Alex Brockman

CSC 391: Introduction to Computer Vision

3/1/19

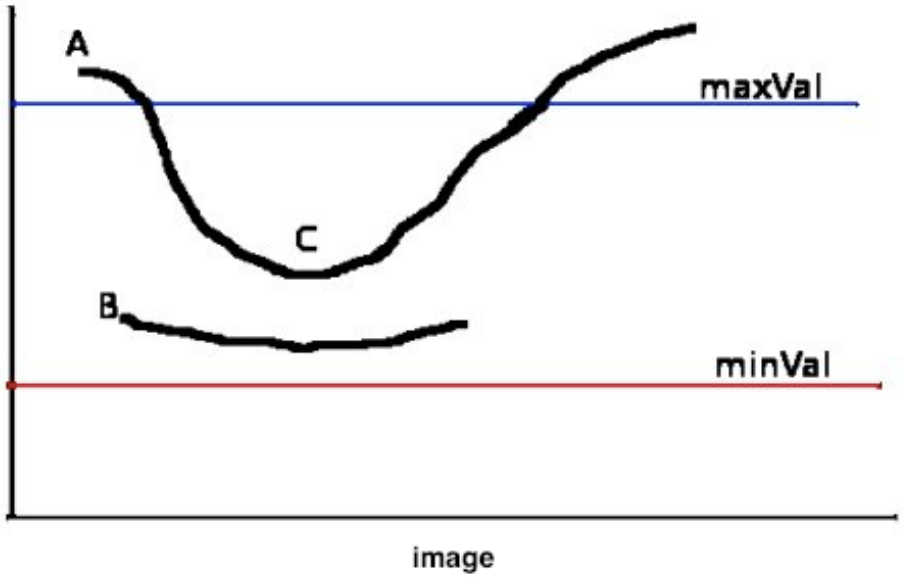
Project #2: Local Feature Extraction, Detection and Matching

Part 1: Edges and Corners in Video

---

a) Edge Detection

Canny edge detection is one algorithm for detecting edges in images. In general, an edge in an image is a place of rapid change in an image intensity function. In this part, I analyzed the results of the Canny edge detection algorithm on different parameter values within our code. I utilized the cv2.Canny() function within the OpenCV library. It allowed me to specify a threshold from a minVal to maxVal which would determine which edges would be marked. The algorithm contains several stages but only in last and final stage is a user able to modify the edge detection parameters.



Edge A, B and C thresholding

If A, B and C are edges, A would be marked as a valid edge because it exists above the maxVal threshold. Edge C would also be marked because it is connected to the A edge and is above the minVal threshold. However, edge B would be excluded because no part of it exists above the maxVal.

Below are screenshots of the results of the edge detection method in the video feed:

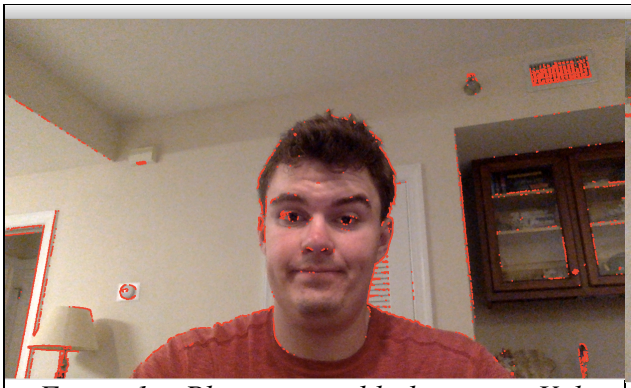


Figure 1a: Photo in good lighting, minVal: 200, maxVal: 200

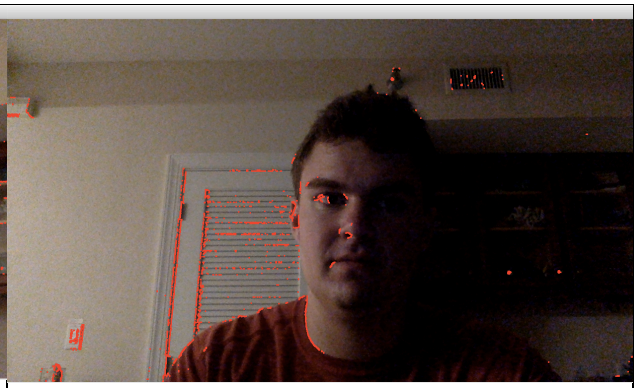


Figure 1b: Photo in bad lighting, minVal: 200, maxVal: 200

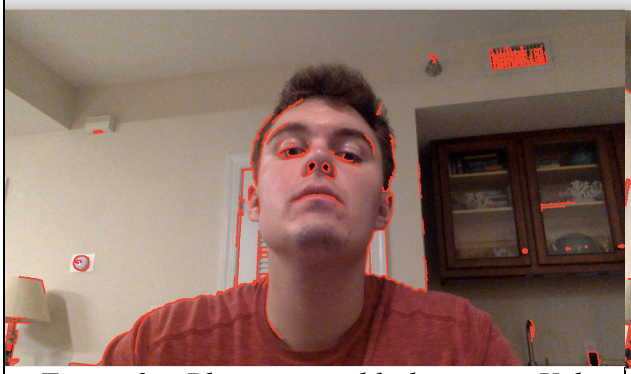


Figure 2a: Photo in good lighting, minVal: 100, maxVal: 300

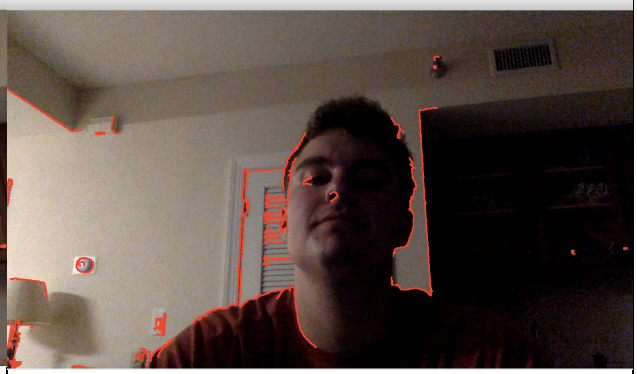


Figure 2b: Photo in bad lighting, minVal: 100, maxVal: 300

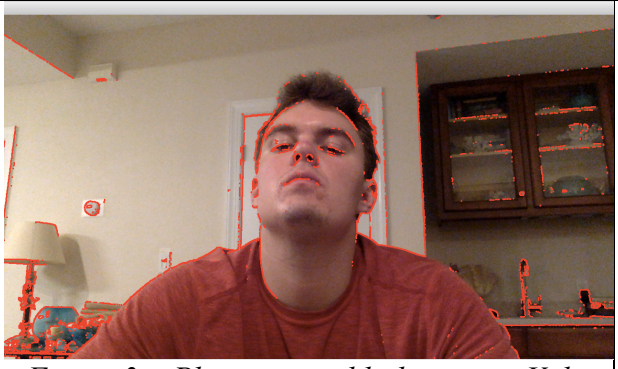


Figure 3a: Photo in good lighting, minVal: 175, maxVal: 175

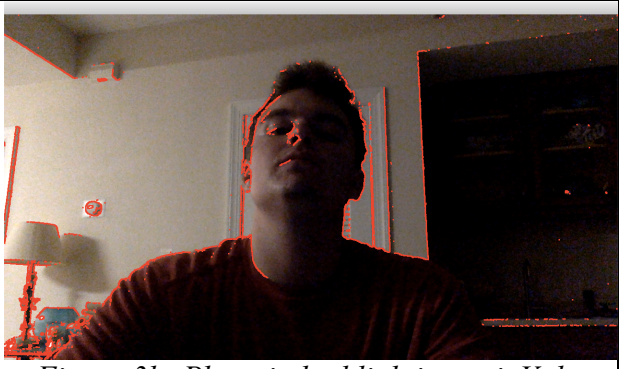
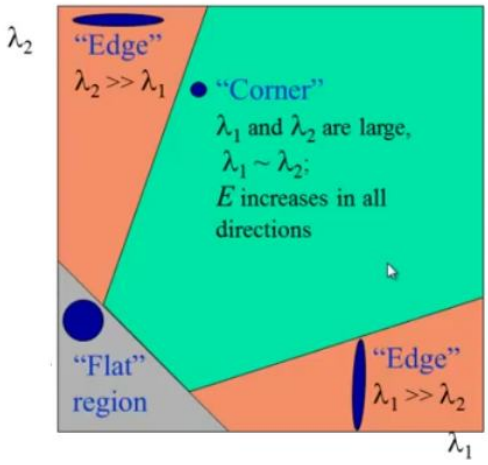


Figure 3b: Photo in bad lighting, minVal: 175, maxVal: 175

I took each photo in a well-lit scenario and a badly lit scenario to test the effectiveness of the algorithm. It is clear that a minimum and maximum value of 200 left the algorithm with edges that were not fully defined. I suspect this is due to the minVal being set too high. When changing the minVal to 100 and the maxVal to 300, the edges were not defined correctly. I believe the minVal is too low here. In the cabinets behind my head, hardly any of the edges were defined, even in the well-lit photo. The best combination of minVal and maxVal I found was a minVal of 175 and a maxVal of 175. The edges were very well defined on the cabinets, the wall, the thermostat, the door and my body in both lighting conditions. I suspect this is due to the minVal and maxVal capturing edges in the sweet spot of the video feed.

b) Corner Detection

The corner Harris algorithm is one method of identifying corners in an image. It contains several functions that result in a score to determine whether an image contains corners. These eigen values are compared to determine whether the feature in the image contains a corner, an edge or is flat.



The eigen values correspond to an image feature after an R function is evaluated

In the cornerHarris function in OpenCV we can specify thresholds for the part of the image considered for corner detection (blockSize) and the region for analysis (ksize). The last parameter (k) is not important. The Harris algorithm was applied with various parameter values on two types of images captured from live video:

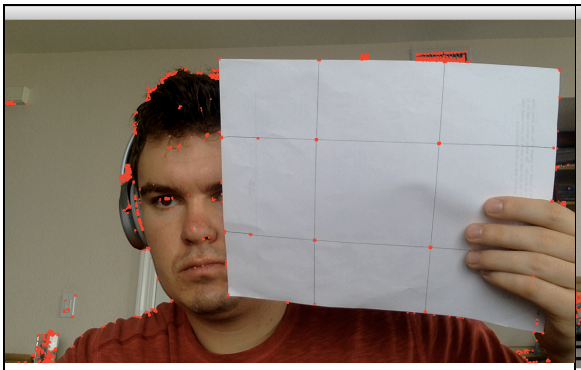


Figure 1a: Harris corners with blockSize: 3, ksize: 3

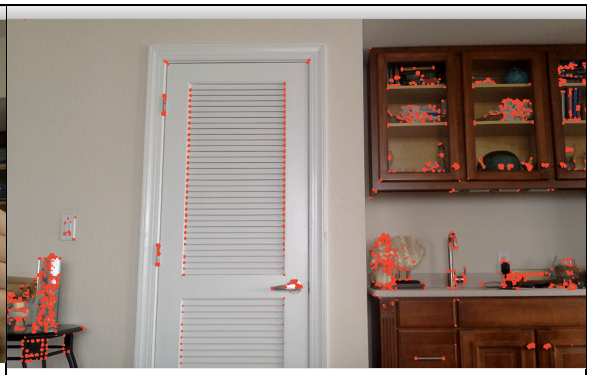


Figure 1b: Harris corners with blockSize: 3, ksize: 3

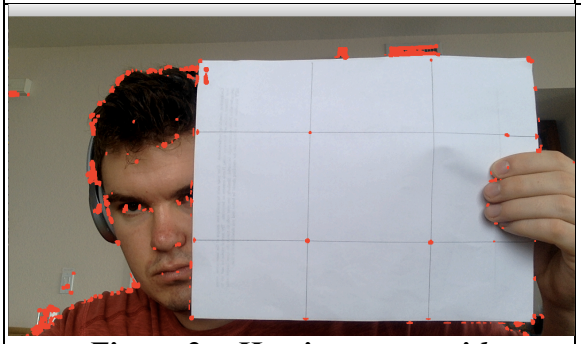


Figure 2a: Harris corners with blockSize: 7, ksize: 7

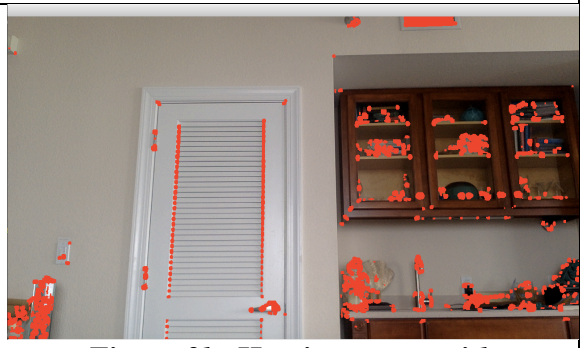


Figure 2b: Harris corners with blockSize: 7, ksize: 7

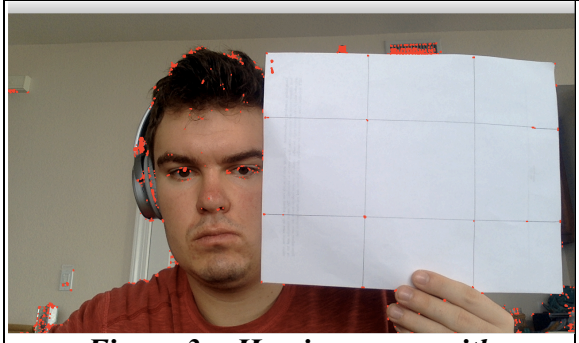


Figure 3a: Harris corners with blockSize: 2, ksize: 5

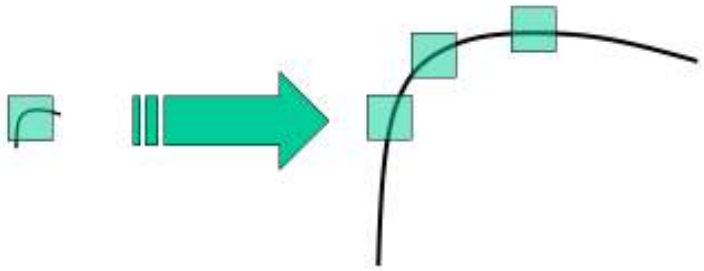
Figure 3b: Harris corners with blockSize: 2, ksize: 5

The Harris corner algorithm is more effective with static images than live video, but it is clear that lower blockSize and ksize values are most effective in producing the most accurate corners. I suspect that when the blockSize and ksize are larger like in Figure 2, too many points are determined to be corners because the threshold is too large. An intersection between points in the image is incorrectly determined. The most prominent corners in the images on the left side are on the paper grid and on the right, the door in

the background. In all cases, the corners on the grid and door were well defined although the pictures in Figure 2 displayed more corners than the images contained. It is interesting to note that the ksize seemed to have much less effect on the corners displayed than the blockSize value. Therefore, Figures 1 and 3 best displayed the corners in the images.

Part 2: SIFT Descriptors and Scaling

The SIFT algorithm extracts keypoints from images and computes their descriptors. When an image is translated and rotated, these keypoints and descriptors are used to recognize the locations of important features in the image. Image extrema detection, keypoint localization, orientation detection and keypoint descriptors make up the steps of the SIFT algorithm.



The SIFT algorithm creates a keypoint based on several factors

When experimenting with the SIFT algorithm, I discovered that images with well separated colors and objects retained their keypoints and descriptors. For example, I used a simple photo with a yacht in the ocean and a basketball court layout:

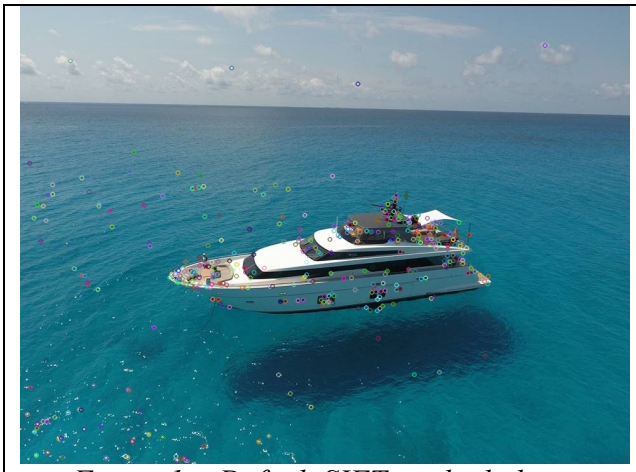


Figure 1a: Default SIFT applied photo

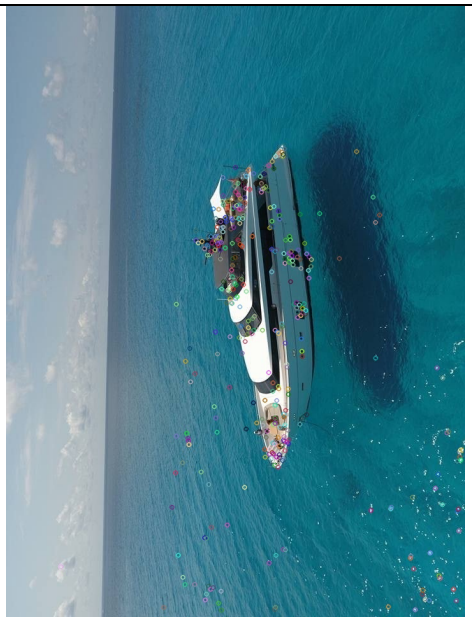


Figure 1b: SIFT applied rotated photo

Figure 1c: SIFT applied photo at 75% resolution

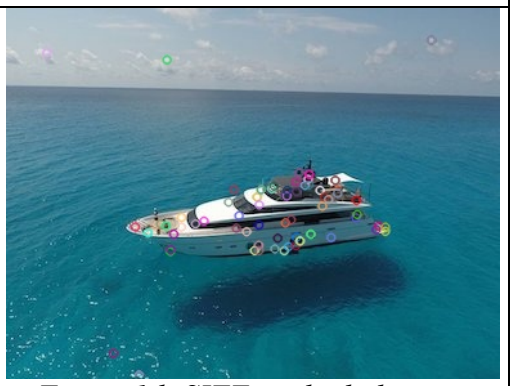


Figure 1d: SIFT applied photo at 40% resolution

It is clear in these images that the amount of keypoints identified was roughly identical when the image was rotated and the resolution was decreased to 75%. However, when I reduced the resolution further to 40%, some of the keypoints were lost in the resulting image. Many of the keypoints in the water were lost because the detail in the image was diminished when the resolution was decreased. The amount of displayed keypoints in a variable the user can modify. When I specified the amount of keypoints to display as a

SIFT parameter in my implementation, the result was similar to the default in this example.

A different picture was analyzed to determine if SIFT keypoints and descriptors would be retained after a translation and a resolution reduction:

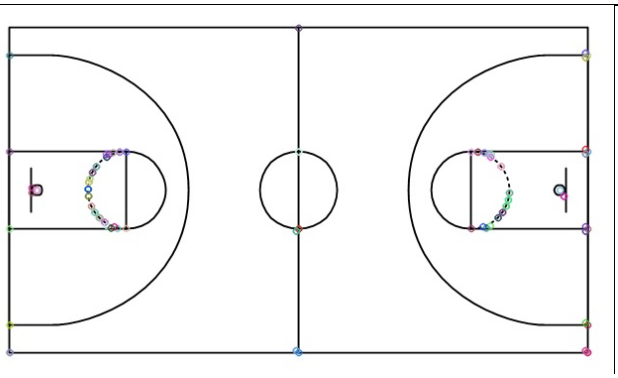


Figure 2a: Default SIFT applied photo

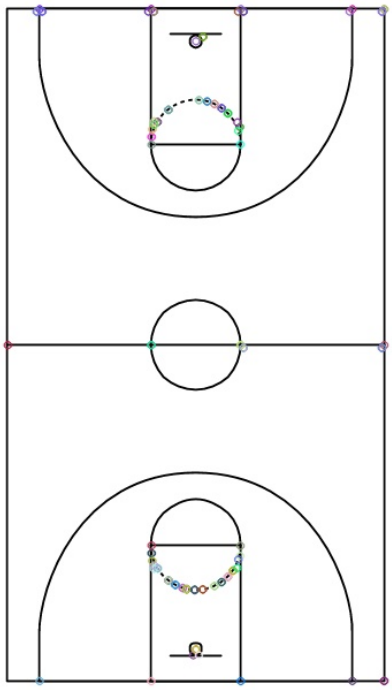


Figure 2b: SIFT applied rotated photo

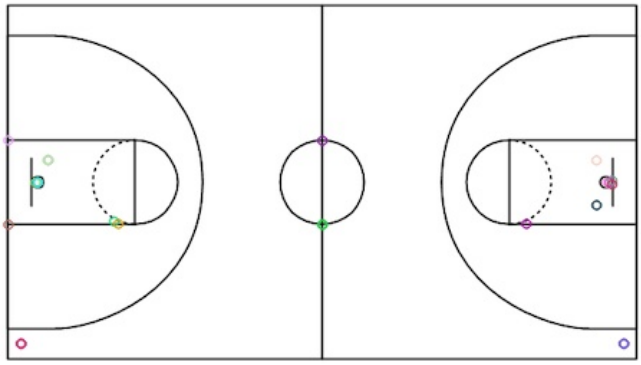


Figure 2c: SIFT applied photo at 75% resolution

This photo produced a different result. The default and rotated photos produced a similar amount of keypoints when the SIFT function was applied. However, when the image resolution was reduced by a mere 25%, the amount of keypoints produced decreased

dramatically. I imagine this type of image is much more likely to respond aggressively to changes in scale because of its simplicity. The details of the image blend together more easily when the image resolution is decreased.

Part 3: Keypoints and Matching

The OpenCV orb function is an efficient alternative to the SIFT and Harris functions. ORB is short for Oriented FAST and Rotated BRIEF and was developed in 2011 as a low performance SIFT alternative. The function takes two images as inputs and computes their keypoints and descriptors. The BFMatcher function then checks for matches between the two images. When finding keypoints it uses the Harris corner algorithm and it uses the BRIEF method to create descriptors. The efficiency in the method can be attributed to the lack of rotation detection features from the SIFT and Harris functions. ORB utilizes clever math to replicate the rotation detection from the other methods. The ORB function was applied to several image examples:

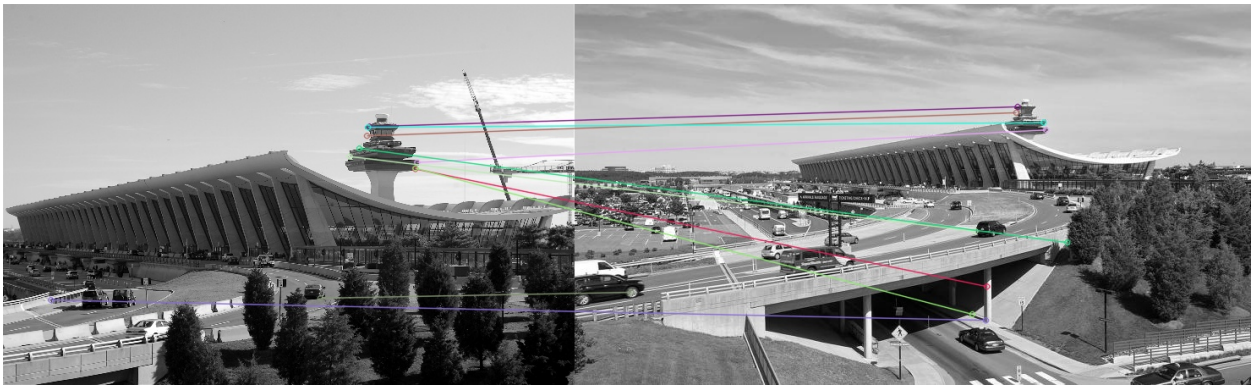


Figure 1: Airport photo with two distances

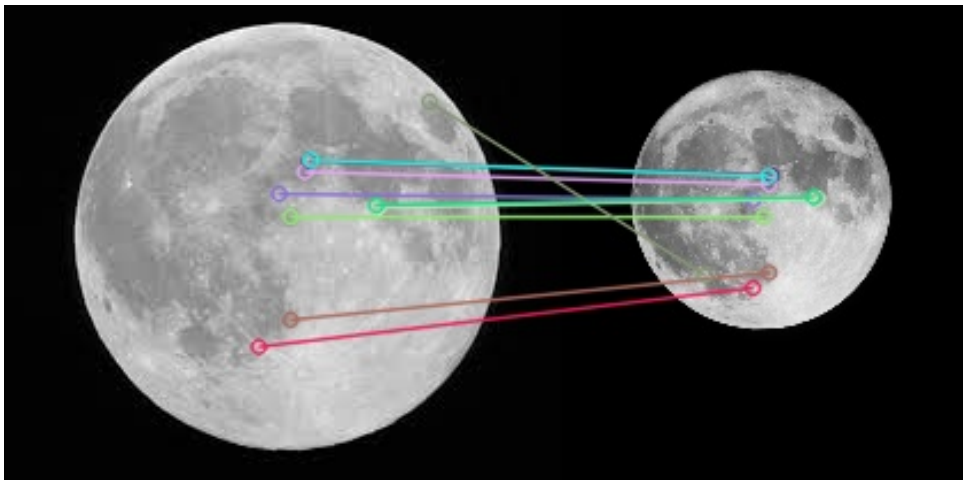


Figure 2: Moon photo taken from different sources

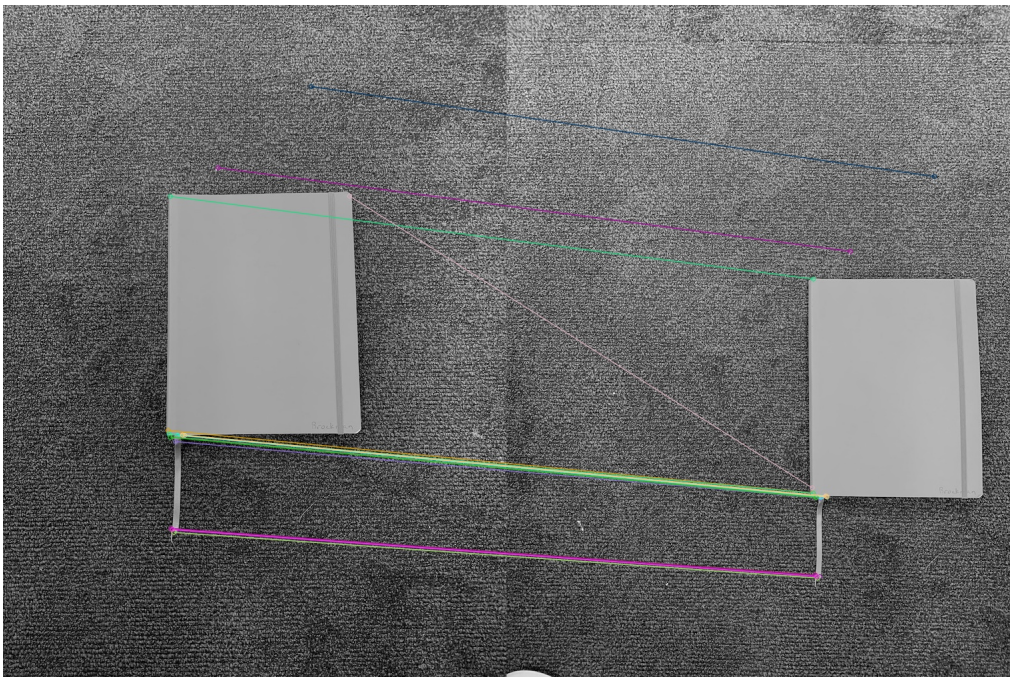


Figure 3: Book photo taken from different perspectives

The ORB function performed well on these photos. However, it is not perfect. In Figure 1, the perspective change causes mismatching in a few cases in the comparison. The features around the control tower are captured well, but features around the road and landscape are mismatched. The combination of Harris corner detection and SIFT keypoint detection in the algorithm occurring might be the reason for the mismatching. Analysis of keypoints using one of these methods at a time would likely produce different results. In the moon and book example, the ORB method performed exceptionally. The resolution of the moon photo was 418 x 207 pixels and the keypoints were well matched

with a couple exceptions. In the book example, the perspective was modified and the keypoints were well matched. I suspect the high resolution of the images (756 x 1008 in both cases) and the clear contrast between the subject and its background contributed to the good result.

It is important to note that using SIFT and Harris corners individually to produce keypoints may lead to different results but the built in ORB function in the OpenCV library is far easier to implement and produces strong results in the majority of cases.