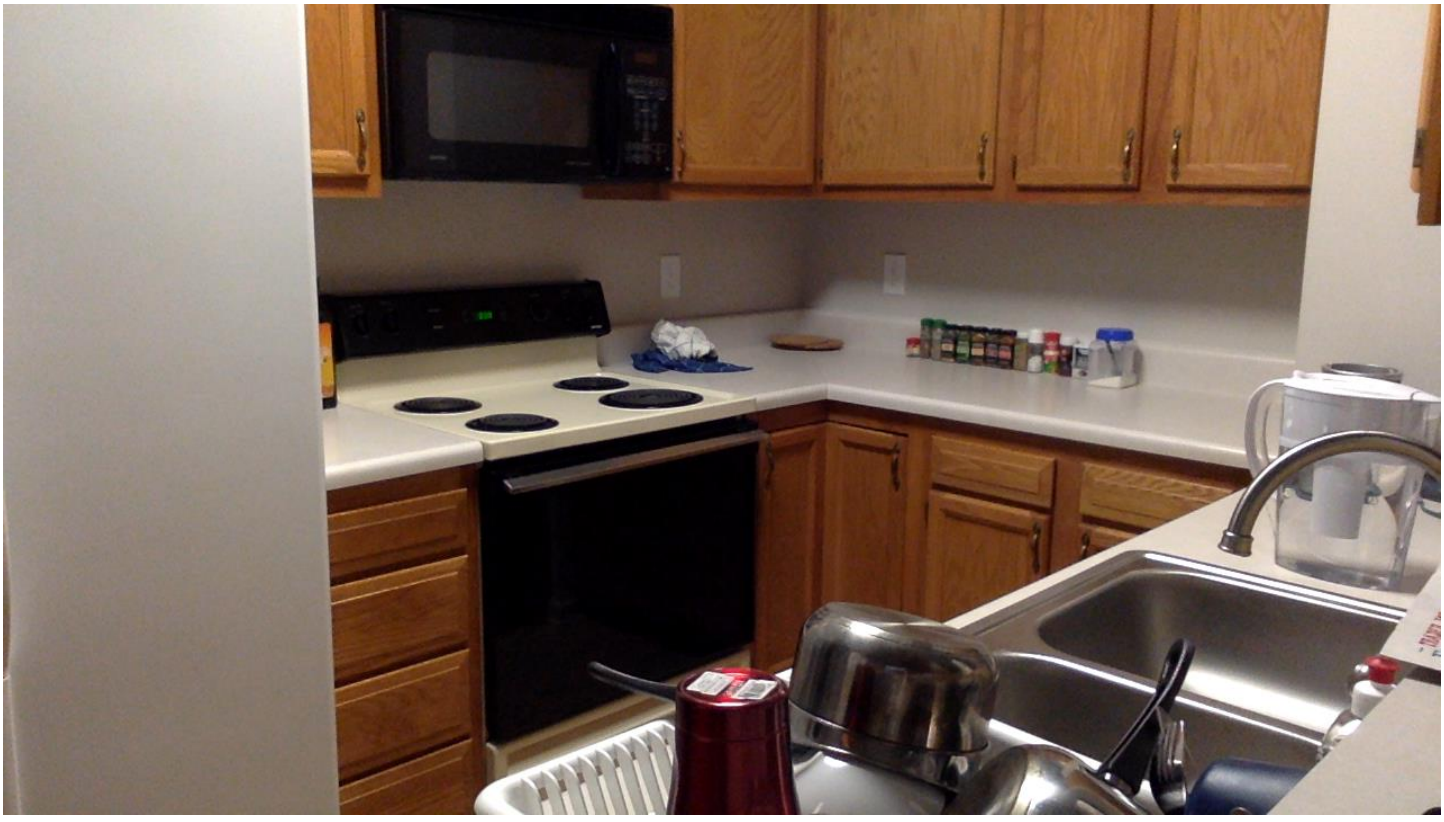
Thomas Tavolara
CS691: Project 2
"Welcome To My Condo"

1 Edge and Corners in Video

Edge detection

The first imaging scenario was of my kitchen under well-lit conditions. The low and high thresholds for Canny edge detection were 50 and 70, respectively, and the sigma value for the initially blurring was default ($\sqrt{2}$). Generally, under these well-lit conditions, the ranges of these thresholds were relatively lax - plus or minus 10 for both produced visually similar results. Edges were generated from the grayscale conversion of the color images captured by my laptop's web camera. Overall, the edge detection is fairly good. The edges of the refrigerator, cabinets, drawers, handles, microwave, stove, stovetop, oven, countertops, sink, and various kitchen items were clear and distinct. Some missing edges included the edge where the countertop meets the back paneling and some of the microwave window and buttons. There were spurious edges in the flat part of the cabinets and drawers due to their wood grain finish. There were also incorrect edges on the pan in the foreground due to its reflectance. Comparing frame to frame, the edge detection is stable for strong edges, relatively stable for weaker edges, and somewhat spurious for low frequency areas. As discussed in the previous project, most of these spurious edges can generally be dealt with by increasing the magnitude of the initial blurring, though that is not shown in this report. The results are shown in the following 5 images.



Original scene

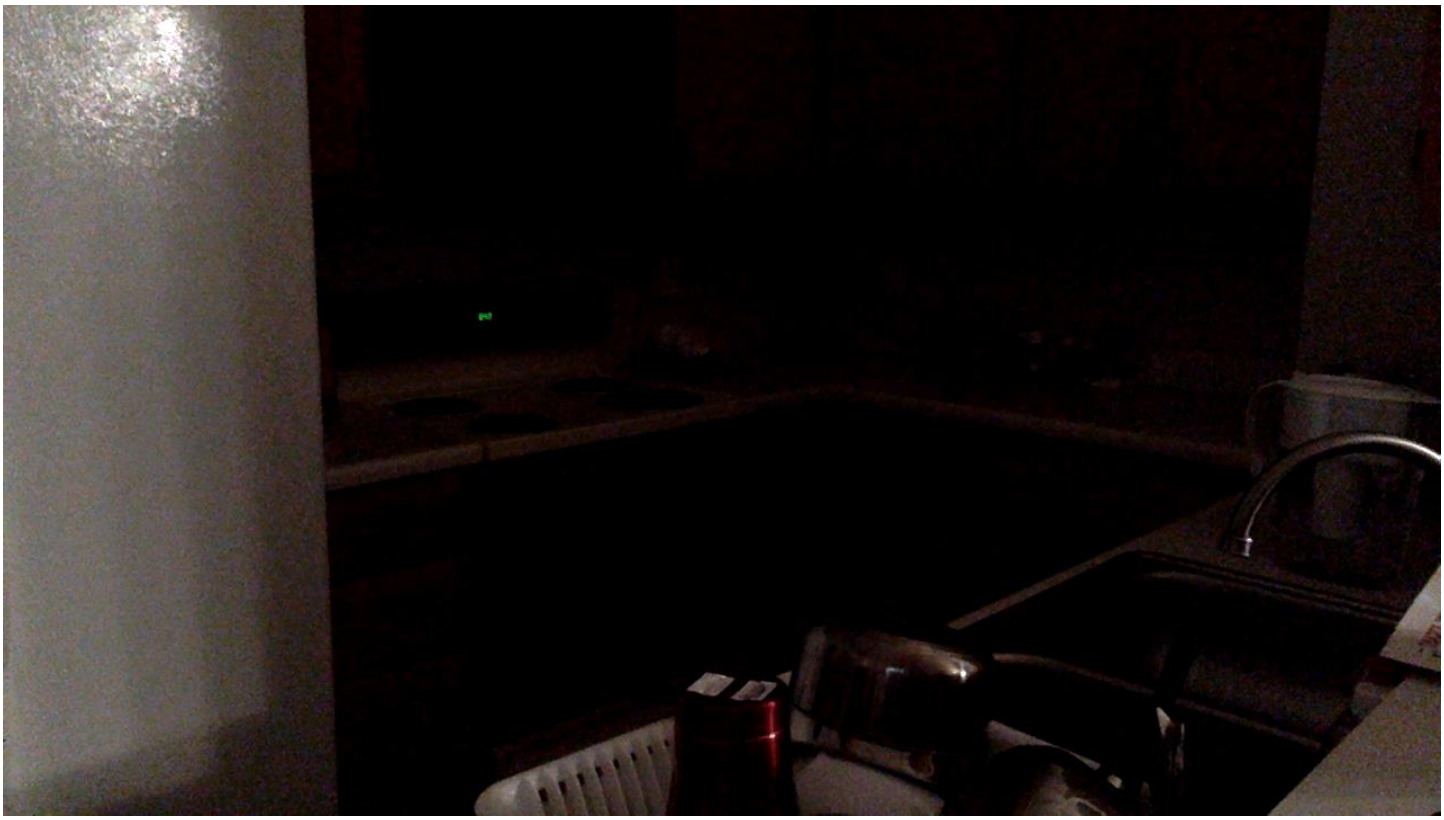
Frame 1

Frame 2

Frame 3

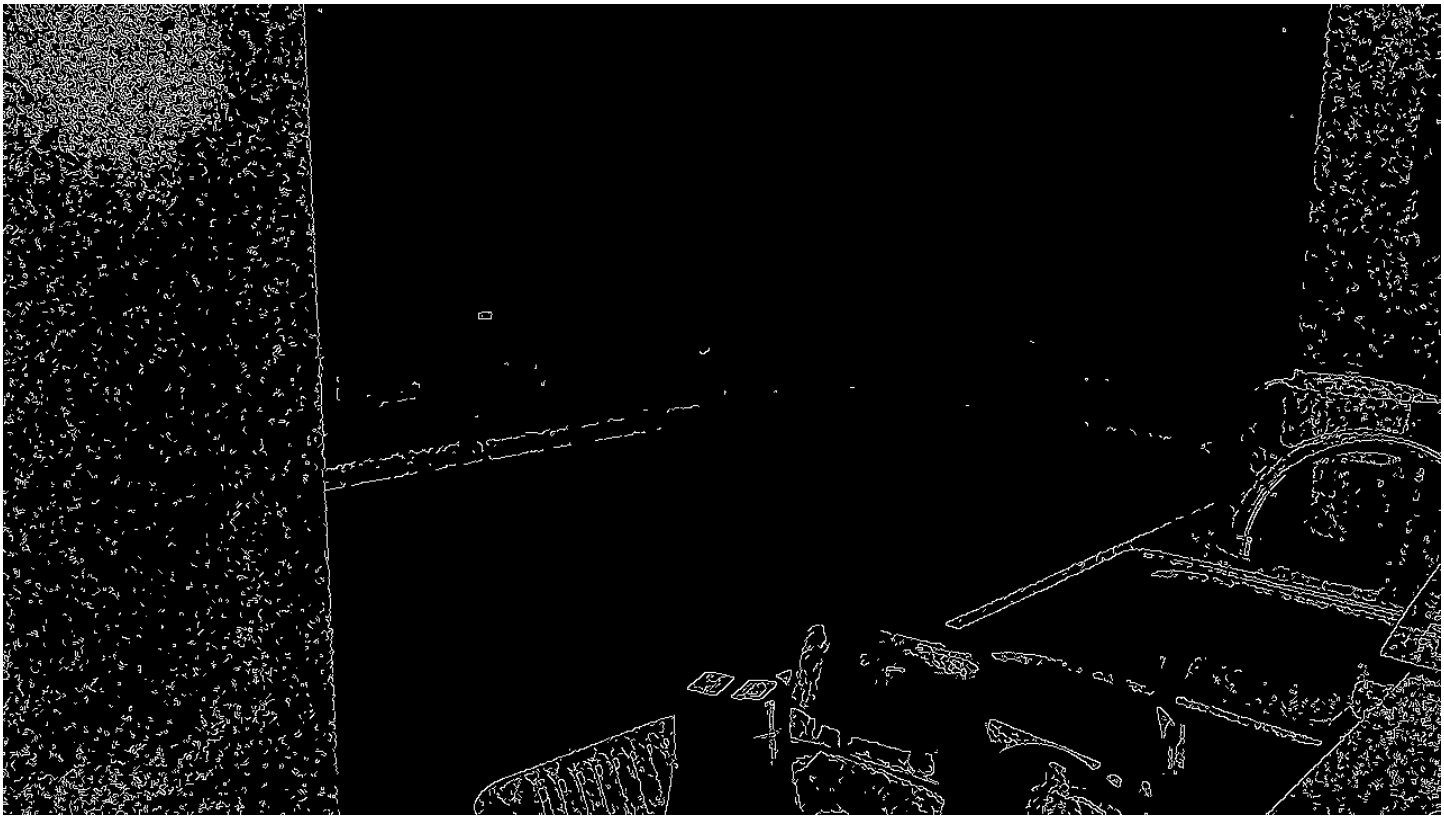
Frame4

The next imaging scenario involved the same exact kitchen scene except this time in low lighting conditions. For the sake of comparison, the low and high thresholds were kept the same (50 and 70) as well as sigma. Using these same parameters, many of the edges disappear with only the sink, refrigerator, and some countertop remaining. There was also considerable noise in the refrigerator (left), wall (top right), and countertop (bottom right) resulting in many incorrectly identified edges. Additionally, the light source was reflecting off the (textured) refrigerator which resulted in a blob of edges, “spaghetti”, on the top left. Edge detection is generally sensitive to low-light noise. Extra blurring can alleviate the problem but can easily result loss of true edges. Comparing frame to frame, these 'noisy' edges are very unstable and jump around constantly, while the sink/countertop and dish rack edges remain stable. Finally, the sensitivity of the edge detection to the parameter choices was much greater in this scenario compared to the previous (well-lit) scenario. Again, this was probably due to the noise cause by capturing low-light images. The results are shown in the following 5 images.



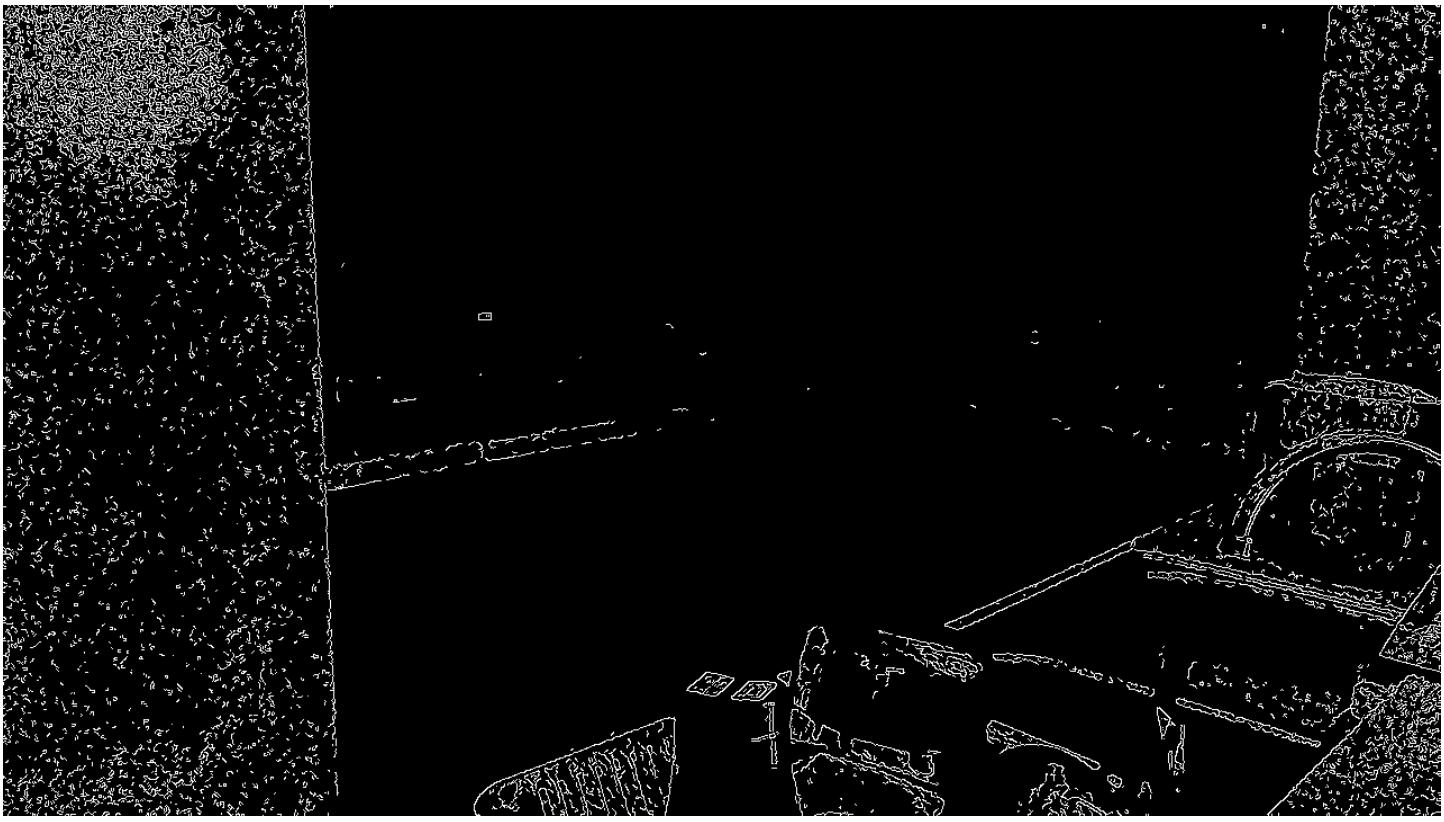
Original scene

Frame 1



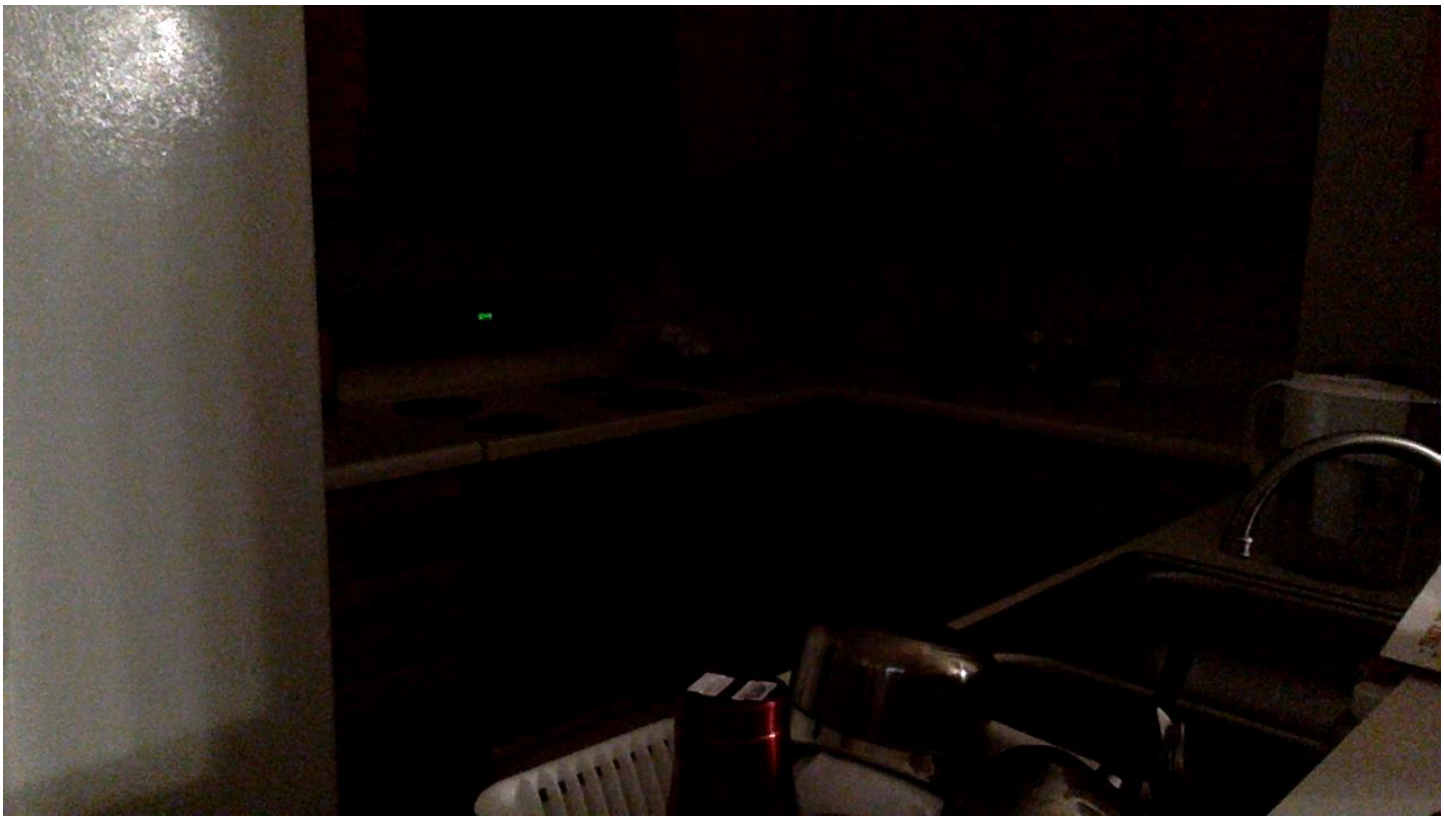
Frame 2

Frame 3

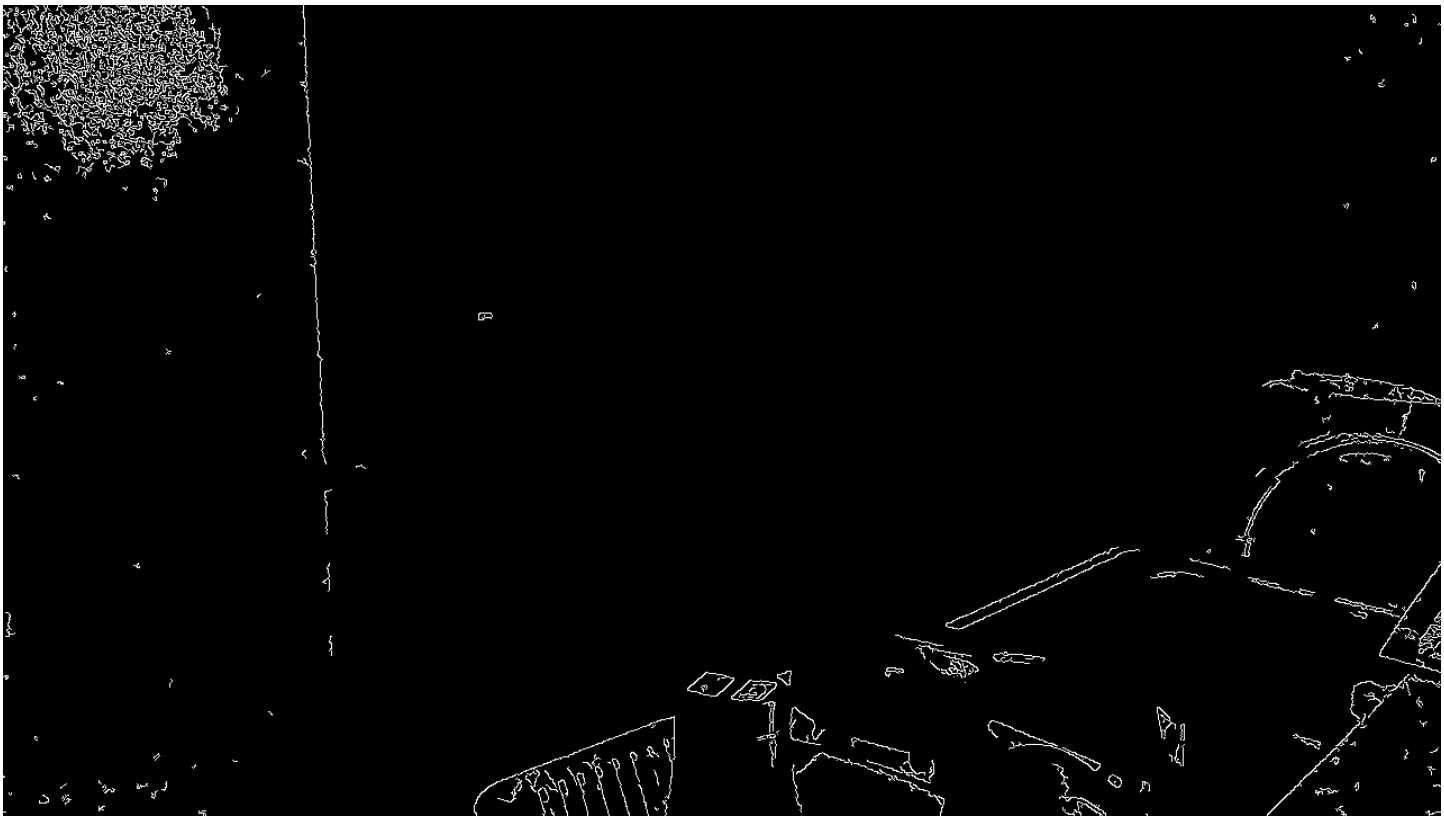


Frame 4

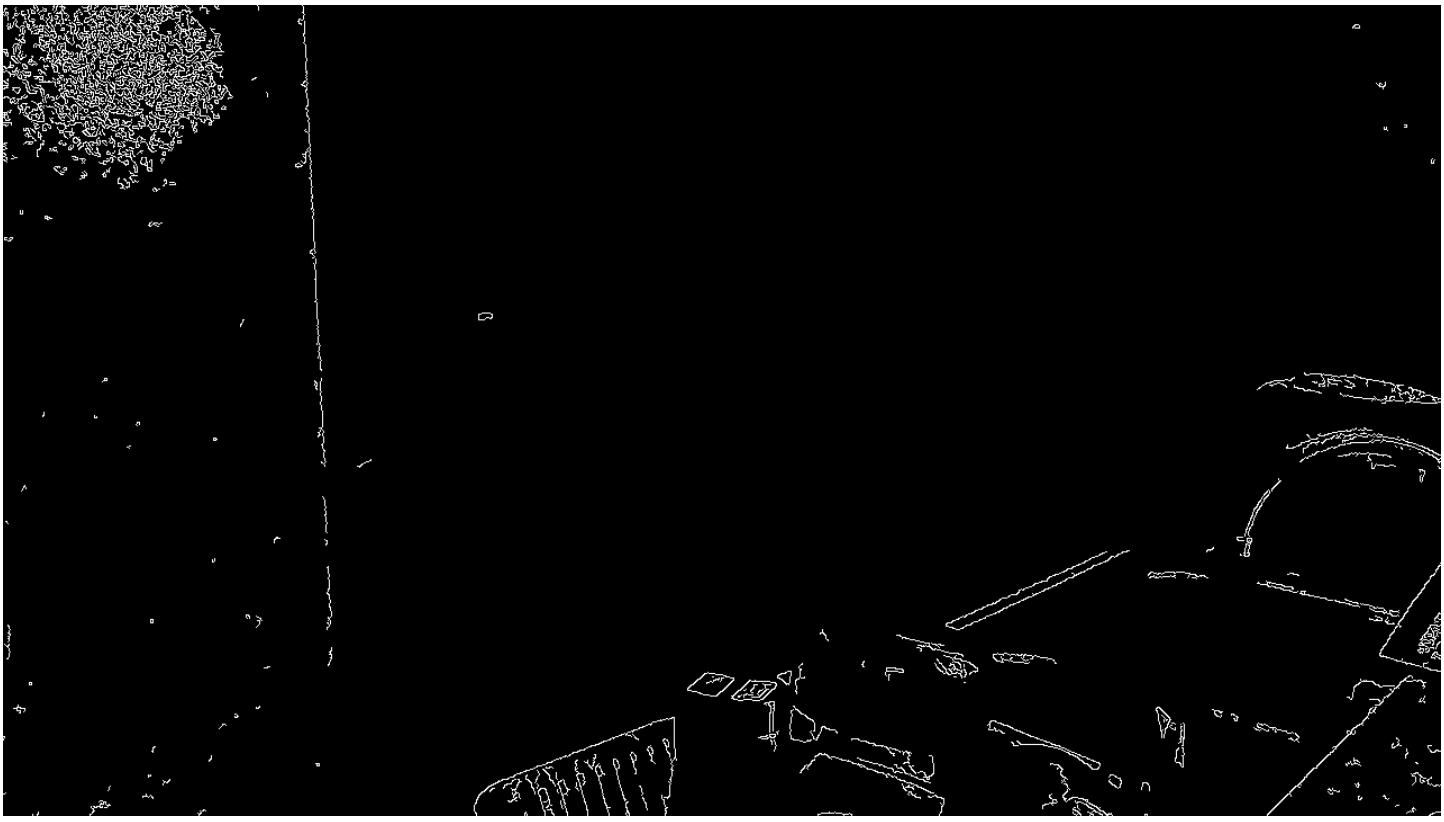
In the subsequent imaging scenario, the thresholds for low lighting conditions were adjusted as to remove 'jumpy' edges frame by frame. The final thresholds were 90 and 130, for low and high, respectively. Compared to the initial low light condition edge detection, some strong edges were lost, such as part of the refrigerator and wall (right). The cost of removing noisy, aberrant edges in low lighting conditions is apparent - true edges are also lost. Edge detection is low light conditions is difficult. The results are shown in the following 5 images.



Original scene

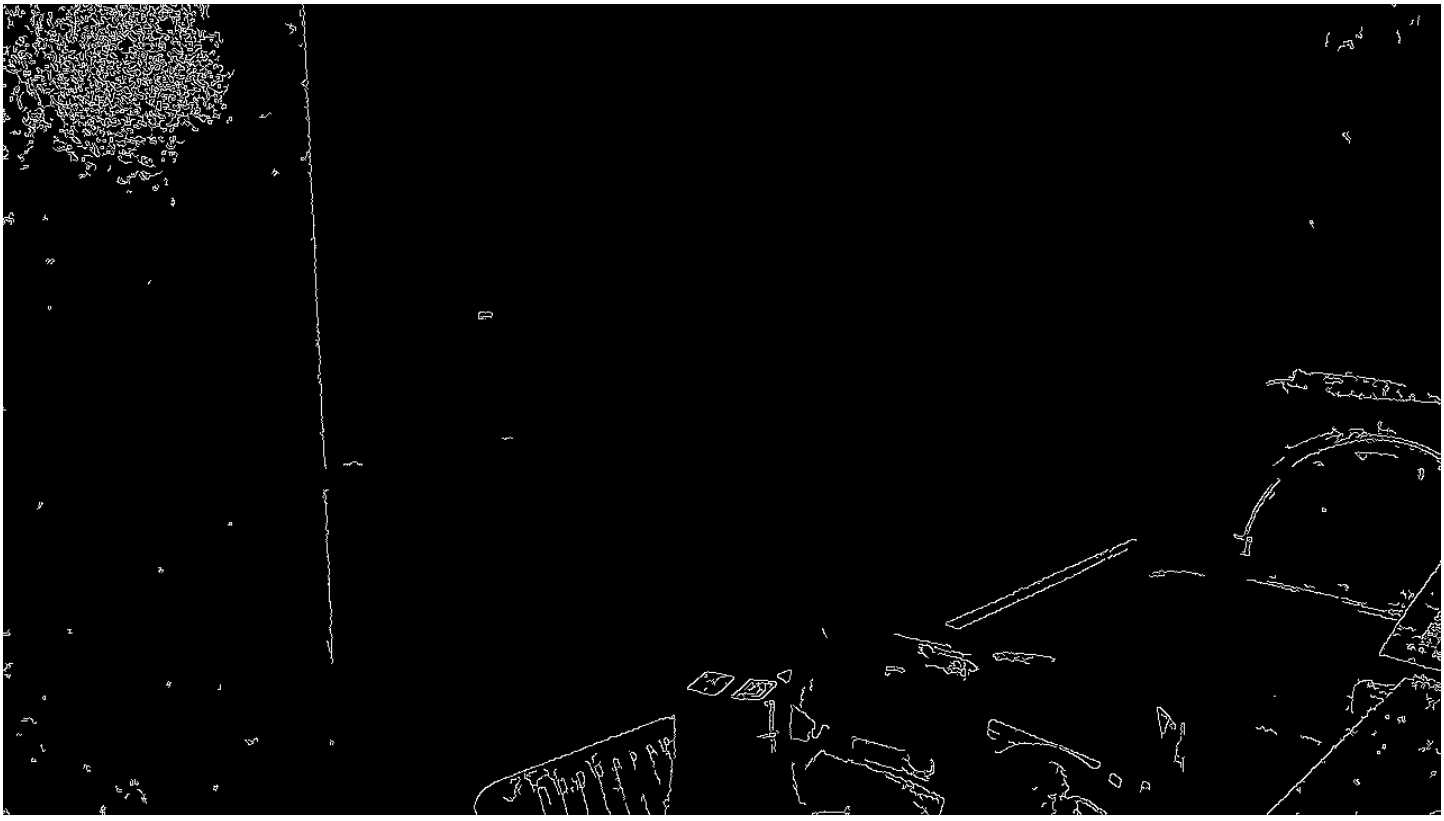


Frame 1



Frame2

Frame 3



Frame 4

In the final imaging scenario for edge detection, the thresholds are set to the same as in the first imaging scenario - 50 and 70. The frames capture edges as my hand moves through the shot. Here, I was trying to demonstrate an odd side effect I noticed while my web camera. Notice than these imaging scenarios are relatively static. That was on purpose. I noticed that if things were moving around in frame, not only would the edges of dynamic component change (because of different angles and lighting changes), but also edges of static elements would also change. This is counterintuitive. The light source in these images was a flood light. Moving elements in the foreground should not affect the lighting in the background. Additionally, after application of global thresholds, Canny edge detection works in local neighborhoods. Yet, the edges of static elements in the scene still change quite a lot. Compare this set of frames to the first set. Low frequency areas like sink, wall, cabinets, and refrigerator have no internal edges (i.e. incorrect ones), but as I pass my hand through the shot as in the images for this scenario, some edges just magically appear. I suspect that my web camera is doing some dynamic brightness adjustment/enhancement depending on the average brightness of the raw image. The results are shown in the following 5 images.

Original scene (hand not shown)

Frame 1

Frame 2

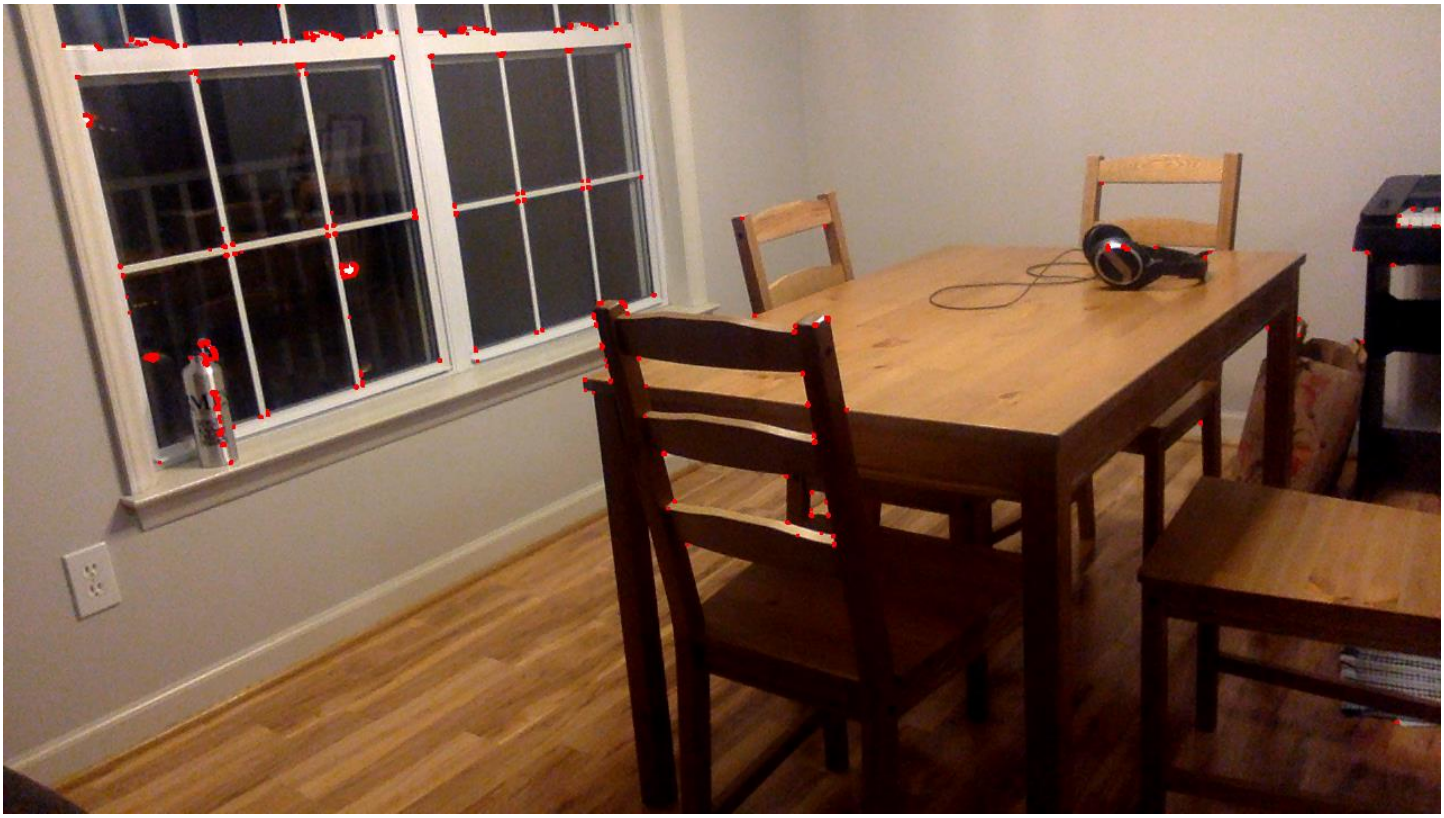
Frame 3

Frame 4

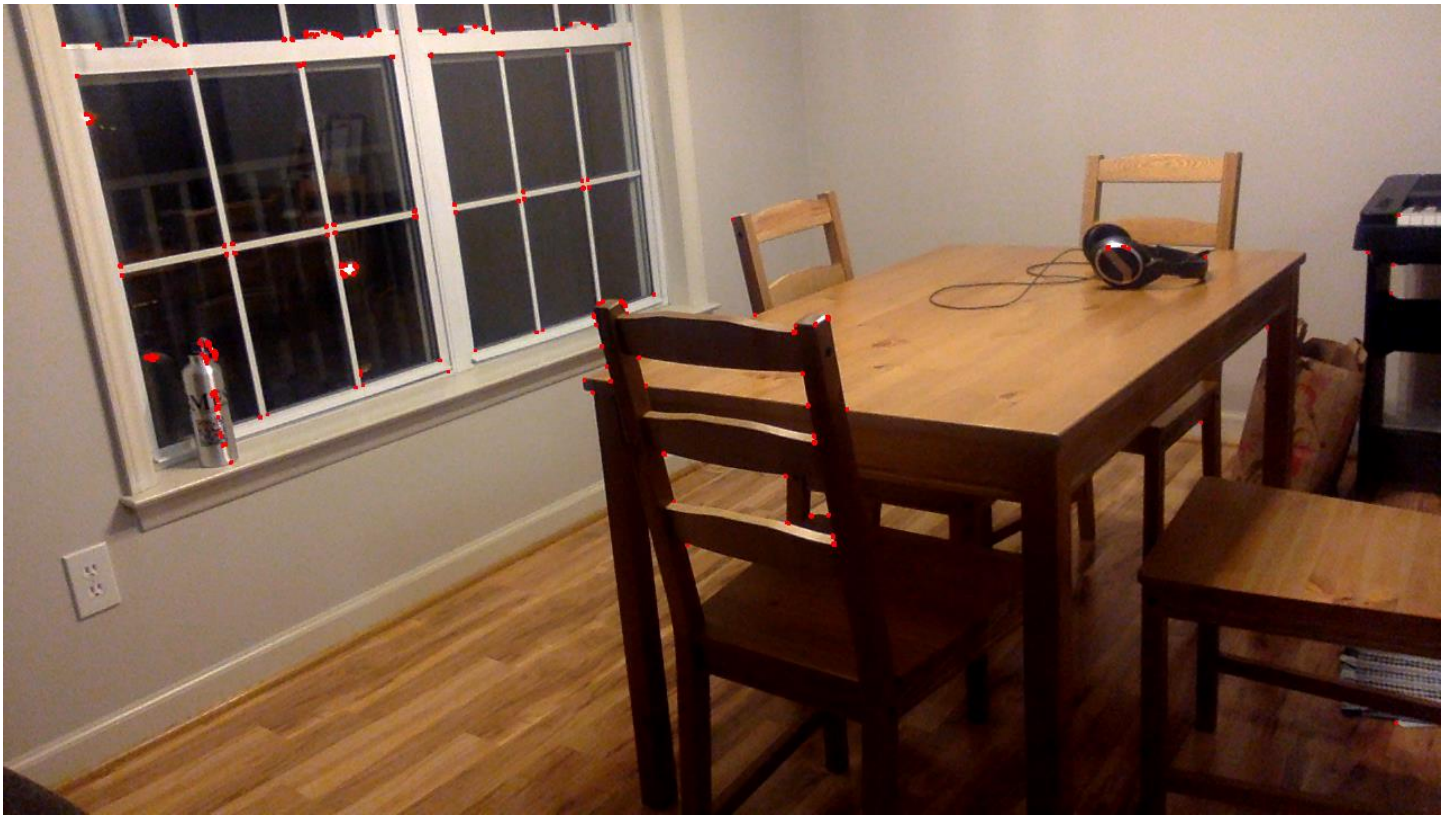
Corner Detection

The next set of imaging scenarios featured some windows and a dining set and were evaluated using Harris corner detection. Three different angles were used under the same lighting conditions. The parameters were as follows - blockSize=2, ksize=3, k=0.04, and threshold=0.05. Though no images are provided, generally speaking, adjusting ksize (in a range 3-21) and k (in a range 0-1) did not change the overall corner detection noticeably. In the previous case, perhaps since the edges were so long in this imaging scenario, the choice of Sobel filter did not matter. Perhaps if the there were smaller edges (in terms of length, not magnitude), this parameter would have had a more significant effect. k is empirically determined (usually within a range of 0.04 and 0.06), so I'm still a bit confused why changing it had little to no effect. Changing the block size did have a noticeable effect. Namely, corners with shorter edges disappeared (such as the piano keys or top of the chairs), whereas the corners of the windows did not change at all. This is expected, as this parameter changes the size of the neighborhood considered when examining potential corner points. Changing the threshold for lambda comparison also had noticeable effect, especially on the bottle in the image. As this threshold was increased, the detection of corners of the letters on the bottle (on the window sill) disappeared. The results are shown in the following images.

Comparing different frames from the same angle, the corner detection was robust, pretty much detecting the same exact corners. This can especially be seen in the window grilles - all four corners of the two center crosses are detected though each frame and all angles. And though not the primary subject of the image, the corners of the piano keys were also consistently detected in all frames across all angles. The chairs are where the corner detection fails. In the first angle (from the left), there were no corners detected on the far chairs (though there are corners). Corners were detected on the near chair. From the centered angle, these corners were not detected in addition to corners detected on the far and near chairs. The angle from the right failed to be consistent with either of these corner detections. These failures could be due to too low of a lambda magnitude threshold, though many incorrect corners were identified when this was set too low. It could also have something to do with the the greyscale counter part to these images. It could be that after grayscale conversion, the contrast between the chairs/table and wall is relatively small. This is supported by the observation that corner detect fails when the chairs are posed in from of the wall. Still, it is clear that choosing a proper threshold is critical for Harris corner detection.

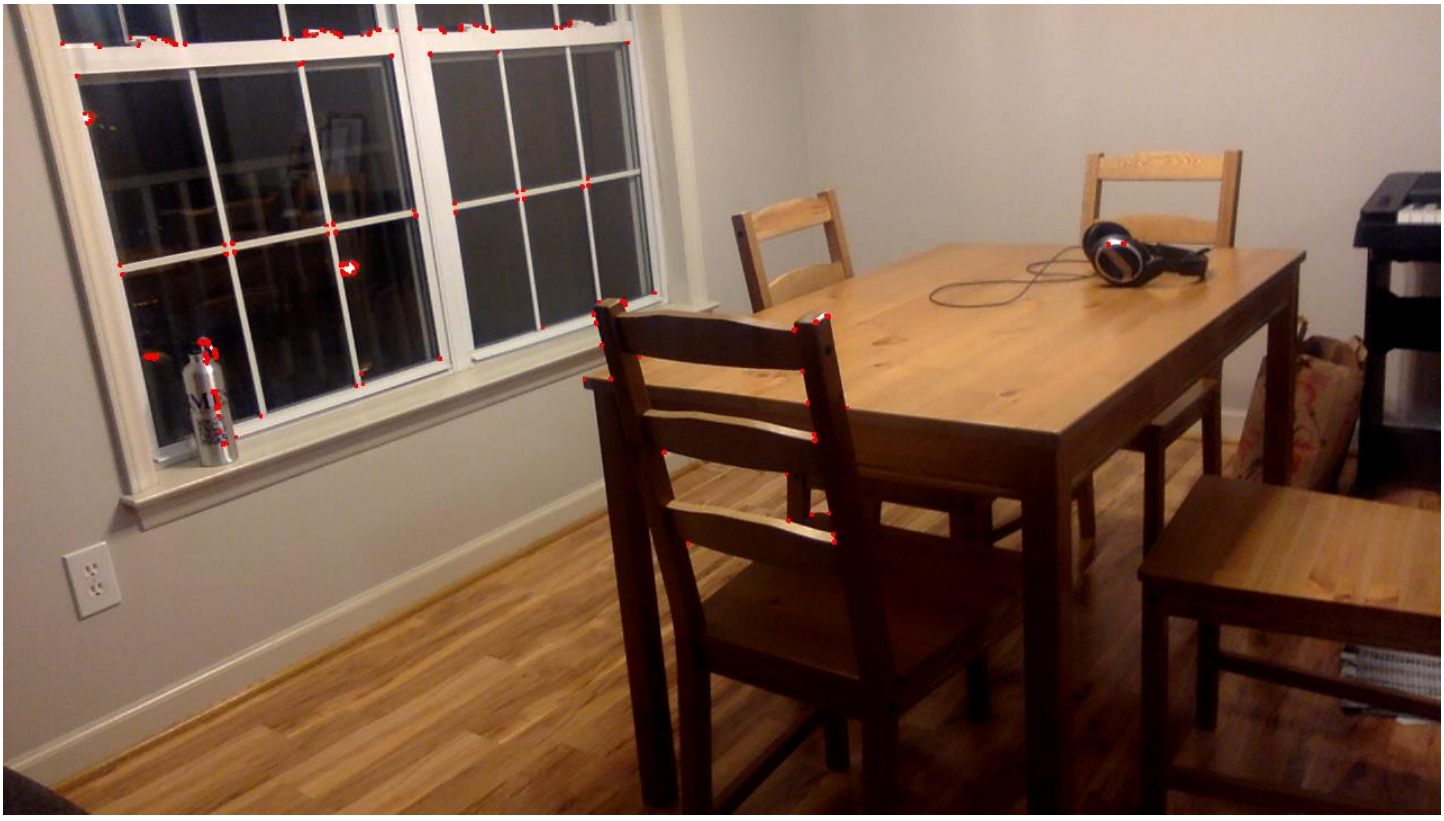


Frame 1, left side

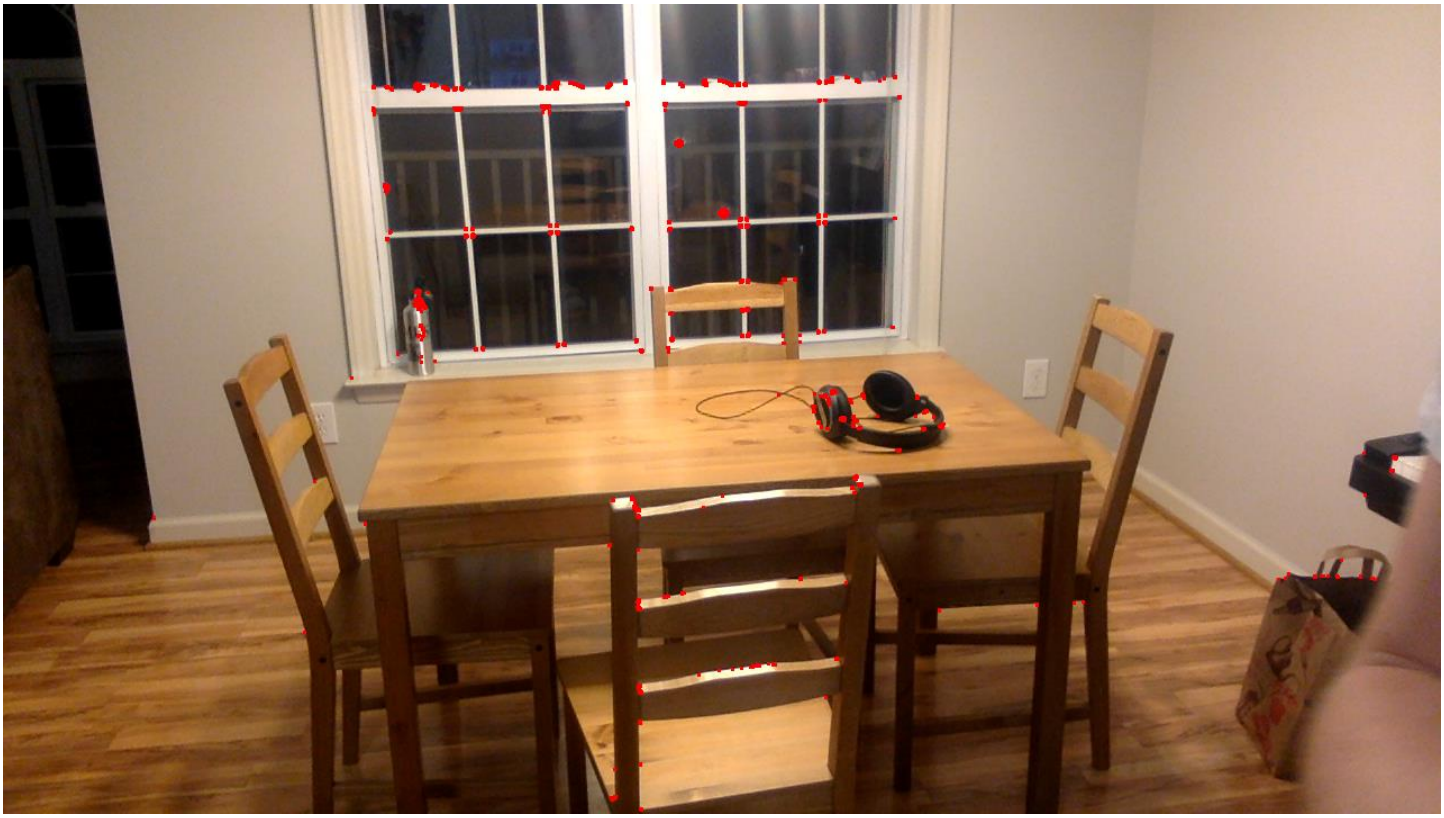


Frame 2, left side

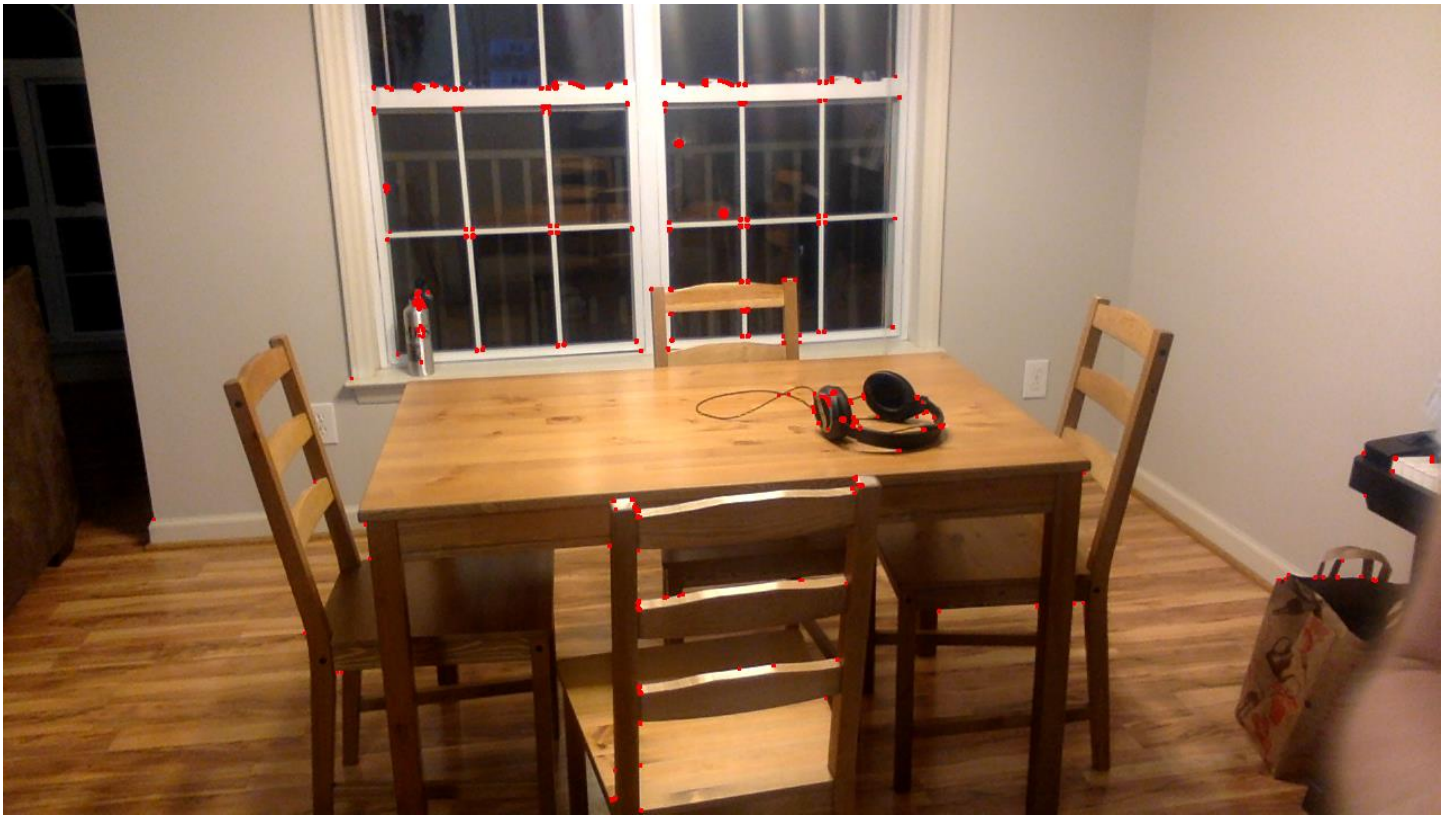
Frame 3, left side



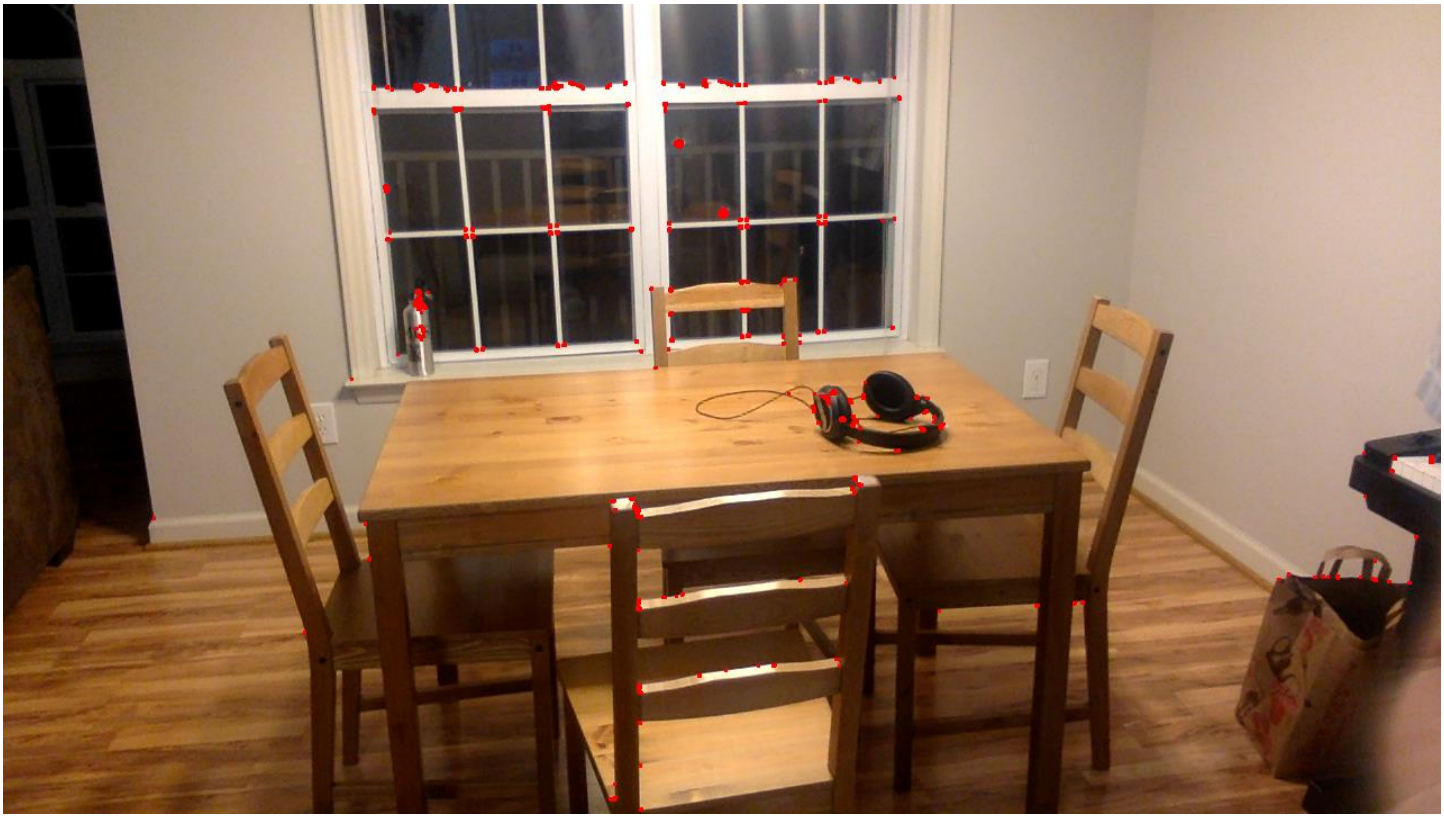
Frame 4, left side



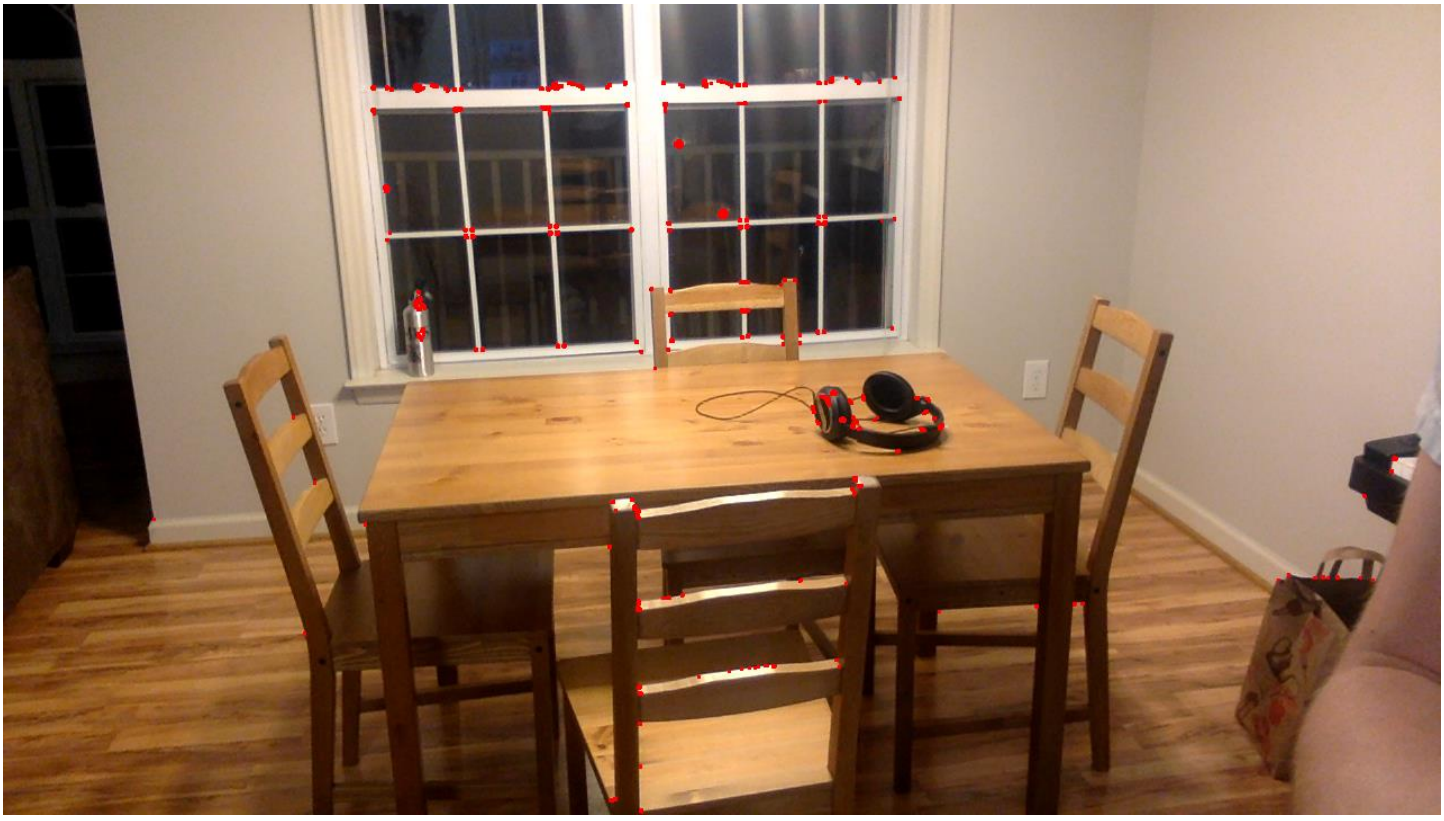
Frame 1, center



Frame 2, center



Frame 3, center



Frame 4, center

Frame 1, right side

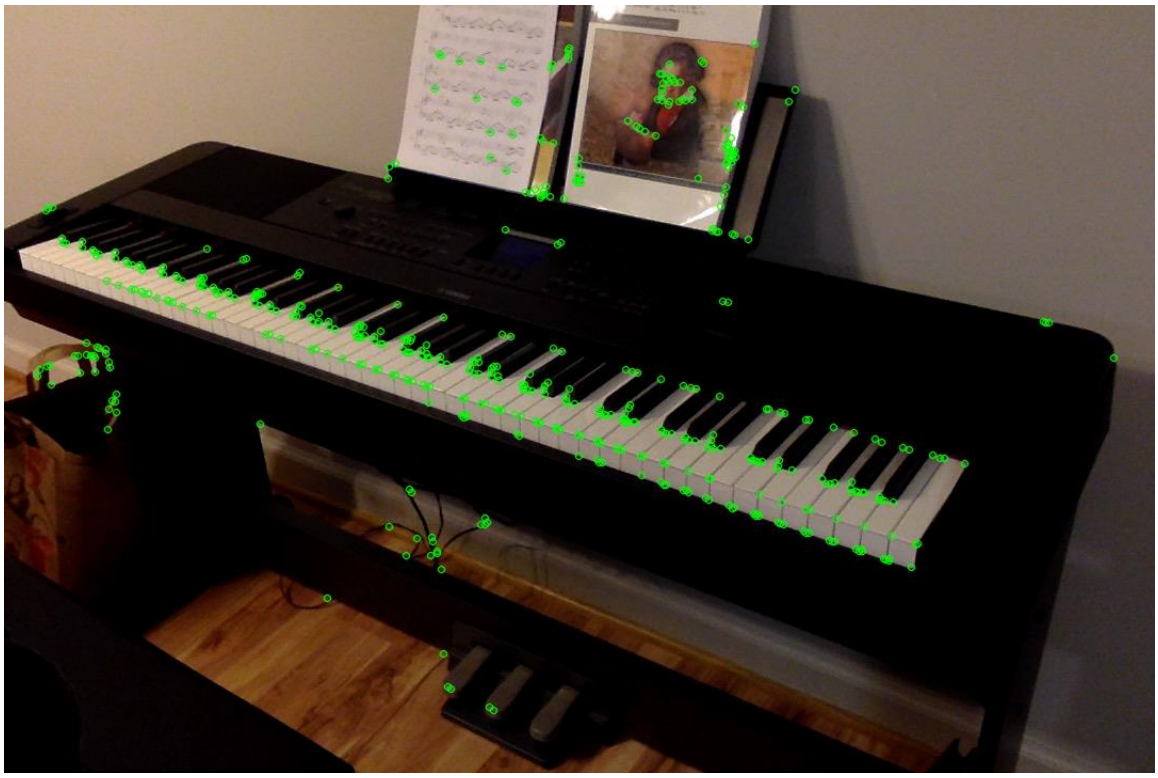
Frame 2, right side

Frame 3, right side

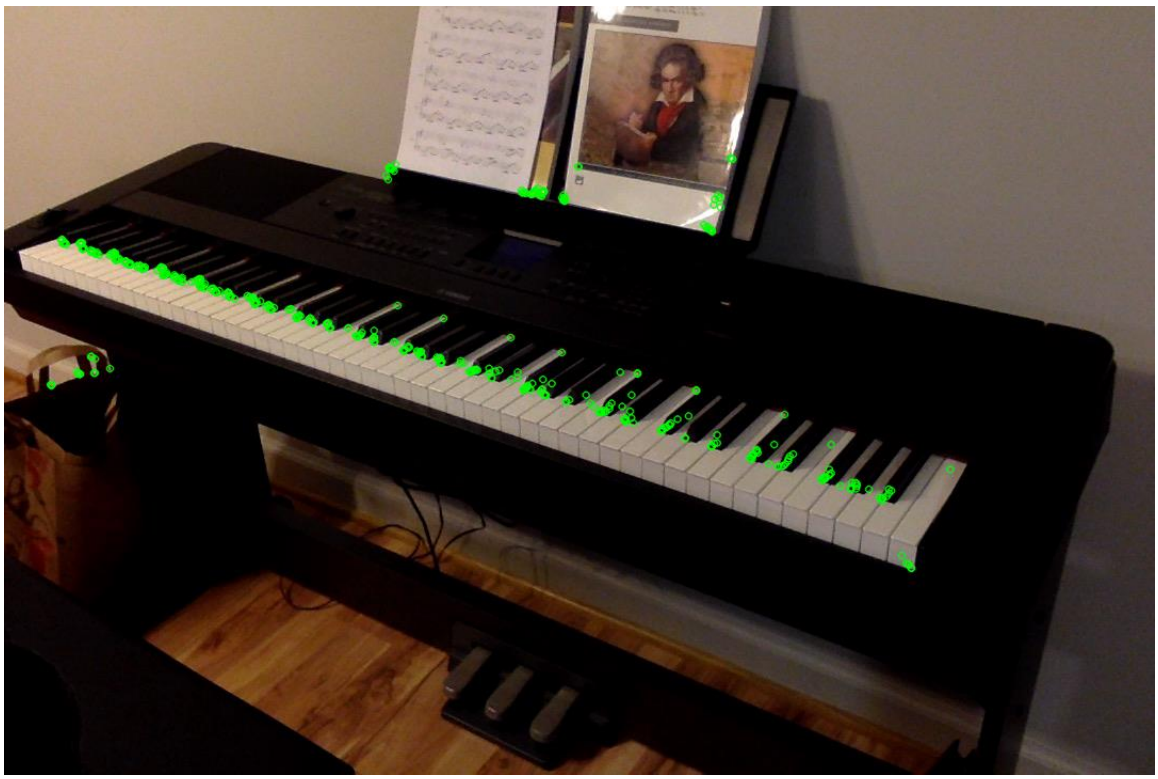
Frame 4, right side

2 SIFT Descriptors and Scaling

The following set of images were taken in order to get an idea of the 'limits' of SIFT parameters – namely, number of features, number of levels, and edge thresholds. I chose a relatively difficult image – a piano – for its repetitive features (i.e. the keys) to see how well I could get SIFT to work. Spoiler – I didn't.



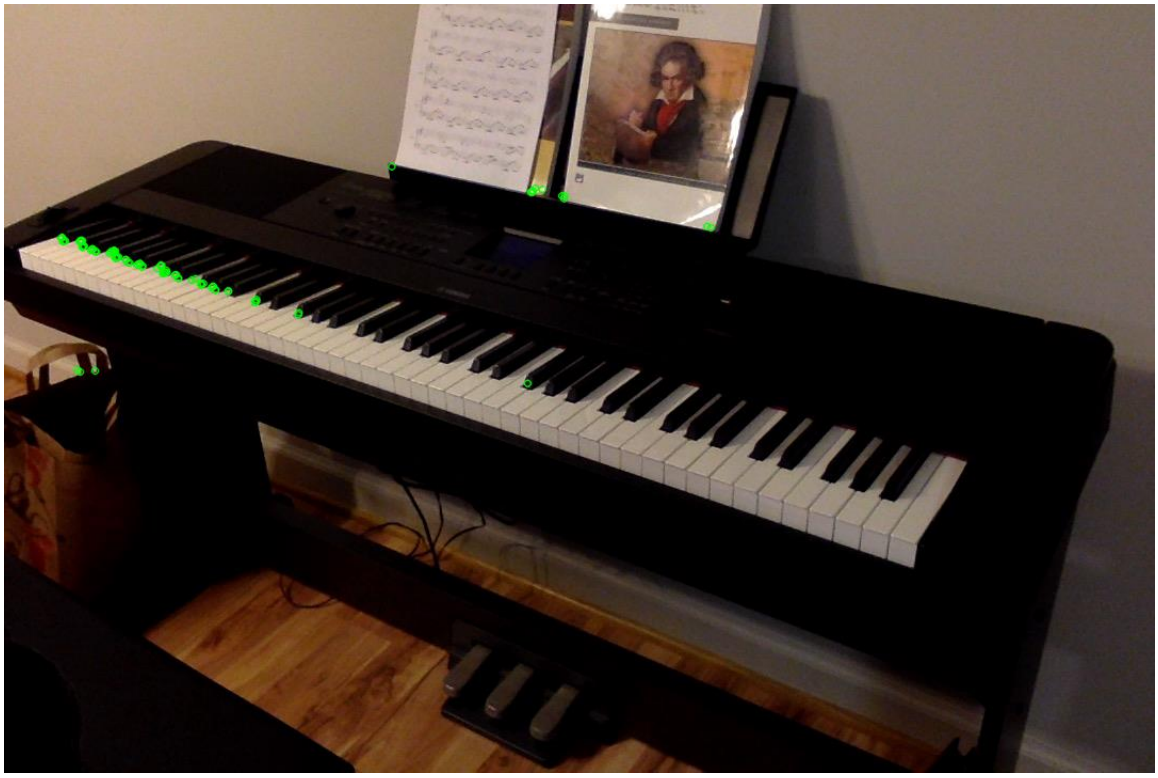
Piano, nfeatures=500, nlevels=1, edgeThreshold=31



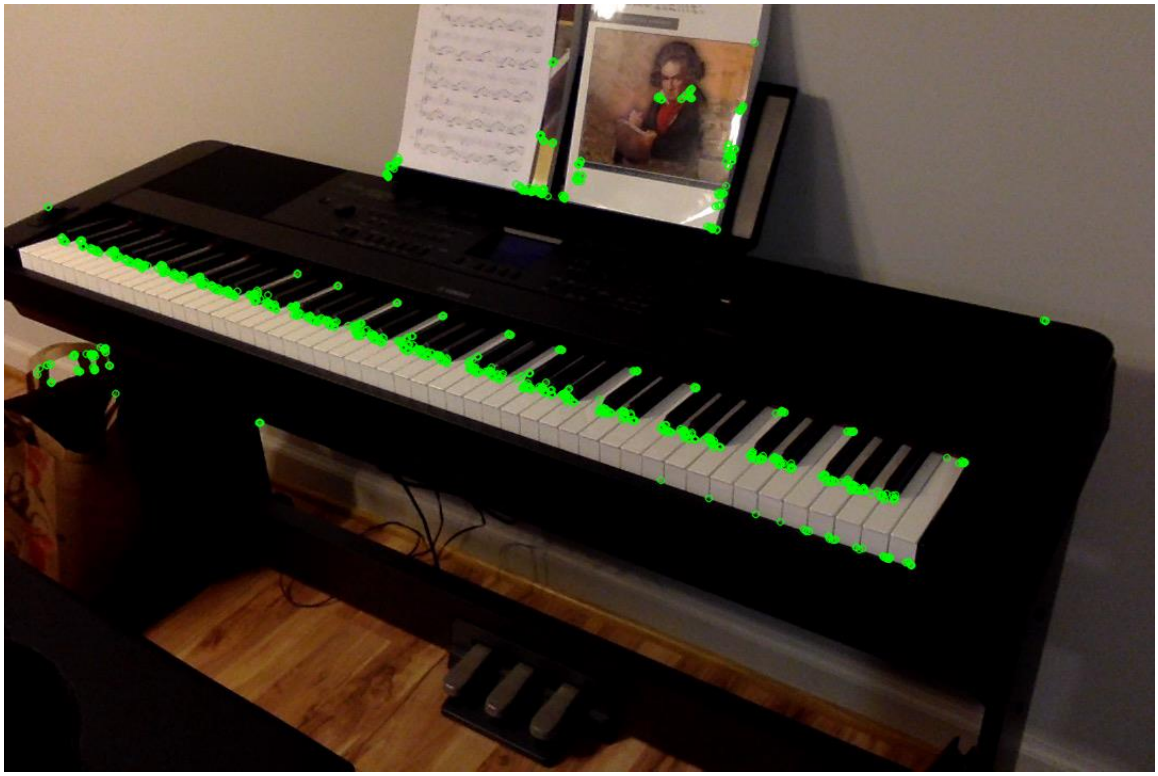
Piano, nfeatures=500, nlevels=20, edgeThreshold=31

Number of levels describes the number of scales that SIFT considers when finding features. The idea is that the features scale invariant (hence, **SI**FT). The first thing I noted when playing with the number of levels was that the fewer the number of levels, the more 'unique' the features became. Notice how with 1 level, the features identified are more spread out and tend to cluster less than with higher levels. I'm not sure why this is the case, but it might have something to do with the saliency of the features. If at a small scale, SIFT detects some features, then that feature may manifest as many smaller features at a larger scale, hence at these larger scales we see a bunch of features detected that 'cluster'. It also may just be some quirk about this particular subject (i.e. the piano). I don't think it's good that these features are clustering. I imagine it makes matching more difficult.

The next set of images looks at the limits of number of features. What I notice first (again) is this sort of clustering of features when number of features is high. I imagine this is redundant and may lead to matching errors. I'm also not sure of the quality of these features (in terms of distinguishability) given that these piano keys pretty much all look the same with the exception of lighting and angle relative to the camera. I'm not sure why SIFT prefers the piano keys near the left side of the keyboard when the number of features is limited.

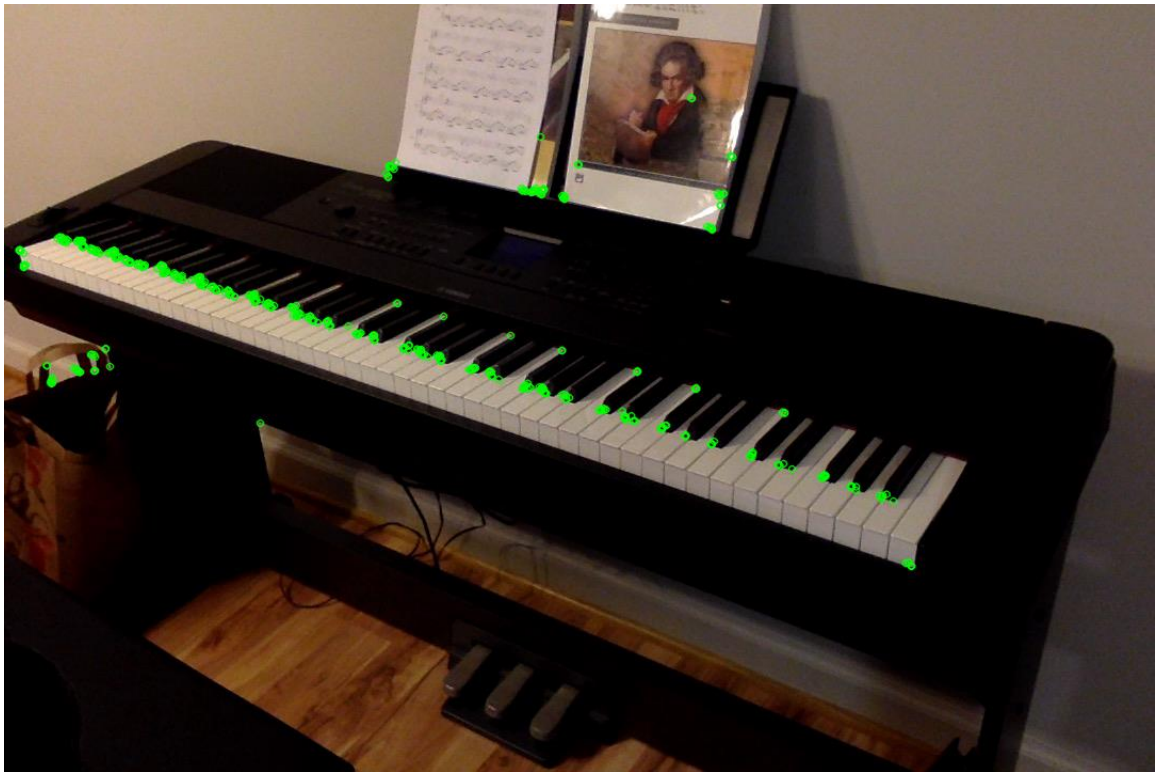


Piano, nfeatures=100, nlevels=8, edgeThreshold=31

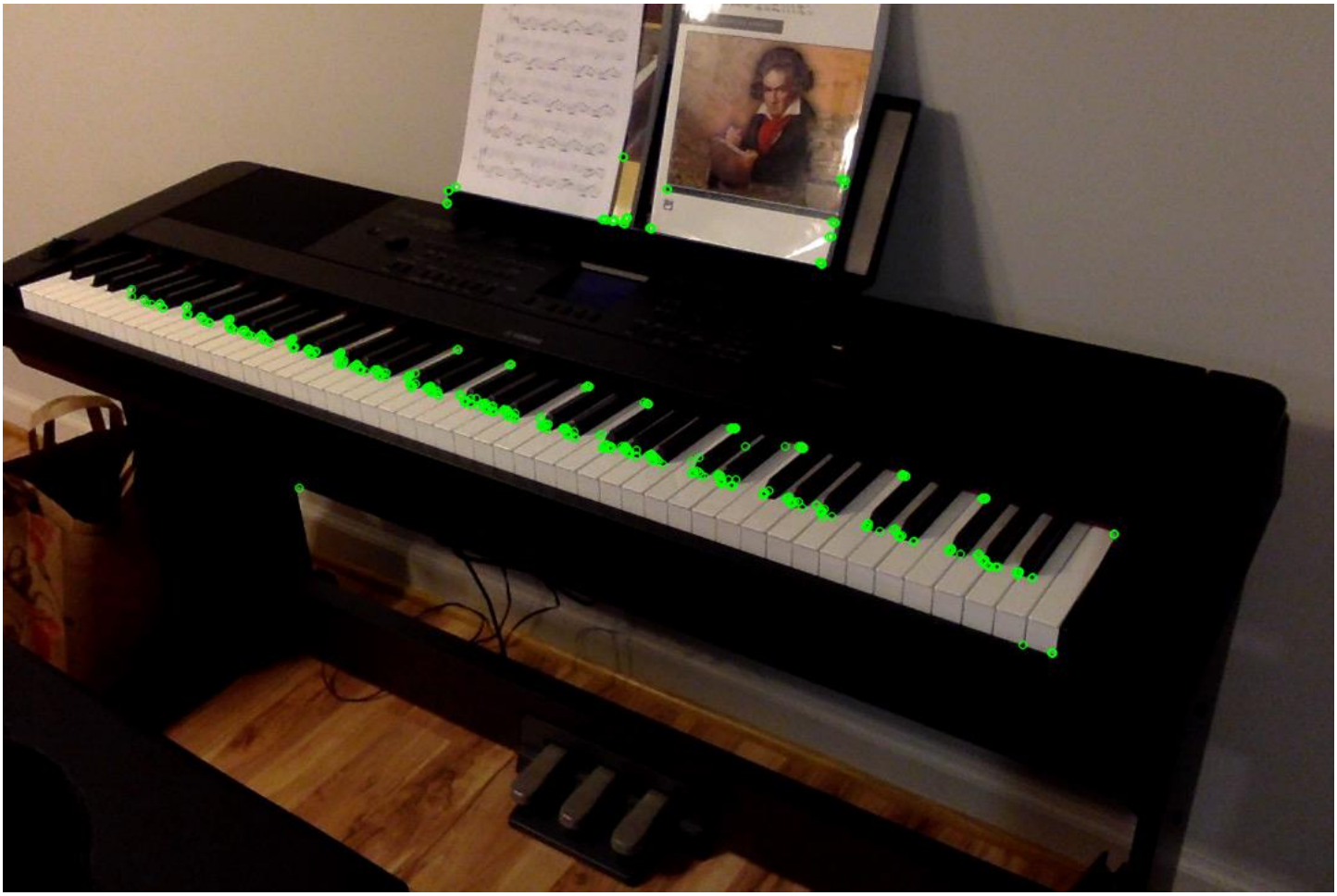


Piano, nfeatures=1000, nlevels=8, edgeThreshold=31

The last variable that I played with was the edge threshold. This controls the size of the border where features are not detected.



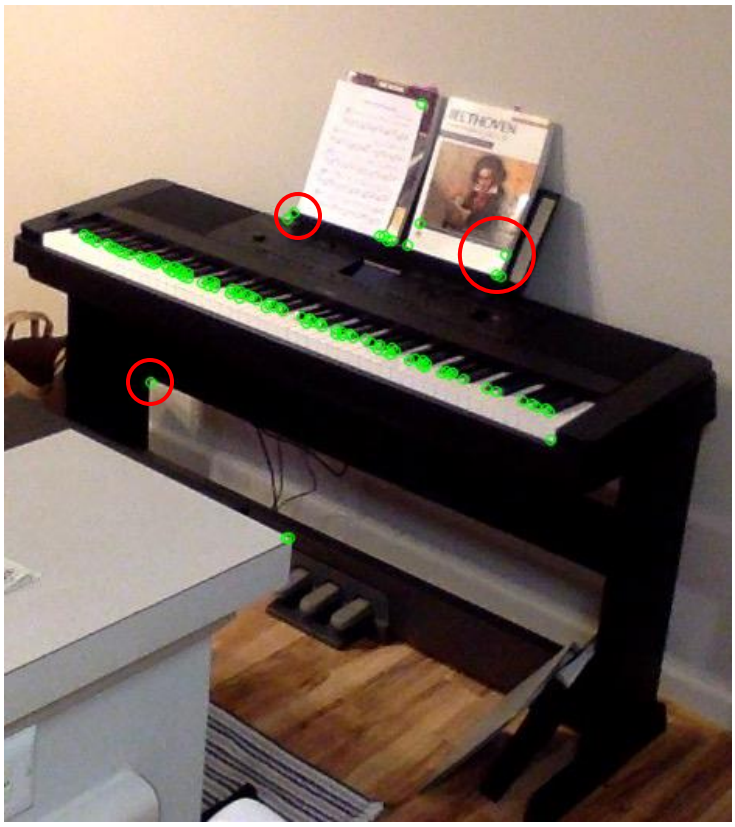
Piano, nfeatures=500, nlevels=8, edgeThreshold=1



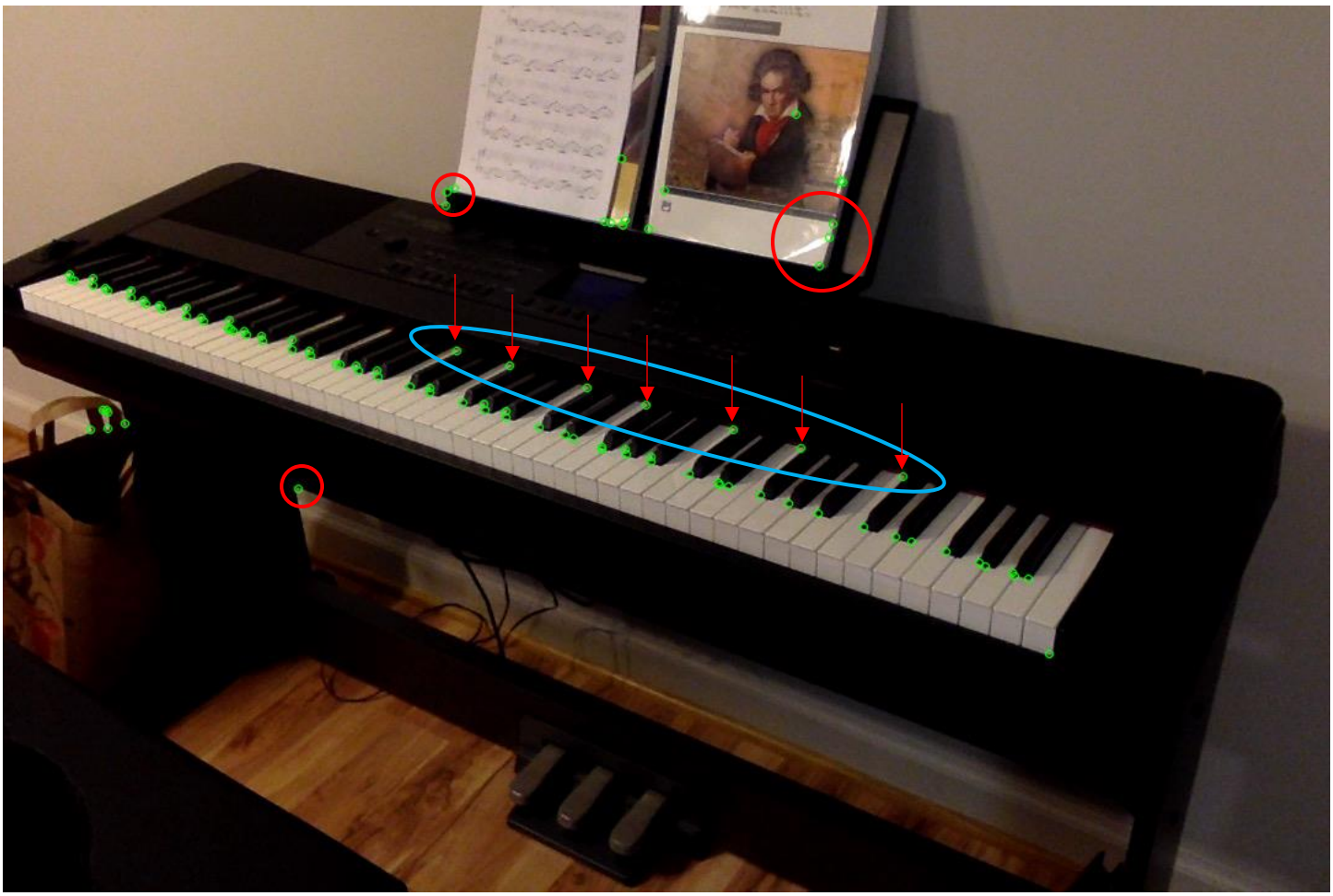
Piano, nfeatures=500, nlevels=8, edgeThreshold=100

The difference in these extremes seems to be a preference for the left or right side of the keyboard. I'm not sure what about adjust the edge threshold would result in this effect. The only difference I can see in the left and right side of the keyboard is key angle and possibly luminance or brightness. I'm also a bit surprise that it's not picking up more of the portrait of Beethoven.

In the next pair of images, I tried to visually tune these parameters on two images at differing scales (and slightly different angles). The final parameters values I ended up with were 99, 1, and 45 for number of features, number of levels, and edge threshold. There were a few common features detected, including certain corners of the music and album cover as well as the underside of the keyboard. There are circled in red. Piano keys were similarly detected, but the far ends of the key were not in the smaller scale (in blue). The limit of scale in this case is when the white space between the keys vanishes, as can be seen in the smaller scale.

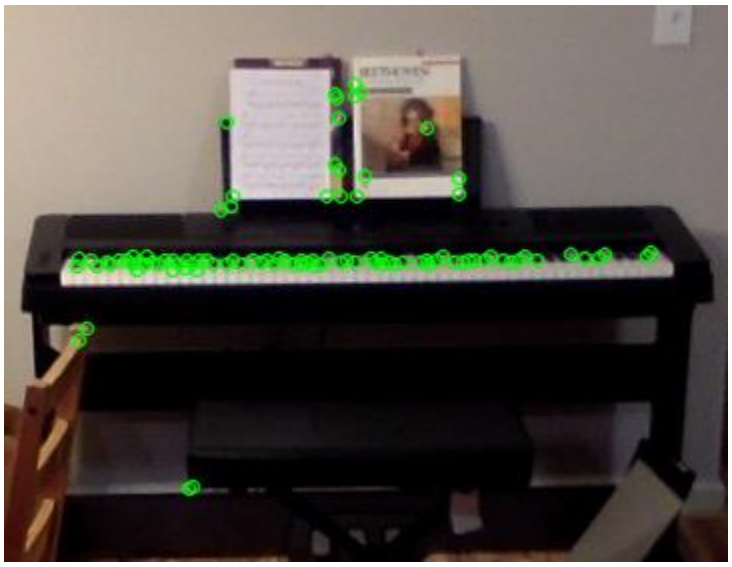


Piano at small scale

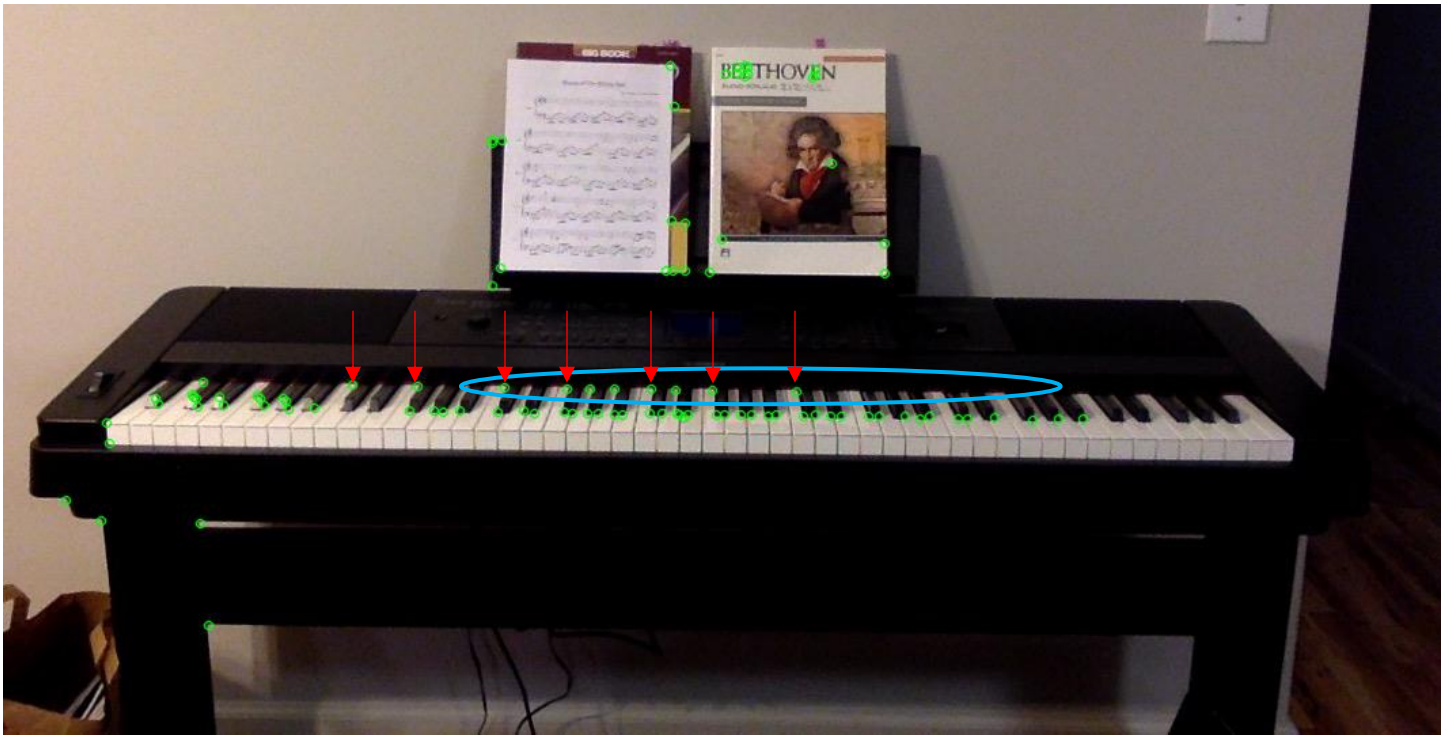


Piano at large scale

The next pair of images look at scale again, this time from a frontal view. The parameters remain the same as before – 99, 1, and 45 for number of features, number of levels, and edge threshold. At this particular angle and scale, letters on the album cover are picked up at the larger scale, which were not present in any other image. As before, corners of the album cover, its art, and the sheet music are detected by SIFT. Very interestingly, the far side of the black keys (circled in blue here and previously) are again detected. Even more so, even though the same keys are not detected in this manner, the same *pitch classes* are (C sharp and F sharp), which I must say is pretty cool. This is probably because both of these keys present the similar-looking corners (arrows pointing to them). Beethoven's color is also present in these images as well as the larger scale from the previous set.



Piano, front view, small scale



Piano, front view, large scale

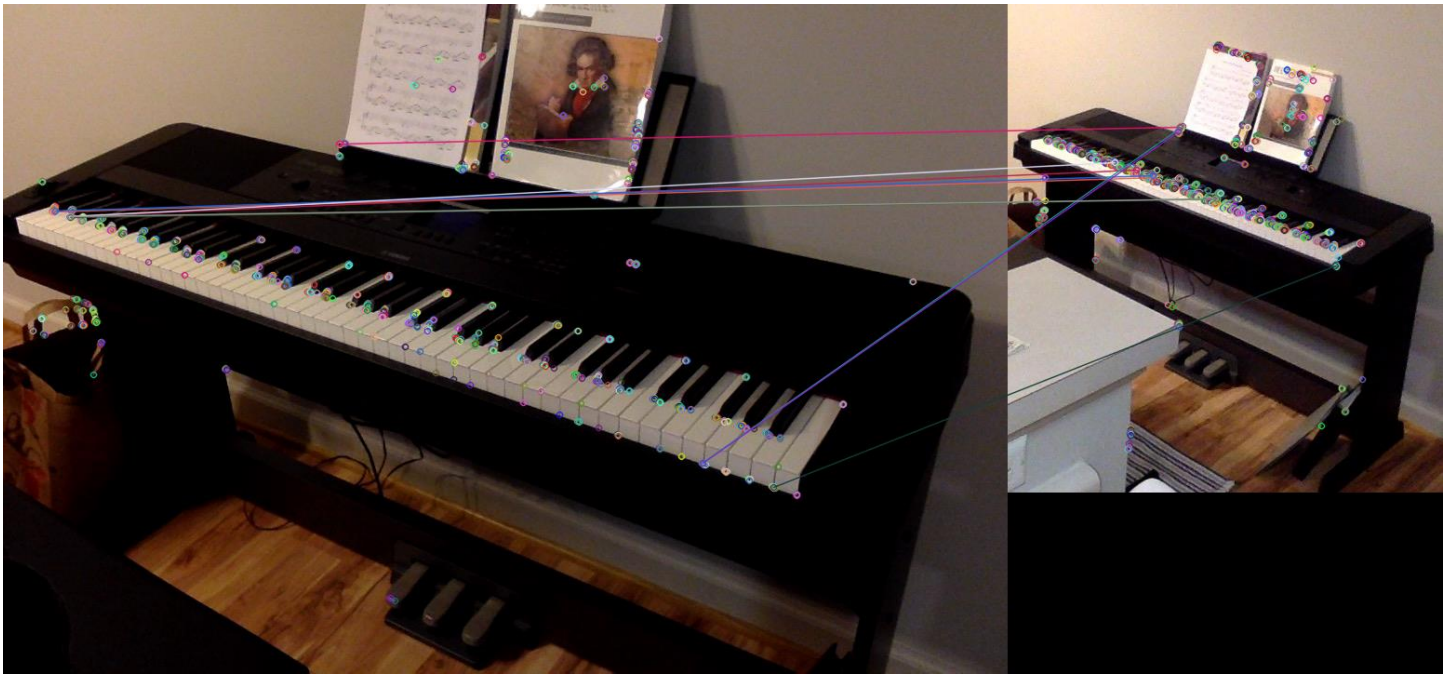
Piano, front view, large scale, rotated

The last in this set of images is a rotated front view of the piano, comparable to the large scale front view from the previous set of images. Among the shared features are Beethoven's collar, some sheet music / album cover corners, a few points on the underside of the piano, and a couple points on the left-most side of the keyboard. Again like previously, the ends of many keys are also picked up.

These results strong support the scale invariant-ness of SIFT descriptors. At two different scales ***and*** at two different angles, several of the same features were detected for the same object. In addition, these same features were also detected in rotated version of the object, further supporting the robustness of SIFT.

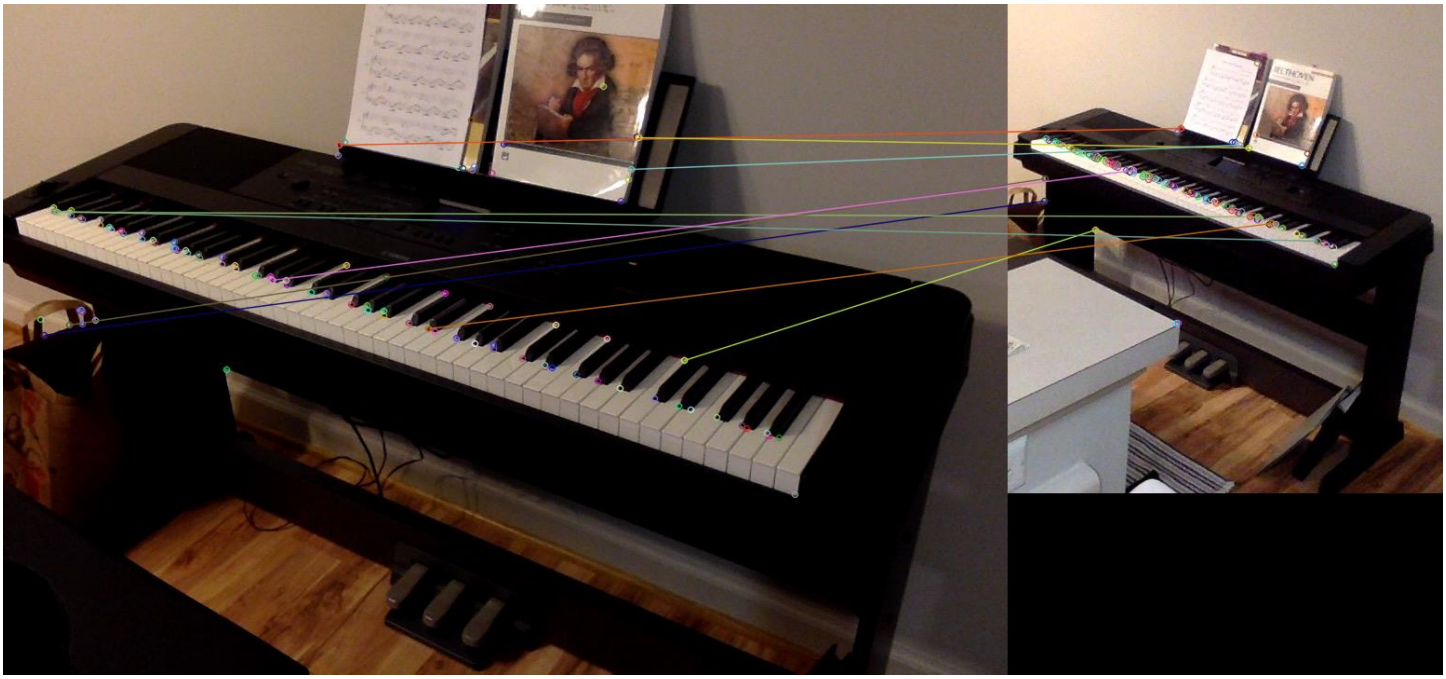
3 Keypoints and Matching

Finally, we come to an application of SIFT and corner detection – point matching. This kind of solution allows us to determine whether the same object exists in two different images and in the case that it does, allows us to do cool things like image stitching! Here, I provide three examples that compare matching using SIFT descriptors and Harris corners based off the results presented in section 2. The parameter values used here were 99, 1, and 45 for number of features, number of levels, and edge threshold. Keypoints were ranked, and the top 10 from each image were considered for matching.



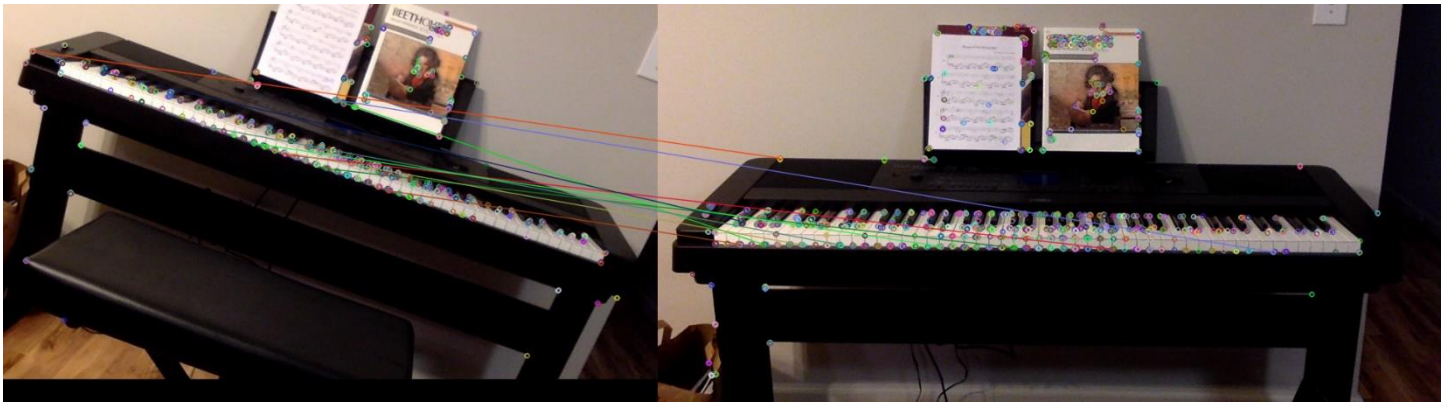
Piano, keypoint matching, Harris corners

Only 1/10 matched in this scenario. This was the bottom left corner of the sheet of paper. Keys were also matched, but not correctly (in pitch class or absolute pitch). Everything else was a mismatch.



Piano, keypoint matching, SIFT descriptors

Again, only 1/10 matched exactly in this scenario. It was the same feature in this case. Again, keys were matched up, pitch class was not. However, there was a bit more success in this image, as paper features matched up with paper features and album cover features matched up with album cover features.

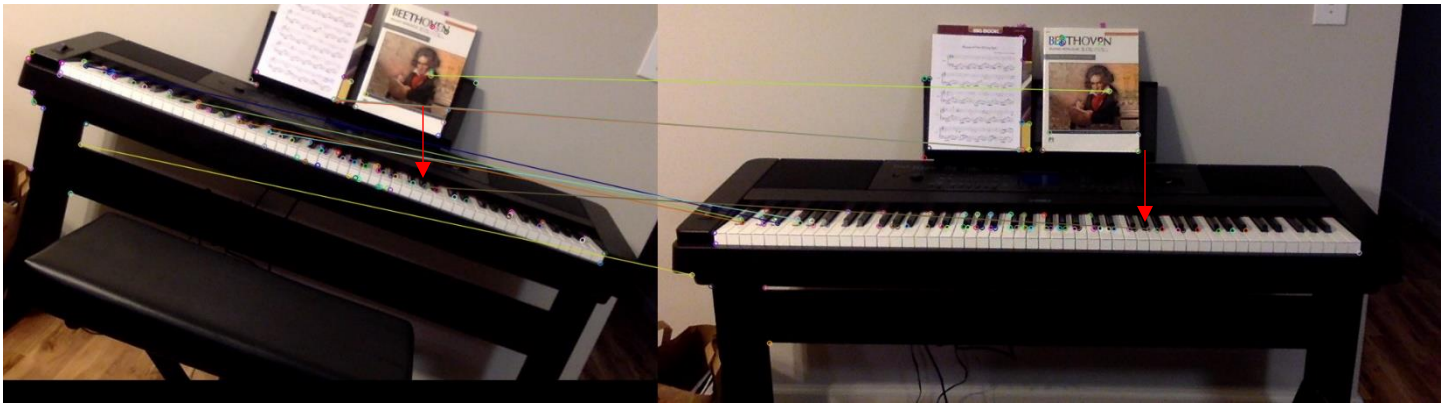


Piano, keypoint matching, Harris corners

Nothing matched correctly here. You can see that some paper and album cover features were matched with the keyboard. Some keys were matched with other keys, but absolutely.

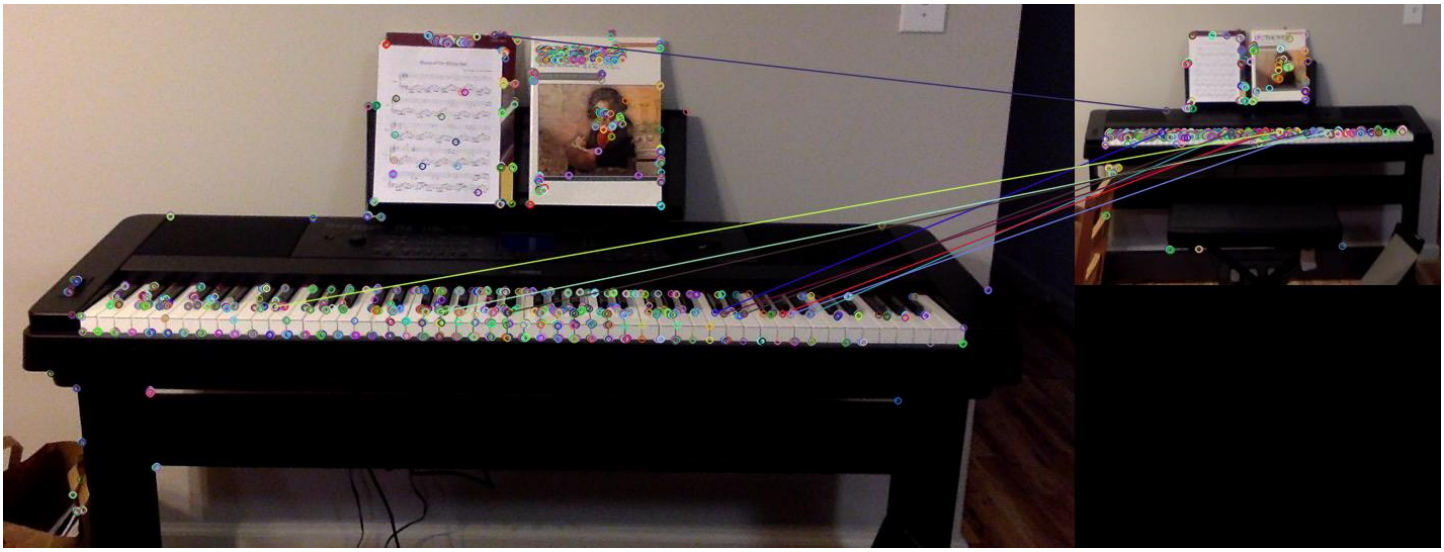
Piano, keypoint matching, SIFT descriptors

Wow! Everything matched exactly except for the magenta line at the bottom. This is so awesome, I'm including the next set of 10 in the following image.



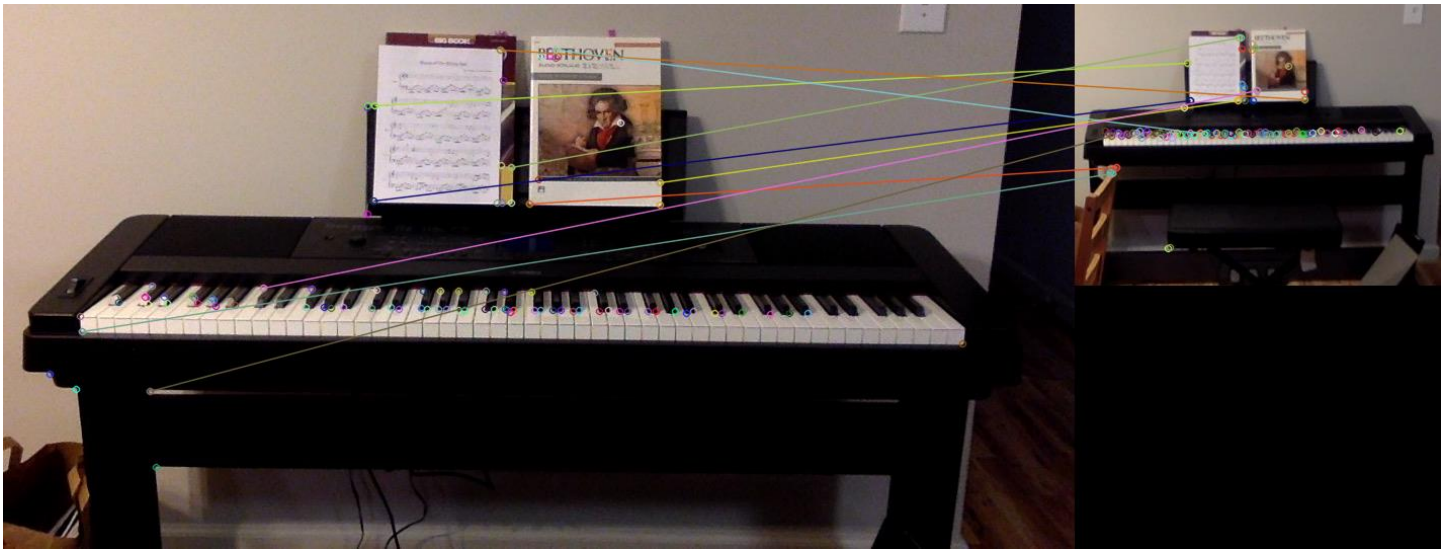
Piano, keypoint matching, SIFT descriptors

A few more descriptors matched here, including two on the paper, Beethoven's collar, and one of the notes (exact match!). This last one is difficult to see, so I'm including an arrow.



Piano, keypoint matching, Harris corners

Nothing matched exactly here. Key are mapped to keys correctly in all cases, though.



Piano, keypoint matching, SIFT descriptors

Only the two points on the left side of the paper are matched correctly here. Everything else is absolutely wrong.

The results of keypoint matching on using Harris corners and SIFT descriptors suggest that SIFT is superior in terms of rotations and scale. In fact, in this scenario, I would say that Harris corners were completely useless. I was especially surprised that SIFT descriptors were capable of identifying the exact note on the keyboard in both a normal and rotated version of the object in the image.