Final Group Project

Hand-Detection Write-Up

12/1/2021

Quintin Nguyen, Akhil Lal, Matthew Cho

**Hand Detection**

**What We Were Trying To Accomplish**

When we first determined our project idea, we imagined a program that takes in a video of someone moving their hand around a screen and moving it in different positions. It would then output a video that is similar to the input, but with a red box around the hand as it moves throughout the screen. The hand's coordinates and type of hand would also be displayed in the bottom left corner of the video. We also originally wanted the program to detect what is and is not a hand and whether the hand (once detected) would be in a thumbs up position, ok sign, fist, or holding 1-5 fingers up. At the start, we thought it wouldn’t be too hard to use feature-matching with template images and keypoints to match each hand position with each frame of the video. We thought that this would be a very applicable idea as it can be applied to many different valuable situations, like improving accessibility of technologies, and especially important currently, where everyone is more conscious of what they come in contact with, interacting with technology without physical touch. To implement this program, we aimed to use only technologies and concepts learned in this class, such as OpenCV in C++. We also assumed that the camera and background remain static and that the hand (when in frame) must be properly lit and be in an upright position.

**Implementation and Results**

​​Throughout the implementation of our project, we ran into many issues and errors, and we tried different strategies to implement hand detection until we finally got our program working. When given an MP4 video, our program will first extract the background and use this background throughout the process. We extract the background by going through 30 random frames throughout the video and building an image made up of the average of each pixel throughout each random frame. We found that using 30 frames made the most accurate background without including too much of the hand and wasting too much time. An example of an extracted background from a video can be seen below:

On the left we have the first frame of a video of a busy street in India, where cars are moving in and out of frame. On the right is our program's computed background for the video.

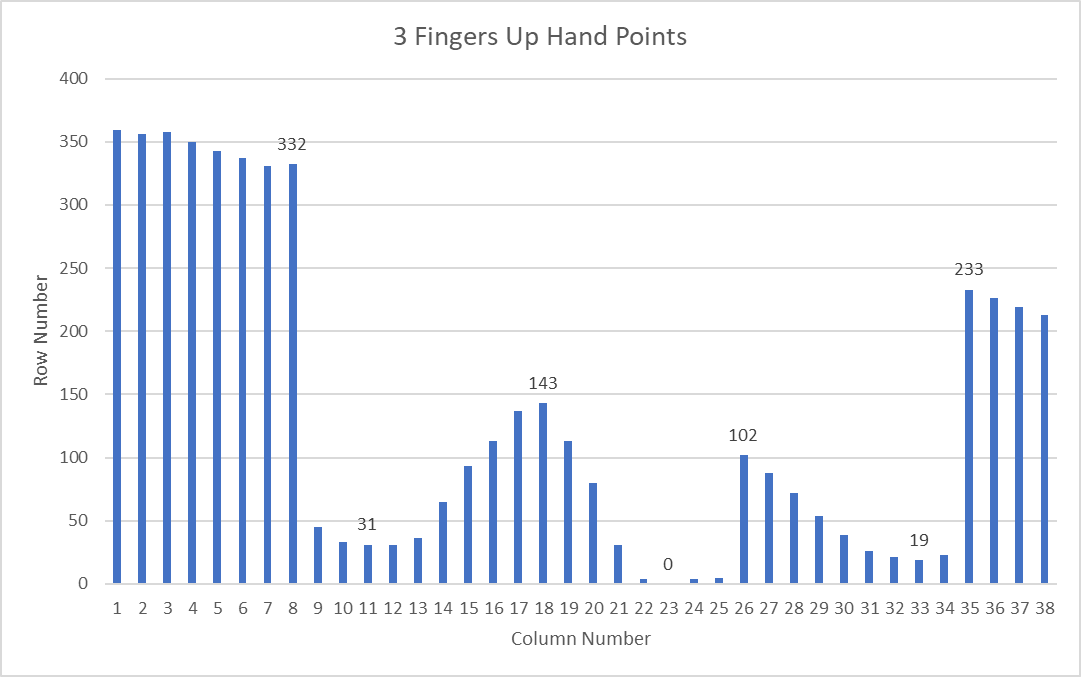
Then we prepare this background image and every frame here after. We prepare images throughout this program by applying gaussian and median blurring and increasing the saturation, contrast, and brightness. We applied gaussian blurring to get rid of much of the noise and also median blurring as it gets rid of noise while still maintaining most of the edges. Increasing the contrast, saturation, and brightness help make the hand stand out from the background and make it easier to create a mask using the background. We also skipped processing some frames in order to speed up the processing of each video. So, for every third frame in the video, we prepare the frame and isolate the hand by determining differences between the current frame and the previously computed background. For pixels different enough (using a decided threshold), they are assumed to be part of the hand and are set to white. If a pixel is not different enough from the corresponding background pixel, it is assumed to be a part of the background and is set to black. An example of this can be seen below:



Then, we find all the image contours (which are the continuous "blobs" of white pixels) in this newly created image representation and sort them by contour area. Then, we go through each contour, from biggest to smallest, until we detect the hand (in any position). When checking a contour, we crop just around it so we can focus just on this current contour. In most cases, the biggest contour will be the hand and if it’s not we just move on to the next biggest contour until we hit a certain threshold. An example of a cropped contour to be examined can be seen below:



We then find the top edge of the contour (up most white pixel). From here, we then determine the number of fingers in the contour. We do this by finding the local minimums and maximums of points in the contours. We noticed that this is an indication of the number of fingers the hand is holding up. For example, if there are 4 local minimums and 3 local maximums, then there are 4 fingers held up. Minimums represent the top of the image (which are the fingertips), and vice-versa, due to how images are structured (from top to bottom). See graph below of points taken from a 3 finger up contour.

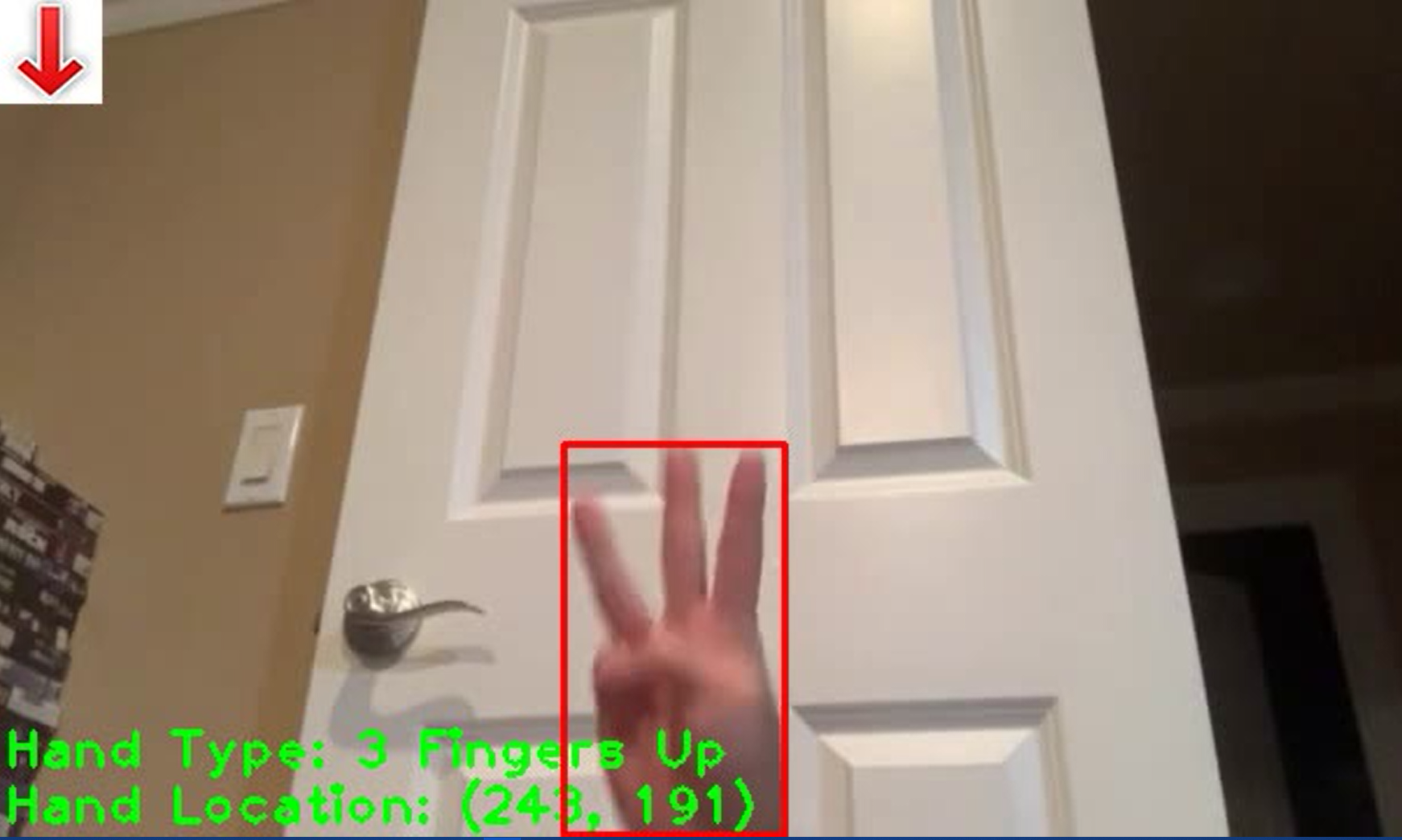


These values were obtained by analyzing the top white edges of a contour and seeing for each column, how many rows would it take to meet a white pixel. As you can see from the graph, these points display a very specific pattern that is prevalent in various hand positions. Each local minimum is a finger tip and each local maximum is the space between the finger tips. Here is another example from a 5 finger up contour. While this may not be the most robust method, it was able to fairly accurately detect the type of hand in each frame of video.



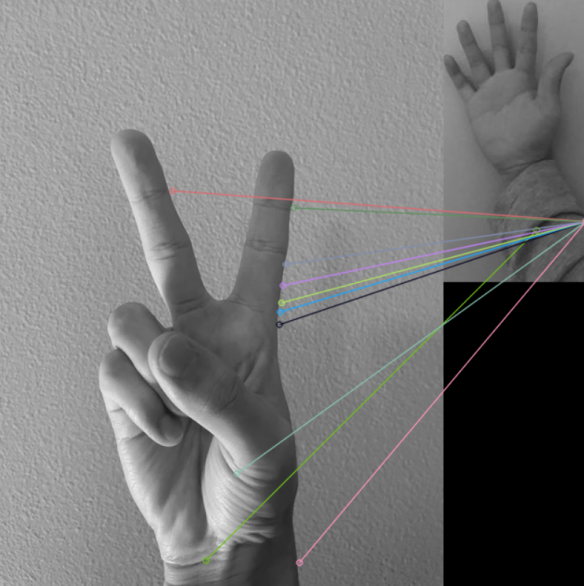
After determining the number of fingers the hand is holding up, this information, along with the coordinates of the hand, is displayed on the screen with a red box around the hand. If no hand is detected (-1, -1) is the shown location with a grey circle and no box. Our program is also able to detect if the hand is moving and which direction it is moving by comparing the current hand position to the previous hand position. In addition to displaying the hand coordinates, hand position, and box around the hand, the program also displays a symbol in the top left corner representing the hand's movement. If the hand is stationary (not moving), the symbol will be a red circle. If the hand is moving, the symbol will be an arrow pointing in the direction the hand is moving. Lastly, if a hand is not detected, then the symbol will be a gray circle. Screenshots from our output video can be seen below:





**Lessons Learned**

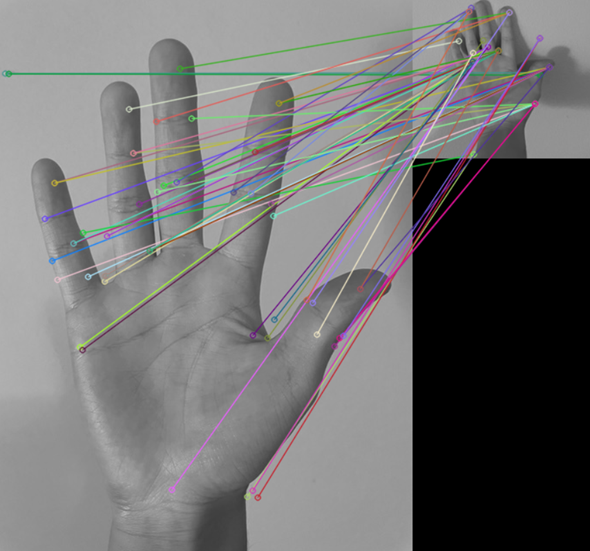
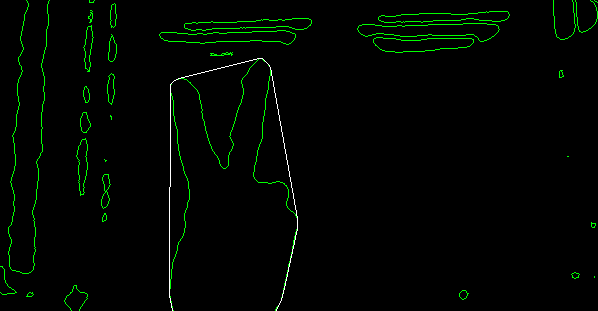
The lessons that we learned were that SIFT wasn’t a very good detector for our project and we weren’t able to detect a lot of the right keypoints using FLANN based matching. A lot of keypoints in the search image did not get properly matched with the key points in the template image. We tried using binary and grayscale images, but they still did not get matched correctly. Perhaps with more time, we could more accurately configure these keypoints and better match them accordingly. In the left image below, no matching keypoints could be detected at all and in the right image below, the key points in the search image are not matching at all correctly.

We also learned that image detection is a lot more challenging than we first thought. There are a lot of parameters and variables that we have to decide to work best for the situation at hand. For example, when preparing the image, we went through many iterations on what filters should be applied to the image before processing it and what values should each filter use on the image. It took a lot of testing to find values that worked best for the most images that made the hand stand out the most from the background. A lot of times, when computing a background image, parts of the hand get sampled and computed as part of the background. This leaves a dark stain where the hand was sampled from in the ExtractedBackgrounds function (see images below of before and after preparing the image). Choosing filters and testing values to minimize this grey area was something we had to test multiple times. We even looked into using a background segmentation tool like grabcut to hopefully better isolate the hand and get a cleaner mask of the hand, but eventually decided against it, due to the complexity and lack of time.

We also learned how to use a lot of different OpenCV methods. We learned how to manipulate the image how we want it and before settling on the current algorithm we used a lot of other tools to hopefully better segment the hand from the background and detect the background. We first tried basic template matching by comparing the number of similar white and black pixels in the binary search and template images, but this was really unreliable. Then we tried SIFT and while some pictures did work decently (see left picture). We couldn’t accurately detect the hand types, especially in video form, as there were too many key points that didn’t match up correctly. Then we tried using contour hulls, where each hull of the contour is supposed to represent a finger (see right image). But we couldn’t figure out how to extract this information in an accurate, reproducible, and precise manner. But with each failure, it led us down closer to the correct step until we finally settled on local min and max. And along the way, we learned how to use various OpenCV functions and we would explore how each tool worked and how to get information from the image. Knowing how to read an image and see what the computer sees, while exploring different OpenCV tools and Computer Vision concepts is one of the most important lessons we learned from this project.

We ran into a few issues here and there and have learned a lot from them. During the video output, there were a few false positives because of the background being randomly detected at times. Some objects that aren’t hands would show up as a hand being detected. We were also unable to detect a hand if it was rotated on its side because of the algorithm we chose to go with (local min and max) instead of SIFT. However, it can detect both the left and right hand. We ultimately know what’s causing the major issues of our current algorithm and with more time, we would be able to explore more complex methods that better detect the hand.

In terms of future improvements, we would need to better mask the image we are using, perhaps with a median algorithm instead of a mean algorithm in the ExtractBackgrounds. Or we could limit specific colors that get used as part of the background or look into the K-means algorithm to better segment the image. We also need a better algorithm to remove false positives and more accurately detect a hand from different angles. With our shape-matching algorithm it could be combined with using SIFT and FLANN based matching again by tweaking the algorithm and trying a different approach or perhaps looking into edge detection to better detect the hand and hand type.

**What We Accomplished and Areas of Improvement**

In the end, we were able to meet most of our original objectives. We were able to output a video that would detect the hand, track the hand’s movements, and be able to identify the number of fingers a hand held up. As planned, we were able to print this information onto each frame of the video where we had different symbols displayed representing the hand’s movement. We also were able to print a box that encompassed the entire hand and we were able to print the location and number of fingers the hand held up.

However, we were not able to accurately detect a thumbs up position, ok sign, and fist. And when a hand is displayed, the hand must be displayed in an upright position. Our current implementation only allows for the number of fingers held up on one hand to be detected, but perhaps we could implement other algorithms like SIFT or ORB to someday be able to track these hand positions. Using keypoint detectors and template images would definitely help with this current issue as it doesn’t matter the rotation or type of hand shown, as long as it shares similar keypoints with the template image, it would be able to better detect a hand.

The accuracy of our program is another aspect we hope to improve upon in the future, as there are a lot of false positives that the program thinks is a hand, but is actually just part of the background or is not a hand at all. We could potentially make this more accurate by improving upon the ExtractBackground function by instead of taking an average of frames, we could instead take a median of the frames which would minimize the chance that parts of a hand appears as a background image. We could also improve upon the ExtractBackground function by detecting a range of hand colors and limiting those pixels from contributing to the background image. As for minimizing false positives, perhaps we could use 2 algorithms to more efficiently and accurately detect the hand. Using shape detection to detect a hand is what’s used currently, but we could combine this shape detection with other algorithms such as SIFT/keypoint detection, or using color to minimize a lot of the false positives.

Also, because the program has to analyze hundreds of frames for each video, we hope to always decrease the time it takes to process each video and hopefully come up with a more efficient and faster algorithm.

**Credit**

These websites helped guide us to create our program. No algorithm was directly copied from these websites, but they instead showed us how to better use the OpenCV tools in our own program.

<https://learnopencv.com/contour-detection-using-opencv-python-c/>

<https://answers.opencv.org/question/32140/draw-largestrect-contour-on-this-image/>

<https://stackoverflow.com/questions/13495207/opencv-c-sorting-contours-by-their-contourarea>

<https://www.geeksforgeeks.org/find-indices-of-all-local-maxima-and-local-minima-in-an-array/>

<https://docs.opencv.org/3.4/d5/d6f/tutorial_feature_flann_matcher.html>

<https://vovkos.github.io/doxyrest-showcase/opencv/sphinx_rtd_theme/page_tutorial_feature_flann_matcher.html>