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:

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:

Parameters	Inliers	Good Matches	Ratio
SIFT Lowe @ 0.6	256	402	0.64
SIFT Lowe @ 0.7	293	499	0.59
SIFT Lowe @ 0.8	385	646	0.60
ORB Lowe @ 0.6	36	50	0.72
ORB Lowe @ 0.7	88	144	0.61
ORB Lowe @ 0.8	190	400	0.48

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;

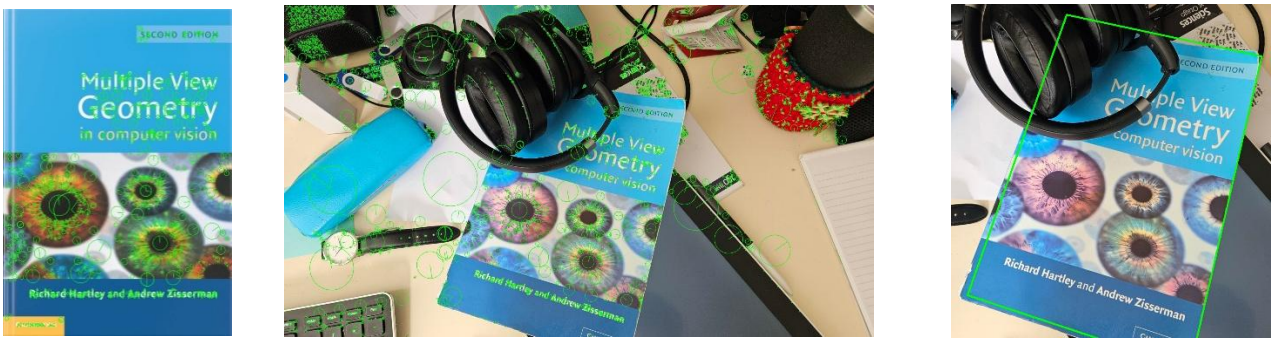


Figure 2 Object and Scene KeyPoints with Boundary Box drawn using the SIFT detector and FLANN matcher

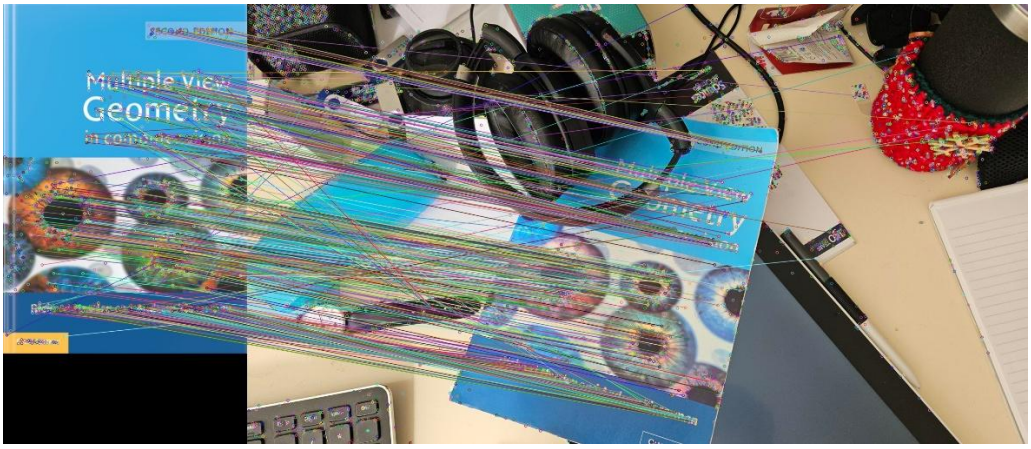


Figure 3 Feature Matching using OpenCV and SIFT KeyPoints

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.

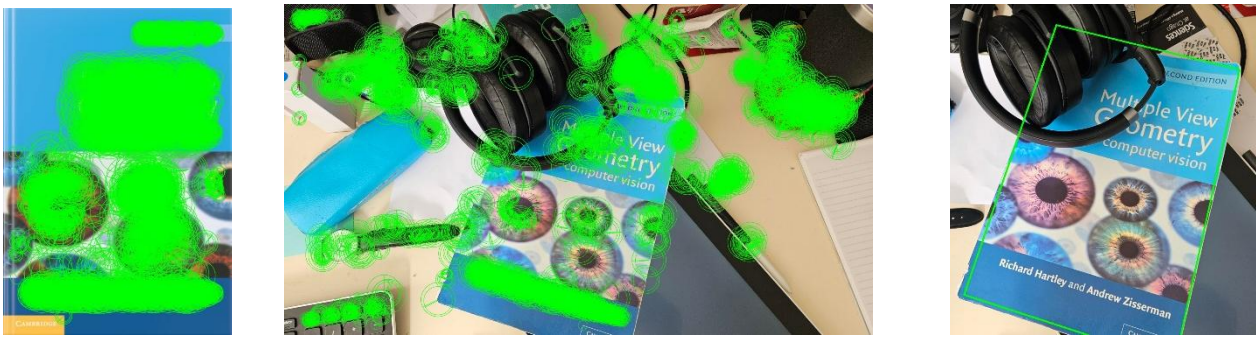
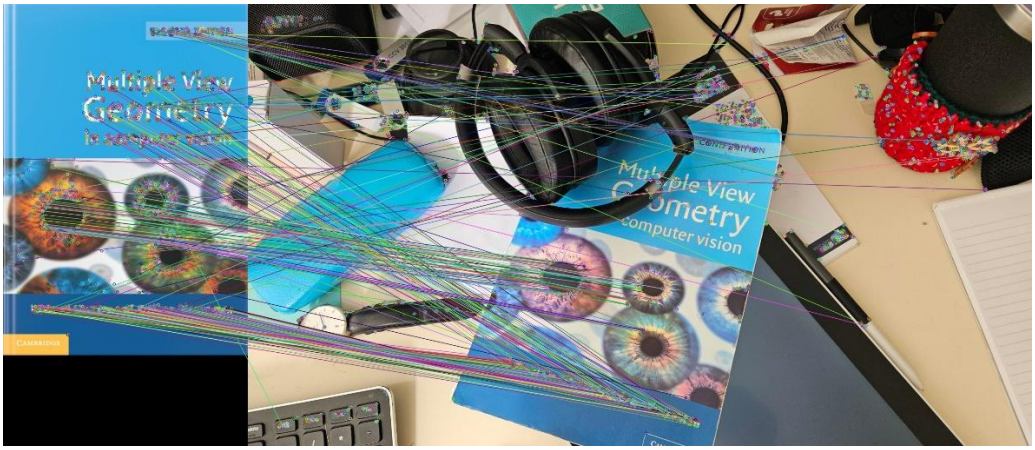


Figure 4 Object and Scene KeyPoints with Boundary Box drawn using the ORB detector and FLANN matcher



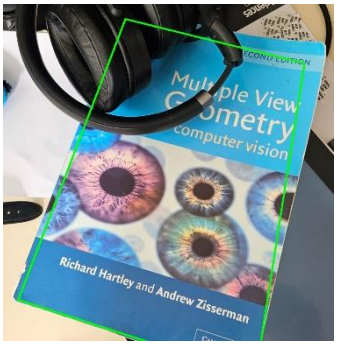


Figure 5 SIFT with a Lowe Ratio of 0.8

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.

- 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.)*.

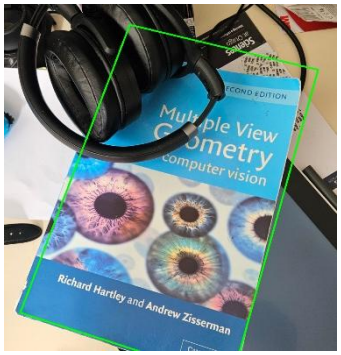


Figure 7 SIFT with a Lowe Ratio of 0.6

- 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.

Parameters	Inliers	Good Matches	Ratio
SIFT INF Lowe0.6	300	463	0.65
SIFT INF Lowe0.7	332	573	0.58
SIFT INF Lowe0.8	420	720	0.58

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.

Matcher	Matches	Good Matches	Good Features	Inliers	Time Taken (s)
FLANN	1954	720	1440	420	0.658
BFM	1954	730	1460	478	4.409

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.

- 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.
- 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 not work without disabling the "default" Boolean and changing the 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 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;
    - 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

Figu

Figure 14 Book scene cluttered.

F

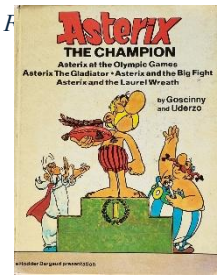


Figure 13 Asterix

3.2 Hypotheses and Experimental Questions

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.

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 size of the object in the scene, OpenCV features were used to present the user with a photo of the object (for reference) and the corresponding scene image, the user would then select the visible corners of the object in scene, this gave the “ground truth” corners (with a user error margin of 5pixels), allowing the boundary box detection corners to be compared using Euclidean distance between the vectors. To minimize the user errors in corner selection, default corners were defined and used as so the variables remained constant across the different tests. 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.

---

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.

Detector/Matcher	Total Features object+scene	Good Matches	Average Execution Time (s)
SIFT/FLANN	6955	646	5.15
SIFT/BFM	6955	629	10.52
ORB/FLANN	9998	404	1.56
ORB/BFM	9998	343	1.26

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 giving better inliers, its "good matches" count was lower compared to using FLANN, this would be due to the fact that BFM is exact in that it finds the true closest and second-closest matches, making the overall distance smaller. Due to this fewer "good" matches pass Lowes ratio test, whereas FLANN is approximate and often inflates the distance gap allowing more matches to pass through, leading to the higher count observed.
- 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 $O(n^2)$ 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 $O(n^2)$ its bitwise operations are hardware accelerated.

4.2 Detection Accuracy

Test Results:

Firstly I ran the program defining the corners of each object in each scene myself to define some defaults to be used consistently across the averaging tests, the results of these tests are shown in the tests below with the detected corners. Note to fit these have been abbreviated, a table is provided which shows a summarised overview of the pixels errors;

Both @ 50k, 0.8 lowes ratio									
ORB					**SIFT**				
Test	**Good Matches**	**Good Features**	**Inliers**	**Pixel Error Average**	**Test**	**Good Matches**	**Good Features**	**Inliers**	**Pixel Error Average**
T1	1106	2212	783	11.91	**T1**	570	1140	412	11.27
T2	953	1906	474	10.55	**T2**	804	1608	529	10.7
T3	6086	12172	4219	12.8	**T3**	1863	3726	1399	13.98
T4	4995	9990	3666	5.58	**T4**	1802	3604	1292	5.76
T5	2436	4872	1570	12.65	**T5**	821	1642	479	13.55
Average	3115.2	6230.4	2142.4	10.698	**Average**	1172	2344	822.2	11.052

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	Test 2: md + md-scene2	Test 3: book + b-scene1	Test 4: book + b-scene2	Test 5: book + b-sceneD
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	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	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	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	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	Test 2: md + md-scene2	Test 3: book + b-scene1	Test 4: book + b-scene2	Test 5: book + b-sceneD
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	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	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	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	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:

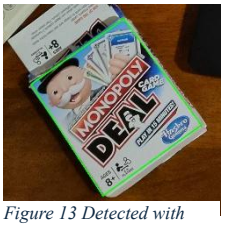


Figure 13 Detected with workaround

Figure 14 Failed Detection Scene1

Figure 15 Failed Detection Scene 2

- In running the initial tests, I struggled to get the ORB detector to find the object in Test 1 and 2 (*see figure 13 & 14)*, this was most likely due to the fact that there was a large scale difference between the objects file size and the relative size of the object in the scene (object file size of 2171x 2648 compared to 446x505 in scene), since ORB isn't as robust in scale variance as SIFT. Though raising the feature cap of the detector to 50,000 did help, a workaround involving scaling the object by a scale of 0.4x before running it through the detector was the minimum scale that allowed me to detect the object in the scene with any degree of accuracy (*see figure 12)*.
- With such a large features cap, SIFT created a larger set of possibly less robust features, filtering these out with a more lenient Lowes Ratio of 0.6, and the same 50K feature cap, seemed to result in a larger pixel error for ORB, yet a slightly lower result for SIFT (*see figure 15*). But ultimately lowering this cap to 5000 only resulted in a <1% average decrease in pixel

errors for SIFT, yet a >23% increase in average pixel errors for ORB (*see figure 16)*.
In an attempt to refine this further I lowered the feature cap for ORB to twice its default (1,000) and SIFT even lower to 500 (following the trend of how fewer features result in higher accuracies), resulted in an increase of >25% average pixel errors in SIFT, and a larger >84% increase of average pixel errors for ORB (in comparison from the initial test in *figure 12*) *(see figure 18)*.

Additional Test Results:

Both @ 50K, lowes at 0.6									
	ORB					SIFT			
Test	Good Matches	Good Features	Inliers	Pixel Error Average	Test	Good Matches	Good Features	Inliers	Pixel Error Average
T1	342	684	265	12.47	T1	302	604	281	11.33
T2	116	232	94	10.38	T2	462	924	359	11.44
T3	2305	4610	2070	14.29	T3	1126	2252	949	13.38
T4	2176	4352	1983	5.64	T4	1149	2298	1034	6.03
T5	675	1350	525	14.34	T5	420	840	356	12.88
Average	1318	2245.6	987.4	11.424	Average	691.8	1383.6	595.8	11.012

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

Changing the nfeatureCap to 5000, at 0.8 lowes ratio									
	ORB					SIFT			
Test	Good Matches	Good Features	Inliers	Pixel Error Average	Test	Good Matches	Good Features	Inliers	Pixel Error Average
T1	267	534	140	14.32	T1	397	794	267	11.48
T2	238	476	85	12.57	T2	649	1298	430	10.86
T3	897	1794	623	16.98	T3	888	1776	620	13.5
T4	1283	2566	986	6.24	T4	1270	2540	866	5.99
T5	740	1480	433	16.15	T5	799	1598	496	12.95
Average	789.5	1370	453.4	13.252	Average	800.6	1601.2	535.8	10.956

Figure 17 Feature cap of 5000, Lowes Ratio of 0.8

ORB @ 1K, SIFT @500, lowes at 0.7									
	ORB					SIFT			
Test	Good Matches	Good Features	Inliers	Pixel Error Average	Test	Good Matches	Good Features	Inliers	Pixel Error Average
T1	20	40	18	16.02	T1	10	20	6	16.67
T2	23	46	7	685.908	T2	31	62	30	10.82
T3	11	22	7	33.62	T3	76	152	36	17.86
T4	173	346	141	5.205	T4	141	282	95	5.82
T5	17	34	9	24.29	T5	70	140	34	18.2
Average	56	97.6	36.4	19.78375	Average	65.6	131.2	40.2	13.874
				(excluded T2 for average)					

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

Final Observations:

- The need for preprocessing the image in test 1 with ORB image detection, demonstrates that ORB is not situated for cases of large scale differences between object size and in scene object size. Note that this was not needed for SIFT, due to its robust handling of scale variance. Lowering the feature cap for ORB to below 5000 resulted in the Test 2 ultimately failing, further reinforcing this fact.
- One thing that was noticed was that with both detection methods on Test 4 (book.jpg + b-scene.jpg) resulted in a lower pixel average error than the other combinations, in this scene the corner is not visible and I needed to approximate the corner selection with rulers on a screen. This demonstrates how the underlying homography calculation works, by approximately using KeyPoints in a similar method to how I estimated the “true” corner by overall geometry. Showing that the method works better at corner approximation rather than precisely defined visible corners.

---

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.
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.
Due to the large number of test data gathered for this project, not all was included in the report and therefore some sections were excluded in order to make the document more concise, not that it is, but smaller than it would've been. Originally there was included testing with non-planer objects though that added 2 extra pages. Additional testing notes are included in a public GitHub repository, where I uploaded the project for testing.

https://github.com/BrooklynT101/ObjectDetection