Object Detection Report

Brooklyn Taylor - 4308085

# 1. Introduction

This report outlines the development and evaluation of an object detection program implemented using C++ and OpenCV. The project’s primary focus was to detect a specific object within a scene by leveraging feature detection, descriptor matching, homography computation, and using a bounding box for visualization. The goal was not real-time performance but rather to achieve high accuracy in the detection. To that end, various methods were evaluated with method experimentation, with a particular emphasis on detection accuracy, even if it meant accepting longer execution times.

# 2. Implementation of Object Detection

This was section details the implementation of the object detection for the given assignment parameters, in the report I refer to this as the ‘Standard Test.’ Below is a brief overview of the variables and methods used in the program with testing results near the end for justification on my 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:**

* After testing with the different matching methods with the SIFT detector I found the results to be similar in terms of feature matching but due to the selection of SIFT and that it gives a large number of float type descriptors, FLANN was selected based on test results.

## 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.

**Test Results:**

A table with numbers and symbols

AI-generated content may be incorrect.I began by running both tests with a feature cap of 5000, since both detectors maxed out that number I considered it a good bench to compare their accuracy. For consistency, both detectors are using FLANN for feature matching, using the LSH Index parameter for the ORB detector. I also adjust the Lowes ratio between 0.6 and 0.8 for comparison and logged the results in the table above:

Figure 1. Test Results

So, while on average, ORB gave a better inlier to match ratio, SIFT delivered a substantial increase in the total number of inliers.

Another important thing to note was after observing the results of the feature matching, in the program I’d set it to output the results of the features matching to visually interpret what was happening “under the hood”. And what became evident was the fact that while both detectors in my use case produced a boundary box with some degree of accuracy, the ORB detector managed to detect a lot of non-ideal KeyPoints in the scene to match with object KeyPoints, for instance we can see the KeyPoints with the SIFT detector were accurate on both accounts, with it matching features between the object and scene with little background interferrence;  
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

Whereas with the ORB detector, we can see that its key points were a lot more focused on gradient/colour differences and corner selection, which meant that it’s feature matching focused a lot more on the background rather than 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

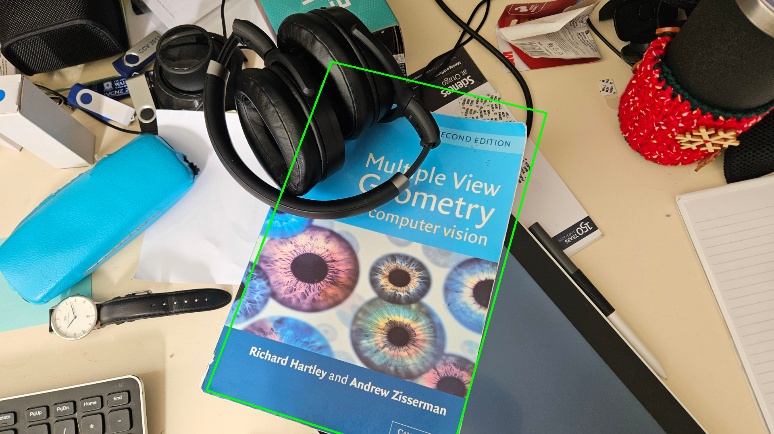
AI-generated content may be incorrect.

Figure 5 Figure 3 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.

Figure 6 SIFT with a Lowe Ratio of 0.8

* In further testing, I tried using SIFT with a feature cap of 1000 which gave unsatisfactory results (*see figure 1.*), loosening the Lowes ratio from 0.8 to 0.6 did give marginally better results, with it encapsulating more of the object but was still 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 implemented into the test to compare the results they provided, using k-nearest neighbours matching where k=2. The Lowes ratio test was used to filter matches, using a ratio of 0.8, ensuring that only the best matches contributed to the 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 the same match counts, Lowes ratio test filtered out better matches and features from the output of the Brute Force Matcher, though the execution times were 6.7x slower compared to the FLANN matcher  
Testing on varied images (different objects and scenes) in later accuracy tests revealed that FLANN offered more accurate match filtering with SIFT compared to being used with BFM. Researching into this I discovered that FLANN is better suited for SIFT because it efficiently handles high-dimensional float descriptors using approximate nearest neighbour search, making it significantly faster and more scalable than Brute-Force Matching (BFM). While BFM is accurate, it became computationally expensive with large numbers of features like those produced by SIFT, especially with larger image sizes. FLANN struck a good balance between speed and accuracy, making it the preferred choice for large-scale matching with the various test cases I used.

## 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 the arguments provided, instead using a terminal menu for running different methods.

### Execution:

The provided skeleton code was altered significantly to suit a more robust testing environment for myself, implementing a simple terminal menu for changing and assigning variables. Adding options for running different tests and changing settings.

* Though to run the code is still straightforward; to run the object detection you simply select the first option in the menu (typing 1 into the terminal and hitting enter). This will give you the output given the chosen parameters I deemed best for this project; this uses the provided object and scene images for object detection.
* To find the usage of the program there is a provided usage option in the menu (option 2), this will print the current settings for the object detection. Which by default are the values I used for the assignment (standard test).
* In addition, there is included a “settings” menu option that allows changing the object and scene file (by providing the filename of the image in the working directory), as well as the detector and matcher type using my default settings.

# 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.

### A computer and a mouse on a table AI-generated content may be incorrect.Accuracy Test

* A box of monopoly game

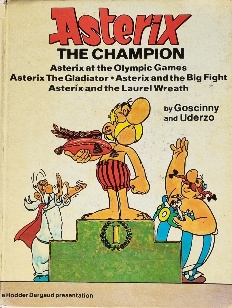
  AI-generated content may be incorrect.A photo of a card playing game “Monopoly Deal” was used in conjunction with;

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

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

Figure 11 md-scene1

Figure 12 Monopoly Deal

* + A scene image of books laid in a grid formation. (*see figure 14*)
  + A scene image of books scattered around, covering ~50% of the object. (*see figure 15*)

## 3.2 Hypotheses and Experimental Questions

Figure 14 Book scene cluttered.

Figure 13 Asterix book

**Speed Test:**

* While ORB was known to be faster in speed than SIFT, I wanted to observe the difference between the two detection methods for comparison, to determine if the speed trade-off was worth the degree of accuracy difference, by comparing their matches and feature detection.

Figure 15 Book scene grid

**Accuracy Test:**

* It was hypothesized that while SIFT takes longer than ORB, its accuracy in detecting the correct object in the scene is significantly better. The object was to determine the actual degree of accuracy by some determined metric. It was also examined that BFM would produce better matching results during lab projects, it was this that determined its standard use in the accuracy test.

**Non-Planer Object Detection:**

* It has been mentioned how the homography matrix calculation and usage in our object detection is much more suitable for flat objects like the fronts of books, this statement led to the question of how does the object detection methods we use fair against complex faces and rounded edges

## 3.3 Benchmarking Approaches

**Method for Testing Speed:**

* Each detector/matcher combination was executed twenty times.
* 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:**

* The true (ground truth) corners of the object were selected using OpenCV features, these were corners that the user can select, though I determined some default for use.
* The homography matrix was calculated to determine the boundary box corners, the offset of the corners compared to the ground truth (calculated using Euclidean distance between two vectors) were used to determine the accuracy of the resulting object detection in pixel error values.
* 5 tests with different object and scene image combinations were used for each detector to determine the average pixel error values per corner, and per test.

**Non-Planer Object Detection Method:**

* Using objects with complex faces in different environments to evaluate the limits of the ORB/SIFT detection methods in conjunction with the current homography matrix calculations to determine the level of accuracy in detection.

# 4. Test Results

## 4.1 Detection Execution Times

**Final Observations:**

* SIFT was slower than ORB in terms of raw execution time; however, SIFT’s output was significantly more accurate.
* For descriptor matching, FLANN and BFM performed similarly on the standard test image but diverged when evaluated on a broader dataset, with FLANN showing more consistent accuracy.

## 4.2 Detection Accuracy

**Final Observations:**

* The SIFT & FLANN combination provided robust detection of the card playing game images.
* The face detection experiment revealed weaknesses: the bounding box was overly large and deformed, suggesting that the current approach is more suited to detecting planar objects (e.g., book faces) rather than curved or complex shapes like faces.

## 4.3 Object Detection Using Non-Planer Objects

**Final Observations:**

* The face detection experiment revealed weaknesses: the bounding box was overly large and deformed, suggesting that the current approach is more suited to detecting planar objects (e.g., book faces) rather than curved or complex shapes like faces.
* The detection results for faces were poor, resulting in an oversized and deformed bounding box.

# 5. Discussion

## 5.1 Analysis of Results

* **Method Selection:**
  + Although SIFT was slower, its higher accuracy was the deciding factor in this application, given that speed was not the primary concern.
  + FLANN was favoured for its superior accuracy with SIFT, despite some performance trade-offs, as the primary goal was to correctly localize the object in the scene.
* **Performance vs. Accuracy Trade-offs:**
  + In applications where real-time performance is critical (e.g., VR or facial recognition), these methods might need further optimization. Using **ORB** with parameter adjustments would be However, for the purpose of this project, accuracy was prioritized.

## 5.2 Limitations and Weaknesses

* **Experimental Limitations:**
  + The approach worked well for objects resembling planar surfaces, such as a card or book face, but struggled with non-planar objects like faces.
  + The testing dataset could be expanded to include a wider variety of images to further validate the robustness of the method.

## 5.3 Potential Improvements

* **Algorithmic Enhancements:**
  + Investigate additional preprocessing steps (e.g., image smoothing) to improve detection in more challenging images. Similar to how in assessing the accuracy of the detection methods, image preprocessing was needed to detect the object for one use case.
  + Explore adjustments to detector or RANSAC parameters to potentially enhance match accuracy.
* **Future Work:**
  + Consider integrating alternative detection methods for non-planar objects.
  + Experiment with dynamic parameter tuning based on scene content.
  + Experiment with greyscale preprocessing when using SIFT and ORB.

# 6. Conclusion

This project successfully implemented an object-in-scene detection program using C++ and OpenCV. The SIFT & FLANN combination, despite longer processing times, provided the necessary accuracy for detecting the provide object in the scene. The experiments highlighted the trade-off between speed and accuracy, using different images for determining accuracy in different conditions and environments, and the limitations observed with non-planar objects point to areas for future improvement.   
Though not necessary for the given assignment, I enjoyed testing the different parameters of this project and thus I have formatted my program in a way that this work lays a solid foundation for further personal exploration into more robust and adaptable detection methods, and how these methods can be applied for different situations.