Computer Vision Project2
Bill Xu

Edge and Corners in Video

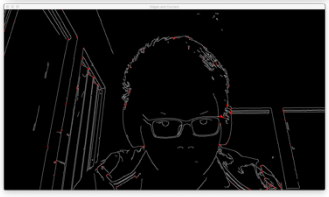
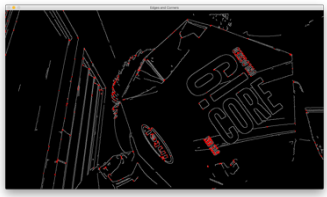
In this part I'm experimenting on the best parameters for an edge & corner detector. The settings for the pictures I've used is me myself (as an object of curved/smooth edges and corners) and a Intel i9 CPU box(as an object of sharp edges and corners) in my room. I will also adjust the light, distances and blurriness to test its performances.

Edge (Canny) Parameter Tuning

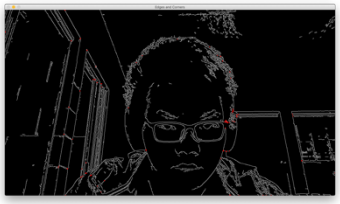
First let's keep the light/distance to be fixed in a natural extent, and no blurriness is applied.

Original

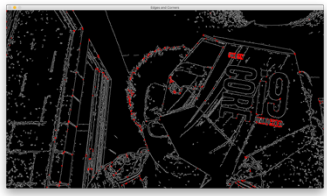

100-200

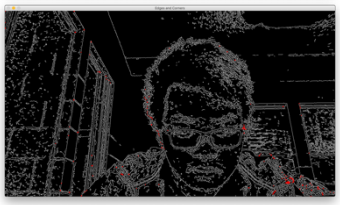
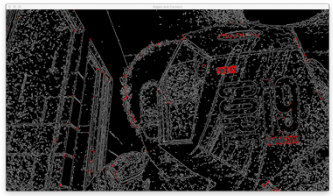
50-100




40-80

30-50

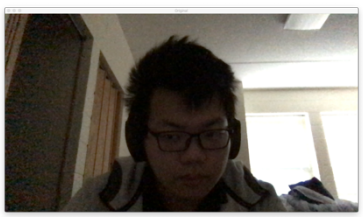



The parameters used for tuning in Canny edge detector are basically max threshold and min threshold during Hysteresis Thresholding. The max threshold decides the minimum of “sure edge”, while the min threshold decides the minimum “possible edge” to consider. I started with 100 for min threshold and 200 for threshold, and the results are bad in terms of weak edges detection (one cannot observe my mouth). After decreasing the threshold pair to (50,100), a majority of edges are observed, even though there is one obvious missing edge on the lower left of CPU box, and a missing edge of my forehead. After decreasing the pair to (40,80), all the detailed edges on both human face and CPU box seem clear, even though several little wiggly edges appear. When we decrease the threshold to (30,50), the wiggly edges seem to be everywhere.

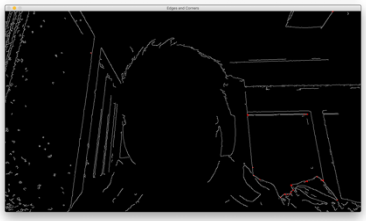
Thus, we can roughly conclude that a good max threshold for Canny edge detector is in (80,100) while a good min threshold in (40,50). We will then examine these intervals in extremely dark and lightened situations.

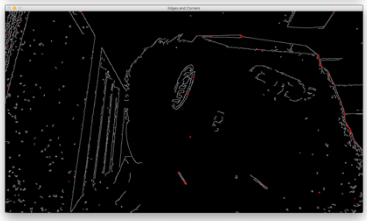
Now let’s fix the distance, but darken the background.

Darkened

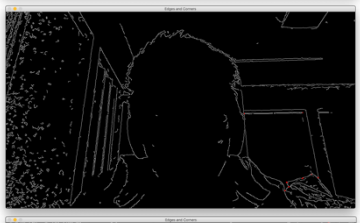


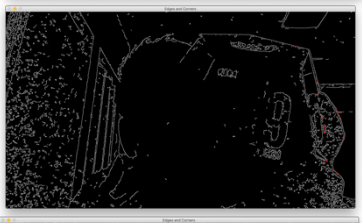
50-100



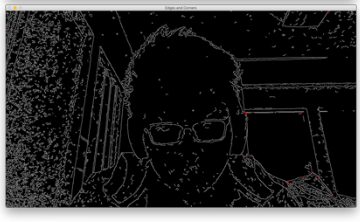


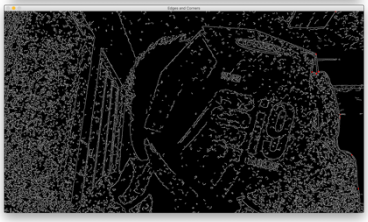
40-80





30-50





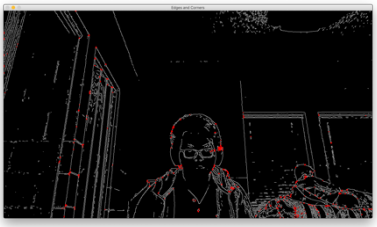
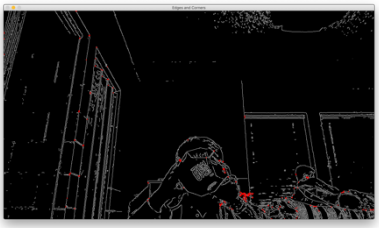
I tried the rough interval concluded in the previous part, but even if I decrease the threshold to the lower bound (40,80), lots of edges are still missing. So I have to decrease even more and it’s

not until (30,50) that I started to get roughly every edges I want. However, as demonstrated before, low threshold like this generates a lot of little wiggly edges.

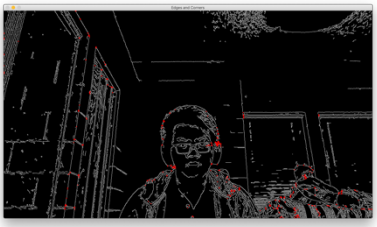
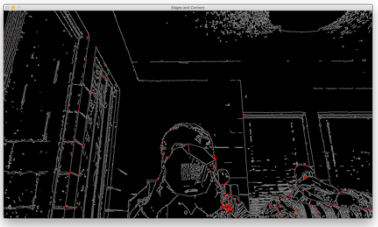
Let's now look at the effect of distances.

Distanced

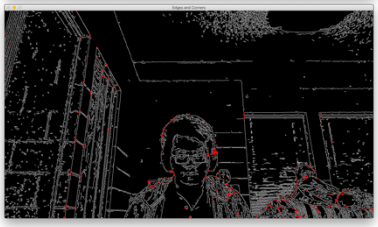
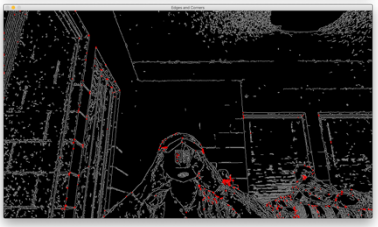

50-100

40-80

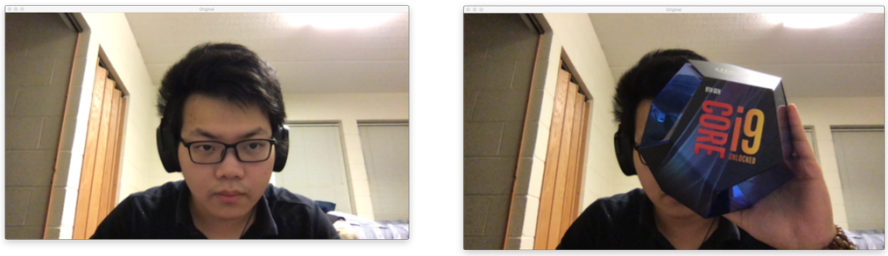
30-60

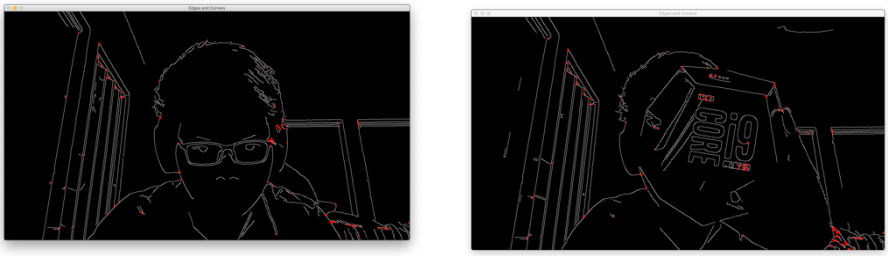
As before, I started experimented with (50, 100) and (40,80) for min/max threshold. Both pair of parameters are good except not detecting a shadowed edge of the CPU box. I decreased the threshold even further but still no such edge displayed. At the same time, tiny wiggly edges becomes obvious when max threshold goes below 80.

Now let's check the effect of blurriness on Canny edge detection.

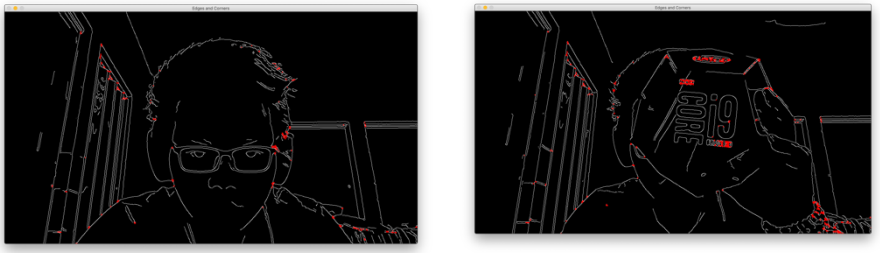
Blurred



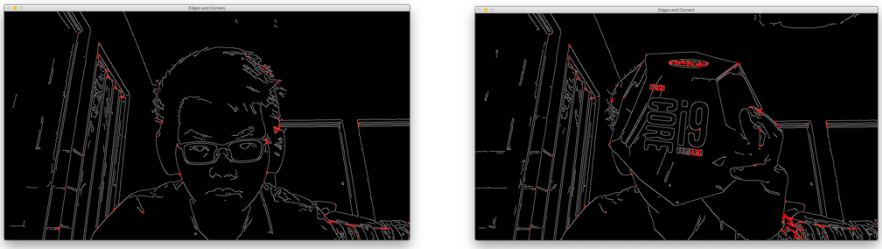
50-100



40-80



30-60



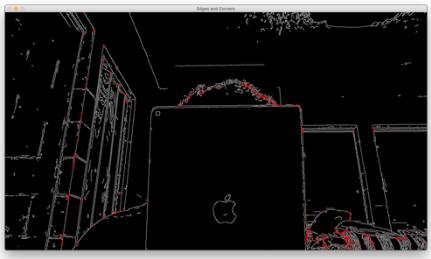
Blurring image using a Guassian filter removes the high frequency parts and thus the gradient of edges become smaller. Then, we need a lower threshold to detect the blurred edges. In this case, a good edge detector appears until I decrease the thresholds to (30,60).

After countless trials, we can conclude that a good Canny edge detector have max and min thresholds around 40 and 80. The detector is generally covariant with distance changes but influenced by light intensity and blurriness. When the background is dark or when image is blurred, a lower threshold should be considered to detect all the edges.

Corner(Harris) Parameter Tuning

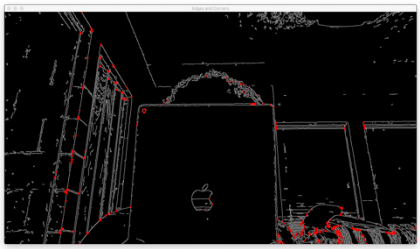
Now let's fix the edge parameters to be (40,80) and adjust parameters in Harris Corner Detector.

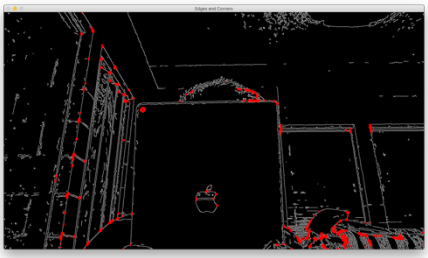
Original



bs = 2

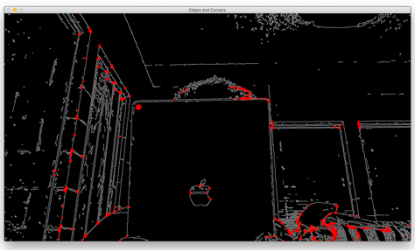
bs = 4

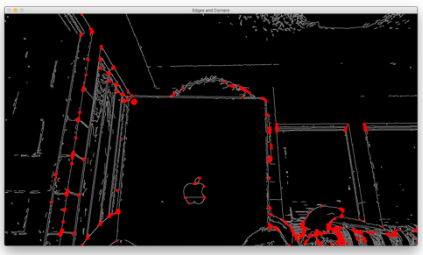




bs = 6

bs=8





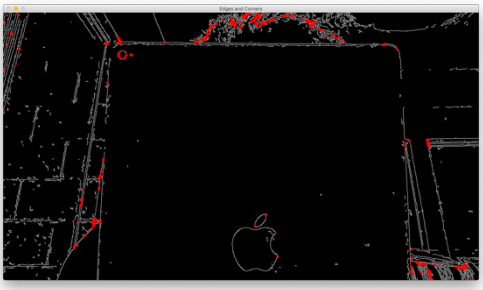
bs=10

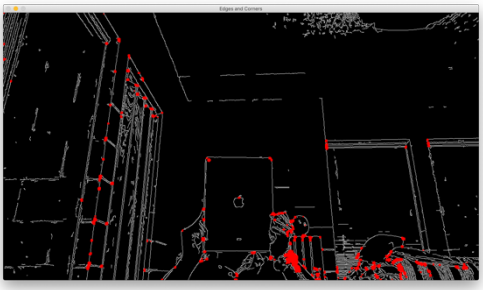
There are 3 parameters that could be tuned for Harris corner detection, namely block size, ksize and k. After multiple trial, I decide that block size, which controls the shift amount to calculate change in intensity, is the leading variable that affect performance.

In a natural setting as the previous session, I used an iPad for a clearly cornered object. Starting from blocksize=2, it is clear that corners of the apple logo are not detected. As I increase bs, bigger (or less sharp) corners are detected. At bs=6, we get a nice appearing corner detector. However, as we increase bs further, more non-edges are colored as well (such as quilt on bed). Notice that iPad camera is fully detected as a color at bs=10, which suggest it approaching to a blob detector.

Now let's try the detector's ability of invariance on geometric transformations, using bs=6 we have just chose.

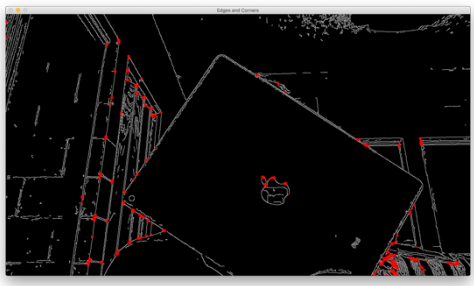
Closer

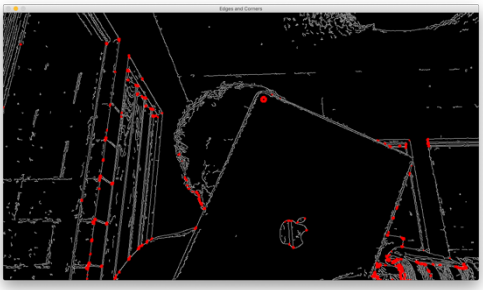




Farther

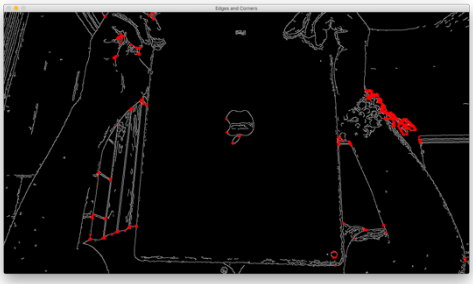
Rotate

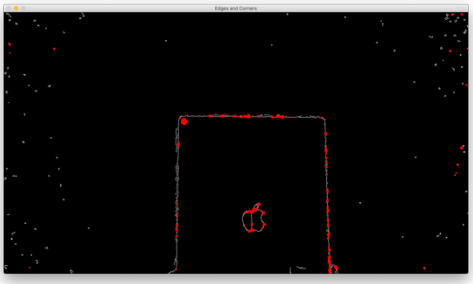




Rotate

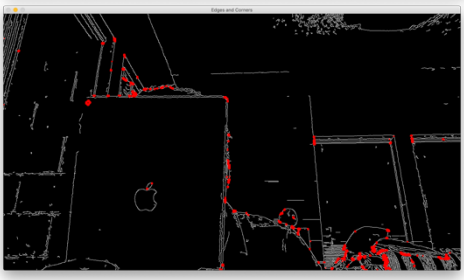
Rotate

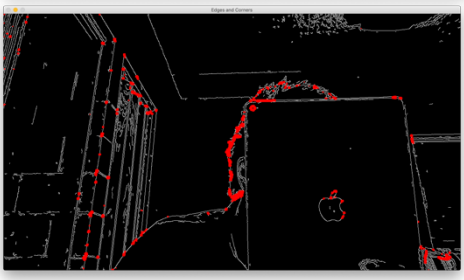




Dark

Trans





Trans

Amazingly invariant to translations, rotations and intensity changes! And even short-distance scaling doesn't affect the corner detection. In each picture above, the corners of iPad and Apple logo are almost perfectly detected.

SIFT descriptors and scaling

In this section I'm using a bottle cap to test the SIFT descriptor because this blue circle is easy to detect and only contains one feature as key point.

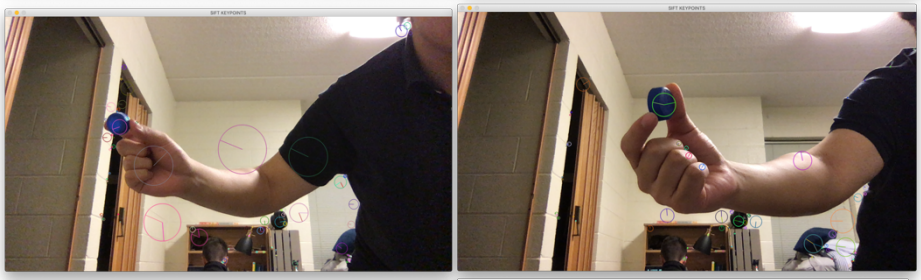
There are lots of parameters for tuning in SIFT, but after some experiment I didn't find significant changes on feature detections. (Maybe the reason is that my focus here is simply a well-detected and single featured bottle cap.) So the only thing I changed from default parameters is "nfeatures = 50" to hide minor features.

First, let's look at how rotations, translations and light changes affect SIFT descriptor.

Original

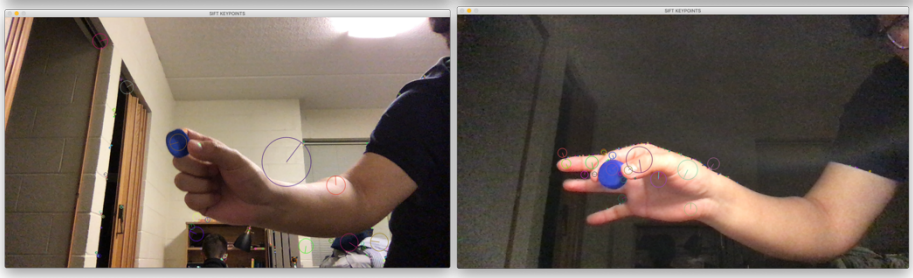
Trans

Trans



Rotate

Rotate



Dark

From the results above, we have proven that SIFT is invariant with respect to translation, rotation and light changes. On every picture above, the blue bottle cap is identified as a feature.

Now it's time to check SIFT's tolerance and limitations to scaling.

The pictures above are a sequence of bottle cap detection by SIFT with increasing distances. Every picture except the final one has a marked circle on the cap, indicating the available range of SIFT detection. After multiple testing, the limitation of scaling for this SIFT detector is roughly as small as that in picture 5.

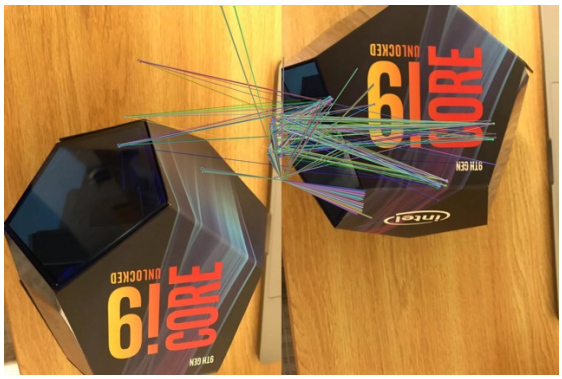
Key Points and Matching

In this section I will implement Harris Cormer Detection and SIFT key point detector to match the features between two images of the same object, taken from different directions and distances.

Here is a full Harris corner matching with knn matching algorithms. A full feature matching using corner detection is complex and hard to distinguish. So I used other objects that yields less matches. One can easily observe that some matches are inaccurate.

Deep Learning, Harris

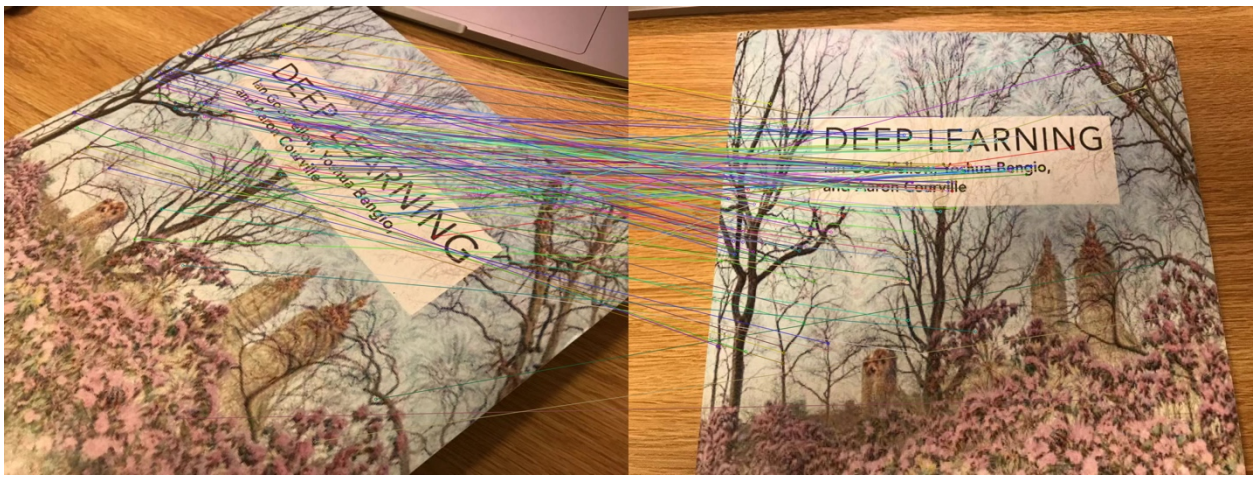
CPU box, Harris



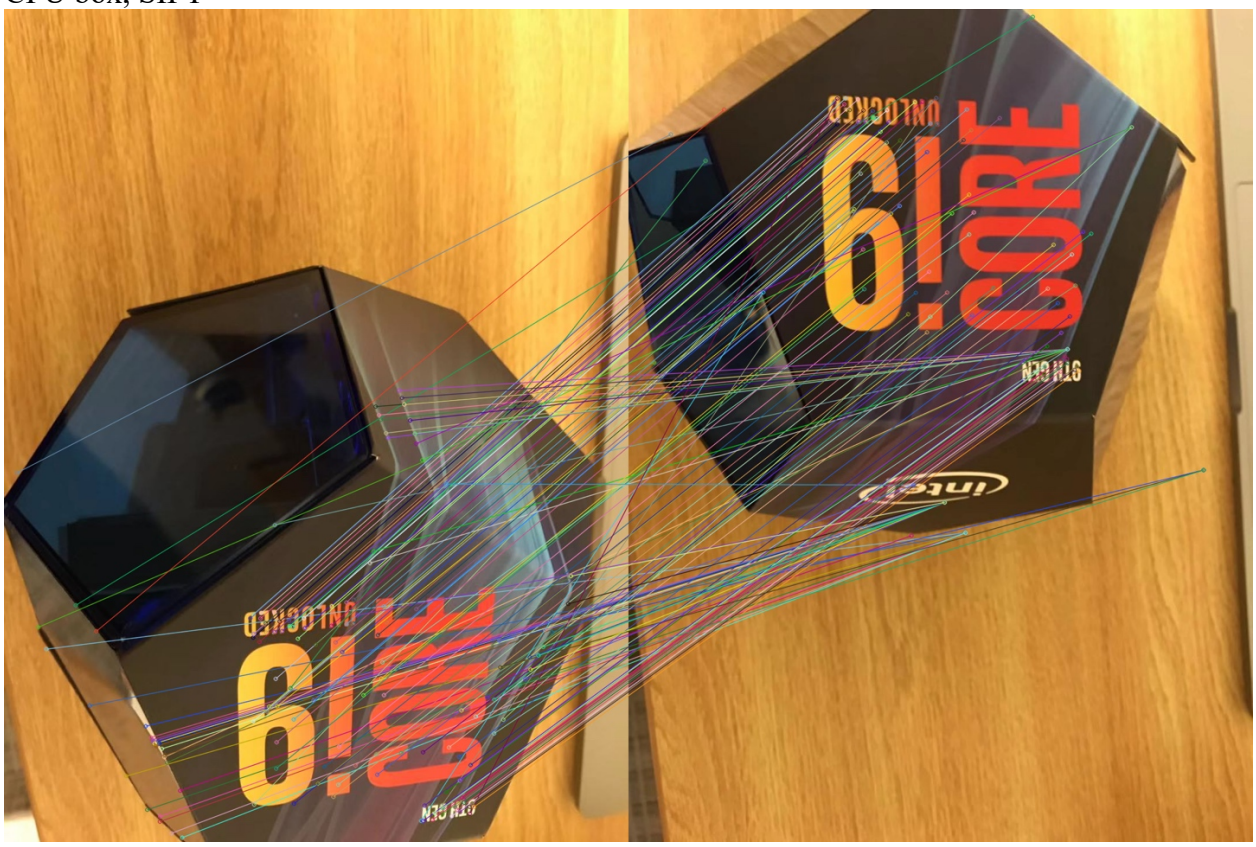
Chair, Harris

The performance gets better when I'm using SIFT. Here are some demonstrations. The examples I used here are: Deep Learning by In Goodfellow (with skewed transformation) , Intel i9 9900k CPU(with vertical translation), a chair(with mirrored transformation) and a door (with scaling)

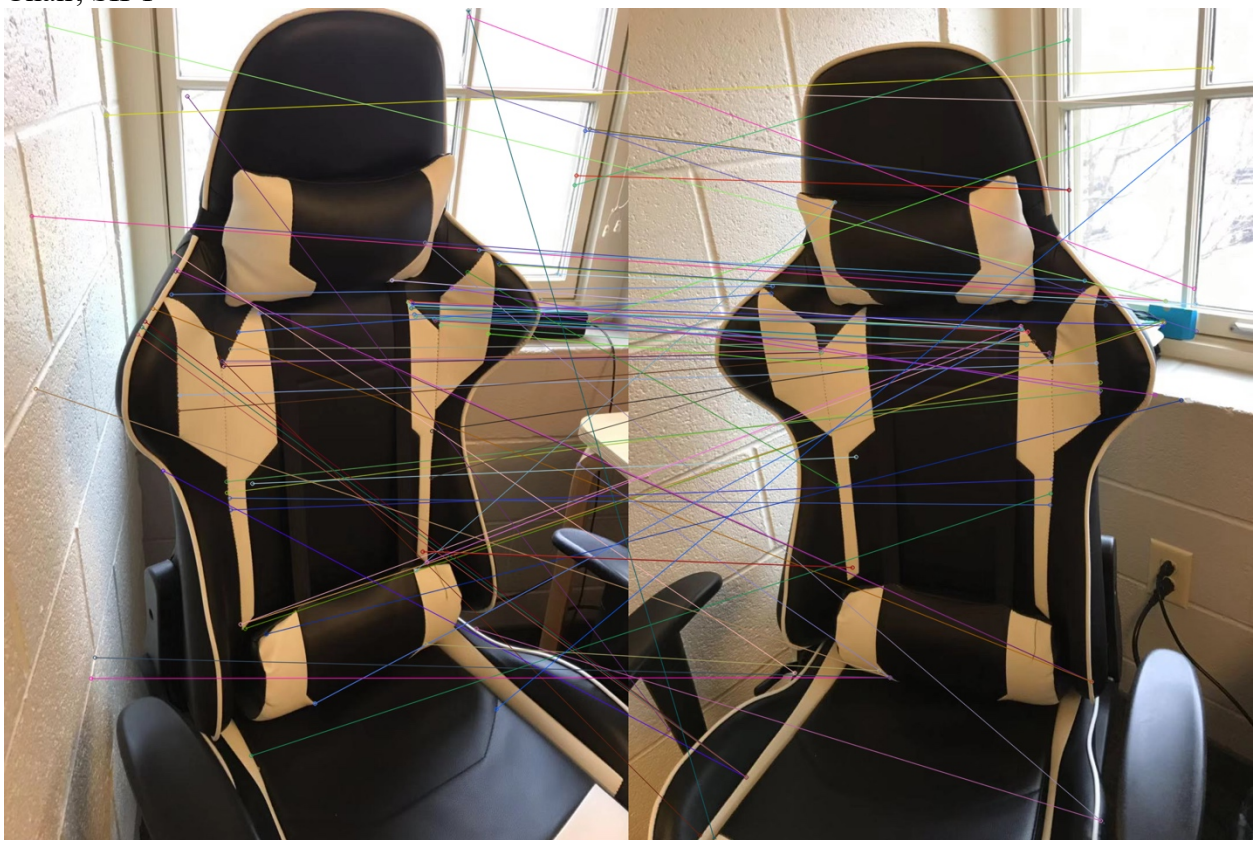
Deep Learning, SIFT


DEEP LEARNING
DEEP LEARNING

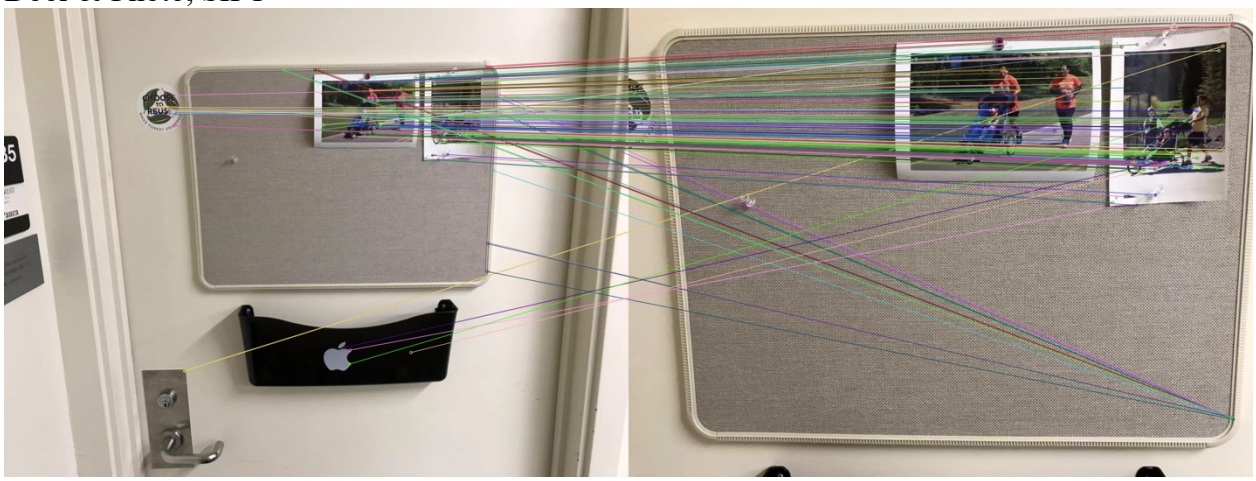
CPU box, SIFT


UNLOCKED
9TH GEN
intel
UNLOCKED
9TH GEN

Chair, SIFT



Door & Photo, SIFT



By this point, we can conclude that SIFT kypoint matching performs better than Harris corner matching in most cases. SIFT is generally a nice algorithm for feature matchings of two images after skewing, rotating, mirroring and scaling.