# Project #2: Local Feature Extraction, Detection and Matching
CSC391: Introduction to Computer Vision

Grace Newman

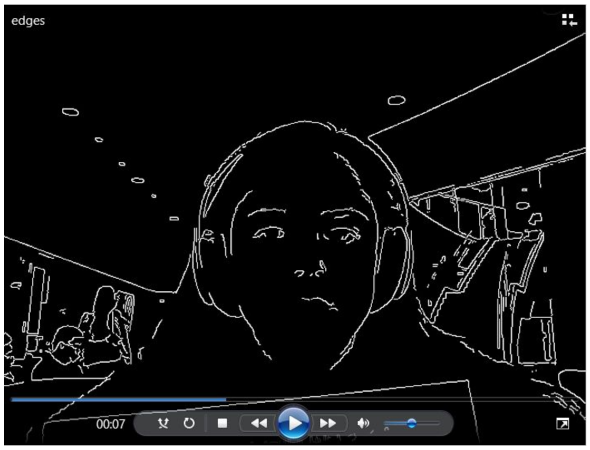
Due date: Friday, March 1st

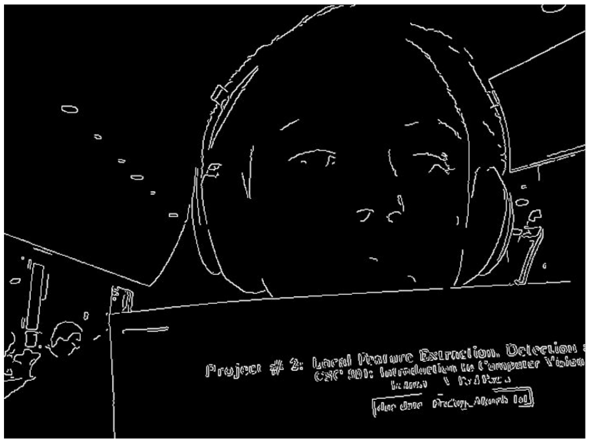
## 1 Edges and Corners in Video

<u>Edge Detection Experimentation:</u>

Scene 1:

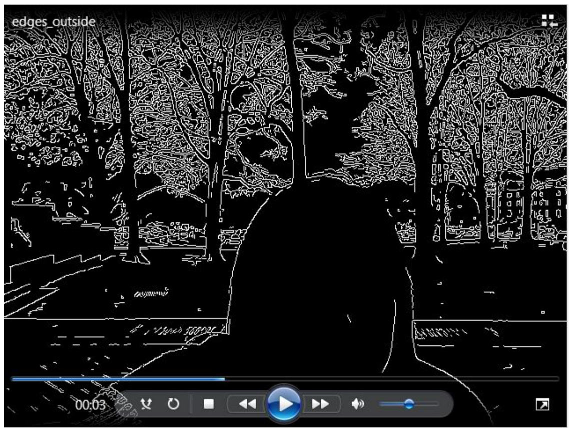
My first scene is pretty well lit, not extremely bright, inside a large open room. I ended up choosing for my parameters cv2.Canny(grayVid, 100, 150) as a nice balance of low and high thresholds for my video edges. When I tried decreasing the first threshold to value of 0, the video footage was very shaky and added wiggly edges in place they didn't exist and didn't detect a lot of the edges that did exist. When I tried 50 for this value, the same shakiness occurred but had less extra edges. When I decreased the second parameter to 0, there was a lot of extra wiggly edges added. When I increased to 50 and then 100, the edges got more stable but appeared granier (maybe detecting some of the softer edges). These values were all bad because the upper threshold was lower than the lower threshold. I think that using cv2.Canny(grayVid, 100, 200) was also a good option because it is a more stable edge detector than the 150, but I think doesn't detect quite as many edges as using 150 did. This filter detected the lines of the ceiling, doorways, and staircase around me, as well as the outlines of my head and upper body. It was able to detect the major features on my face, but didn't have a completely solid outline of my head or shirt. When I held up pieces of paper with words on them, up close it was able to detect the edges of letters very well, but was more unclear and blurry further away from the camera. Whereas up close, it was less able to detect my facial features as it was when I moved further away from the camera.

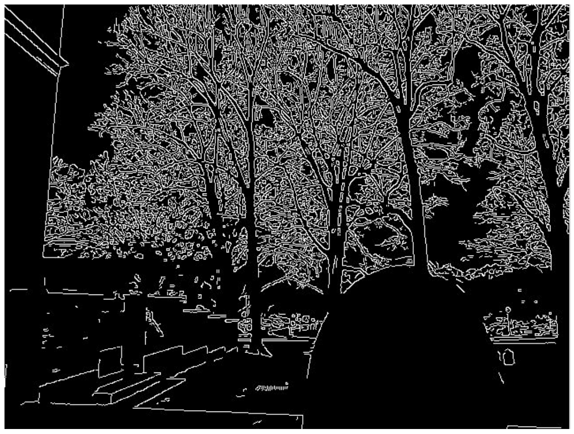




Scene 2:

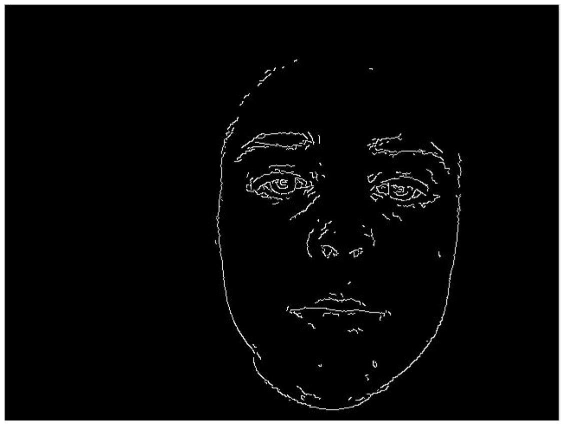
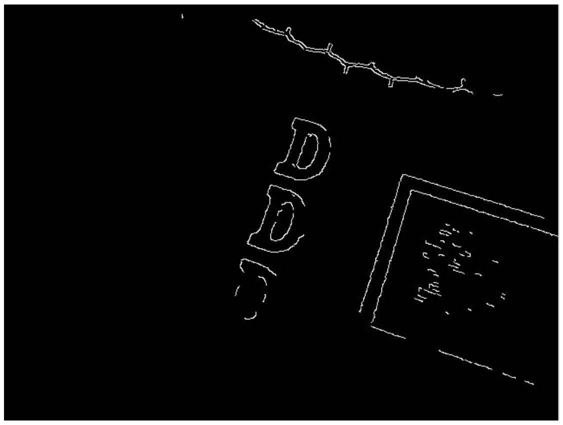
My second scene was outside of the business school, where it was kind of gloomy out and there were lots of trees and also some buildings in the background and edges of buildings. I ended up choosing for my parameters cv2.Canny(grayVid, 100, 150) as a nice balance of lower and upper thresholds for my video edges again. When I played around the values, some of them with the tree branches were crazy wiggly and busy all over the screen, and some of them showed no detail at all. I found that being in the outside lighting with the scenery behind me made it so that the video and edge detection did not find any details on my face, only the silhouette of my upper body and head. The natural lighting was behind me so it didn't brighten my face like it might have if I was facing the sunlight. The scale of me moving closer and further away from the camera didn't change its ability to detect my facial features. The edge detector very well detected the main edges of buildings in the background, the detailed brick pattern on the ground, as well as all of the tree branches behind me.





Scene 3:

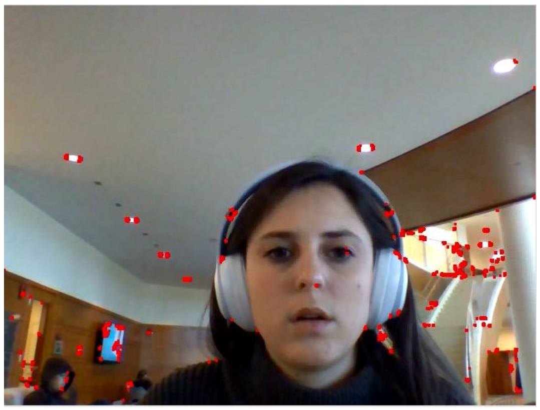
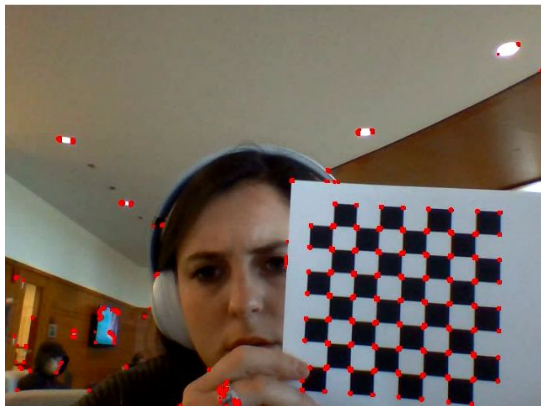
My third scene was inside with the light turned off completely dark. The light of my computer lit up my face and area around me a little bit. For the edge detection, I changed my parameters since the previous ones were detecting hardly any features in the dark. I used cv2.Canny(grayVid, 75, 50), which had wigglier lines than the previous scenes and shorter lines, but allowed me to capture the most edges in this scene. When I moved around, the edges were a little shaky and it was specific on where it would pick up edges of the things on the wall behind me. Scale also really mattered because it hardly detected any of the features on my face the further away I was, but was able to completely outline all of my facial features up close to the camera. It also was able to detect some of the lines on the ceiling, but since it was so dark that was pretty far away for it to pick up on.

Corner Detection Experimentation:

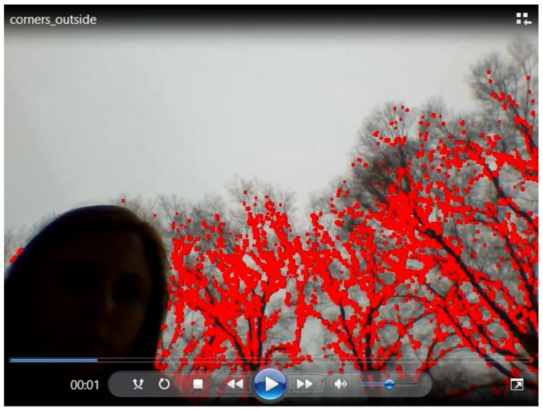
Scene 1:

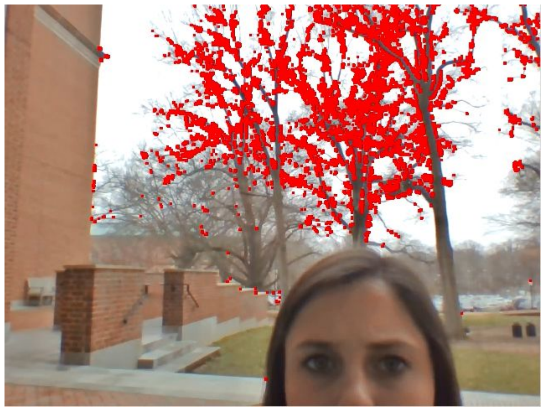
For my first scene that was inside and fairly well lit in a large room with doors and staircases and lots of straight lines. The arguments of cornerHarris are the blockSize of the neighborhood of detection, ksize used in the derivative, and k. I used the parameters cv2.cornerHarris(grayVid, 3, 3, 0.01) as a balance between detecting corners, but had a higher threshold so not everything looked like a corner. Some of the larger values I tried resulted in a thick ring of red dots around my outline and around other edges. Especially when I decreased the k parameter to 0.00001, it was a very low threshold and a lot of things were detected as being corners. These parameters were able to detect corners on the ceiling and staircase in the background well, and on my facial features the corner of my eyes and mouth. However, far in the background there are a lot of dots for people moving around that are confused, so I don't think it is very good at detecting corners on things that are small very far away. When I held up a sheet with a checkerboard on it, the corner detection worked really well on finding all the corners. It also worked well with transformations when I moved the sheet around, it continued to detect the corners. However, as I moved the sheet closer to the camera, it had a harder time detecting a lot of the corners. The corner detection didn't work well on things held very close of big scale, and also of the things far in the background there were lots of dots for things that don't seem to be corners.

Scene 2:

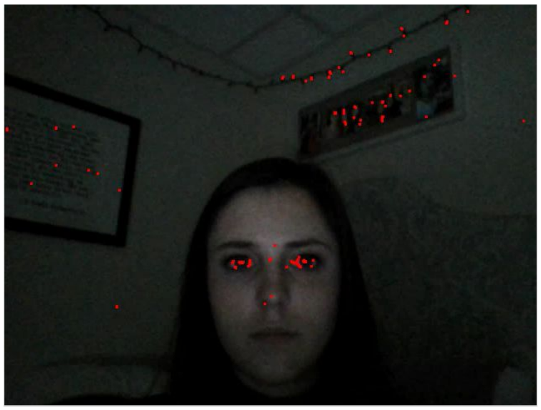
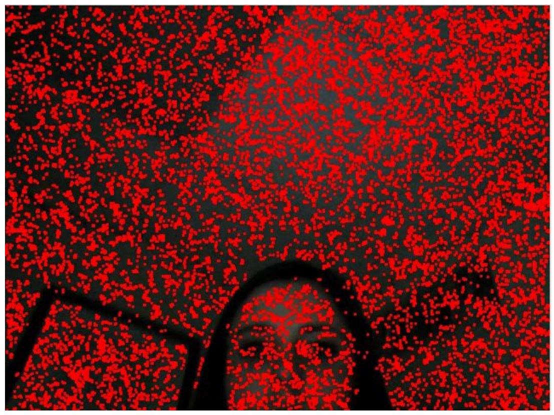
My second scene that was outside of the business school, where it was kind of gloomy out and there were lots of trees and also some buildings in the background and edges of buildings. It was a very busy background with all of the tree branches, so a whole bunch of corners were detected back there from the tips of all of the little branches. When I moved my laptop around side to side and angled it up and down for transformations, it was consistent in detecting corners in the same places. When I angled the computer towards the building in the background, it was able to detect the corners of steps in the wall, the corner in the tip of the building, and the corners between my silhouette and ground in the background. However the building that was quite far in the background, it didn't detect many of the corners of the outline of it or of the windows of the building. The corner detection was again, not very good at detecting things far in the background. Also, because of the lighting again, the corner detection did not trace any of my facial features because the natural light was coming from behind me.





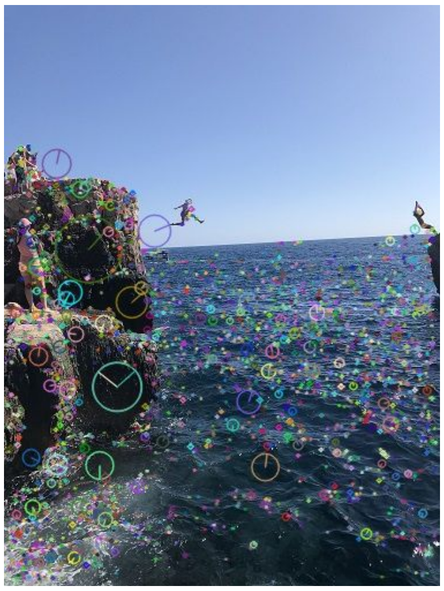
Scene 3:

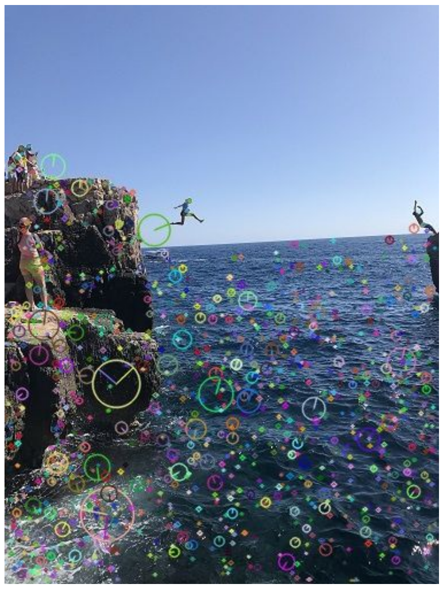
For my third scene which was inside in the complete darkness, the brightness of my laptop screen really helped with lighting up the footage. However, I think the darkness cause my video footage to be very grainy, because the results created a lot more corners than the previous scenes had, especially in places without corners. For my parameters for this scene, I chose cornerHarris(grayVid, 3, 3, .2), because the .2 decreased the amount of excess corners detected on my facial features. I adjusted my parameters and no corners would show up with anything but 3 for the first two values, so I left those the same. However, I tried decreasing my threshold on the third parameter to 0.00001, but it created a thick ring of red dots around me and certain features. When I increased the third parameter to .5, nothing at all showed up. However, even with the values I chose for the corner detection, when I would move my laptop screen around for translation, red dots covered the screen everywhere until it stopped moving. It was still bad at scale and wouldn't detect corners in the background when the laptop was still, but whenever the laptop moved around, the whole screen was covered in red corners. Lighting seems to be very important in the ability of this function to detect corners.

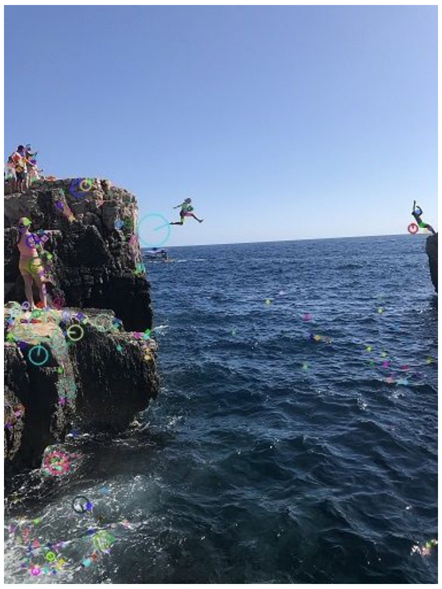



## 2 SIFT Descriptors and Scaling

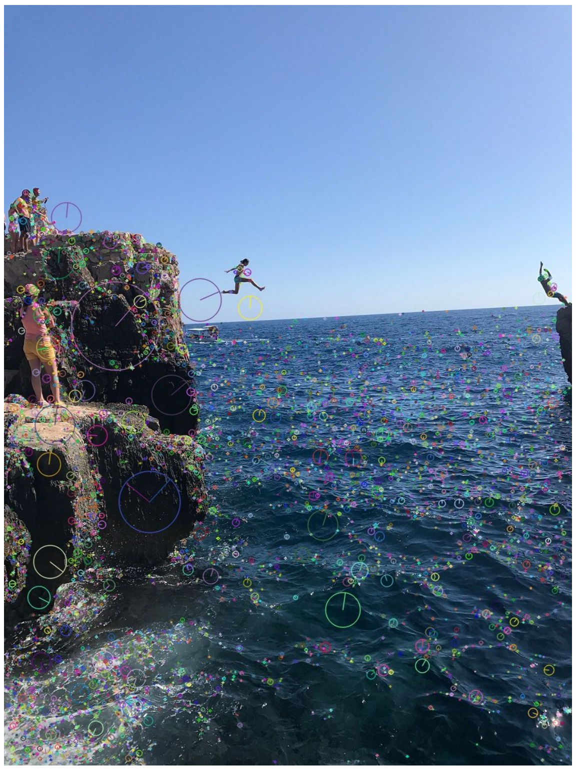
For my parameter values, I thought that the default values were the best values for the image I selected. I used the values cv2.xfeatures2d.SIFT_create(0, 3, 0.04, 10, 1.6). When I increased and decreased the contrast threshold parameter, either not enough keypoints showed up when I increased the value to 0.1, and when I decreased it to 0.01, there were way more key points than we needed, and also of things that weren't really distinct features in the image. I also experimented with the values of the fourth and fifth parameters. The edge threshold parameter, when I made it 0 had some more key points show up further out in the horizon of the photo, that I didn't think were key features. When I played with the value of the sigma, no key points were given when the value was below 1. A value of 1 showed less key points out in the water, but focused just around the rocks and people features. Also, some of the gradients and key point circles became a lot larger, because I think it was less accurate at detecting where exactly the key point was and characterized a larger region as being a keypoint. I think it is important to have a lot of the key points in the waves identified because if matching up with wave caps of another image.

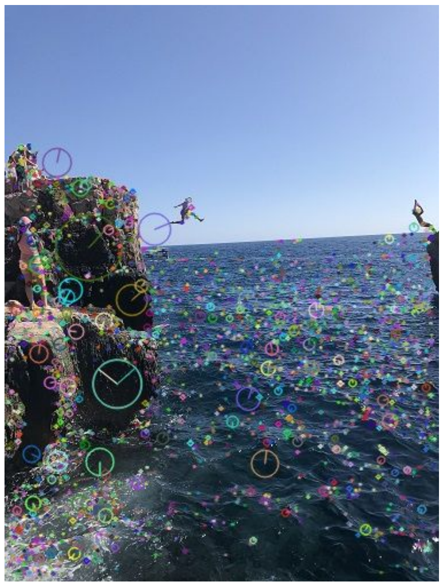

(0, 3, 0.04, 10, 1.6)

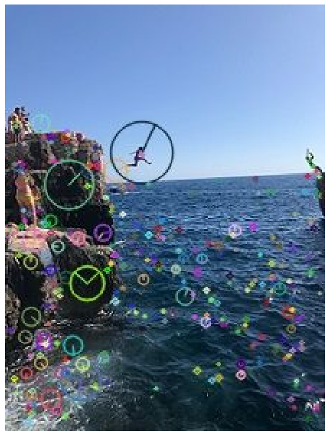

(0, 3, 0.01, 10, 1.6)


(0, 3, 0.1, 10, 1.6)

<u>Scale change</u>

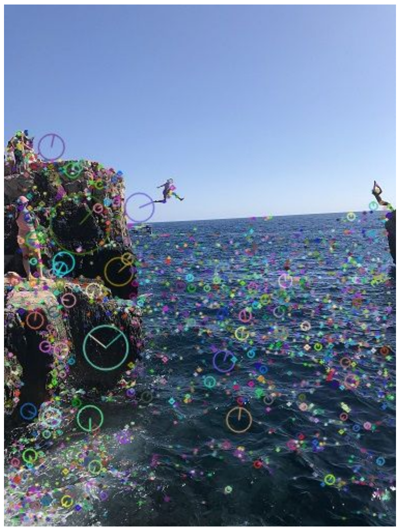


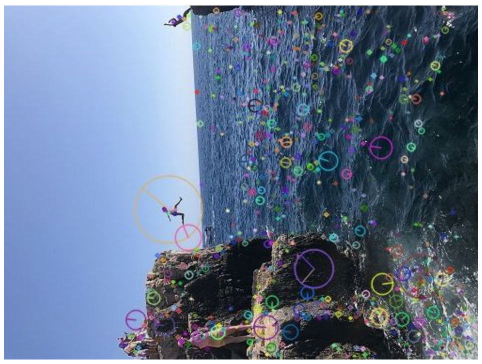




When I changed the scale of the image, the first be more than twice the size of the second, and the third being less than have the size, the key points found changed drastically. I think that the quality of the photo affected this also, because when I downsized the image the amount of detail and clarity was likely changed. The photo of the largest scale detected a ton of small key points in great detail. There were more key points of specific gradients all over, while the medium scaled photo seemed to group some of these features together and detect them with more general gradients, rather than very detailed. The amount of key points found in the largest image continued out into the water much further than the originally sized image and was able to track down much more detail further out. The smallest image hardly detected any key point compared to the two larger images. On me jumping, it characterized my whole body as a keypoint and was very large in scale because it found me to be an important feature in the photo and found the gradient of that rather than breaking it into smaller points like the larger photos allowed. The points out in the water hardly even have gradients and are just small dots.
The performance got worse with each downscaling of the image. The largest image found the most key points that could be used to detect features in the image. The medium size image found a lot of points that would probably be the best to work with without having so many. The smallest image when the scale was downsized by a lot, was not effective at all in detecting key features of the image. This SIFT function becomes a lot less effective and has limits with the scaling when working with smaller images and less detail.

<u>Rotated</u>

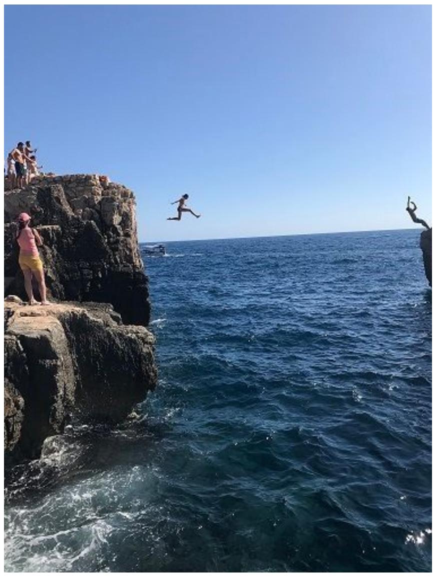


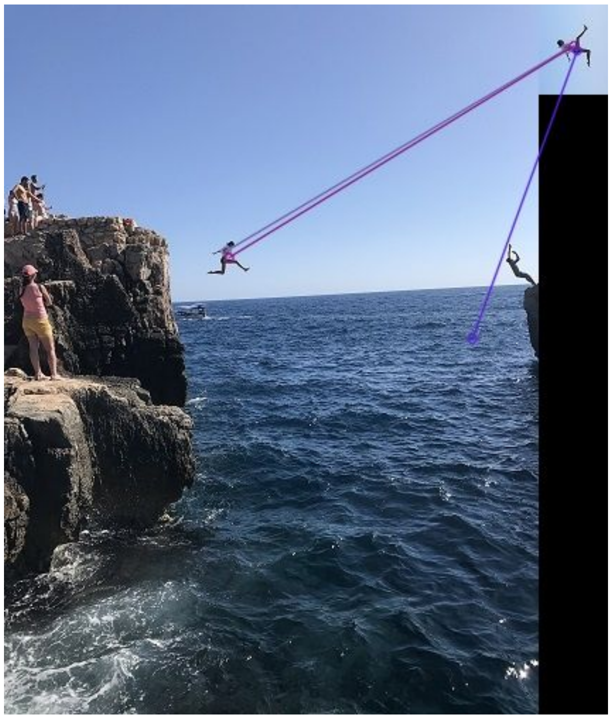


When I rotated the image using the same parameter values, the gradients did not come out the same. I found this interesting because I was expecting everything to be the same, just rotated. However, there were extra key points added on some of the larger features that weren't detected on the original. Also, some of the smaller features were not detected as key points in some of the water and waves. I am surprised that rotation affects the keypoint finder when it doesn't affect the detection of edges or corners.

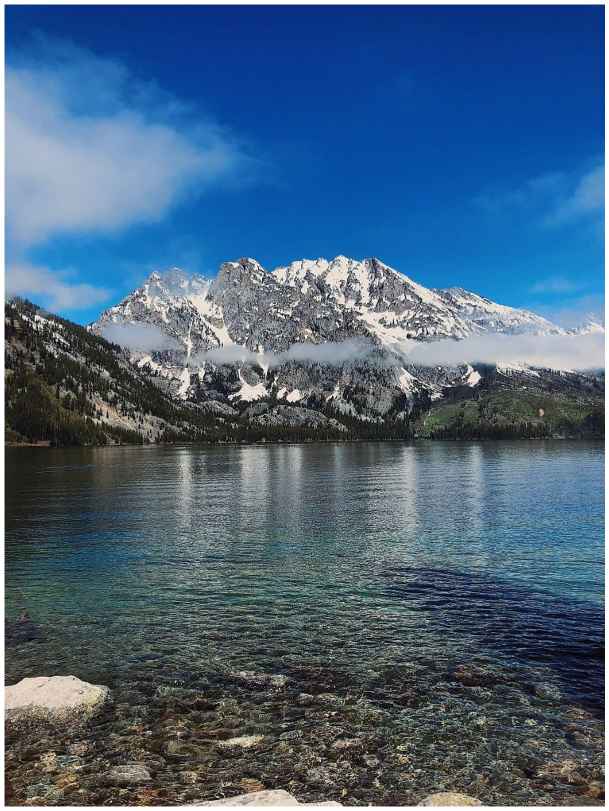
## 3 Keypoints and Matching

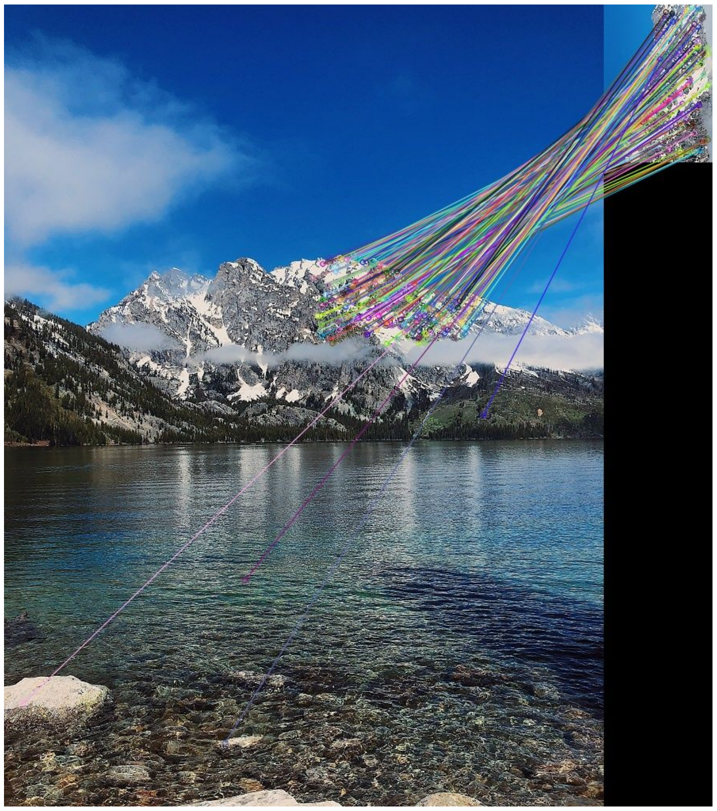
Different angles of photos:





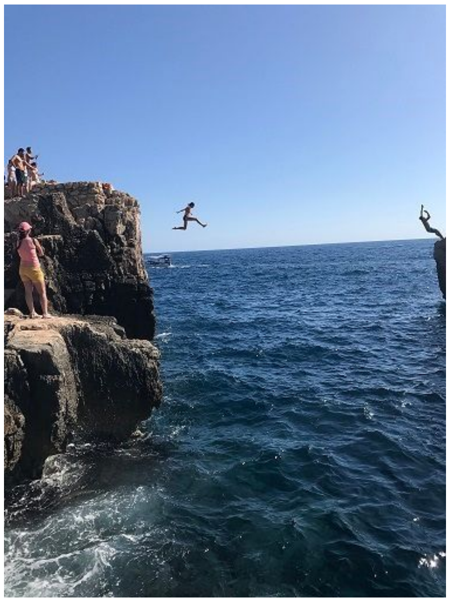
When I used the matching, it was able to for the most part determine where in the original image the small, rotated image was taken from. In this example, I used a pretty distinct part of the image without many extra keypoints in it so it should be able to pretty accurately determine where the photo was from. This is useful in this scenario to determine which part of the photo the smaller photo came from.

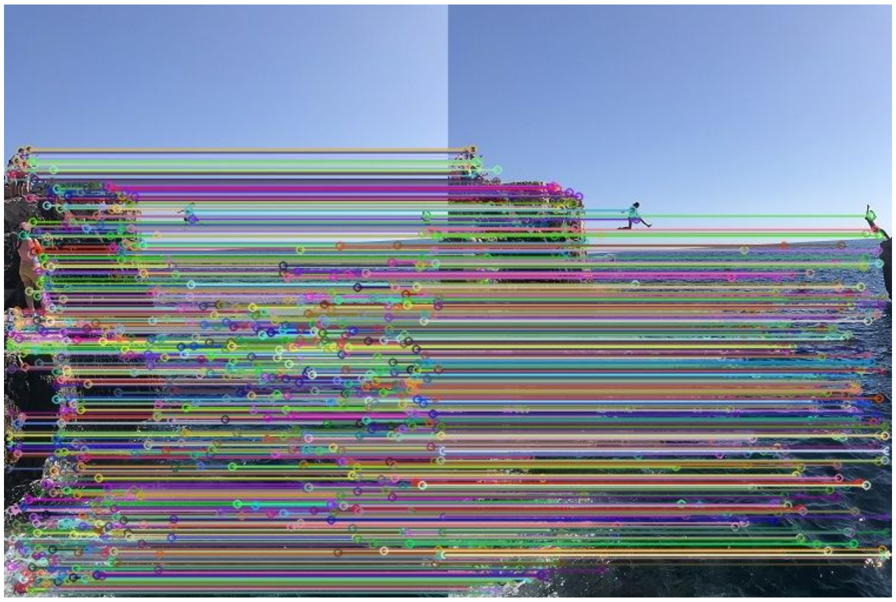


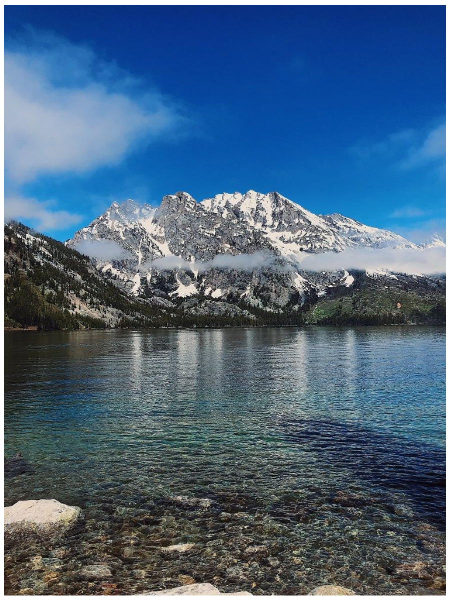


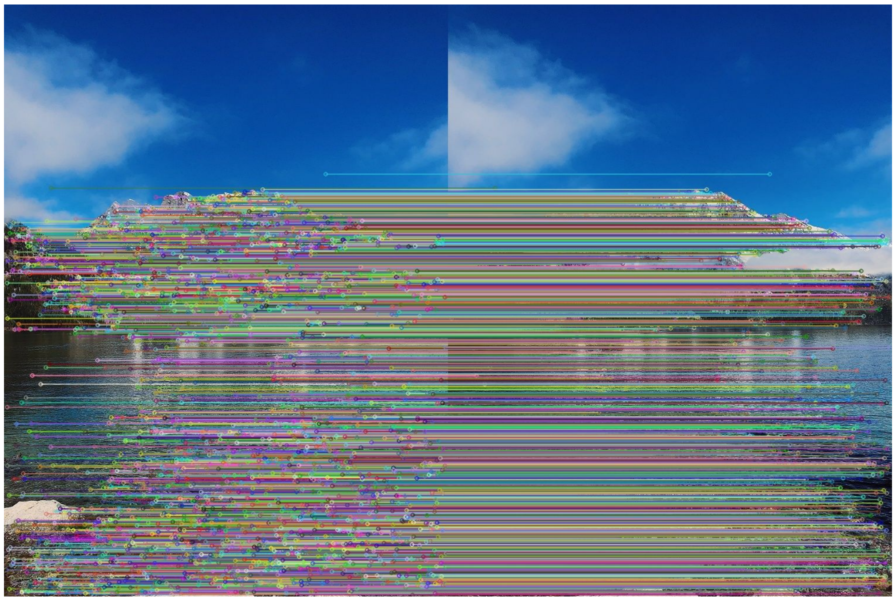
When I used the matching in this case, it correctly identified most of the keypoints from the smaller, rotated image, except a few that were matched elsewhere in the scene. In this example, I took a subset of the mountain peaks that looks fairly similar to the other parts of the mountain and did not match it up with any other part of the mountain, which I was surprised about. This had good matching for this image. There were a lot more keypoints in this mountain peak than the image of me.  In this case, the matching could be useful to determine where different photos from the same scene might overlap and match them together.

<u>Same Images</u>



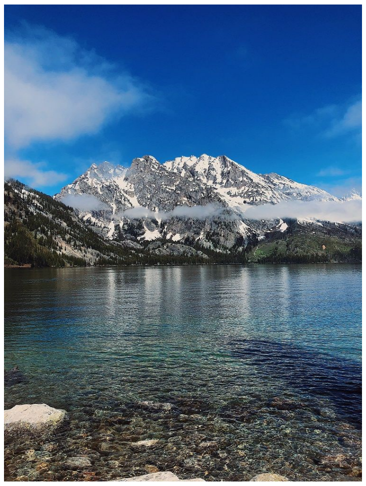
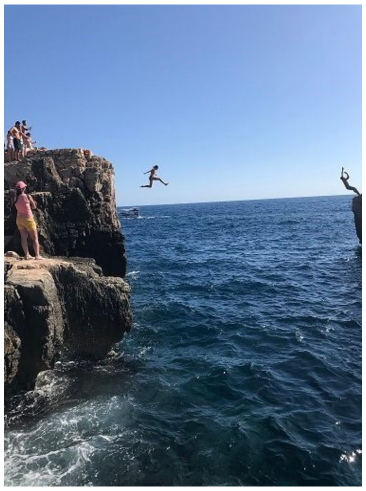
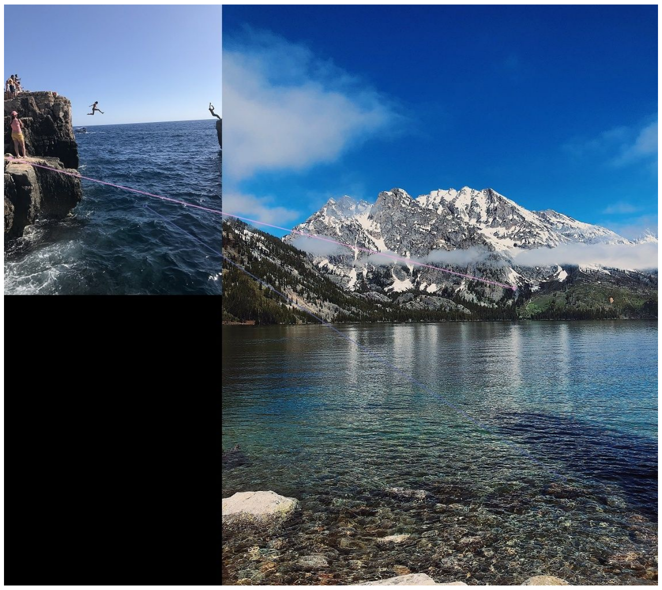






When I used the matching on the photo of me jumping against the photo of me jumping, as well as the mountain photo against the mountain photo, it was able to match many keypoints in each photo. You would be able to deduce from the amount of lines present that they images were the same. The matching tool is useful in this case to test if images are the same, or detect as many similarities as possible in photos.

<u>Different Images:</u>

Out of curiosity, I also used the matching code on the two completely different images that I was using to experiment. It detected two matching keypoints between the two photos. I would've thought there would be more similarities than this because of the rippling water in each photo, or at least that that would be where the matching points come from. However, the keypoints matched come from the brown rocks and match with darker parts of the mountain image. In this case, the tool would be used to detect different features of completely different photos that are similar based off of their gradients, which may not be that useful.