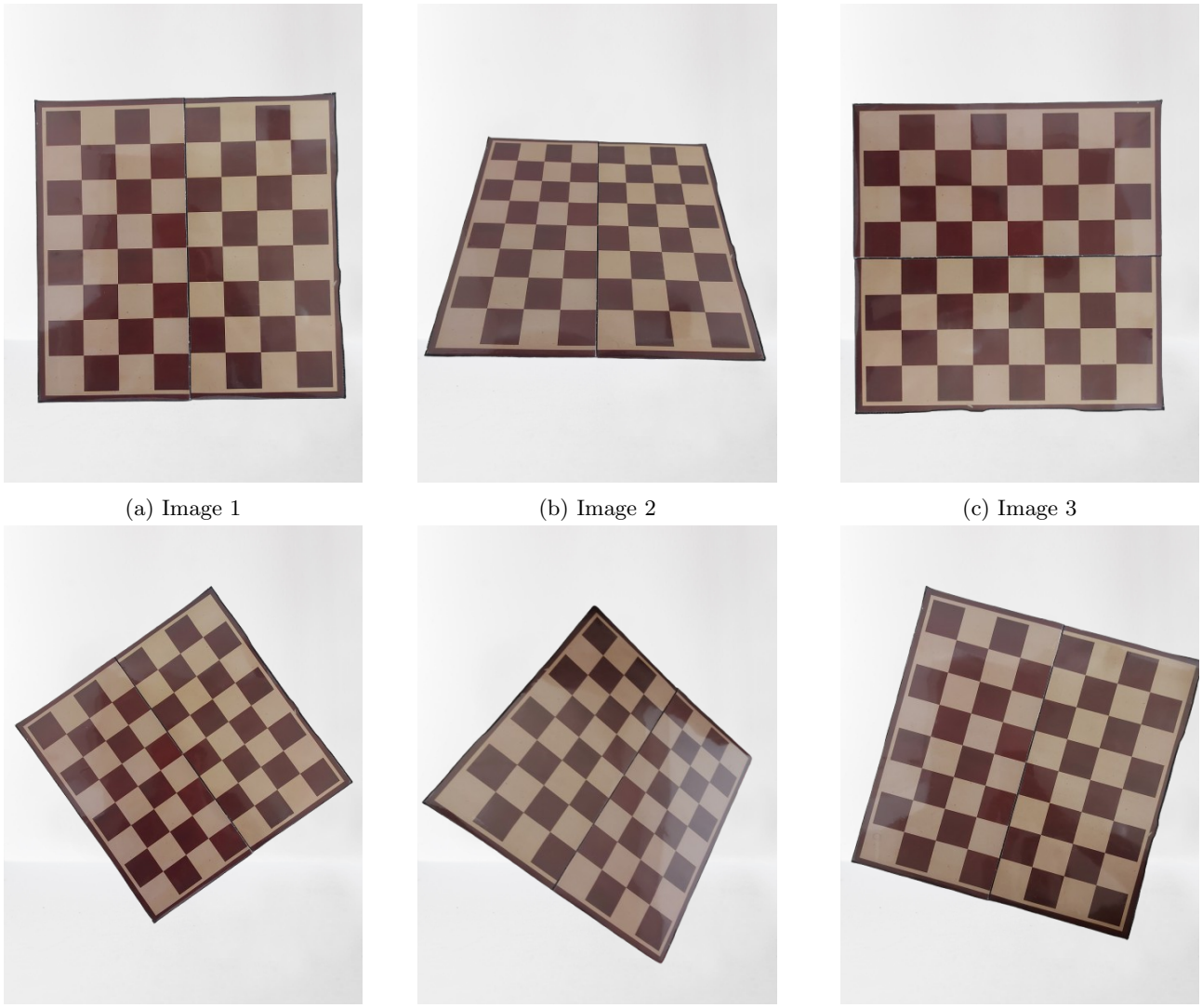
3D Vision Midterm Project Report: Camera Calibration and Homography Computation

Siddharth Kothari (IMT2021019)

October 31, 2024

Contents

1 Camera Calibration



(a) Image 1 (b) Image 2 (c) Image 3

(d) Image 4 (e) Image 5 (f) Image 6

Figure 1: Chessboard images chosen.

1.1 Introduction

For the camera calibration task, I selected 6 images of a chessboard taken from various angles, and I use OpenCV's inbuilt functions to first figure out the chesboard corners, and then use the chessboard corners thus found to calibrate the camera, upto the chessboard's scale.
I assign the chessboard corners to have the coordinates (0,0,0), (1,0,0), and so on, and relative to this world coordinate frame, I define the camera parameters. The z coordinate of the chessboard plane is taken to be 0, and the x and y coordinates of the points vary.
The camera projection matrix is written as -

$$P = K[R|t]$$

where P is the camera projection matrix, R is the rotation matrix of the camera relative to the world frame, and t is the translation vector of the camera frame centre relative to the world frame. R and t represent the extrinsic parameters of the camera.
The intrinsic camera matrix (or simply the camera matrix) K is of the form

$$K = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix}$$

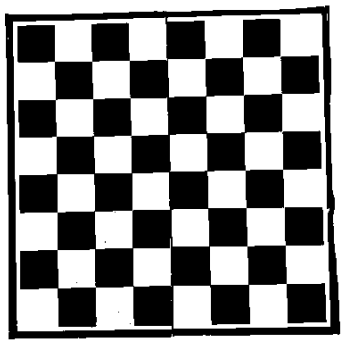
The assignment focuses on finding the camera intrinsic matrix for the camera, along with the rotation matrix and translation vectors. The camera calibration function of OpenCV can also be used to find the radial and tangential distortion coefficients.

1.2 Images

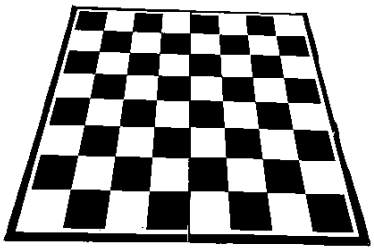
I use a total of 6 images. The images are shown in Figure 1. All the images represent the same chessboard, taken from the same mobile camera, at different orientations and distances.

1.3 Preprocessing Steps

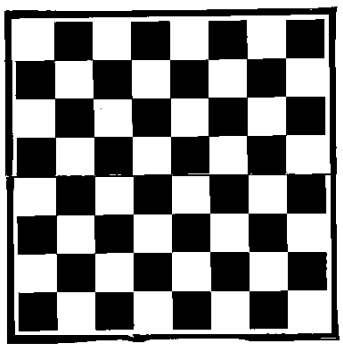
As is clear from the Figure 1, there are a few images in which the automated function may not be able to identify the chessboard of the relevant size, as there is glare on the chessboard due to lighting issues. Hence, to tackle this I perform a little preprocessing on the images.
Specifically, I first convert the images to grayscale, and then apply a threshold to boost the presence of the chessboard for the detector. The results of the same are shown in Figure 2.



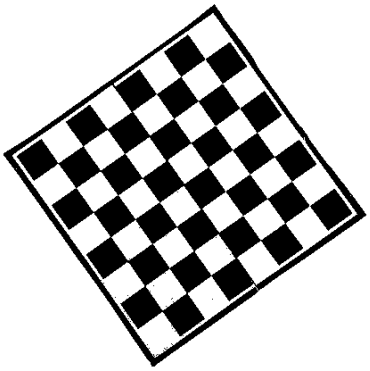
(a) Image 1



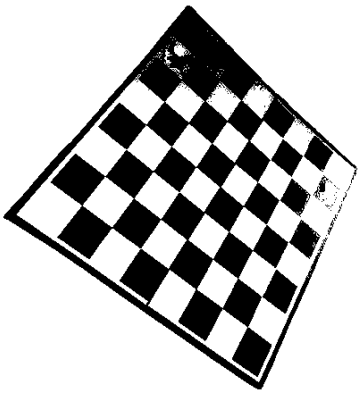
(b) Image 2



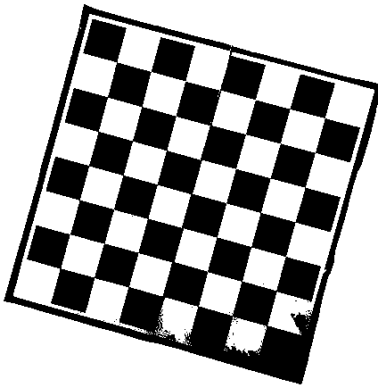
(c) Image 3



(d) Image 4



(e) Image 5



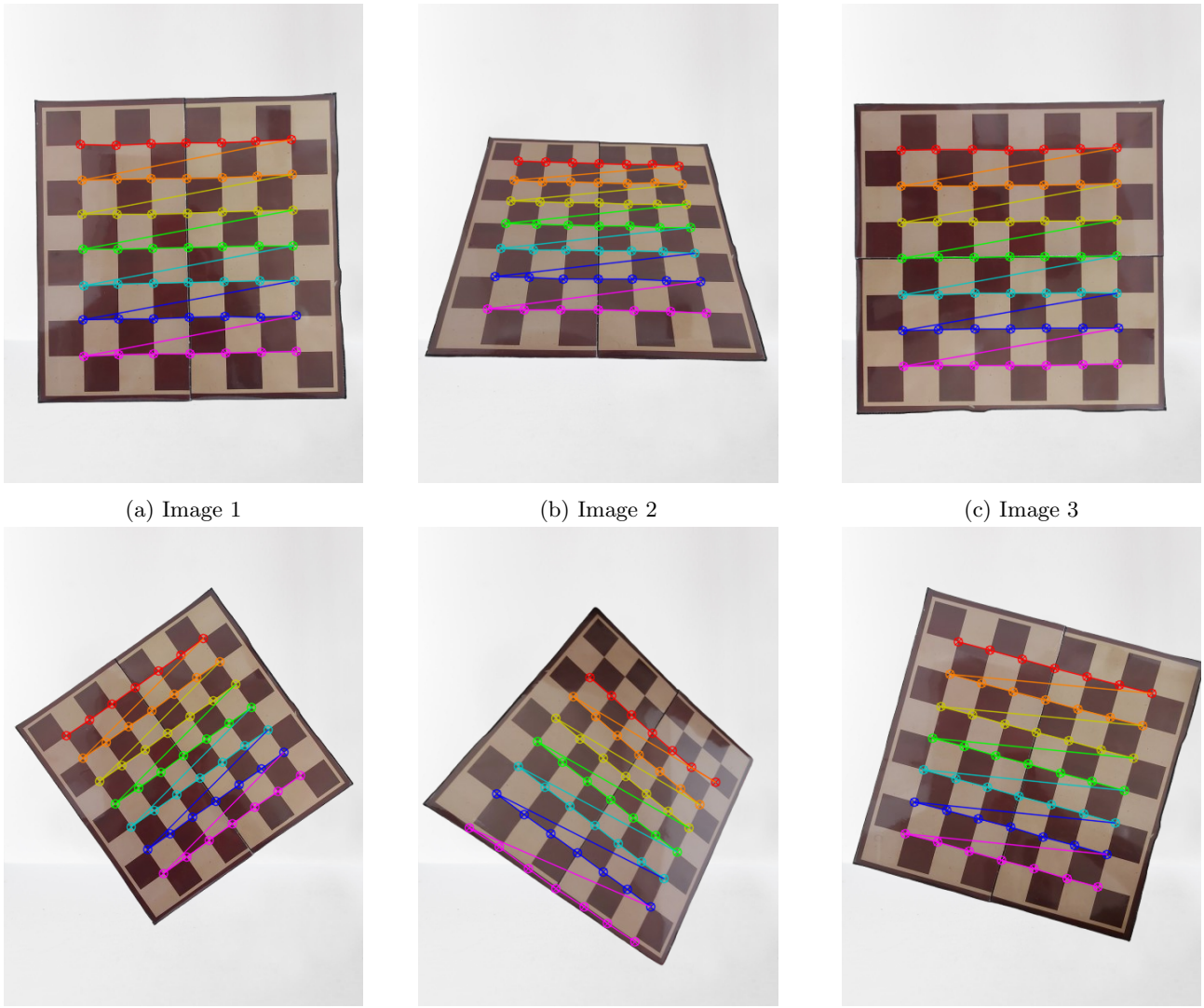
(f) Image 6

Figure 2: Chessboard images after grayscale and thresholding.

There is only one problematic figure which emerges, which is Figure 2e. Apart from this one, all the other images have an easy and errorfree detection of chessboard corners.

1.4 Corner Detection

I use OpenCV's inbuilt function findChessboardCornersSB for detecting corners, and then I refine the chosen corners using the cornerSubPix method. The results for this section are shown in Figure 3.



(a) Image 1 (b) Image 2 (c) Image 3

(d) Image 4 (e) Image 5 (f) Image 6

Figure 3: Corners detected in the Chessboard images.

As stated previously, the corners in chessboard 5 are not as accurate as expected. This shows how the lighting conditions can impact detection using automated methods. In the remaining 5 images, I see the same set of points being mapped. I can now find the camera parameters by forming correspondences between the 3d coordinates that I have assigned to these points, based on our world frame, and their pixel locations in the images.

1.5 Output Values

I use OpenCV's inbuilt function cameraCalibrate to get the intrinsic and extrinsic parameters of the camera for the 6 images. The function returns the reprojection error (after applying the calculated matrix back on the 3D points, the camera intrinsic matrix, the distortion coefficients and the rotation and translation vectors.
I provide 2 sets of results along with a comparison of the reprojection errors in both cases. The sets correspond to when the faulty image 5 is included or not during the computations.

1.5.1 Reprojection error

The first comparison is based on Reprojection error. In images 1,2,3,4 and 6 the points are mapped accurately, while in image 5, a different set of points is mapped. When the parameters are calculated including the image 5, I get a reprojection error of **0.915**, while the reprojection error when the parameters are calculated excluding image 5 is **0.456**.
The lower reprojection error is indicative of the increased accuracy when the faulty image is removed.

1.5.2 Camera Intrinsic Matrix

The camera intrinsic matrix for the case when image 5 is included is -

$$K = \begin{pmatrix} 427.836 & 0 & 213.711 \\ 0 & 426.325 & 282.468 \\ 0 & 0 & 1 \end{pmatrix}$$

While, for the case when the image 5 is excluded, the camera intrinsic matrix is

$$K = \begin{pmatrix} 418.969 & 0 & 215.825 \\ 0 & 418.375 & 291.102 \\ 0 & 0 & 1 \end{pmatrix}$$

Note: From here onwards, I only report the values for the case when image 5 has been excluded, due to the increased accuracy.

1.5.3 Rotation Matrices

The camera calibrate function returns the rotation vectors, not the rotation matrices. The rotation vectors can be converted to rotation matrices using OpenCV's inbuilt Rodrigues function. The rotation matrices for the images 1,2,3,4 and 6 are shown below.

$$R \text{ for image } 1 = \begin{pmatrix} 0.999 & 0.020 & 0.015 \\ -0.021 & 0.999 & 0.029 \\ -0.015 & -0.029 & 0.999 \end{pmatrix}$$

$$R \text{ for image } 2 = \begin{pmatrix} 0.999 & -0.008 & -0.025 \\ 0.021 & 0.813 & 0.581 \\ 0.016 & -0.581 & 0.813 \end{pmatrix}$$

$$R \text{ for image } 3 = \begin{pmatrix} 0.999 & 0.010 & 0.009 \\ -0.010 & 0.999 & 0.007 \\ -0.009 & -0.007 & 0.999 \end{pmatrix}$$

$$R \text{ for image } 4 = \begin{pmatrix} 0.815 & 0.579 & 0.007 \\ -0.579 & 0.815 & 0.018 \\ 0.004 & -0.019 & 0.999 \end{pmatrix}$$

$$R \text{ for image } 6 = \begin{pmatrix} 0.964 & -0.264 & 0.021 \\ 0.264 & 0.964 & 0.006 \\ -0.022 & -0.0004 & 0.999 \end{pmatrix}$$

Ifind that the rotation matrices indeed do correspond roughly to the amount of rotation required to ensure that the rotated image has perfectly horizontal and vertical chessboard corners.

1.5.4 Translation Vectors

Finally, the translation vectors obtained in order for the images 1,2,3,4,6 are as follows -

$$T \text{ for image } 1 = \begin{pmatrix} -2.928 & -2.885 & 10.068 \end{pmatrix}^T$$

$$T \text{ for image } 2 = \begin{pmatrix} -2.958 & -3.159 & 13.056 \end{pmatrix}^T$$

$$T \text{ for image } 3 = \begin{pmatrix} -3.354 & -2.651 & 9.782 \end{pmatrix}^T$$

$$T \text{ for image } 4 = \begin{pmatrix} -4.171 & -1.186 & 12.544 \end{pmatrix}^T$$

$$T \text{ for image } 6 = \begin{pmatrix} -1.895 & -3.831 & 10.568 \end{pmatrix}^T$$

1.6 Distortion Coefficients

1.6.1 Definitions

Some pinhole cameras introduce significant distortion to images. Two major kinds of distortion are radial distortion and tangential distortion.
Radial distortion can be represented as -

$$x_{distorted} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \; y_{distorted} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

On the other hand, tangential distortion can be represented as -

$$x_{distorted} = x + [2p_1 xy + (p_2(r^2 + 2x^2))] \; y_{distorted} = y + [2p_2 xy + (p_1(r^2 + 2y^2))]$$

The algorithm will compute the 5 distortion coefficients $(k_1, k_2, p_1, p_2, k_3)$, and will return them in this order.

1.6.2 Results

The distortion coefficients calculated are as follows -

$$(k_1, k_2, p_1, p_2, k_3) = (0.00268, 0.722, -0.00285, -0.00198, -2.8394)$$

2 Homography Computation using Chessboard

For the homography computation, I work with 2 kinds of settings - one uses 2 images of the chessboard, which is a planar object, and explores 3 methods for finding keypoints and obtaining matches between the points, on the basis of which the homography is computed.

2.1 Images

(a) Image 1

(b) Image 2

Figure 4: Corners detected in the Chessboard images.

For the chessboard, the images used are shown in Figure 4. I will try to warp image 4a to 4b using the homography matrix, to observe the effectiveness.

2.2 Method 1

The first method Iuse to find keypoints, and create matches between the keypoints involves the usage of the ORB detector for detecting keypoints and features, following which it uses Brute Force KNN Based Matcher to match the keypoints.

2.2.1 Keypoint Detection using ORB Detector

ORB is a fusion of the FAST keypoint detector, along with the BRIEF descriptor. FAST is used to find the keypoints, following which the Harris Corner Measure is applied to get the top N points. ORB also provides us with rotational invariance in the keypoints. For descriptors, ORB uses the BRIEF descriptor, with modifications to allow for rotational invariance. ORB is a much faster descriptor calculator as compared to SIFT and SURF, and also outperforms SURF.
The computed keypoints using the ORB descriptor are provided in Figure 5.

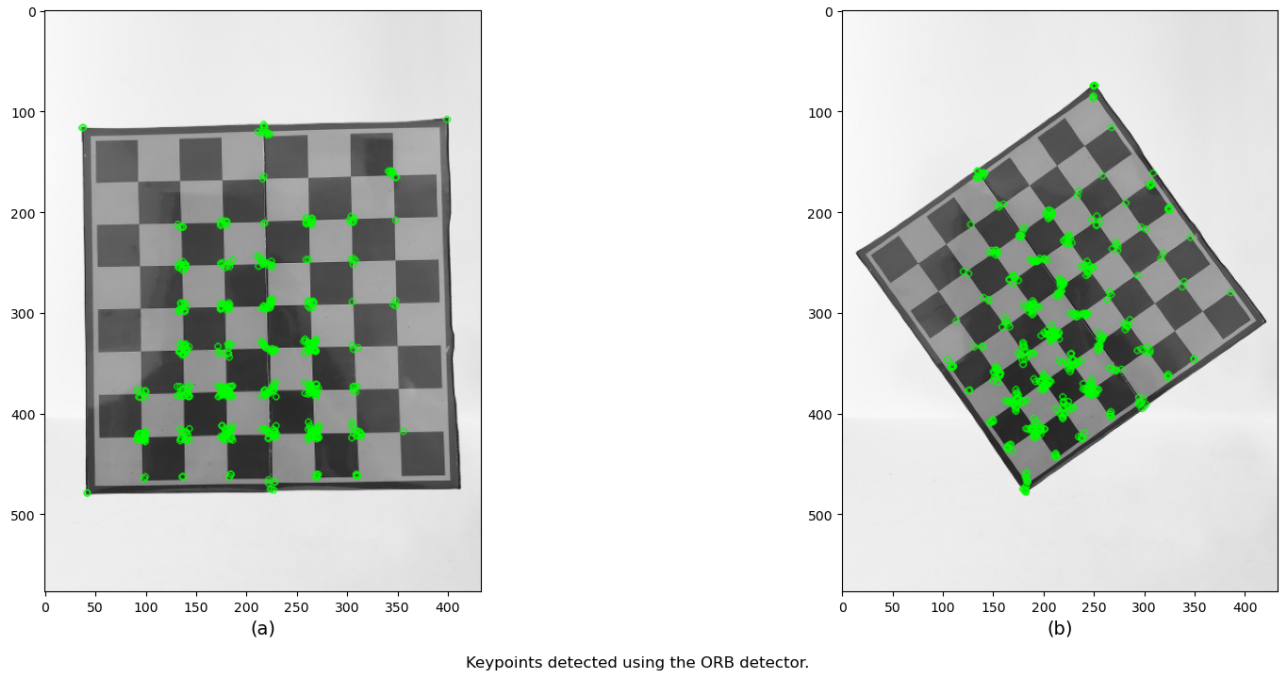


Figure 5: Keypoints computed using the ORB detector

2.2.2 Keypoint Matching using KNN Matcher

In this method, I utilize the K-Nearest Neighbors (KNN) approach to identify and filter matching keypoints between two sets of image features. For this approach, I use a Brute-Force(BF) Matcher, with the L2 norm. I then find the 2 nearest matches for each point. I then apply Lowe's ratio test to retain only reliable matches, and removing ambiguous matches. RANSAC algorithm is used for handling outliers.
The matches produced are shown in Figure 6. We can see that quite a few matches are wrong, and hence, the homography matrix has high chances to be incorrect. I explore the results of warping in a separate subsection. It is also noted that there is a high chance of the matches being incorrect for a chessboard by automated methods, due to the highly similar nature of feature descriptors obtained for multiple points, due to the nature of the chessboard.

2.3 Method 2

For this method, I use the ORB detector again, but this time I use the FLANN matcher rather than the BF Matcher.

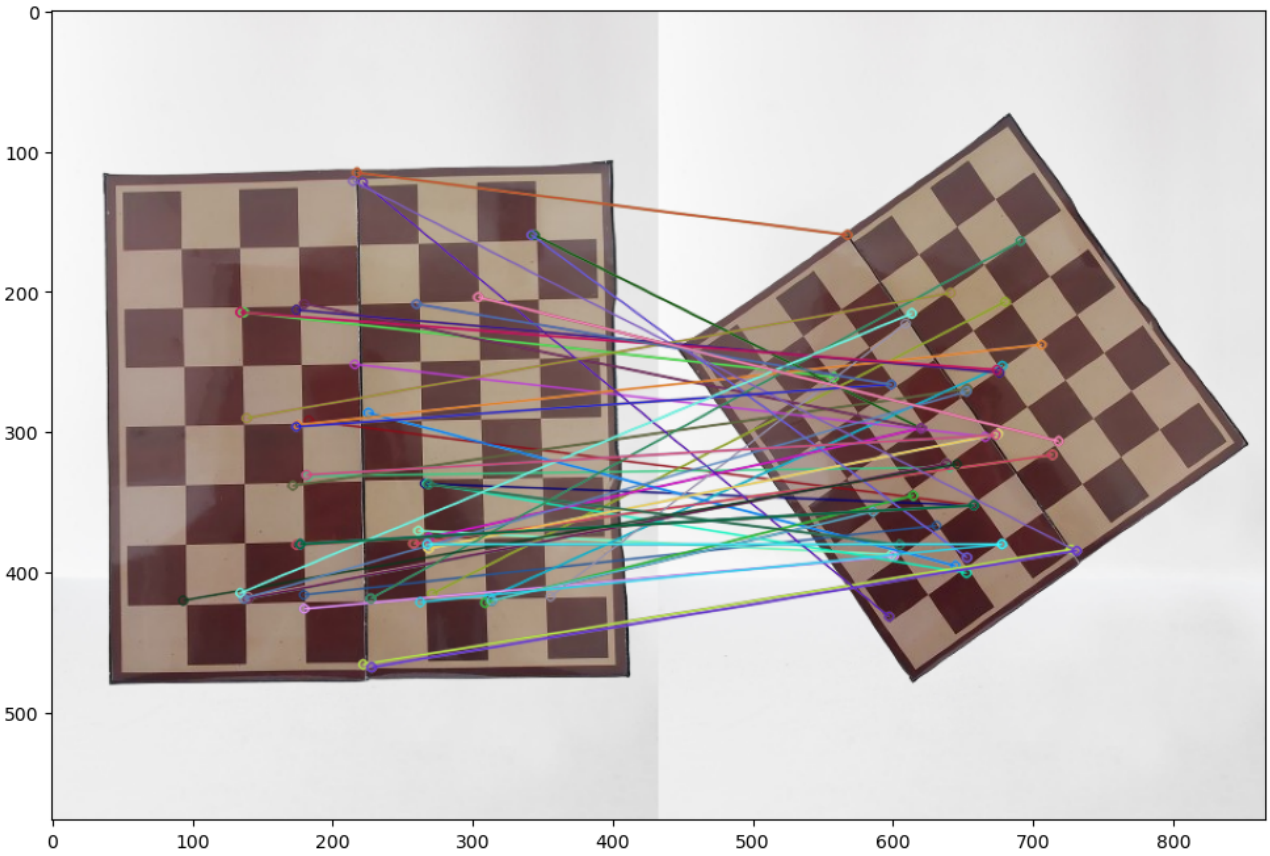


Figure 6: Matches detected using the BF Matcher

2.3.1 Keypoint Detection using ORB Detector

Since I use the same detector, the keypoints found in this case as well are the same as shown in Figure 5. Hence I directly talk about the results using the FLANN Matcher.

2.3.2 Keypoint Matching using FLANN Matcher

The FLANN (Fast Library for Approximate Nearest Neighbors) is a collection of algorithms optimized for fast nearest neighbors search. It works faster than the BF Matcher.
The matching results are shown in Figure 7. As we can see, the number of wrong matches is reduced, which, as we shall see later, results in a more robust homography. We again use the RANSAC algorithm to handle outliers.

2.4 Method 3

2.4.1 Keypoint Detection using Chessboard Corner Detection

For this method, I use the findChessBoardSB function which I used for Camera Calibration to find the chessboard corners. These corners serve as the keypoints, and there exist natural correspondences between each of the detected corners.
These matches are directly passed to the function to compute the homography. This is equivalent to manually finding the keypoints and matching them, and results in the highest quality homography and warping, as we shall see later.
The detected corners which act as the keypoints are shown in Figure 8.

2.5 Homography Computation using RANSAC

The homography matrix obtained using the BF Matcher is shown below.

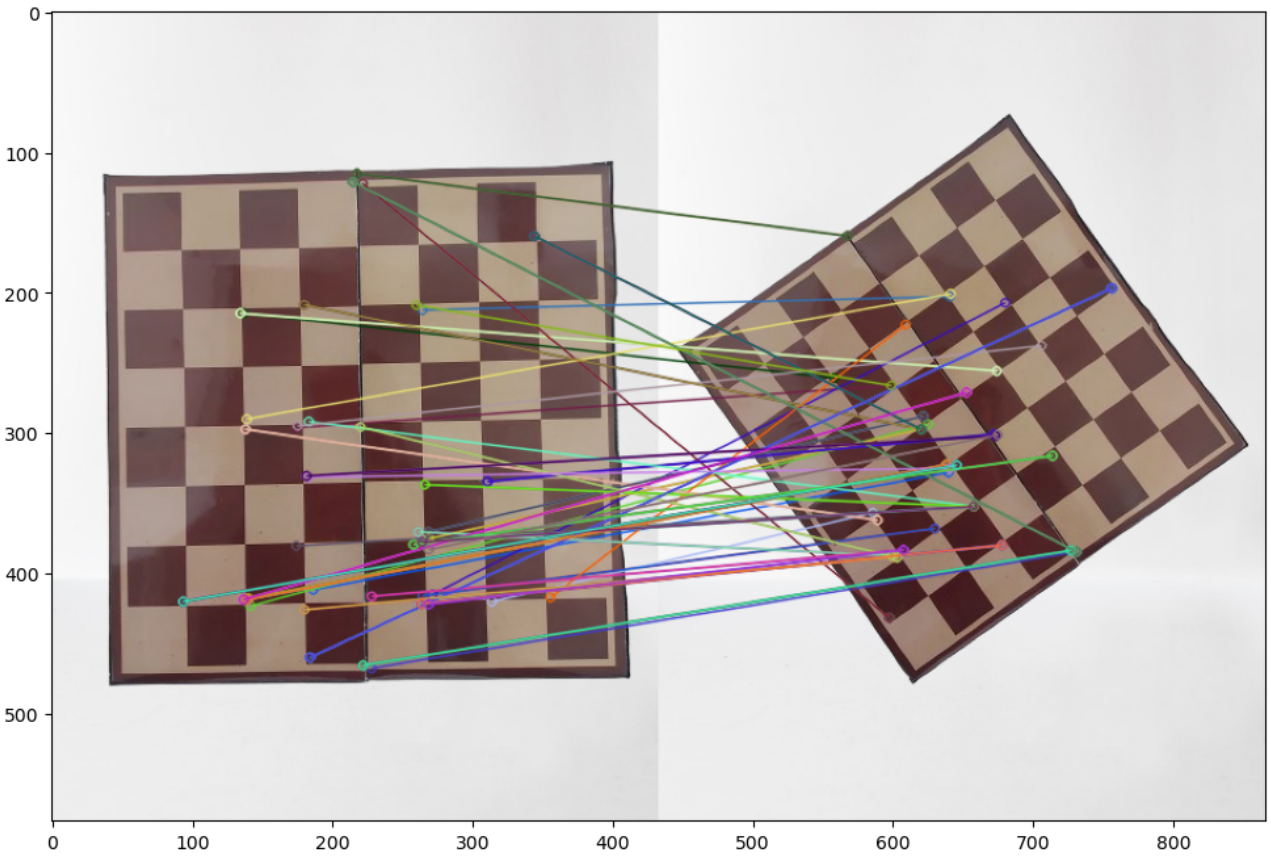


Figure 7: Matches detected using the FLANN Matcher

(a) Query Image

(b) Train Imge

Figure 8: Corners detected in the Chessboard images.

$$H \text{ using BF Matcher} = \begin{pmatrix} 0.285 & -0.619 & 224.237 \\ 0.450 & -0.972 & 351.610 \\ 0.0012 & -0.0027 & 1 \end{pmatrix}$$

The homography matrix obtained using the FLANN Matcher is shown below.

$$H \text{ using BF Matcher} = \begin{pmatrix} 0.662 & 0.468 & -65.079 \\ -0.461 & 0.678 & 181.296 \\ -0.000055 & 0.000075 & 1 \end{pmatrix}$$

The homography matrix obtained using the keypoints from Chessboard Detection is shown below.

$$H \text{ using BF Matcher} = \begin{pmatrix} 0.682 & 0.4685 & -66.672 \\ -0.445 & 0.686 & 179.2 \\ -0.000045 & 0.000049 & 1 \end{pmatrix}$$

As we can see due to the similarity between the homography matrices obtained in the last 2 cases, it is expected that their warping results will be similar.

2.6 Warping Results

The warping results for the three methods in order are shown in Figures 9, 10 and 11.
As stated previously, the warping results are proper for the FLANN matcher and the Chessboard Corner based matching, but this is not the case for the BF Matcher.

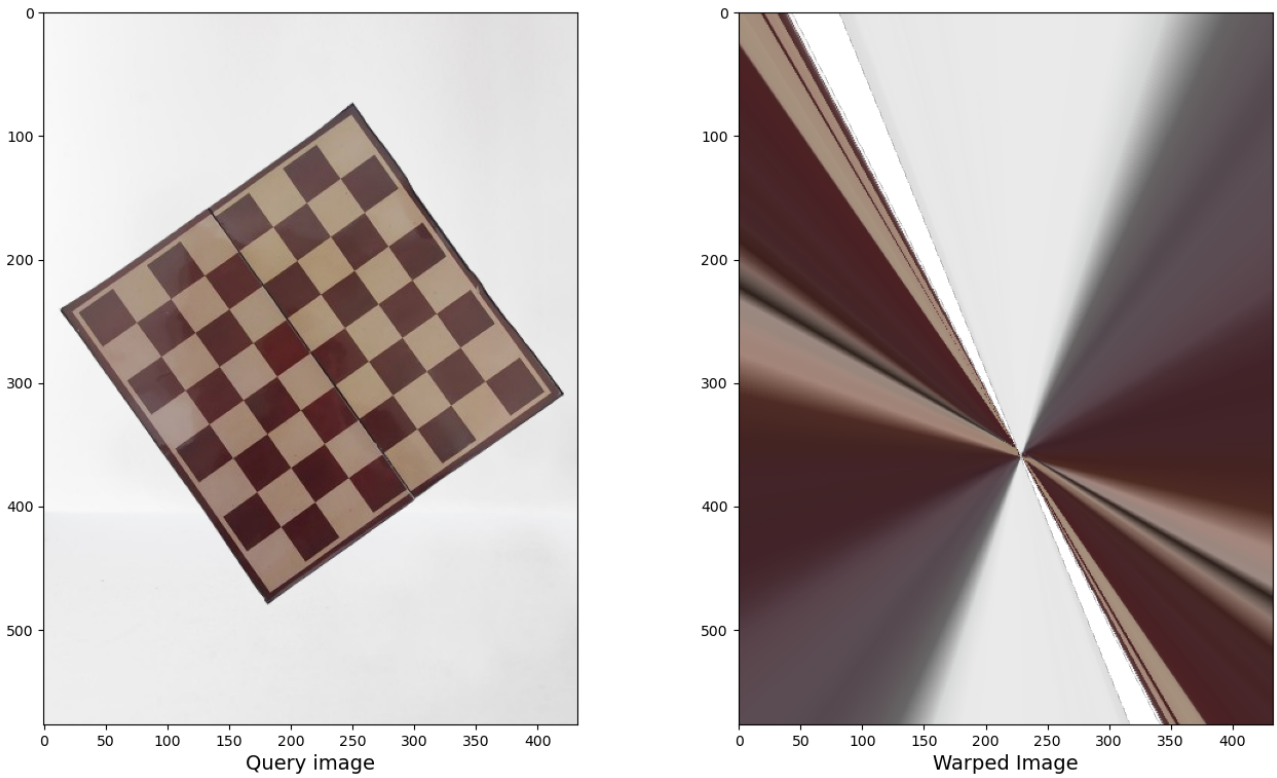


Figure 9: Warping for the Homography using the BF Matcher

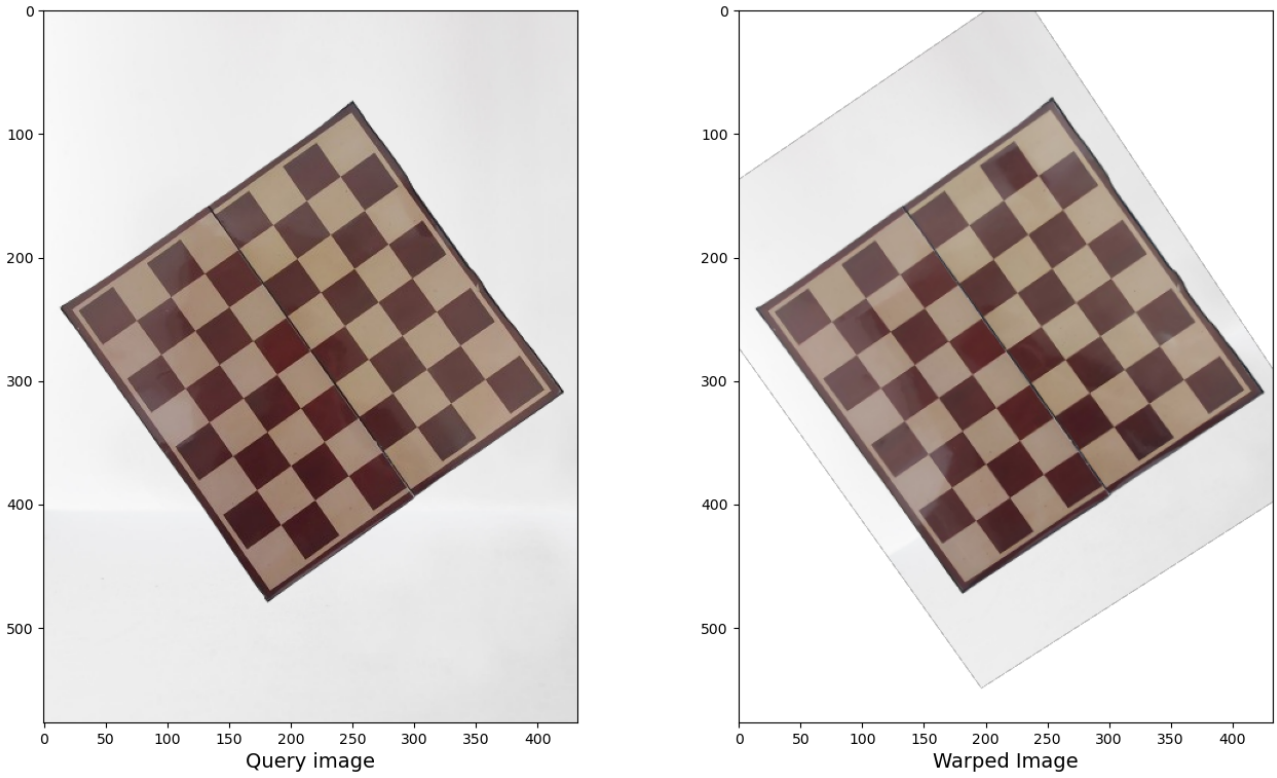


Figure 10: Warping for the Homography using the FLANN Matcher

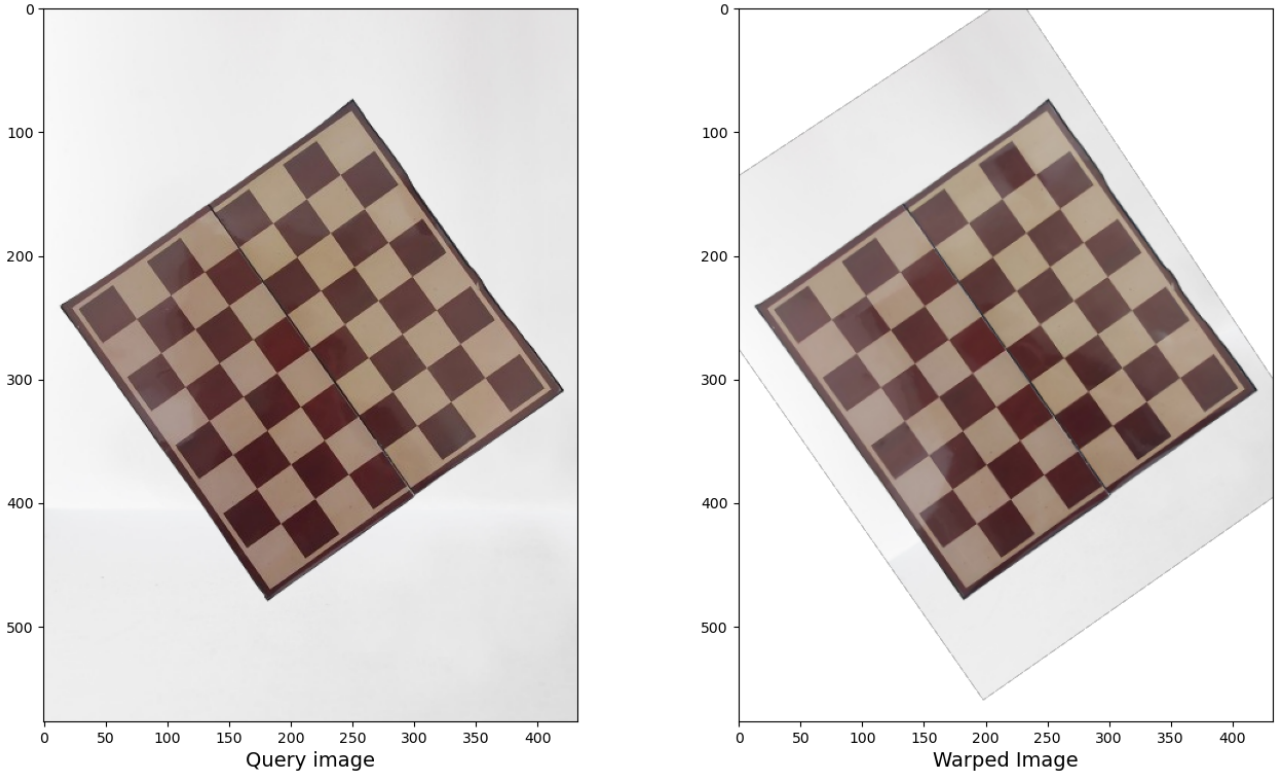


Figure 11: Warping for the Homography using the Chessboard Corners

3 Homography Computation using Rubik's Cube

I use similar methods for the case when a Rubik's Cube is used rather than a Chessboard. In the previous case, the Homography computed for the BF Matcher is poor due to poor matches. The chessboard itself is still a planar object. In this case, I intend to observe how the planarity of points chosen as keypoints may affect the quality of the Homography obtained, and benchmark it against the case when the keypoints and matches are identified manually.

3.1 Images

The Images used for this activity are provided below.

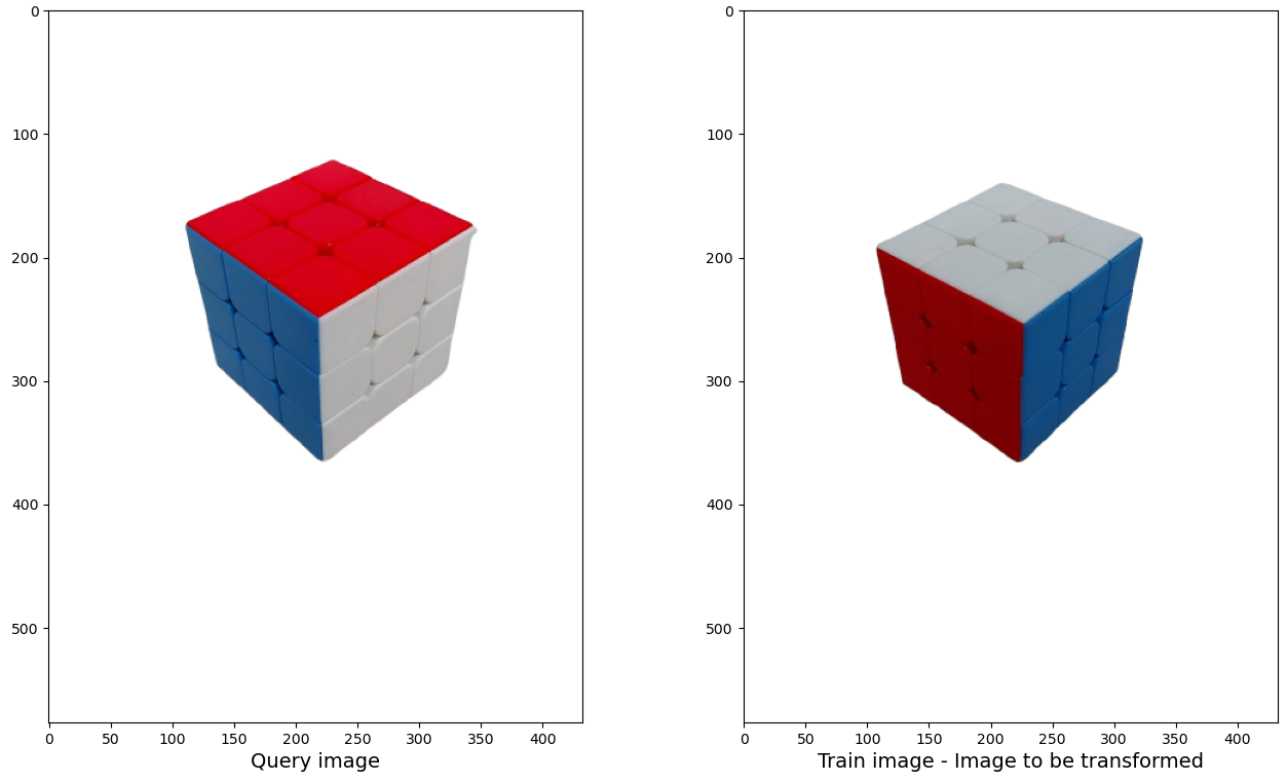


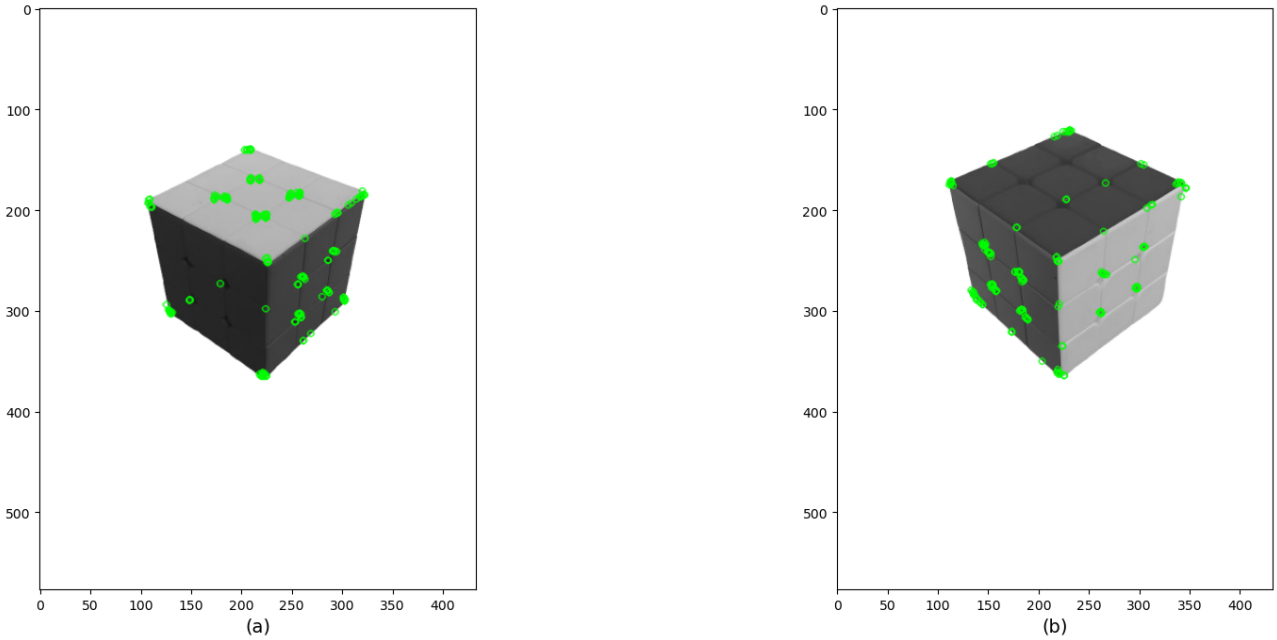
Figure 12: Images for the Rubik's Cube analysis

3.2 Using Keypoint Based Methods

Although I have tested using both the BF and the FLANN Matcher, the results are same. Hence for the sake of brevity, I only include the results using the FLANN Matcher. In this case as well, I use the ORB Detector. The results for Keypoint Detection along with the Matched detected are shown in Figures 13 and 14 respectively. As we can observe, the keypoints and matches detected are almost correct, but don't necessarily lead to the best homography, when compared with manual matching and detection.

3.3 Using Manual Identification

In this method, I manually find the keypoints, and create correspondences between them. I use GIMP to identify the coordinates of the 7 visible corners of the cube, and map them to each other in the two configurations. The points selected are shown in Figure 15.
As we shall see in a later section, the homographies and the eventual warping are similar in the 2 methods, but the manual matching still produces a better outcome, and leads to a warping which is closer to the expected outputs.



Keypoints detected using the ORB detector.

Figure 13: Keypoints Detected in the Cubes using ORB detector

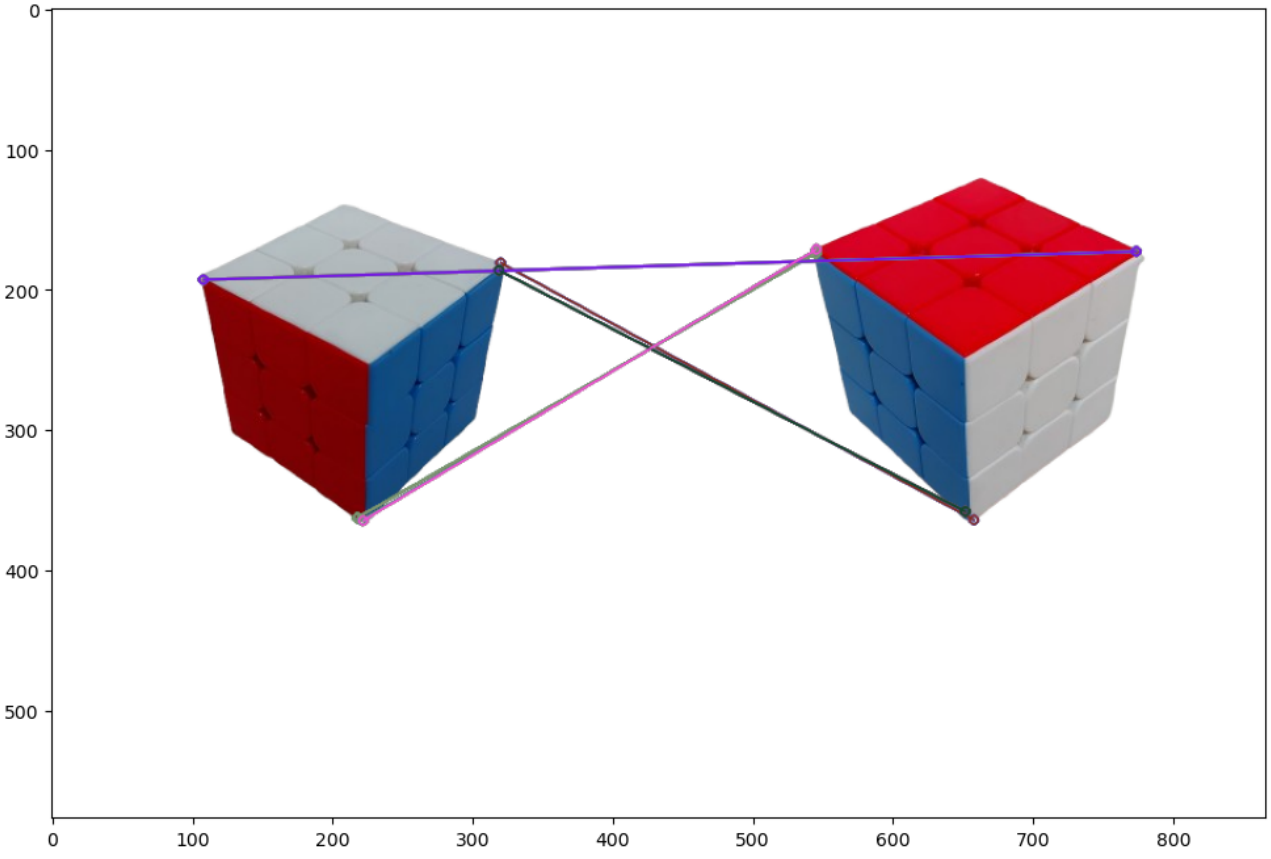


Figure 14: Matches Detected in the Cubes using the FLANN Matcher

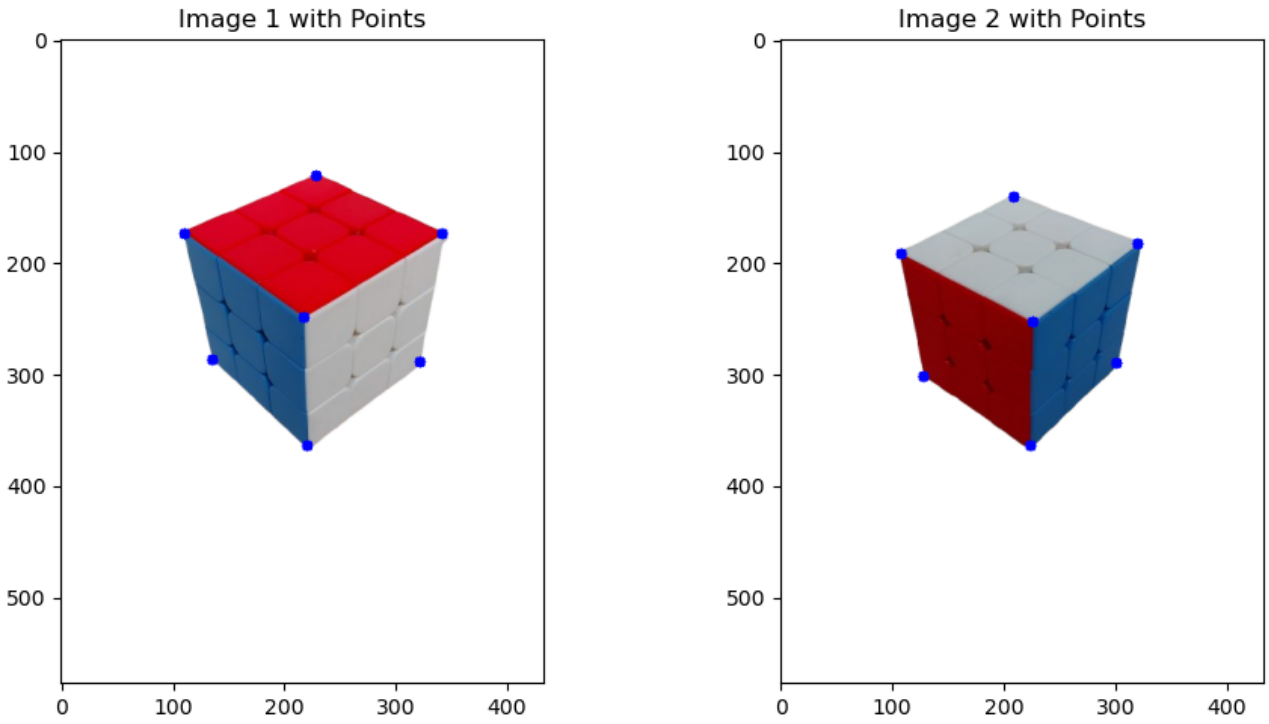


Figure 15: Manually Chosen Keypoints

3.4 Homographies Computed

The Homography matrix computed for the matching with FLANN is shown below.

$$H \text{ using FLANN Matcher} = \begin{pmatrix} -0.997 & -0.902 & 724.027 \\ 0.775 & -0.2746 & 193.735 \\ -0.0009 & 0.00207 & 1 \end{pmatrix}$$

The Homography Matrix for the matching using manually detected keypoints is shown below.

$$H \text{ using Manual Detection} = \begin{pmatrix} -0.545 & -0.932 & 563.187 \\ 0.884 & -0.630 & 190.381 \\ 0.00014 & -0.0003 & 1 \end{pmatrix}$$

3.5 Warping results

Finally, the Warping results for detection using ORB and FLANN and Manual Matching and Detection are shown in Figures 16 and 17 respectively.
It is clear from the warping results that the manual detection and matching yields a homography matrix which is very accurate, and results in a perfect homography.
Overall, it can be concluded that due to the object being non-coplanar in this case, manual matching is able to lead to a perfect homography, as compared to automated identification and matching.

4 Technical Discussion

In this section, I broadly discuss my learnings and observations from this assignment, in each of the three sections. I also try to highlight my experiments and provide an explanation of my methods and inferences.

4.1 Observations and Results of Camera Calibration

1. In Section 1, when I used the images of the chessboard directly without proper preprocessing, the findChessboardCornersSB method is unable to detect the chessboard in the image. The primary reason for

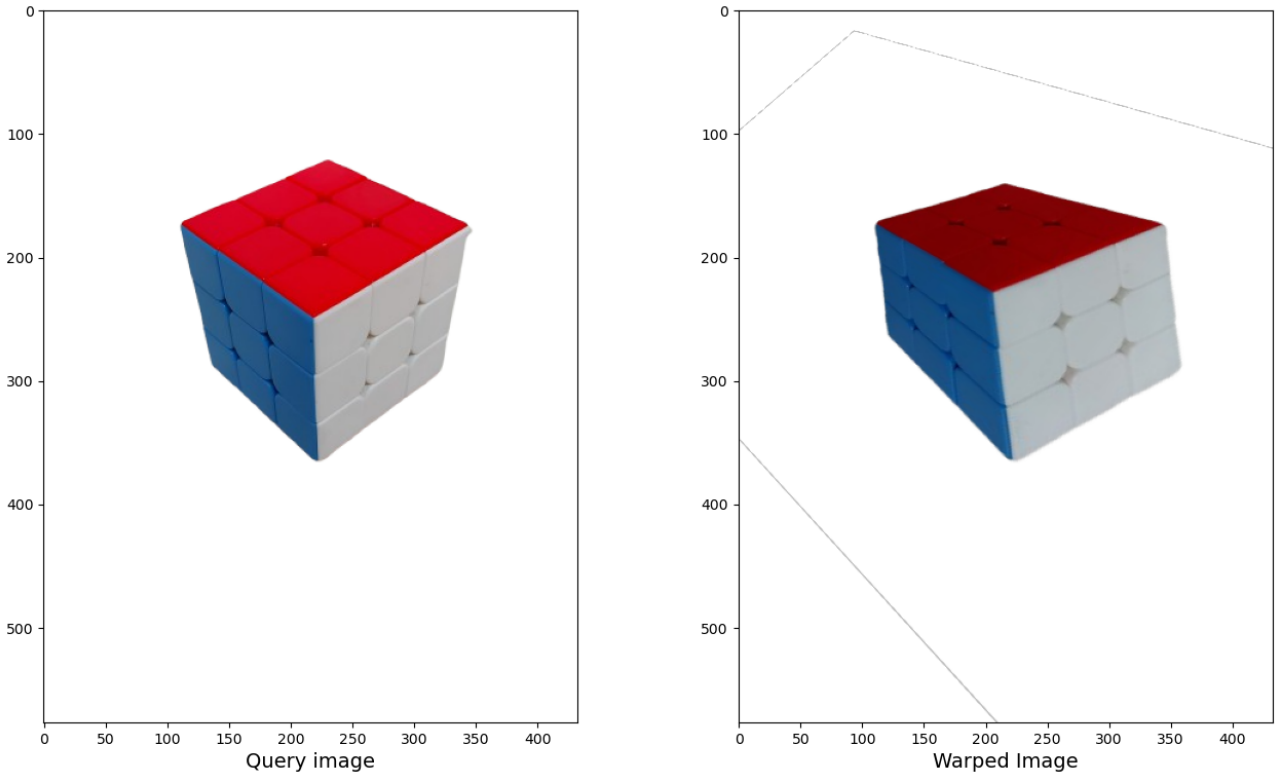


Figure 16: Warping using Keypoints and Matching using FLANN

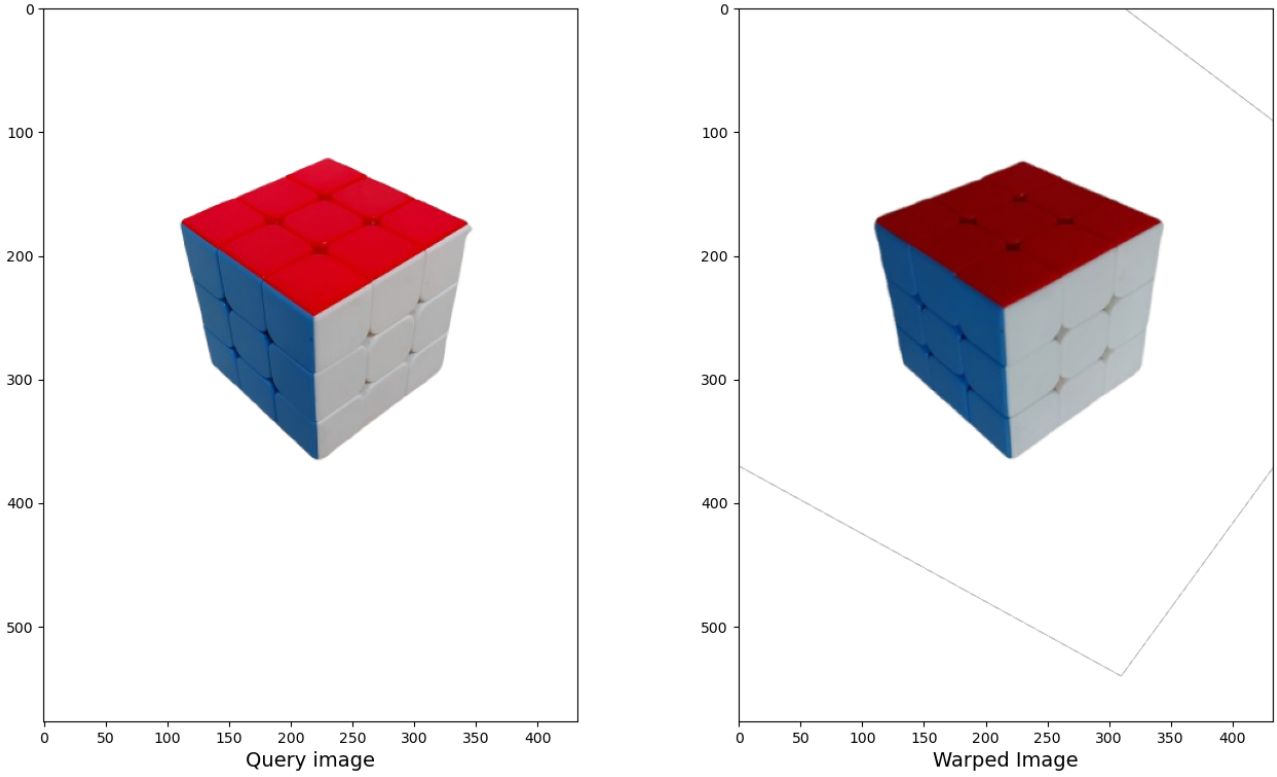


Figure 17: Warping using Manually Chosen Keypoints

this is due to the lighting conditions under which the images are taken. We can clearly observe a glare in quite a few of the images, which makes it difficult to figure out the chessboard.

2. Hence, I use the thresholding based approach described above, to help make the chessboard corners in the image very clear and easily detectable. Still, the lighting conditions for one of the images causes issues in this approach, and the eventual detection.

3. The camera calibration works better when the faulty image is removed. Although a chessboard is detected in the faulty image as well, it is not the same chessboard as the other images, and when this is removed, a lower reprojection error is observed.

4.2 Poor Homographies due to Wrong Matching

1. As we observe in Section 2, the homography computed by using the matches from the BF Matcher is very poor. This is primarily due to the large number of wrong matches, due to which the RANSAC algorithm is not able to perform well.

2. On the other hand, when we used FLANN based Matcher, we observe that although there are still some wrong matches, they are lower in number, and the RANSAC algorithm is able to take care of the outliers, resulting in a proper matching.

3. The most accurate matching is by using corresponding points as detected by the findChessboardCornersSB function. This is equivalent to manually finding and choosing the keypoints.

4. In this analysis, we have still not observed the effect that the non planarity of the chosen keypoints has on the homography computation, and the errors are primarily due to wrong matching.

4.3 Effect of Non-Planarity on Computed Homographies

1. In the previous section, we observed the effect of incorrect matching on the homographies. In section 3, we try to observe how the non planarity of the keypoints can affect the homography.

2. To this regard, we observe that with the keypoint detection methods, we are still able to get accurate enough keypoints and matching. But when we go for homography computation and warping, the results, although somewhat correct, are not fully perfect.

3. When we manually select the keypoints and matching according to the seven visible corners of the rubik's cube, we observe that the homography is near perfect, with the cube undergoing required rotation to ensure a perfect warping.

4. Hence this demonstrates that non planarity of the keypoints may affect the homography, and make it inferior in quality as compared to the homography computed when the points are chosen manually.

5 Relevant Links

The project code and results be found on Github at the following link - https://github.com/siddharth-kothari9403/CameraCalibration-and-HomographyComputation.

The link to the OpenCV documentation for Camera Calibration is available here - https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html.

The link to the OpenCV documentation for Homography Computation is available here - https://docs.opencv.org/4.x/d9/dab/tutorial_homography.html.

The code for homography computation has been borrowed from the following assignment which we did in the Visual Recognition course - https://github.com/SSS-192858/VR_ImageProcessing.