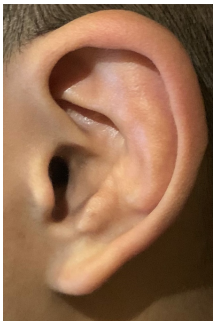
Final Project: Ear Detection and Verification

By: Diego Guadarrama, Fred Chan, Elygh Thao

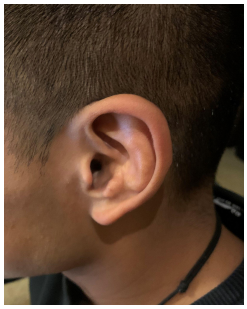
Goals For This Project:

For our project we wanted to create a program using opencv to find a certain ear in an image. We would start with an initial image of a person's ear (we'll call this anchor) and try to find this ear across different comparison images. For this project to be successful, the specific ear will need to be detected and matched in the comparison image. If there are multiple ears in the picture, our program will need to find the anchor ear correctly. If there are only negative ears in the comparison image, our program should clearly state that the ear is not found.

Here are examples of the kind of images our program will deal with:



Anchor Image



Angled Positive Image

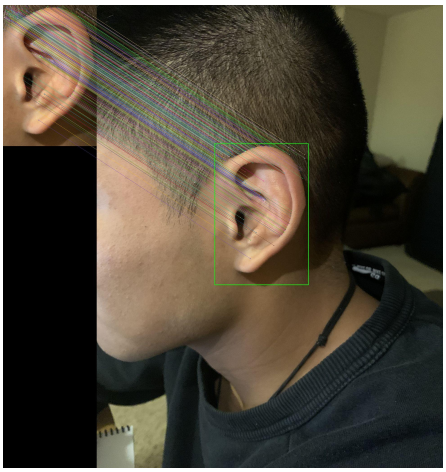
Zoomed-Out Positive Image

True Negative Image

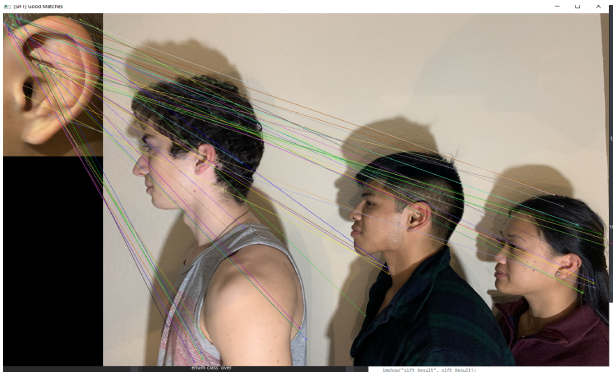
Multiple Ear Image with True Positive in Middle

OpenCV Feature Detection

The first thing we worked on was determining an opencv tool to help us identify the anchor ear in the comparison images. We wanted a key feature detector that would be able to make key points across both the anchor image and comparison image accurately and consistently. We experimented with SURF, ORB, and SIFT, and eventually decided on SIFT because even though it wasn't the fastest, it produced key points that we felt would work better with our closeup ear anchor image. Combining SIFT with the flann key point matcher algorithm gave us great matches that worked on images with no difference. For example, the image below is matching the anchor image with a zoomed out anchor image.



However our feature matching started to give us issues on images where the anchor ear would be smaller, and images with the ear at an angle.



The feature matching is matching all across the comparison image ranging from negative ears, to even the shoulder of the boy on the left.

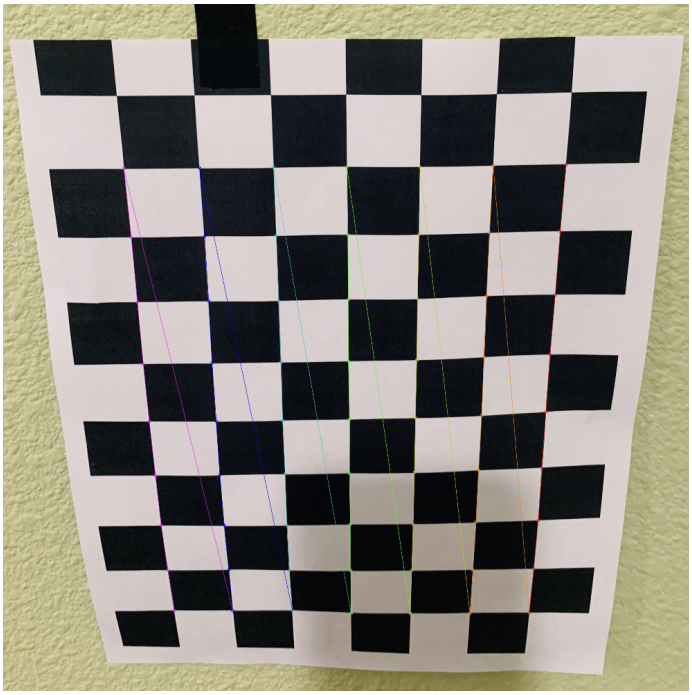
The feature matching matches too much with the hair, and a lot less with the actual ear. Only some of the points on the anchor image actually correlate to the correct location in the ear of the comparison ear.

From here we decide to split our team to tackle each issue: Scale/Resize and Angle

Angle Problem

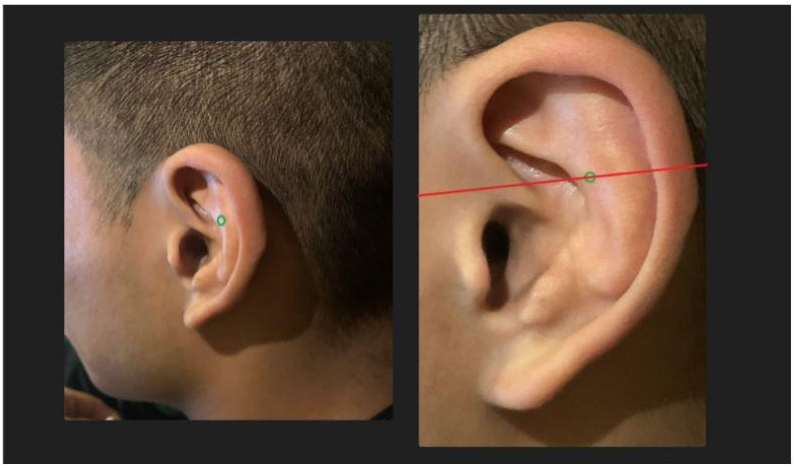
In order to verify an ear, depth information can potentially be helpful since all ears have different shapes. However, we have images all taken at different viewpoints and positions. Depth information can change widely. In order to standardize our input images, we proposed to project all images to two fixed camera positions to get our pairs of stereos images. The processes are to find intrinsic parameters of the camera, create two virtual viewpoints and project images to a standardized location, calculate the depth of the ears, and compare the depth map to verify if they are the same ear.

This image shows we can learn the intrinsic parameters by taking multiple images of the chessboard in the same position but with different angles.



Stereo Vision

- Find correspondence
- Calculate disparity
- Calculate depth



Reference: https://fpcv.cs.columbia.edu/about

Fundamental Matrix

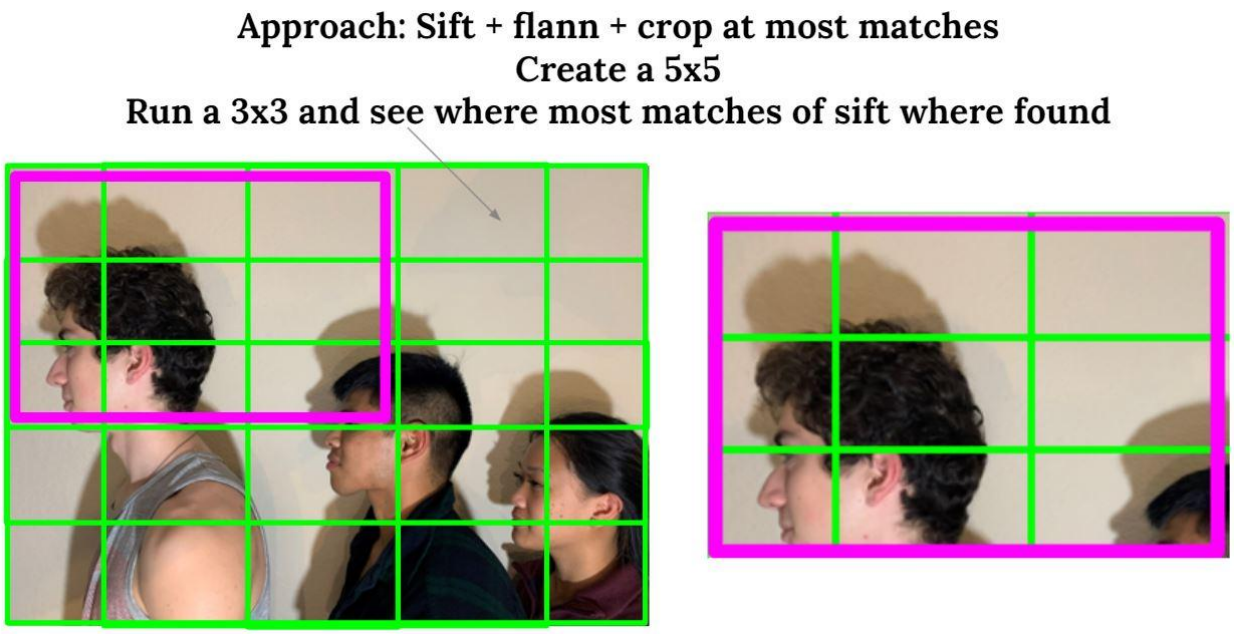
Left Points -> Right Points
U, V -> u, v
U, V -> u, v
U, V -> u, v
.....
.....

$$\underbrace{\begin{bmatrix} u_l^{(i)} & v_l^{(i)} & 1 \end{bmatrix}}_{\text{Known}} \underbrace{\begin{bmatrix} f_{11} & f_{12} & f_{13} \\ f_{21} & f_{22} & f_{23} \\ f_{31} & f_{32} & f_{33} \end{bmatrix}}_{\text{Unknown}} \underbrace{\begin{bmatrix} u_r^{(i)} \\ v_r^{(i)} \\ 1 \end{bmatrix}}_{\text{Known}} = 0$$

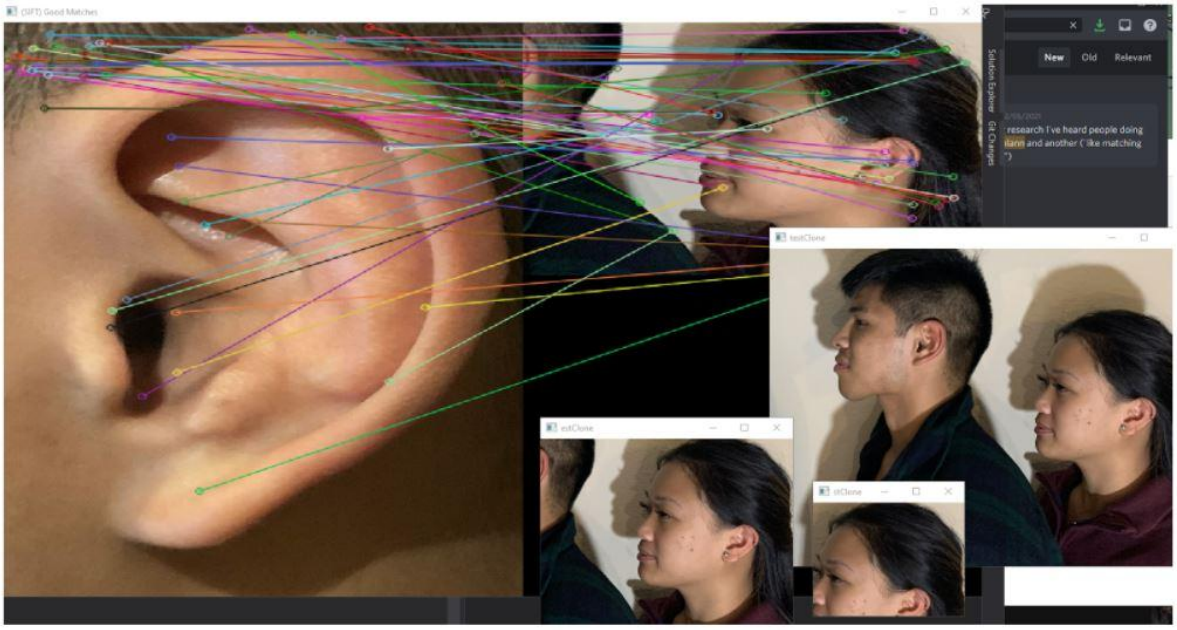
Reference: https://fpcv.cs.columbia.edu/about

Scale/Resize Problem

Having the ear so small in the image with multiple people greatly impacts our SIFT results. It generates matches all over the place and is overall pretty unusable. Our strategy we created was to crop where the most matches are and eventually zero in on the ear. We looked back at our previous program 3 and made an algorithm similar to finding the most common color and storing it in a histogram. We modified this by making the histogram just 1 dimensional in a 5x5 grid representing the comparison image's size and instead of keeping track of common color, keep track of most votes in a 3x3. Cycle through the 5x5 over with a 3x3 template and make a crop of a 3x3 with the most matches. This image below represents how our algorithm works.

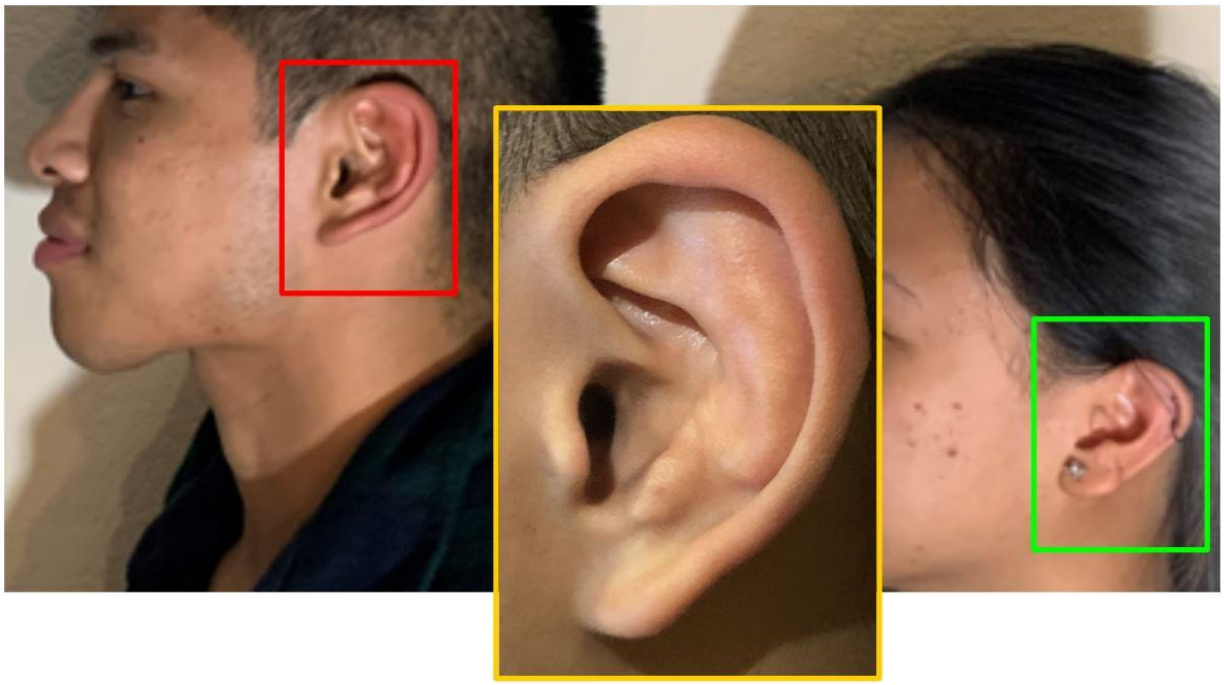


This algorithm worked pretty well, but what we encountered was that most matches were still not broad enough in a 3x3 that it matched onto the target ear. The image below shows what actually happened when we ran our algorithm with the image with multiple people.

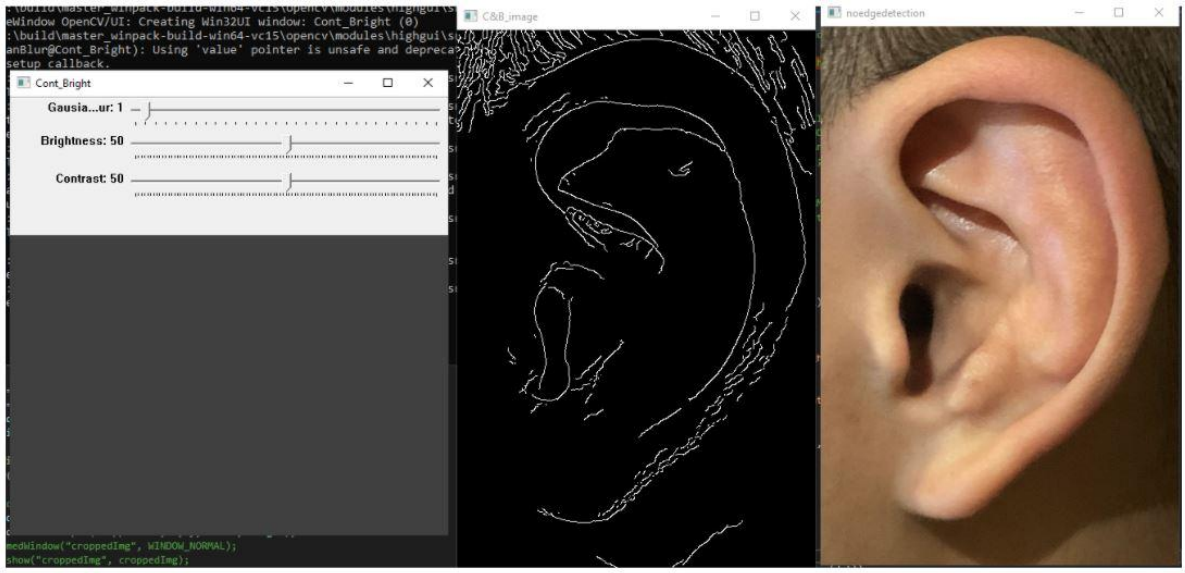


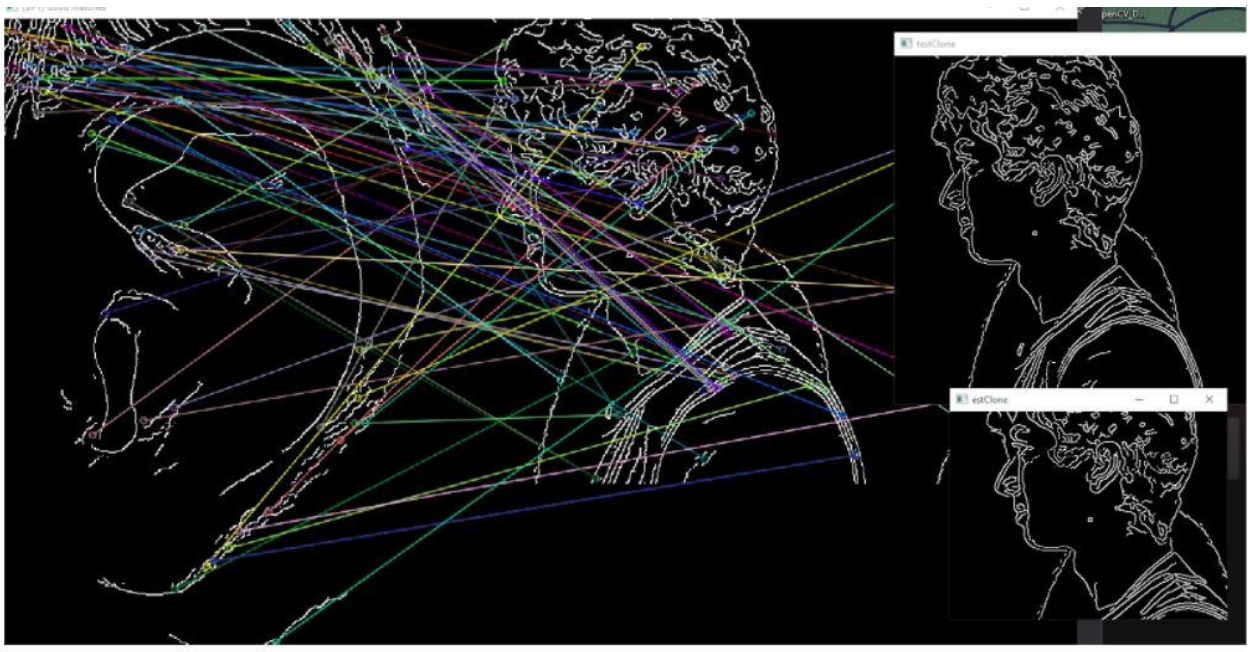
Original Images sift + crop

As you can see, the sift+crop algorithm worked, but on the wrong ear. It actually zeroed in on the girl's ear, instead of the actual target ear which is the boy in the middle. And if you look carefully, the girl's ear does look more like the target ear compared to the boy's ear. After this analysis, we realized that it wasn't that the ear wasn't being recognized, but that the boy's ear actually changed.



Due to him coming back from playing soccer, his ear had actually become more defined and actually turned red. Because of this fact, we realized that it wasn't surprising that our program started to focus on the girl's ear.
From here, we started with experimenting with different approaches, the first being edge detection. Since the ear features became more defined after coming home from a soccer match, we tried using uncanny edge detection and ran our SIFT+crop program from there. We hoped this would have matched more with the shape of the ear and features that otherwise could only be seen with edge detection would be picked up and identified.

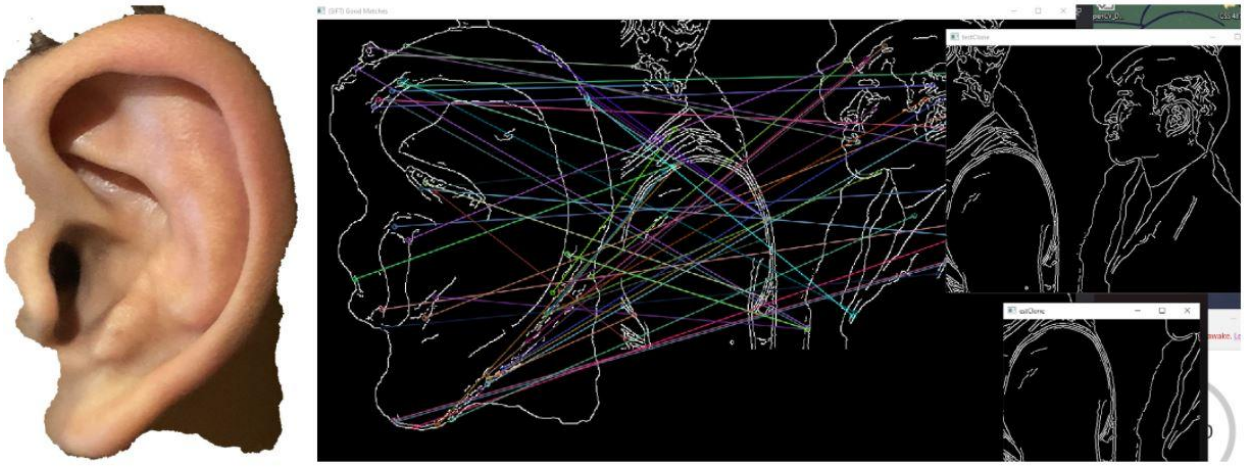




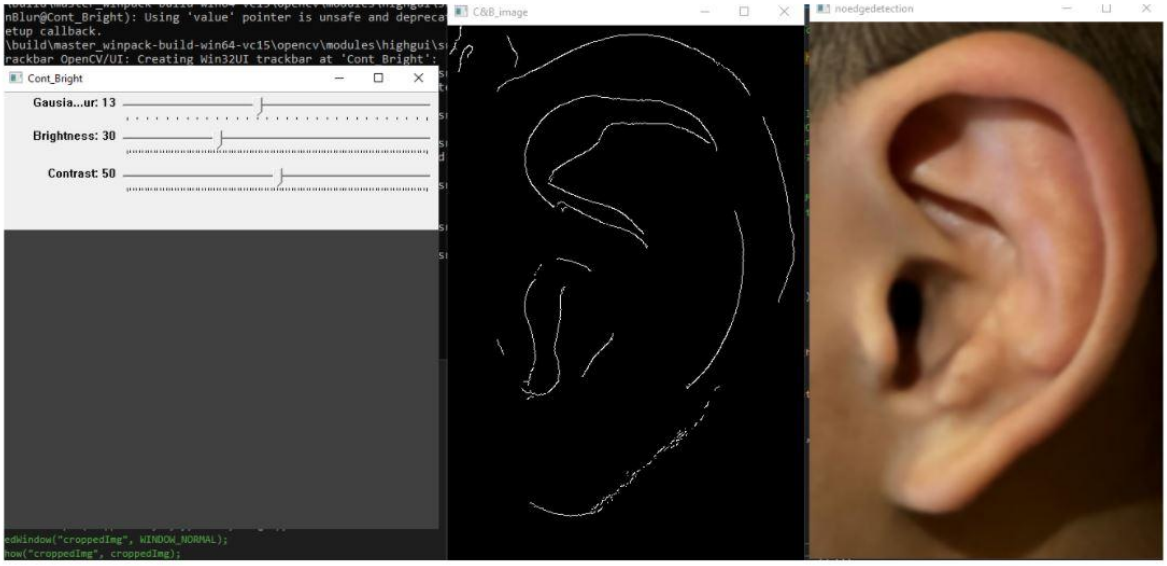
This edge detection ended up not only matching with the wrong person, but also a lot of hair due to the nature of edge detection. This hair issue was annoying so we next tried to get rid of

the hair by using a grab cut of the anchor image. For this we just used the opencv methods for grab cut then ran it with edge detection, then with SIFT+crop. Here were our results.

GrabCut + canny

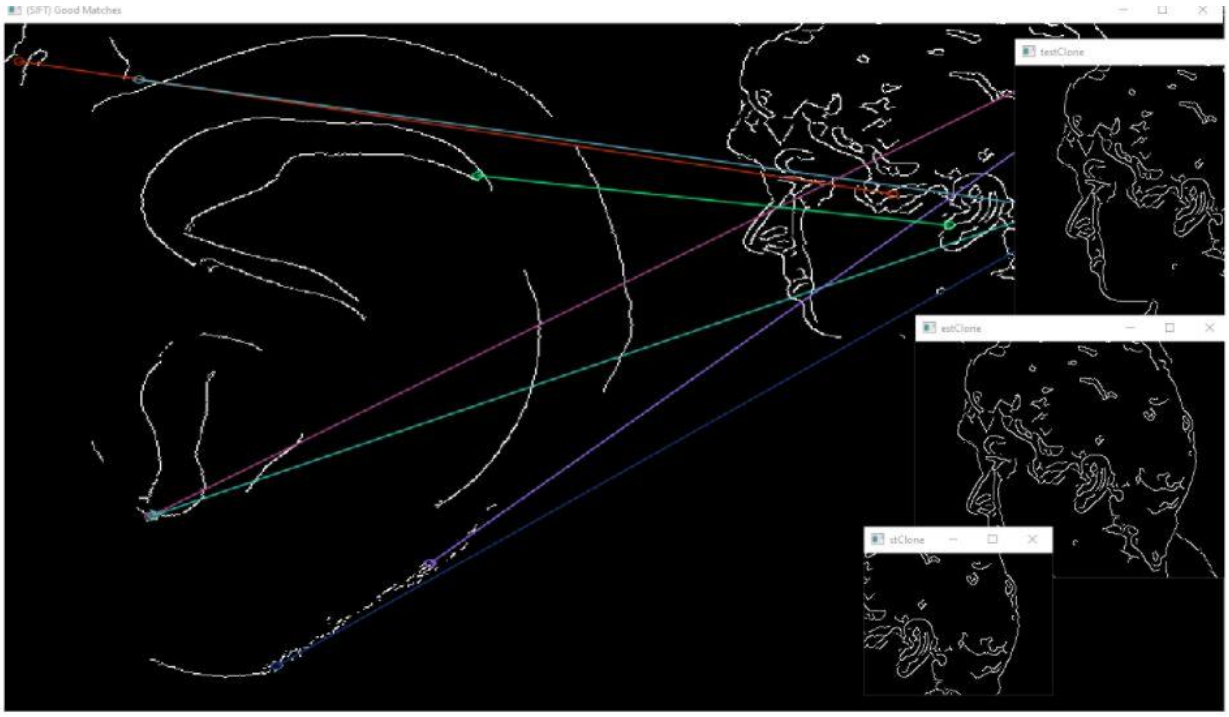


What ended up happening is that the grab cut got rid of the hair, but created a new edge on the left side of the ear that otherwise would never be there. This messed up the SIFT feature detection and resulted in our crop being focused on the left boy's shoulder. Realizing we made a mistake with the grab cut we undid it and continued from there. Our reasoning for the grab cut was to get rid of the hair, but that created a different problem, so a solution we thought of that wouldn't create new unexpected edges would be to use a gaussian blur in addition to altering the brightness and contrast values.



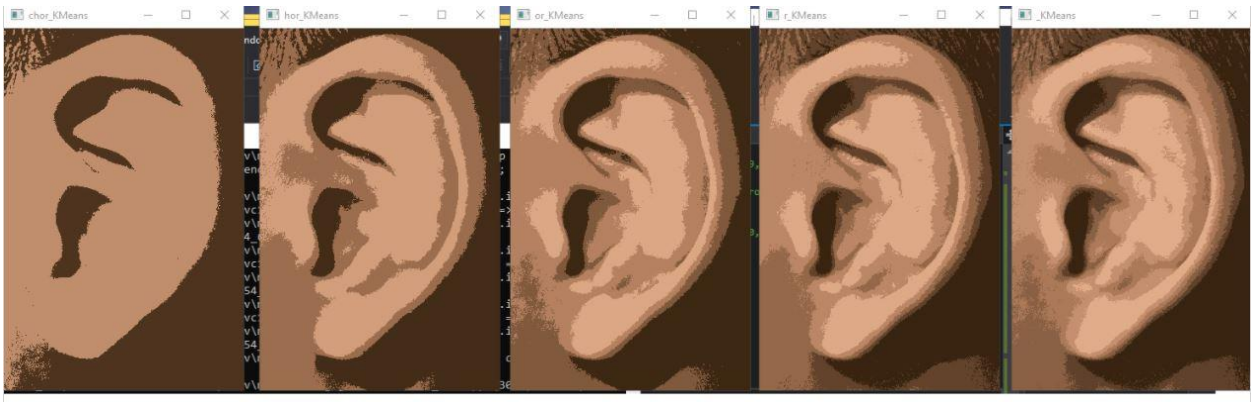
Gaussian: 13 | Brightness: 30 | Contrast: 50

Next run it through our sift+crop

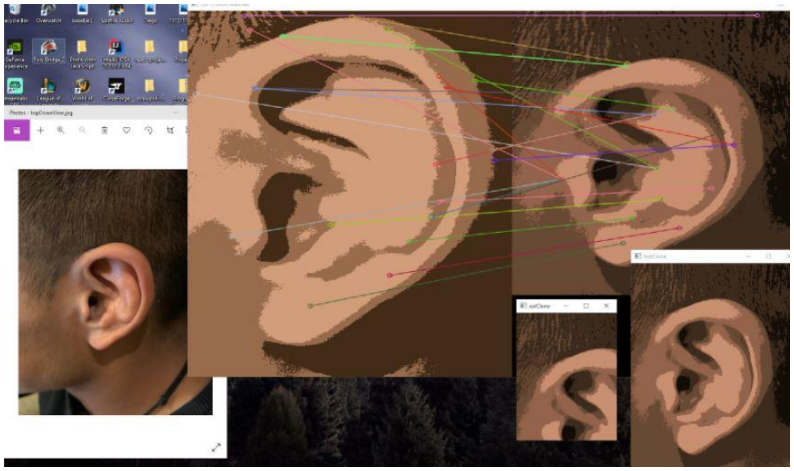


This method gave us a match to an ear, but it matched us to the wrong ear. It matched to the boy on the left's ear instead of the boy's in the middle.

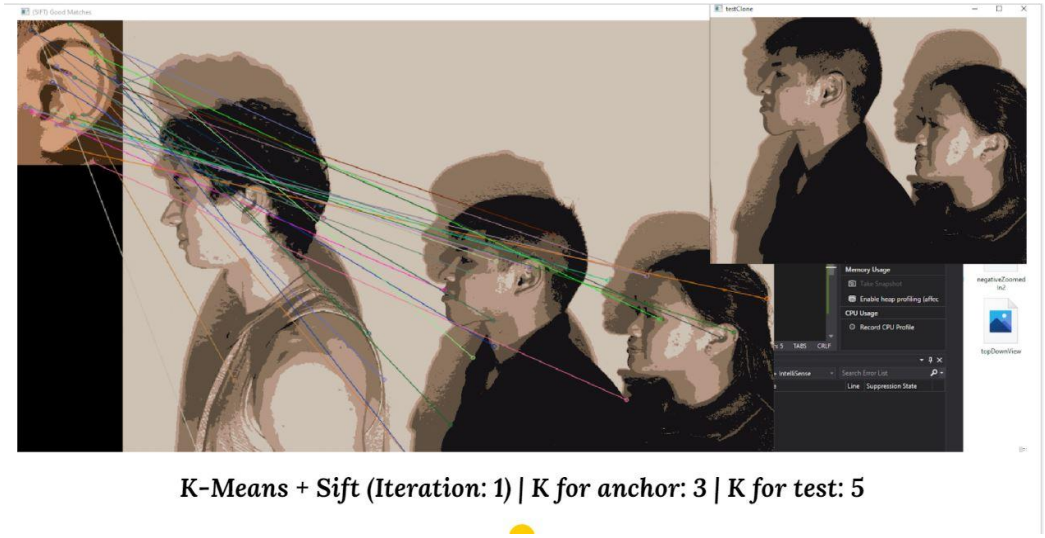
At this point we decided that maybe edge detection wasn't the way to go. It wasn't generating specific enough keypoints to use and match to the correct ear. We needed something that would make both images easier to identify key points and make those key points distinct enough that the algorithm could find the ear. So to do this, we started messing with color by implementing a k-means algorithm. Our k-means algorithm would take in an int and image, and generate a new image with the average number of colors. So if k was 2, it would only generate two colors. Look at the image below for a better understanding.



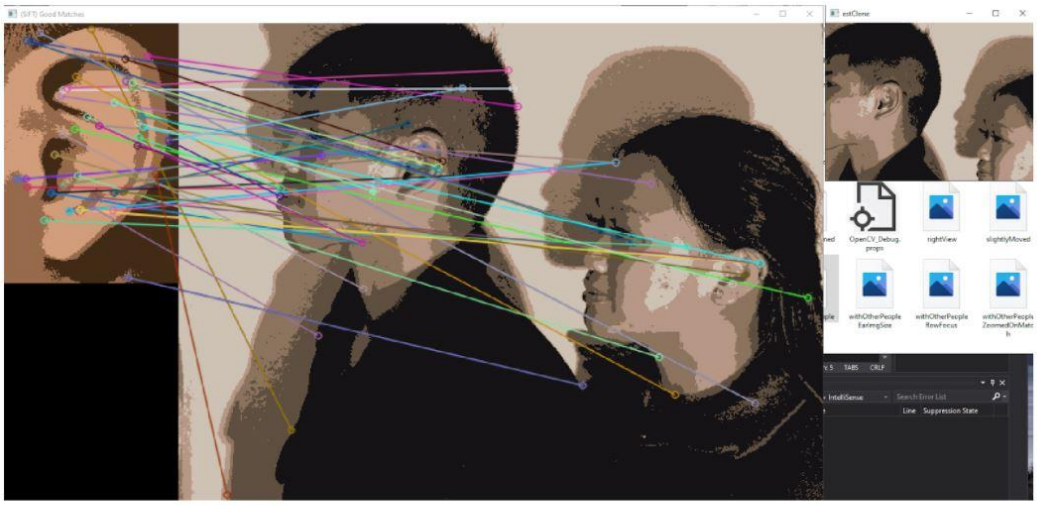
Below is our SIFT key match being used on images with both k-means algorithms being applied.



We were pretty happy with these results, but still needed to test them with the larger image.



K-Means + Sift (Iteration: 1) | K for anchor: 3 | K for test: 5

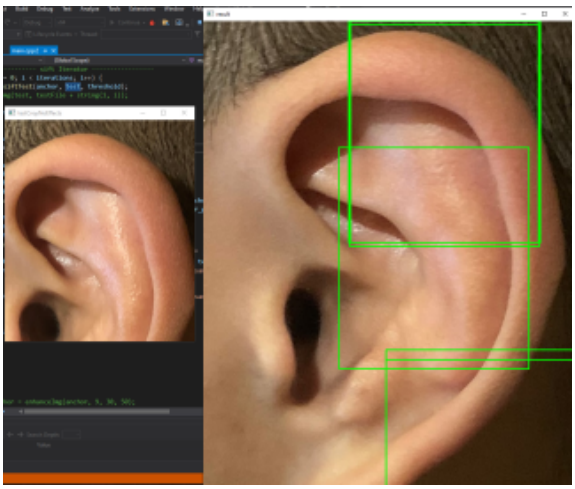


K-Means + Sift (Iteration: 2) | K for anchor: 3 | K for test: 5

K-Means + Sift (Iteration: 3) | K for anchor: 3 | K for test: 5

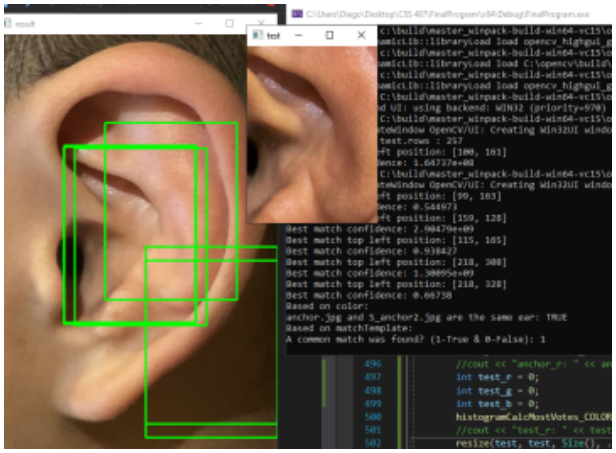
These results were really good. It not only gave us matches to the correct ear, but made crops and created a close up image of the actual target ear. From here we could then use SIFT or any other type of verification method to ensure that these two ears were truly a match.

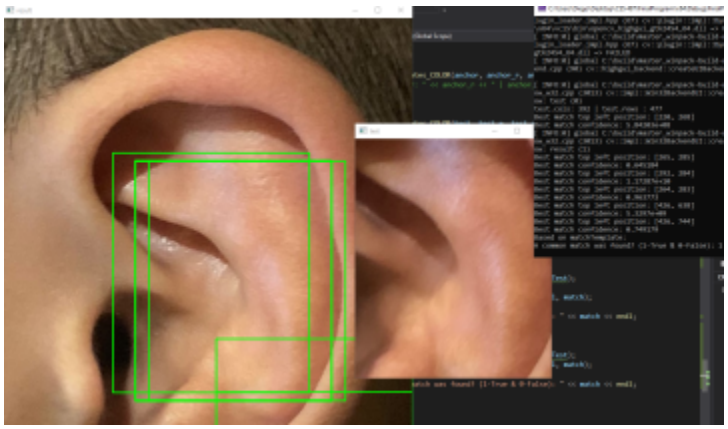
From this point we wanted to focus on verification between ears and while we could have just reapplied SIFT, we realized that SIFT was reaching its breaking point due to the images losing pixel quality due to the constant cropping. Our next idea was to experiment with matching template opencv methods.



Our results were kind of all over the place, and realized this was because our match template wasn't doing so well with different sized images, due to the nature of how the match template worked. This discouraged us, so we tried looking at something else.

Next we experimented with the most common color bucket, similar to the program 3 assignment we did for this class. We found that by doing so it didn't give us very good results across all images, but ended up working really well for our closeup image. This made us think that maybe we should include the color results, but only for the closeup images. But since this still didn't give us results across the board we went back to the match template and spent a considerable more time towards it. We tackled the size/scale issue that match template innately has by hard coding size changes. This took quite a bit of time trying to find the sweet spot for each image type but gave us good results. You see these resizing values in our code and how they impacted our results were very significant.





It was around this time that our deadline started nearing, and we decided to submit what we had despite not fully completing our goals.

What our Batch Files are Doing

NOTE: To Use our code, go to main method and comment/uncomment the “Edit this part”, be sure to only have one section uncommented only

- “calibration” batch file shows our progress on handling angles. Since we chose to work from scratch, our code reflects as much as we got, which was identifying the distCoeffs, rotation vector, and translation vector for an image at different images. For the sake of computing time and reduction in repetition we only include one of our calibration images, but we have many more with the same image just at different angles.

- The next following files shows our progress on handling size and verification
- “Small” verifies the anchor in the test image
    - Resizes both image so that test is relative size to the anchor
    - Acquires most common color between each image and find most common color bucket (No K means)
    - if both buckets are at the same coordinates in the 3d color histogram, show they are a match
    - Apply match template
    - If 3 or more match template squares overlap or close in proximity, print that they are a match
- “Medium” finds and verifies the anchor in the test image of medium sized image
    - Resizes both image so that test is relative size to the anchor (different size than small)
    - Kmeans + sift + crop to find most similar anchor ear image
    - Apply match template
    - If 3 or more match template squares overlap or close in proximity, print that they are a match
- “Large” finds and verifies the anchor in the test image of a larger image
    - Resizes both image so that anchor is relative size to the test (different size than small and medium)
    - Kmeans + sift + crop to find most similar anchor ear image
    - Apply match template
    - If 3 or more match template squares overlap or close in proximity, print that they are a match

What We Didn’t Finish

If we have more time, we will streamline the process and combine our codes. We rectify images to a common plane and search ears using a coarse to fine strategy, and verify if they are the same ears with thresholding key points between two images using SIFT or other feature matching techniques.
We also didn't finish fully implementing handling the angle of ears in our program. While our program did the calibration correctly from what we saw, we were heavily stumped on the computation to use the values that we saw and apply it to ears.

We also didn't finish testing the robustness of our program. The images we used were generally kept the same, and we didn't vary to using different people to see how robust our program would have been. We also didn't do that much testing against the SIFT+crop with k-means with an image with no matching ear meaning that outside of this scope of our project, we don't really know if it would even work at all.

We also didn't finish testing our program against true negative images.

Lessons Learned

Getting depth with stereo vision is a math heavy and error prone process. We first need to find the intrinsic parameters by calibrating cameras using chessboard images. Find the extrinsic parameters with key point matching to learn the relationship between the left image and right image. Then, we can project the images to a common plate to get the correct epipolar lines. Solve the correspondence problem, calculate disparity, and finally calculate the depth of the images. Having to handle each of these problems concerning stereo vision was very hard to do without using code samples.

SIFT keypoint matching is heavily impacted by a lot of different factors, ranging from brightness, contrast, color, size of image, and blurs. When we first started we thought the most we would be changing would be size, but as you can see that quickly changed as the project progressed.

Match Template is very picky on what it will successfully identify. Our handling of sizes and resizes, while did work to an extent, was in no way robust and would probably result in more issues if applied to images outside of our sample.

When dealing with image processing, it is very important to have a standardized size for test images in terms of pixels and resolution. With different image sizes, it not only affected our performance in the code, but also how well our algorithms were used.

A lot of the lessons in the class were easily reviewed and reapplied in our learning throughout this project. From using opencv methods to create different types of images(edge detection, blurs), all the way to what we had learned recently in our lectures about stereo vision.

We underestimated how difficult it would be to do this project, which we thought could initially be solved using a couple of image key features detection and matching. The scope and difficulty of the project grew at an incredible rate as we progressed through our project encountering problem after problem and this was something we were not expecting. It eventually led us to our current point of not being able to fully complete the project and while means we didn't accomplish our initial goal, we had learned so much more about opencv than we had previously.