A table with numbers and symbols

AI-generated content may be incorrect.**Test Results:**

I started with a feature cap of 5000 for both detectors, which they each reached—making it a suitable benchmark for comparing accuracy. To ensure consistency, both used FLANN for feature matching, with LSH indexing for ORB. I varied Lowe's ratio between 0.6 and 0.8 for comparison, logging the results in Figure 1.

While ORB achieved a better inlier-to-match ratio on average, SIFT produced a significantly higher number of inliers overall.

Figure 1. Test Results

Visual inspection of feature matches revealed further insights: although both detectors generated reasonably accurate bounding boxes, ORB often matched non-ideal keypoints within the scene—leading to background interference. In contrast, SIFT consistently matched keypoints between the object and scene with greater precision and minimal noise;  
A blue book with green and blue eyes

AI-generated content may be incorrect. A headphones on a table

AI-generated content may be incorrect. A book and headphones on a desk

AI-generated content may be incorrect.

Figure 2 Object and Scene KeyPoints with Boundary Box drawn using the SIFT detector and FLANN matcher

A pile of papers and headphones

AI-generated content may be incorrect.

Figure 3 Feature Matching using OpenCV and SIFT KeyPoints

In contrast, ORB’s key points focused heavily on gradient and colour differences, often detecting corners across the background. As a result, its feature matching tended to prioritize background elements over the object itself.  
A blue and green cover with green circles

AI-generated content may be incorrect. A book and headphones on a table

AI-generated content may be incorrect. A book and headphones on a desk

AI-generated content may be incorrect.

Figure 4 Object and Scene KeyPoints with Boundary Box drawn using the ORB detector and FLANN matcher

A book and headphones on a desk

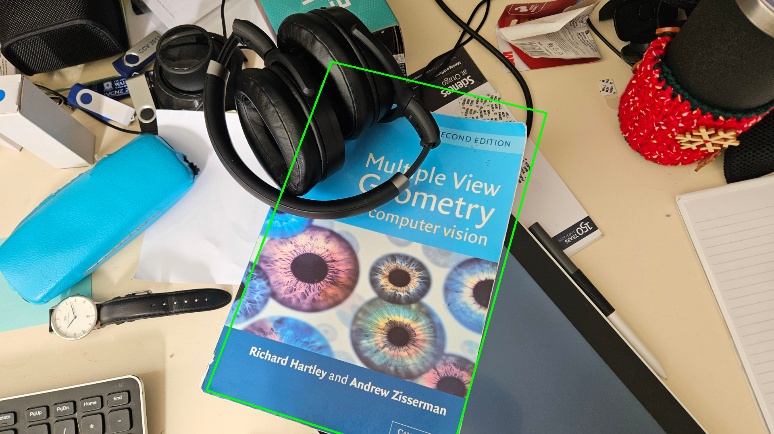
AI-generated content may be incorrect.  
A computer keyboard and headphones on a desk

AI-generated content may be incorrect.

Figure 5 SIFT with a Lowe Ratio of 0.8

Figure 6 Feature Matching using OpenCV and ORB KeyPoints

**Conclusion:**

* SIFT’s higher accuracy output with detecting robust KeyPoints and considerable inlier output outweighed its slower speed performance compared to ORB.
*  Testing with a SIFT feature cap of 1000 yielded unsatisfactory results (see Figure 1). Loosening Lowe’s ratio from 0.8 to 0.6 slightly improved coverage but remained visually inaccurate (see Figure 2).
* I ended up keeping the feature cap at its default ‘0’, which allows the algorithm to detect as many KeyPoints as possible based on its internal criteria. This gave a higher degree of accuracy, I tried to refine this further by adjusting the Lowes ratio, keeping the feature cap at 0 (INF) I adjusted the ratio to compare the number of inliers each result produced. While looking at the test data the ratio of 0.6 produced the best ratio of inliers from the matches, the ratio that produced the greater number of inliers was selected.A table with numbers and text

  AI-generated content may be incorrect.

Figure 7 SIFT with a Lowe Ratio of 0.6

Figure 8 Test Results with Lowe Ratio Experimentation

### Matcher and Feature Extraction

**Approach:**

Fast Library for Approximate Nearest Neighbours (FLANN) and Brute-Force Matching (BFM) were tested using k-nearest neighbours matching with *k = 2*. A Lowe’s ratio of 0.8 was applied to filter matches, ensuring only strong matches contributed to homography computation. For consistency, SIFT was chosen as the detection method for the test

**Test Results:**

Running the test under the same desired conditions as before, with a Lowe Ratio of 0.8, both matchers gave an identical number of matches, but BFM produced a slightly higher output of good matches and features from them after filtering using the Lowes ratio test. This resulted in a higher number of inliers but not by a wide margin.  
A white rectangular box with black text

AI-generated content may be incorrect.

Figure 9 Matcher Test Results using SIFT

**Observations:**

## While FLANN and BFM produced similar match counts, the Lowe’s ratio test yielded better results with BFM—though BFM was 6.7x slower than FLANN. In later accuracy tests with varied images, FLANN provided more accurate match filtering with SIFT compared to BFM. FLANN handles high-dimensional float descriptors efficiently using approximate nearest neighbour search, making it significantly faster and more scalable than BFM. Although BFM is accurate, it becomes computationally expensive with the large number of features SIFT generates, especially on larger images. Overall, FLANN struck the best balance between speed and accuracy, making it the preferred matcher for these test cases.

## 2.5 Build and Run Instructions

### Compilation:

The project is built using CMake and compiled in Visual Studio. The source code values have not been altered much from the provided skeleton code aside from these changes.

* CMakeLists.txt: changed CMAKE\_TOOLCHAIN\_FILE path to "C:\\vcpkg\\scripts\\buildsystems\\vcpkg.cmake" for my own system.
* Launch.vs.json: Commented out preset arguments in favour of a terminal menu for selecting methods.

### Execution:

The skeleton code was significantly modified to create a robust testing environment with a terminal menu for variable configuration and test selection.

* **Option 1:** Runs object detection with the chosen parameters using the provided object and scene images, providing the output given the chosen parameters I deemed best for this project; this uses the provided object and scene images for object detection.
* **Option 2:** Displays usage by printing current settings (default values for the assignment).
* **Settings Menu:** Allows changing the object and scene filenames (from the working directory) and switching the detector and matcher types.
* Additionally, the other menu options are for testing data I noted in this report, while the Speed Test will work with the provided set of object and scene, the Accuracy Test will need disabling the “default” Boolean and changing filenames, additionally these files are available in my public repo noted at bottom. *Note these two methods are not needed for the assignment*

# 3. Experiment Design

## 3.1 Test Data

### Speed Test

* The provided object and scene images (“Multiple View Geometry” book on a cluttered desk) were used.

### Accuracy Test

* A computer and a mouse on a table

  AI-generated content may be incorrect.A box of monopoly game

  AI-generated content may be incorrect.A photo of a card playing game “Monopoly Deal” was used in conjunction with;

Two scene images (md-scene1 & md-scene2) of the game on a desk with clutter, both scene identical with one being closer to the table (md-scene2), resulting in a larger in-scene object presence.

* A photo of a book “Asterix: The Champion” was used in conjunction with;

Figure 10 Monopoly Deal (md)

Figure 11 md-scene1

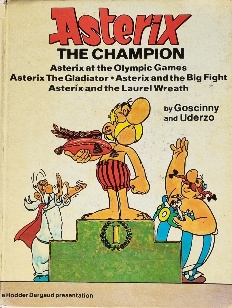
* + A scene image of books laid in a grid formation, denoted b-scene1. (*see figure 14*)
  + A scene image of books scattered around, covering ~50% of the object, denoted b-scene2. (*see figure 15*)
  + A darker image of the b-scene1 with its brightness scaled down, denoted b-sceneD

Figure 13 Asterix book

Figure 14 Book scene cluttered.

## 3.2 Hypotheses and Experimental Questions

**Speed Test:**

* While ORB is faster than SIFT, I compared both detection methods to assess if the speed gain justifies any loss in accuracy by comparing their matches and feature detection.

Figure 15 Book scene grid

**Accuracy Test:**

* It was hypothesized that despite SIFT's longer processing time, its ability to accurately detect the object is significantly better. The test aimed to quantify this accuracy using a defined metric. Additionally, lab observations suggested that BFM yields superior matching results, this was something I wanted to test across a broader range of images.

## 3.3 Benchmarking Approaches

### Method for Testing Speed:

Each detector/matcher combination was executed twenty times. ORB detector used NORM\_HAMMING for the BFM and used the index parameters for the tables, key size, and multi probe levels as 12, 20, 2 respectively, for the FLANN matcher. Detectors had a feature cap of 5000, and used the Lowes Ratio test of 0.8. Performance was measured by averaging the execution times. Total elapsed time over twenty tests divided by 20.

Key performance metrics included:

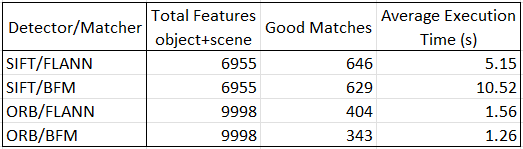
* Execution time for feature detection.
* Execution time for descriptor matching.
* Number of KeyPoints detected and matches produced.

### Method for Testing Accuracy:

To determine the true object size in the scene, OpenCV features were used to present the user with a reference photo and corresponding scene image, then manually selected the object's visible corners. These “ground truth” corners (with a 5-pixel user error margin) allowed the detected bounding box corners—computed via the homography matrix—to be compared using Euclidean distance between the vectors. Default corners were defined to minimize user error and maintain consistency across tests. Five tests using different object and scene combinations were conducted for each detector to compute the average pixel error per corner and per test.

# 4. Test Results

## 4.1 Detection Execution Times

**Test Results:**

The test was conducted to determine the average time it took for each detector matcher combination to find features, and find the good matches from them.

**Observations:**

* ORB was much faster than SIFT in terms of raw execution time, taking around 18% of the total time SIFT executes in; however, SIFT’s output was significantly more accurate, outputting more good matches with fewer total features.
* Despite BFM typically yielding better inliers, its count of "good matches" was lower than FLANN's. This is because BFM, being exact, finds the true closest and second-closest matches, resulting in smaller distance gaps and fewer matches passing Lowe's ratio test. In contrast, FLANN's approximate search often inflates these gaps, allowing more matches to pass and resulting in a higher count.
* BFM performs slower with SIFT yet faster with ORB, this would be because while BFM operates by comparing each descriptor in image A to *every* descriptor in image B using Euclidean distance, obviously slowing down matching with large data sets (big images), so It’s O(n²) and doesn’t scale well with large numbers of KeyPoints which SIFT provides. But ORB uses BFM with Hamming distance between binary strings, meaning that even though its complexity is still O(n²) its bitwise operations are hardware accelerated.

## 4.2 Detection Accuracy

### Test Results:

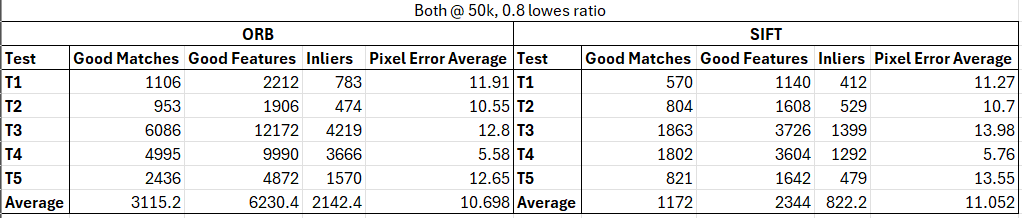
****I initially ran the program to manually define object corners in each scene, establishing consistent defaults for averaging tests. The results below show the detected corners, with abbreviated data for clarity. A summary table presents the average pixel errors;

Figure 12 Initial Accuracy Test with a feature cap of 50,000 for both detectors

**ORB Detector Accuracy Test Results:**

Test 1: md + md-scene1

Default Corners

TopL: (675, 1736)

TopR: (1024, 1478)

BotR: (1328, 1883)

BotL: (961, 2154)

Number of inliers: 783 out of 1106 good matches.

Detected Object Corners:

[679.5, 1732.14]

[1024.02, 1486.65]

[1311.18, 1884.42]

[946.574, 2146.72]  
Pixel Errors;

TopL: 5.93037 pixels

TopR: 8.65482 pixels

BotR Error: 16.8757 pixels

BotL Error: 16.1601 pixels

Avg. Error: 11.9 pixels

Test 2: md + md-scene2

Default Corners

TopL: (961, 1849)

TopR: (1454, 1430)

BotR: (1972, 2044)

BotL: (1471, 2463)

Number of inliers: 474 out of 953 good matches.

Detected Object Corners:

[969.494, 1848.99]

[1451.07, 1435.09]

[1962.21, 2037.25]

[1455.35, 2460.1]  
Pixel Errors;

TopL: 8.49439 pixels

TopR: 5.8753 pixels

BotR Error: 11.8944 pixels

BotL Error: 15.9193 pixels

Avg. Error: 10.5 pixels

Test 3: book + b-scene1

Default Corners

TopL: (41, 1576)

TopR: (978, 1549)

BotR: (957, 2754)

BotL: (16, 2800)

Number of inliers: 4219 out of 6086 good matches.

Detected Object Corners:

[48.7008, 1572.02]

[971.441, 1554.2]

[932.914, 2744.39]

[10.611, 2806.22]  
Pixel Errors;

TopL: 8.66895 pixels

TopR: 8.37035 pixels

BotR Error: 25.9336 pixels

BotL Error: 8.22852 pixels

Avg. Error: 12.8 pixels

Test 4: book + b-scene2

Default Corners

TopL: (1372, 1085)

TopR: (2132, 654)

BotR: (2735, 1638)

BotL: (1987, 2094)

Number of inliers: 3666 out of 4995 good matches.

Detected Object Corners:

[1368.7, 1088.44]

[2134.5, 660.802]

[2736.19, 1641.63]

[1992.65, 2097.16]  
Pixel Errors;

TopL: 4.76814 pixels

TopR: 7.2462 pixels

BotR Error: 3.82283 pixels

BotL Error: 6.47348 pixels

Avg. Error: 5.6 pixels

Test 5: book + b-sceneD

Default Corners

TopL: (41, 1576)

TopR: (978, 1549)

BotR: (957, 2754)

BotL: (16, 2800)

Number of inliers: 1570 out of 2436 good matches.

Detected Object Corners:

[48.0941, 1572.3]

[971.059, 1554.08]

[933.399, 2744.19]

[11.0263, 2806.82]  
Pixel Errors;

TopL: 8.00329 pixels

TopR: 8.60062 pixels

BotR Error: 25.5572 pixels

BotL Error: 8.44303 pixels

Avg. Error: 12.6 pixels

**SIFT Detector Accuracy Test Results:**

Test 1: md + md-scene1

Default Corners

TopL: (675, 1736)

TopR: (1024, 1478)

BotR: (1328, 1883)

BotL: (961, 2154)

Number of inliers: 412 out of 570 good matches.

Detected Object Corners:

[678.189, 1732.36]

[1024.19, 1485.51]

[1311.53, 1885.01]

[946.308, 2147.24]  
Pixel Errors;

TopL: 4.83615 pixels

TopR: 7.50978 pixels

BotR Error: 16.5899 pixels

BotL Error: 16.1724 pixels

Avg. Error: 11.3 pixels

Test 2: md + md-scene2

Default Corners

TopL: (961, 1849)

TopR: (1454, 1430)

BotR: (1972, 2044)

BotL: (1471, 2463)

Number of inliers: 529 out of 804 good matches.

Detected Object Corners:

[966.016, 1848.62]

[1451.38, 1435.76]

[1958.97, 2037.28]

[1454.55, 2459.66]  
Pixel Errors;

TopL: 5.03015 pixels

TopR: 6.32687 pixels

BotR Error: 14.6614 pixels

BotL Error: 16.7858 pixels

Avg. Error: 10.7 pixels

Test 3: book + b-scene1

Default Corners

TopL: (41, 1576)

TopR: (978, 1549)

BotR: (957, 2754)

BotL: (16, 2800)

Number of inliers: 1499 out of 1992 good matches.

Detected Object Corners:

[48.7821, 1573.23]

[968.973, 1556.39]

[935.002, 2745.12]

[9.3464, 2809.16]  
Pixel Errors;

TopL: 8.26169 pixels

TopR: 11.6682 pixels

BotR Error: 23.7236 pixels

BotL Error: 11.3192 pixels

Avg. Error: 13.7 pixels

Test 4: book + b-scene2

Default Corners

TopL: (1372, 1085)

TopR: (2132, 654)

BotR: (2735, 1638)

BotL: (1987, 2094)

Number of inliers: 1360 out of 1820 good matches.

Detected Object Corners:

[1367, 1088.04]

[2134.24, 660.96]

[2731.22, 1640.15]

[1992.59, 2096.82]  
Pixel Errors;

TopL: 5.85317 pixels

TopR: 7.31213 pixels

BotR Error: 4.3527 pixels

BotL Error: 6.25972 pixels

Avg. Error: 5.9 pixels

Test 5: book + b-sceneD

Default Corners

TopL: (41, 1576)

TopR: (978, 1549)

BotR: (957, 2754)

BotL: (16, 2800)

Number of inliers: 479 out of 821 good matches.

Detected Object Corners:

[47.8987, 1572.47]

[969.7, 1555.63]

[933.833, 2743.56]

[9.9246, 2808.44]  
Pixel Errors;

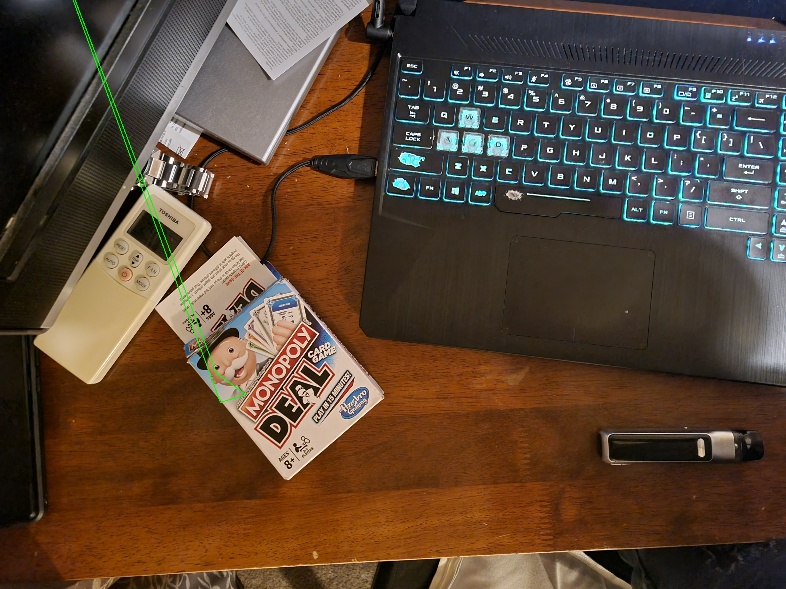
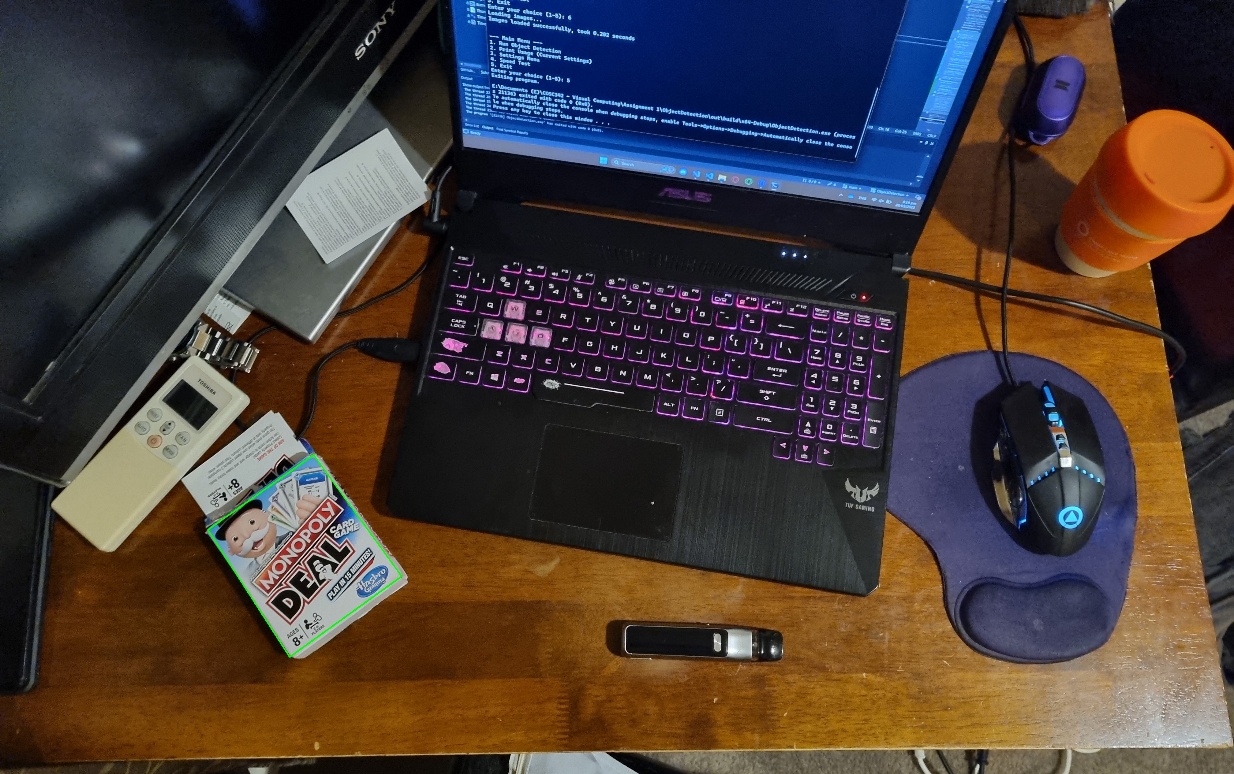
TopL: 7.74744 pixels

TopR: 10.6211 pixels

BotR Error: 25.4124 pixels

BotL Error: 10.401 pixels

Avg. Error: 13.5 pixels

**Observations:**

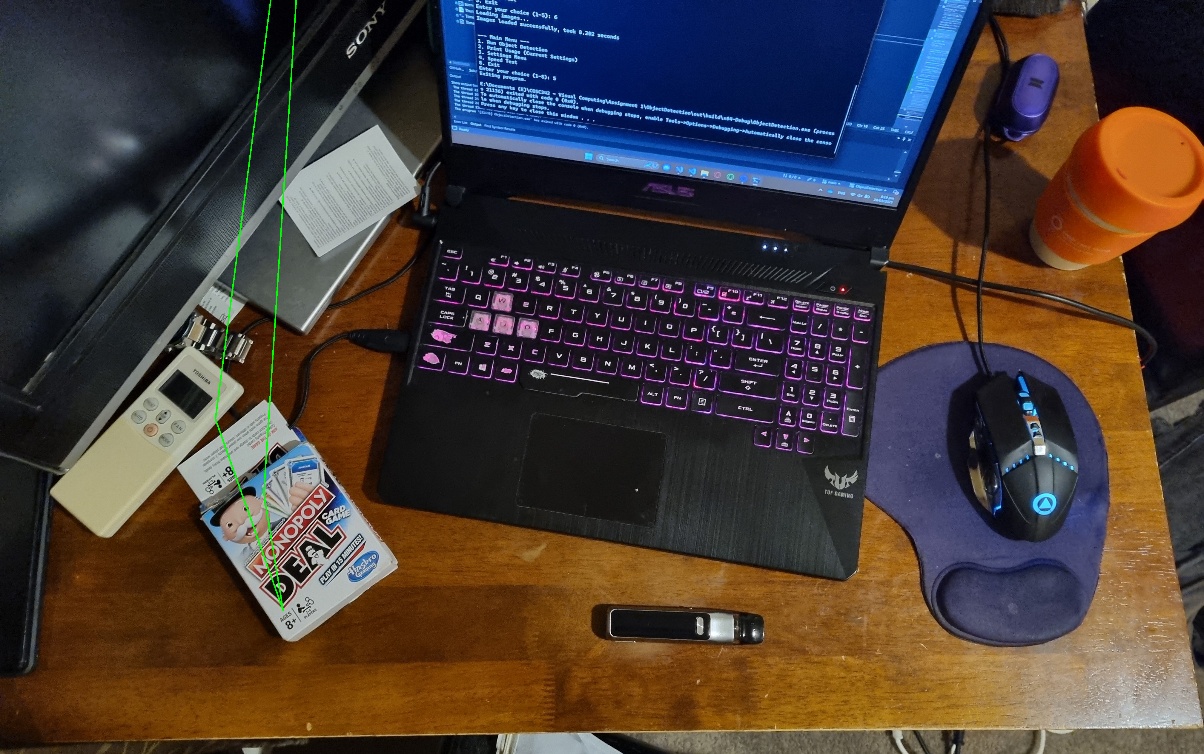
Initial tests showed ORB struggled to detect the object in Test 1 and Test 2 (Figures 14 & 15), likely due to the large scale difference between the object image (2171×2648) and its scene size (~446×505). Since ORB is not as robust to scale variance as SIFT, this discrepancy hindered accurate detection.  
Raising ORB’s feature cap to 50,000 improved results slightly. Additionally, scaling the object image down by 0.4× was the minimum required adjustment for reliable detection (Figure 13).

Figure 13 Detected with workaround

Figure 14 Failed Detection Scene1

At this high cap, SIFT produced more, though potentially less robust, features. A lenient Lowe’s ratio of 0.6 combined with 50K features yielded a slightly lower average pixel error for SIFT and a higher error for ORB (Figure 15). Lowering the cap to 5000 had minimal effect on SIFT (less than 1% error decrease) but caused ORB’s error to jump over 23% (Figure 16).  
Further testing with lower caps—1,000 for ORB and 500 for SIFT—intended to favour robustness over quantity, backfired. SIFT’s error rose by 25% and ORB’s by over 84% compared to the original 50K-cap test (Figure 18).

Figure 15 Failed Detection Scene 2

**Additional Test Results:**

**A screenshot of a graph

AI-generated content may be incorrect.**

Figure 16 Feature cap of 50,000, Lowes Ratio of 0.6

**A screenshot of a computer

AI-generated content may be incorrect.**

Figure 17 Feature cap of 5000, Lowes Ratio of 0.8

**A screenshot of a computer error

AI-generated content may be incorrect.**

Figure 18 Feature cap of 1K for ORB, 500 for SIFT, Lowes Ratio of 0.7

**Final Observations:**

* Preprocessing was necessary in Test 1 for ORB due to its poor performance with large scale differences between object and scene sizes—an issue not present with SIFT, which handled scale variance well. Reducing ORB’s feature cap below 5000 caused failure in Test 2, reinforcing this limitation.
* In Test 4 (book.jpg + b-scene.jpg), both detection methods produced a lower average pixel error compared to other tests, despite the object's corner not being visibly present in the scene. To compensate, I estimated the corner positions using on-screen rulers. This outcome highlights the strength of the homography calculation, which effectively infers object geometry based on matched KeyPoints. Interestingly, the method appears to perform better when approximating corners through overall geometry rather than relying solely on clearly defined visible corners.

# 6. Conclusion

This project successfully implemented an object-in-scene detection program using C++ and OpenCV. The SIFT and FLANN combination, while slower, delivered the accuracy needed to detect the object reliably. The experiments emphasized the trade-off between speed and accuracy across varying images and conditions.

Although not required, I thoroughly enjoyed this assignment and experimenting with different parameters out of interest, structuring my program to support future personal experimentation with more robust and adaptable detection methods.

Due to the extensive test data collected, some results were omitted to keep the report reasonably concise. Despite the 9-page length. For instance, tests with non-planar objects were excluded, as they added two extra pages. Additional testing notes and the full project are available in a public GitHub repository.

<https://github.com/BrooklynT101/ObjectDetection>

## 7. Disclaimer

This version of the report is a shortened copy of the original long version, with help from ChatGPT used only to condense certain paragraphs. No data, results, or conclusions were altered in the process.