CSC 391 Computer Vision
Project 2 Report

Tianen (Benjamin) Liu

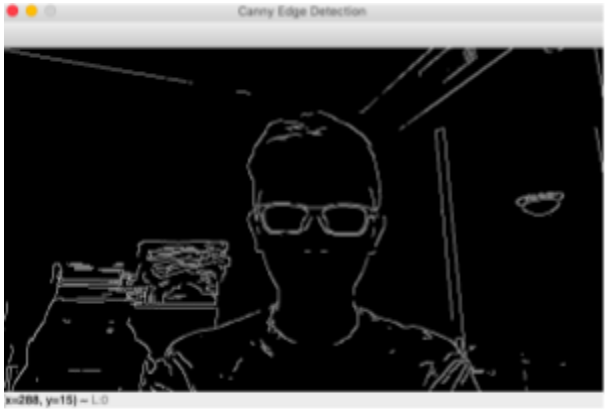
Mar 1, 2019

1. Edges and Corners in Video

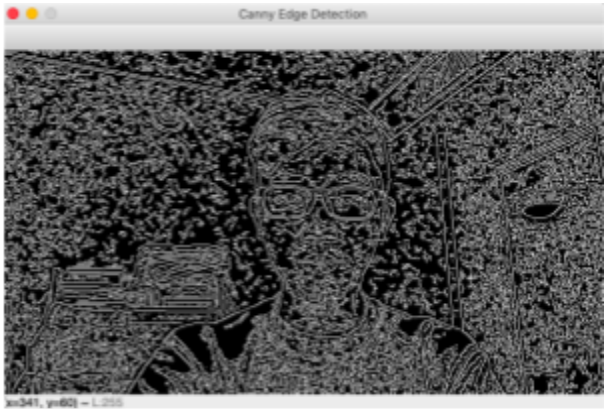
a. Canny Edge Detection

This part of the project utilizes Canny Edge Detection on video camera to detect edges of images captured by the camera. The core command for Canny Edge Detection is `cv2.Canny(image, threshold_1, threshold_2)`. Canny Edge Detector uses these 2 thresholds to adjust how detailed the resulting edges will show. Below are the experiments of Canny Edge Detection using different pairs of thresholds under different environmental conditions including good lighting (brighter) and bad lighting (darker). Both of these 2 conditions take scaling into consideration.

Before trying out the thresholds, we can get an idea of the effect of the magnitude of thresholds by checking the following pictures. When both thresholds are large, there are less detailed edges, whereas when both thresholds are small, more detailed edges are shown. For efficiency purpose, I chose 3 pairs of thresholds, which are (50, 150), (150, 250), and (250, 350) to estimate the optimal threshold.



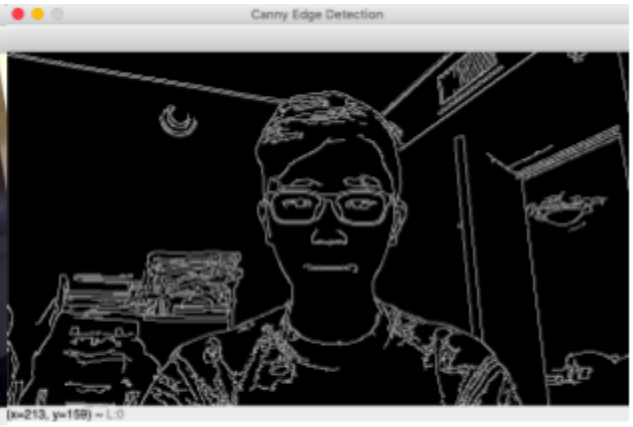
(250, 300)



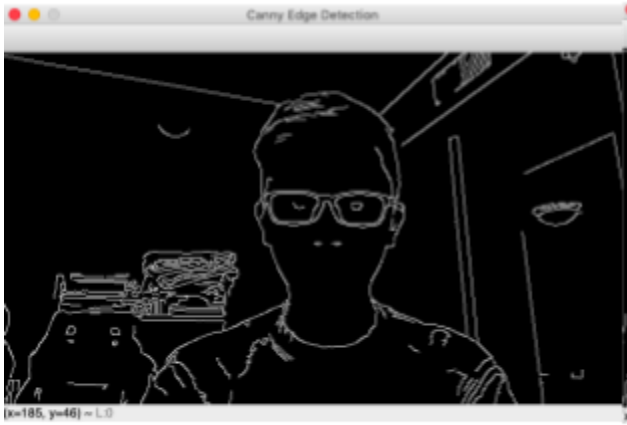
(30, 50)

i. Good Lighting

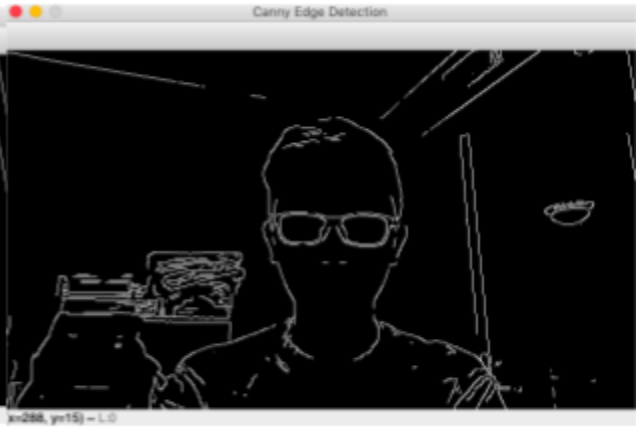
Good Lighting



(50, 150)



(150, 250)

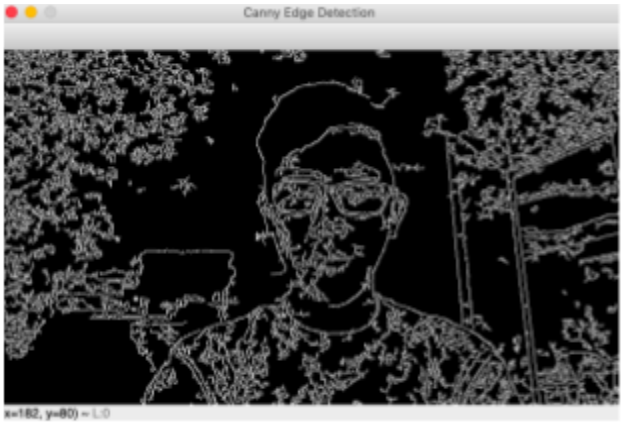


(250, 350)

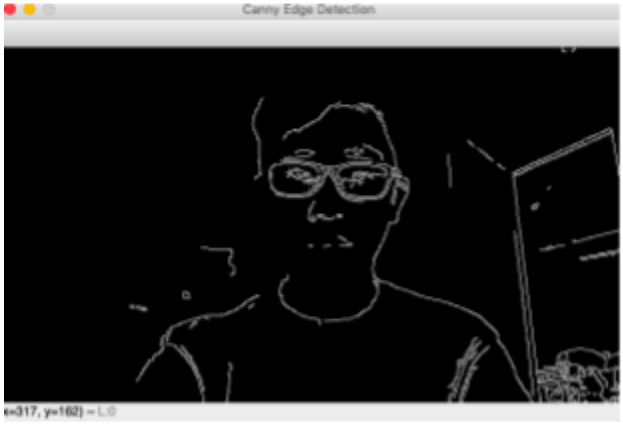
Threshold (50, 150) is a good fit for this good lighting situation as it shows edges of important features such as the walls, the door, the bookshelf, and clear detailed facial features. (150, 250) and (250, 350) are not as good because they do not catch the facial feature clearly.

ii. Bad Lighting

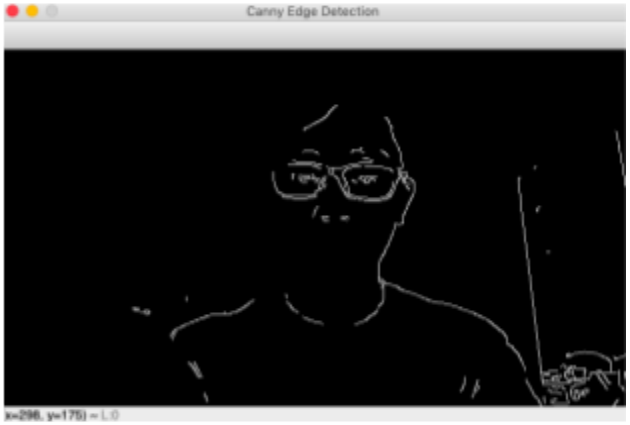
Bad Lighting



(50, 150)

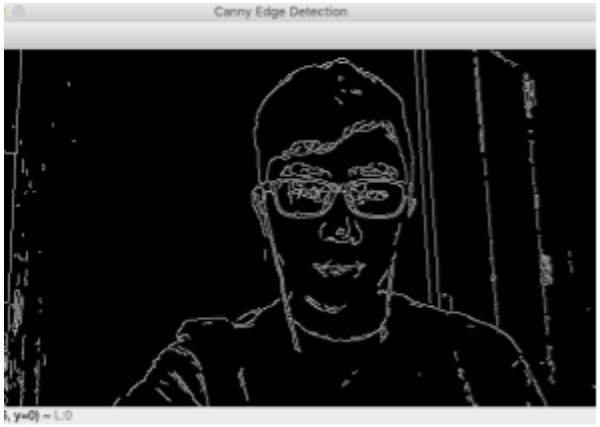


(150, 250)

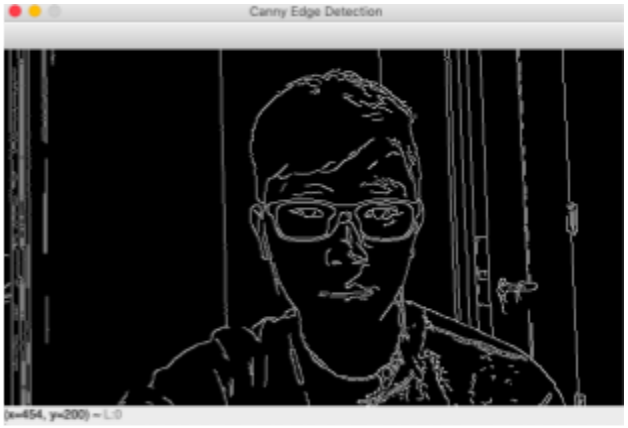


(250, 350)

Based on these trials with 3 pairs of thresholds, we can see that the optimal threshold is somewhere between (50, 150) and (150, 250) because (50, 150) captures too many noises while (150, 250) has many discontinued lines and unclear edges. Now try (100, 150) on Bad Lighting situation, and we can see that the outcome improves as there are more detailed edges on the face.



Bad Lighting (100,150)

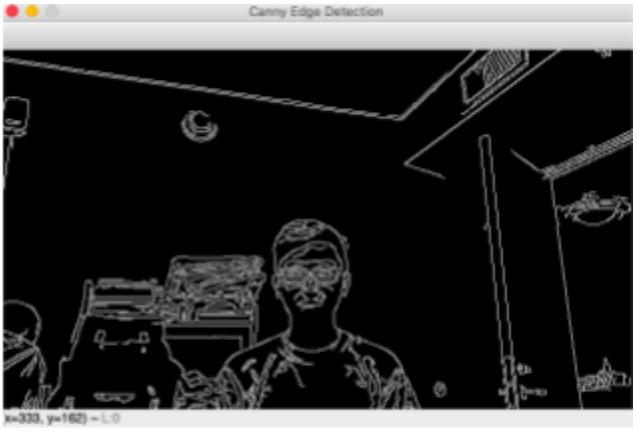


Bad Lighting (100,150)

As a result, I conclude that thresholds (100, 150) is the best for good lighting and bad lighting and is the best overall because it captures most of the key edges and leaves out redundant edges regardless of lighting conditions.

iii. Scaling

Close (100, 150)



Far (100, 150)

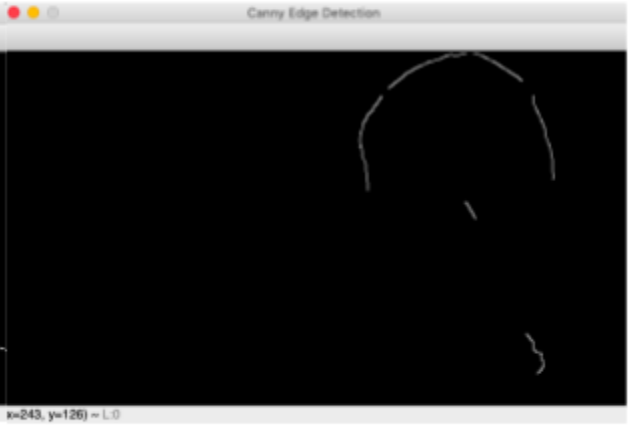
The result shows that (100, 150) yields no major problem with scaling. Whether its a close distance or a farther distance, the facial details are detected as edges and shown clearly.

iv. Blur

Blur

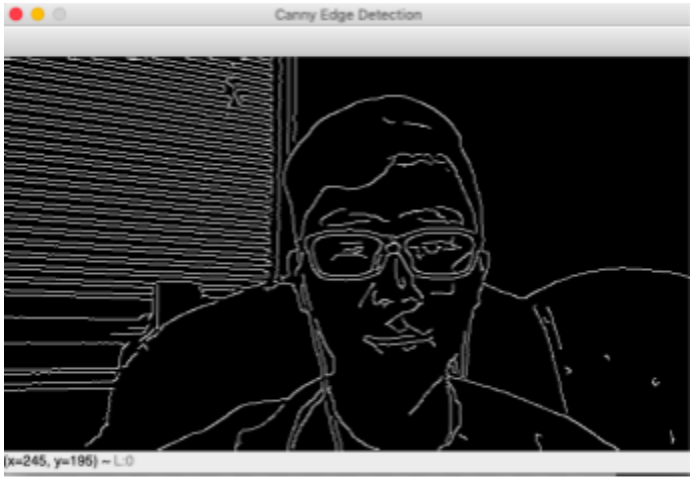
(50, 150)

(150, 250)



(250, 350)

The blurred image results from the method of Gaussian blur. From these results we can see that the optimal threshold should be smaller than (50, 150) since (50, 150) does not capture very detailed edges. Now try (50, 100) on the image. It looks like a good fit for this Gaussian blur.



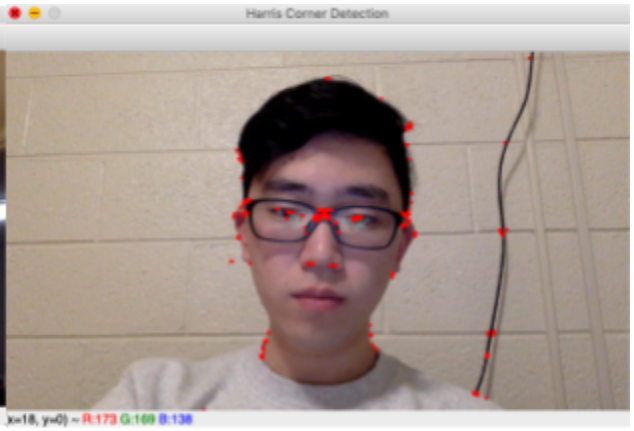
Blur (50, 100)

b. Harris Corner Detection

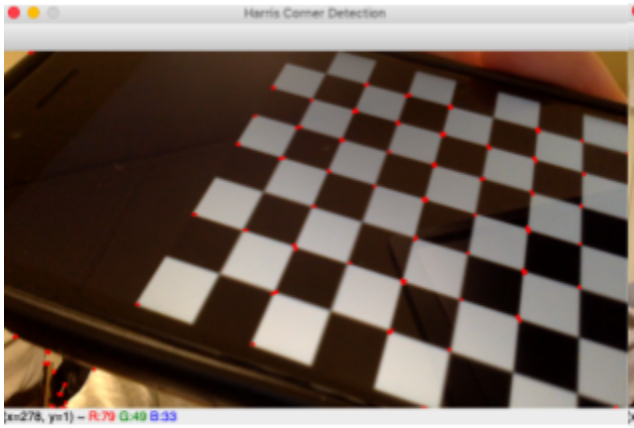
This part applies Harris Corner Detection on video camera to detect corners of images captured by the camera. The core command for Harris Corner Detection is `cv2.cornerHarris(image, blockSize, kSize, k).cornerHarris()` only applies to gray image so I first converted the images into gray and at last plot the detected corner points onto the colored images. Here I will experiment on the influence of parameter blockSize. I tried blockSize = 2, 4, 6, and 8 on girds and different environmental conditions including good lighting (bright) and bad lighting (dark). Below are the results. The result shows that all parameters for blockSize work well on girds, but some work better for different lighting. For good lighting conditions, the optimal blockSize is _. For bad lighting conditions, the optimal blockSize is _.

i. block Size = 2

Good lighting blockSize = 2



Bad lighting blockSize = 2



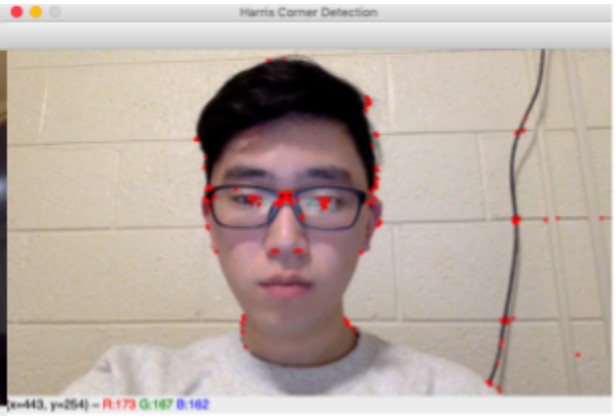
Grid blockSize = 2

Grid rotated blockSize = 2

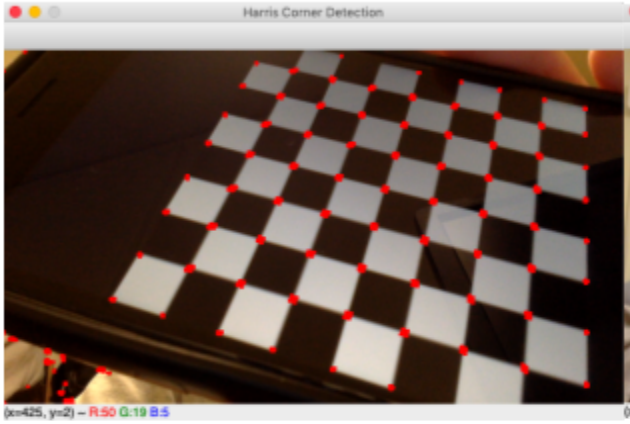
For blockSize = 2, when there is bad lighting (picture on upper right) shows a corner detected on the wall, which is a mis-detection. On the grid (picture on lower left), some corners are not detected.

ii. block Size = 4

Good lighting blockSize = 4



Bad lighting blockSize = 4



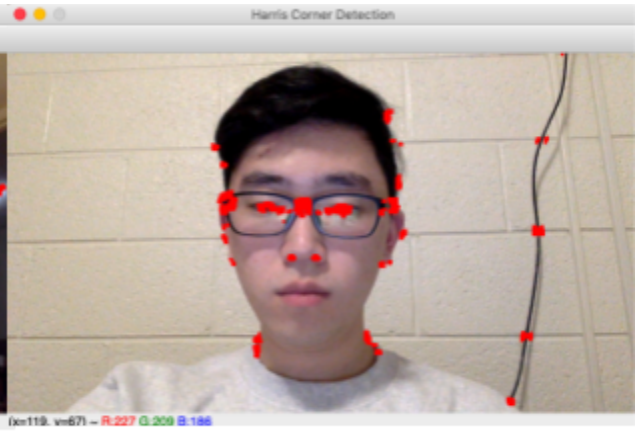
Grid blockSize = 4

Grid rotated blockSize = 4

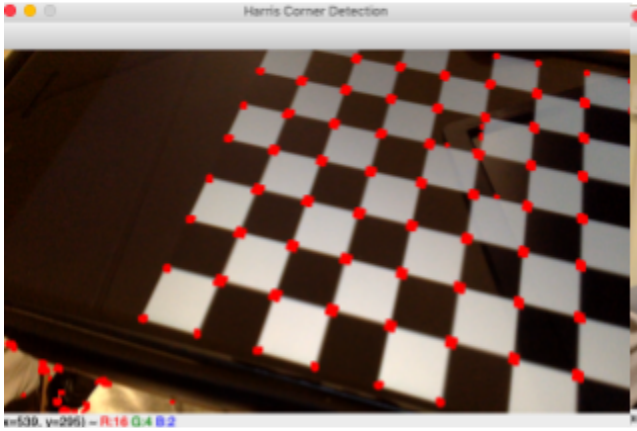
For blockSize = 4, when there is bad lighting (picture on upper right), some corners on the grid of the wall are detected, which are rather unnecessary because they are not obvious corners.

iii. block Size = 6

Good lighting blockSize = 6



Bad lighting blockSize = 6

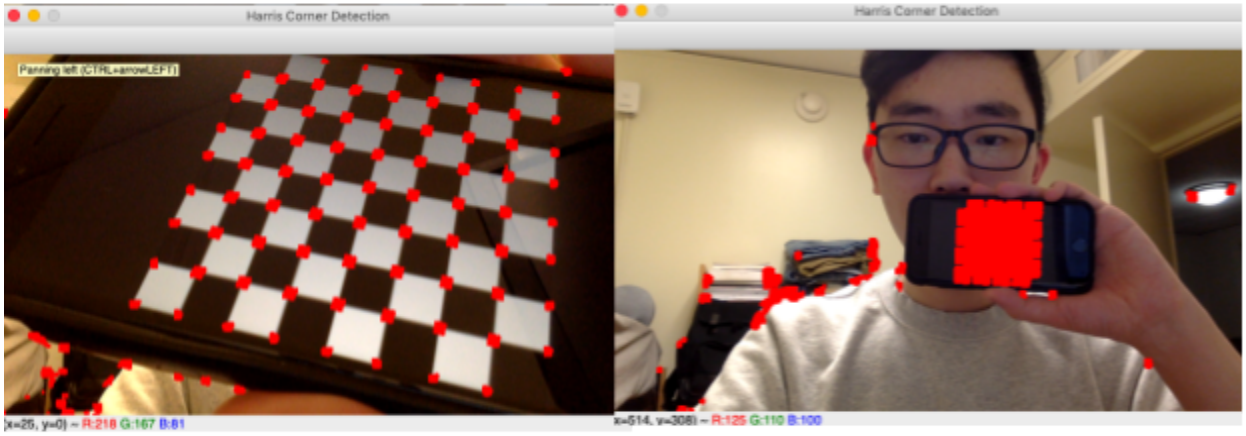


Grid blockSize = 6

Grid rotated blockSize = 6

iv. block Size = 8

Good lighting blockSize = 8 Bad lighting blockSize = 8



Grid blockSize = 8 Grid rotated blockSize = 8

In conclusion, I think blockSize 6 is the optimal parameter for these situations. It gives clear obvious corners with no mis-detection. The size of neighborhood is not as big as when blockSize = 8.

2. SIFT Descriptors and Scaling

This part applies `cv2.xfeatures2d.SIFT_create()` on images with the following transformation. There are 5 arguments for this function. Here I will try to modify nfeatures and sigma to find the best SIFT descriptors. nfeatures is the number of best features to retain, which is the number of keypoints to detect. Sigma is the Gaussian applied to the input image at the octave #0. The default values are nfeatures = 0, and sigma = 1.6. Below I will use the dog image as examples of how transformation changes SIFT descriptors. I will use Empire State Building image to find optimal parameters.

a. SIFT Implementation on Transformed Images & Their Optimal Parameters

i. Regular Image

Original Image

SIFT Keypoints default parameters

When apply `cv2.xfeatures2d.SIFT_create()` with default parameters, the result image will have a lot of keypoints.

nfeatures = 50, sigma = 1.6

nfeatures = 50, sigma = 2.5

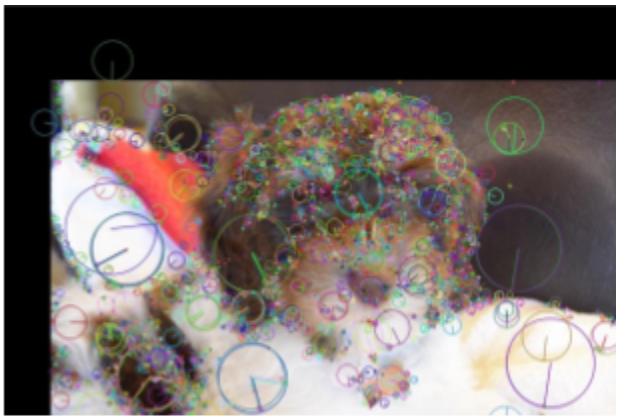
nfeatures = 100, sigma = 1.6

Regular nfeatures = 100, sigma = 2.5

For this untransformed image, I think nfeatures = 100, sigma = 2.5 is the best parameter. Since the image contains a lot of edge that could be keypoints, it is needed to increase nfeatures. Also, since this picture is not soft focus, it is better to increase sigma.

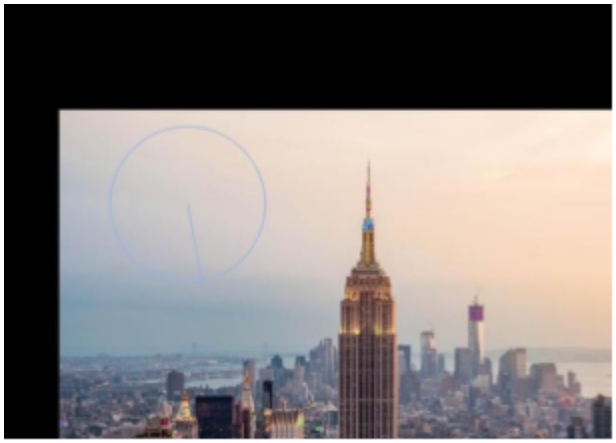
ii. Translated Image

SIFT Keypoints default parameters

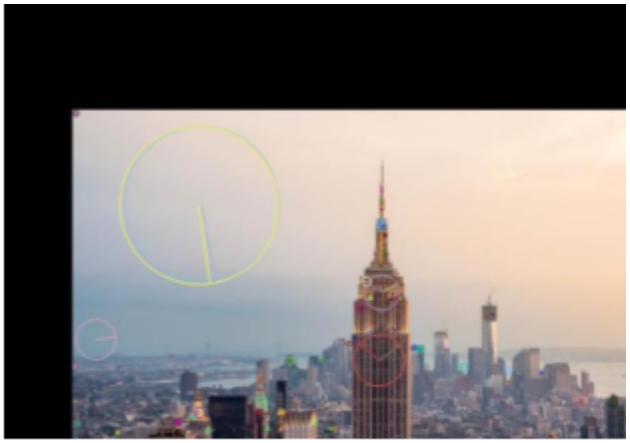


Translation - default parameters

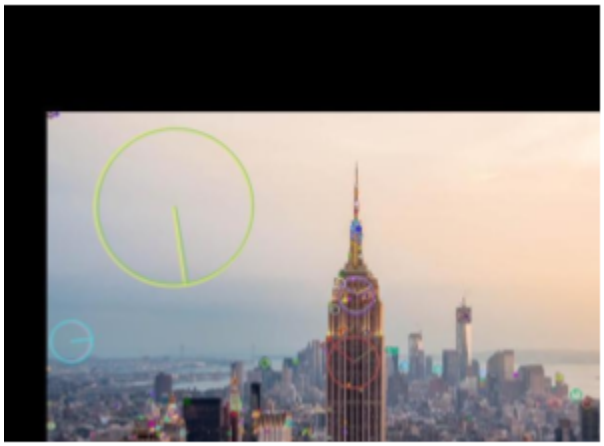
For translated images, the positions of the keypoints change slightly. Some keypoints are even out of the image boundary.



nfeatures = 50, sigma = 1.6

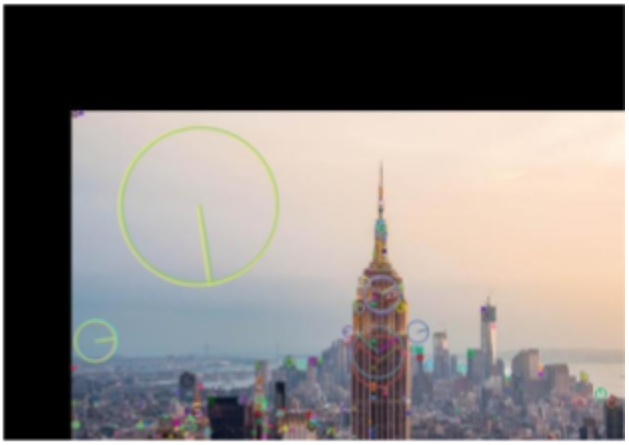


nfeatures = 50, sigma = 2.5



nfeatures = 100, sigma = 1.6

Translation

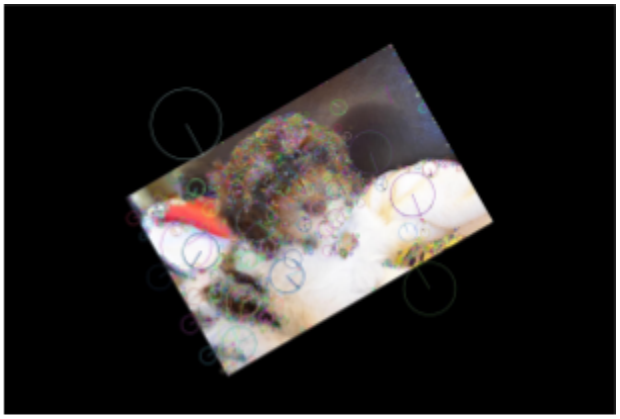


nfeatures = 100, sigma = 2.5

For translated images, I think nfeatures = 100, sigma = 1.6 is the best. Unlike (100, 2.5) which has a lot of keypoints of minor features, (100, 1.6) has a lot of keypoints on buildings without being too cluttered. However, there is a drawback of this parameter, which is the big circle in the sky.

iii. Rotated Image

SIFT Keypoints default parameters



Rotation - default parameters

Rotation of image change the keypoints greatly. Some keypoints are even out of the image boundary.

nfeatures = 50, sigma = 1.6

nfeatures = 50, sigma = 2.5

nfeatures = 100, sigma = 1.6

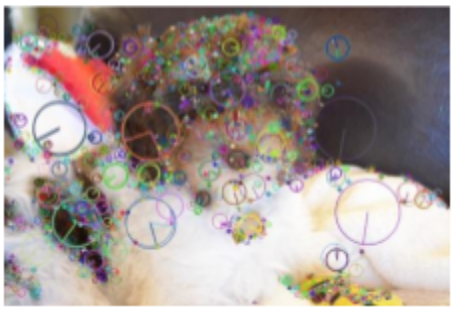
Rotation

nfeatures = 100, sigma = 2.5

I think the best parameters are nfeatures = 50, sigma = 1.6 because the other 3 have a lot of weird circles in the sky.

iv. **Scaled Image**

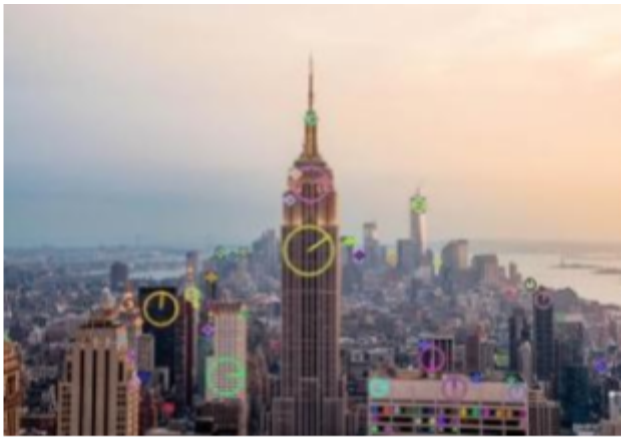
SIFT Keypoints default parameters



Scale - default parameters

Scaling of image changes the keypoints greatly.

nfeatures = 50, sigma = 1.6



nfeatures = 50, sigma = 2.5

nfeatures = 100, sigma = 1.6

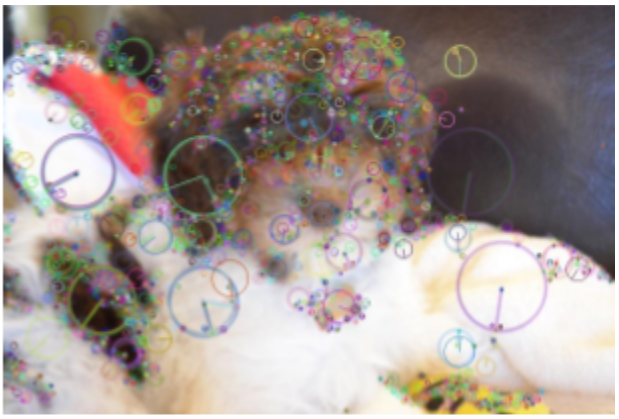
Scale

nfeatures = 100, sigma = 2.5

I think the best parameters are nfeatures = 100, sigma = 2.5. This pair of parameters gives more keypoints and more detailed keypoints.

v. Blurred Image

SIFT Keypoints default parameters

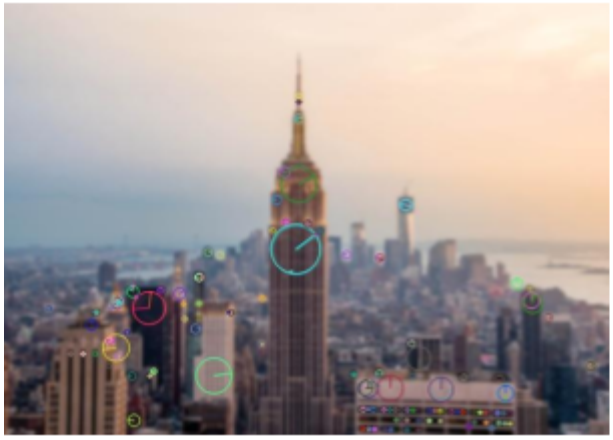


Blur - default parameters

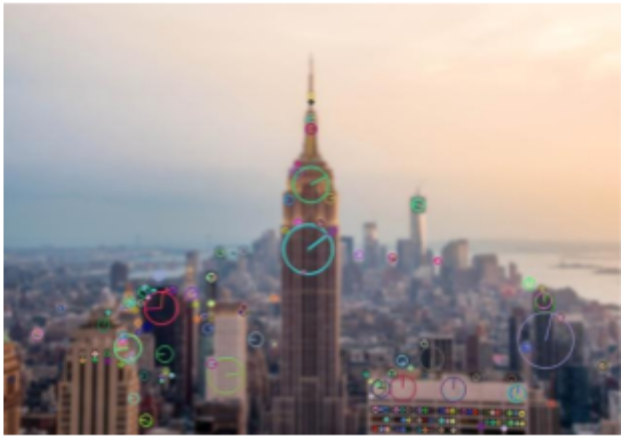
The blurred image has basically kept the same keypoints as the original image has. The blurred image has more new keypoints in new area.

nfeatures = 50, sigma = 1.6

nfeatures = 50, sigma = 2.5



nfeatures = 100, sigma = 1.6



nfeatures = 100, sigma = 2.5

Blur

I think nfeatures = 100, sigma = 1.6 is the best parameters because the keypoints are of fair amount. (100, 25) has an exceptionally large purple circle around a building so we would not want (100, 25).

b. SIFT Implementation on Video

i. Experiment on Parameters

default

nfeatures = 20

Implementing SIFT on video also needs command `cv2.xfeatures2d.SIFT_create()`, which has default nfeatures = 0. This results a lot of keypoints including minor edges and corners. By changing nfeatures to 20, only the top 20 keypoints of significance will show up.

ii. Blur Video

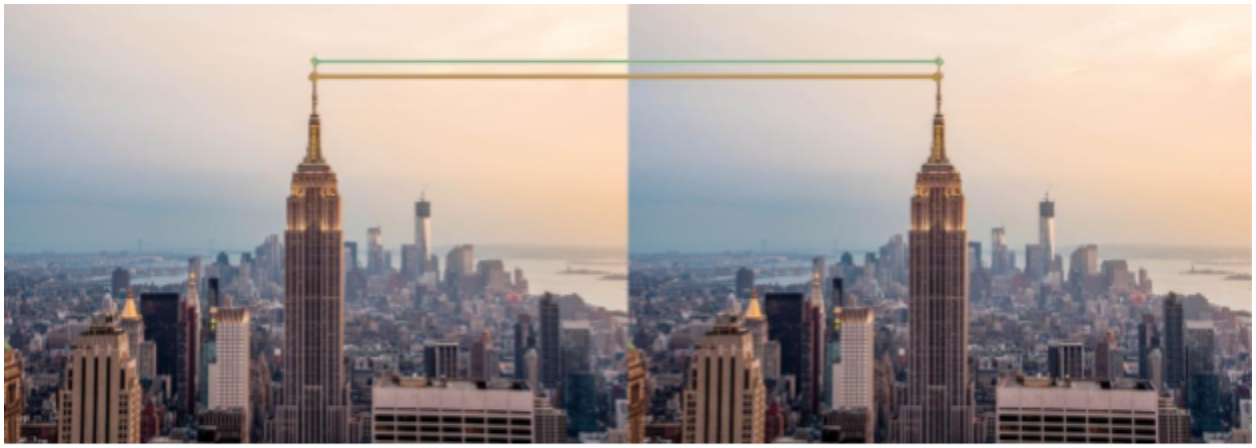
SIFT detects the similar keypoints even if the video is blurred. The following experiment shows the effect of SIFT keypoints on a blurred video.

SIFT nfeatures = 30 | SIFT nfeatures = 30 (blur)

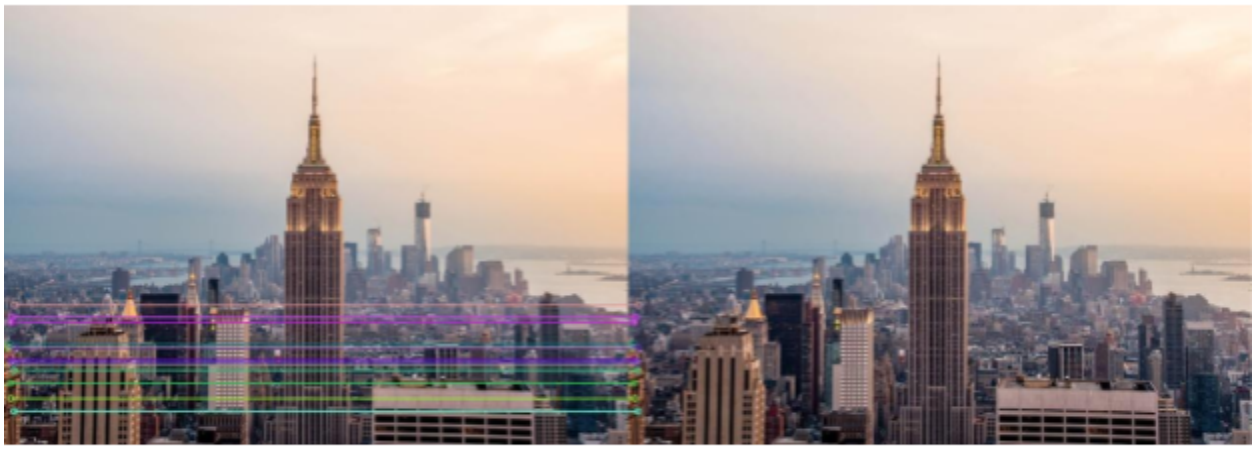
3. Keypoints and Matching

This part compares effects of matching using Harris Corner and SIFT Keypoints. The comparison will be based on an image matching with a regular image (itself), a rotated image, a scaled image, a translated image, an image with different angle, and a blurred image. Each matched image shows 15 matched lines.

i. Regular Image



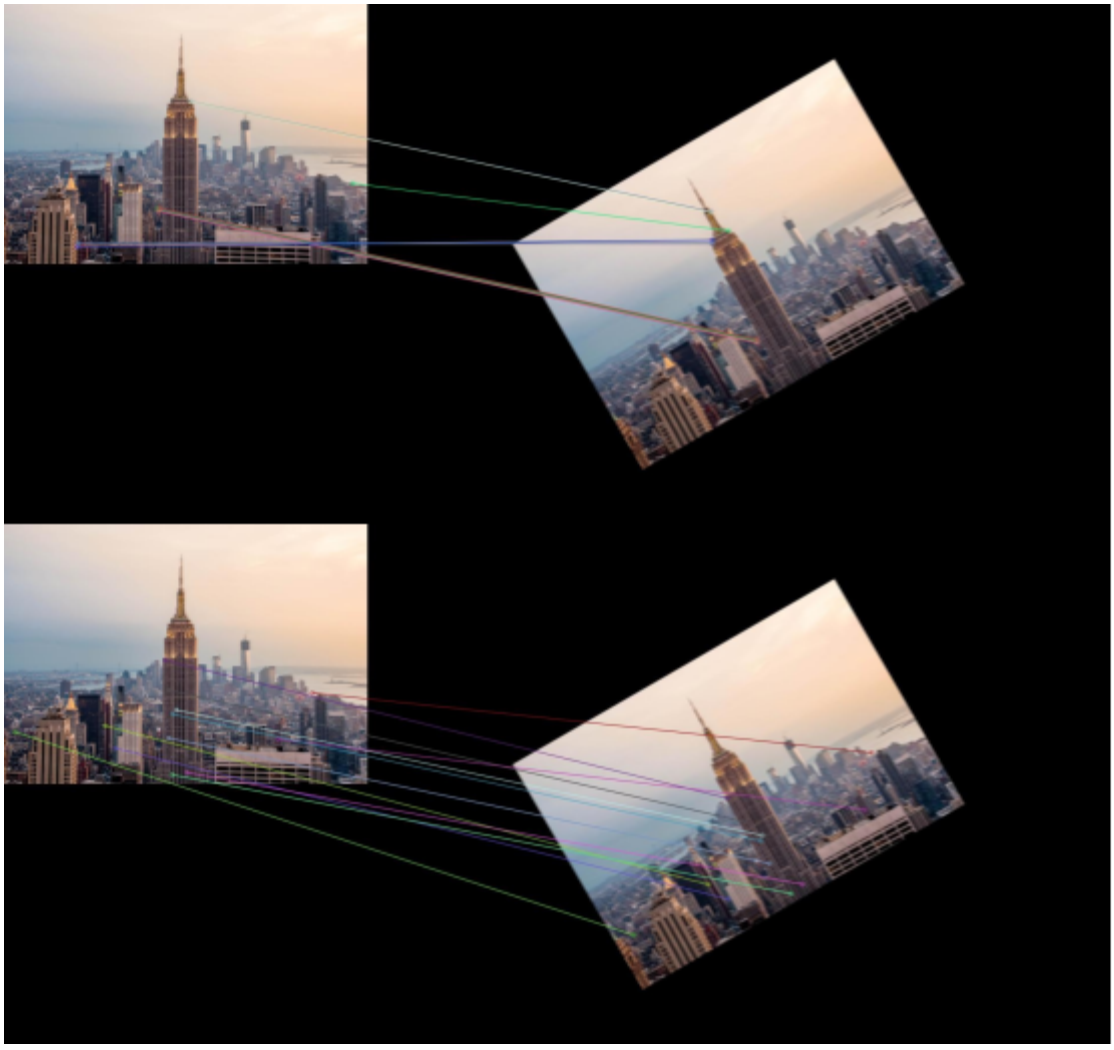
Harris



SIFT

First we compare Harris and SIFT matching to the same picture. We can see that different method focus on different parts of the image. These matches are clearly correct matches.

ii. Rotated Image



Harris

SIFT

Harris shows mismatches, such as the coastline matched to the top of the Empire State Building (green line). SIFT has all correct matches.

iii. Scaled Image

Harris

SIFT

Harris shows mismatches as the yellow, teal, and purple lines are matched to almost the same spot. SIFT has all correct matches because the lines are nearly not intersecting.

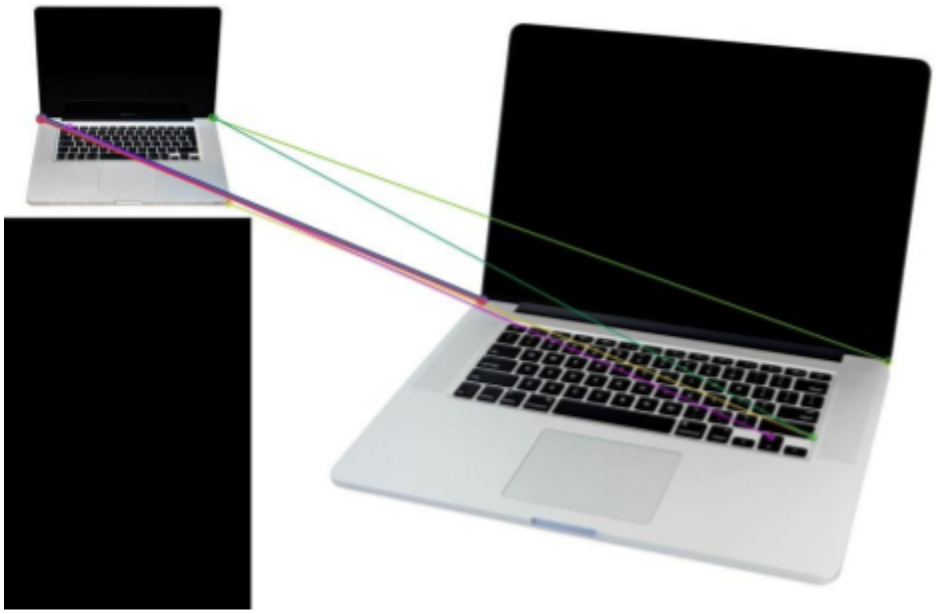
iv. Blurred Image

Harris

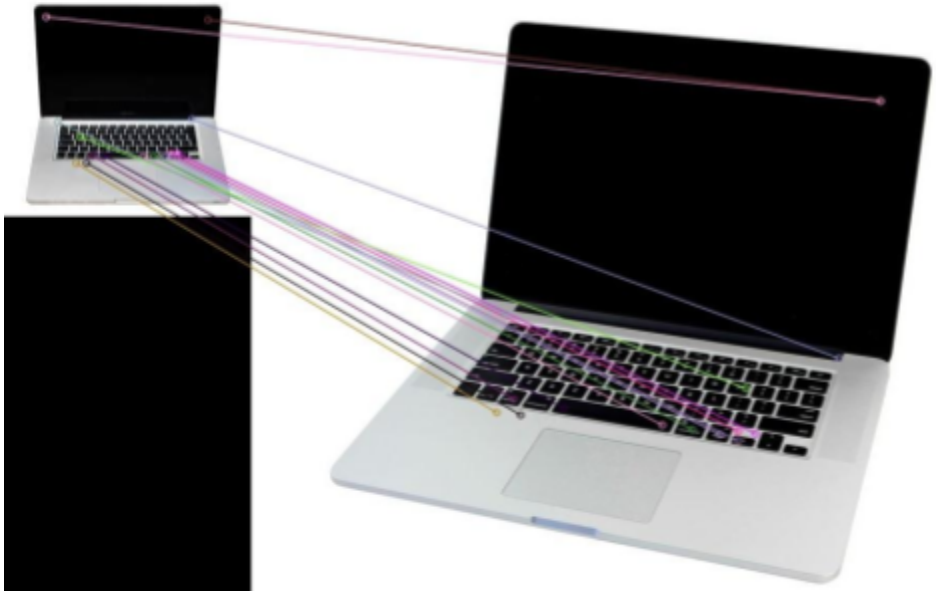
SIFT

Both methods work pretty well on blurred images.

v. Image Taken from Different Angles



Harris

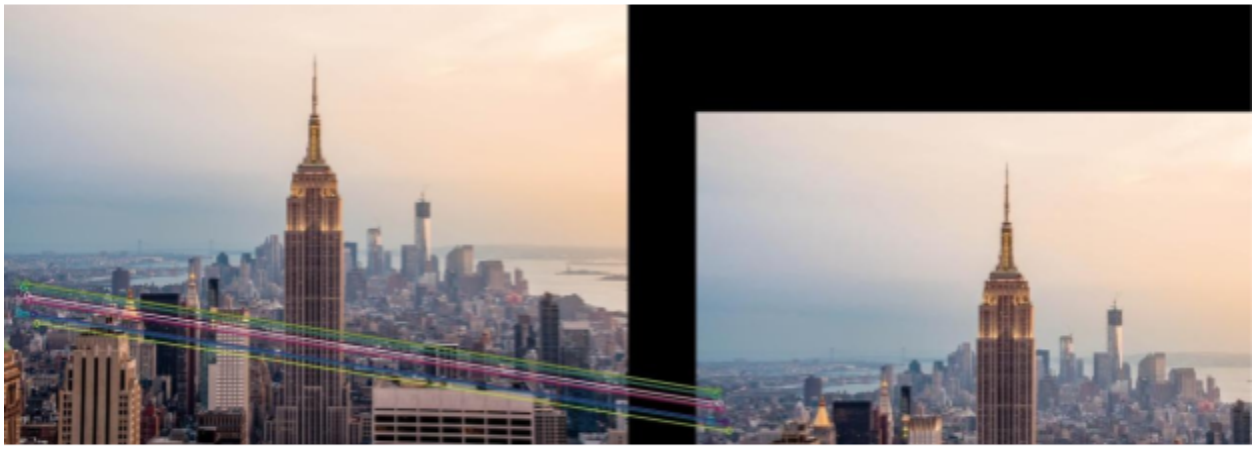


SIFT

Harris shows mismatches such as the edge of the MacBook being matched to the edge of the keyboard (teal line). SIFT has one mismatch too as the left hand side of the screen is matched to the right hand side of the screen (pink line).

vi. Translated Image

Harris



SIFT

Both methods work pretty well on translations since the lines are nearly parallel.