Gesture-Based Control of Augmented Webcam Experience

Ashwin Bhat
abhat4
Johns Hopkins University
abhat4@jhu.edu

Eric Chiang
echiang3
Johns Hopkins University
echiang3@jhu.edu

I. INTRODUCTION

As there is an increased focus on human-computer interaction, there is great emphasis on creating new, user-friendly ways of allowing humans to control computers [1]. In this project, we developed an augmented webcam experience that could be intuitively controlled by users. A large part of creating this environment was developing a robust method of finger-tracking to allow for the users to easily choose different filters to apply in the webcam video and give them a sense of more futuristic interaction than a simple touchpad or computer mouse. In addition, this setup required no extra hardware besides a webcam, which nowadays comes built-in to most laptops. The full implementation of this augmented webcam experience was done using eye tracking and finger tracking.

A. Related Works

Previous methods of hand-tracking that were very advanced or accurate often required extra hardware or were computationally expensive and made it difficult to do finger tracking on real-time video without significant latency. The Kinect sensor is often used for hand tracking because of the infrared camera that gives depth images. Some studies have used it to recognize hands and classify gestures [2]. One of the most advanced articulated hand tracking implementations using the Kinect sensor was done by Microsoft Research. This setup works very well and allows for fully-articulated hand tracking via depth images in real-time [3]. However, it does have a limitation in real-world use because it requires an infrared camera, which most people do not have. The Kinect sensor, while somewhat portable, is also not as seamless an experience as using a laptop integrated webcam or even a USB-interfaced webcam. As that was our goal in making our augmented webcam experience, we steered clear of any camera device besides a simple webcam and made an environment where the user could have their fingers tracked and use their fingers to control the output of the webcam (change filters, remove filters, etc).

B. Our Approach

Our approach, in broad strokes, is below. The details shall be discusses in following sections.

1) Detect and track eyes in video feed using a combination of a cascading object detector and extraction of salient features around the eyes to track them if moved.
2) Track fingers in video feed using thresholding and morphological operators.
3) Apply facial masks/overlays.

II. EYE TRACKING

We describe here the method utilized for tracking of the eyes in the video feed. We used an algorithm that was known for its robustness in face tracking and has also been applied to eye tracking. However, while it is fairly robust the computational cost can become too expensive when running on every frame of a live video while also applying other image processing. This led to occasional lags or freezes in the output video. To reduce the cost of eye-tracking, we implemented a less expensive feature tracking algorithm to detect specific points in the region where eyes were detected. Later on, we decided upon using a combination of the two

algorithms. This shall be discussed in greater detail below.

A. *Viola-Jones Algorithm*

We first utilized the Viola-Jones algorithm, which involves using Haar Features and representing the image with an integral image.

For learning, the Viola-Jones algorithm uses a variant of Adaboost training and then utilizes cascading classifiers. The use of cascading classifiers is highly important as it allows for the use of simple classifiers to pick out negative images and then adds in complex classifiers to reduce to the false-positive rate to an almost insignificant percentage [4].

The Viola-Jones algorithm is often used for face-detection, but it can easily be used for eye-detection, as well. When used for eye detection, it is still highly accurate with a low false-positive rate. In all of our observations, the only false-positive was an instance when the eyebrows were detected as a second pair of eyes right above the actual eyes of the person in the frame.

Fig. 1: Eye Detection/Tracking with Viola-Jones Algorithm at Multiple Instances

However, while the Viola-Jones algorithm is rather robust and has a very low false positive percentage, it does begin to start being computationally expensive when run on every single frame of real-time video. This is even more of an issue when trying to detect the eyes while also tracking the fingers as described later on.

B. *Kanade-Lucas-Tomasi Feature Tracking*

In order to reduce the computational cost, we decided to combine the Kanade-Lucas-Tomasi (KLT) feature tracking algorithm with our existing Viola-Jones eye tracker. The initial difference was that we would run the eye tracker once, at the very beginning of the live video feed, instead of on every single frame.

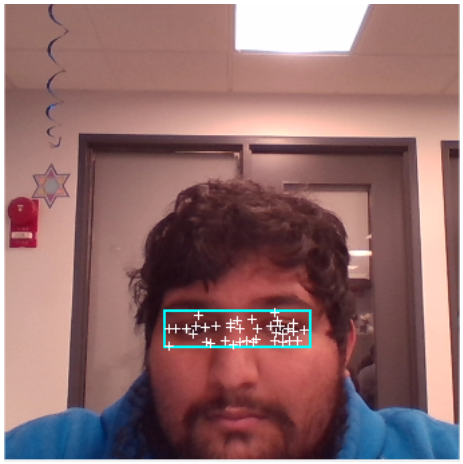

Fig. 2: Eye Tracking with KLT Algorithm

After that, we used the KLT algorithm for feature extraction to find the important points inside the bounded box where our Viola-Jones eye tracker found eyes in the video feed. The points were determined by finding corners using the minimum eigenvalue algorithm. With the salient features now known, those points were tracked and the geometric transformation as the points moved from frame to frame was estimated [5]. This let us keep a bounding box around those same set of points as they moved in the image.

The bounding box was important to have because it was used in the calculations for face masking and face filters. One of the main benefits of using the KLT algorithm, aside from the increased efficiency, was that the points were a bit more resistant to the changes in angle and rotation that we found to be shortcomings in eye tracking with the Viola-Jones algorithm.

This method of a single instance of eye tracking followed by the KLT algorithm to keep track of

the eyes worked fairly well, but it did have a few problems. The primary one was that if part of the eyes were moved off screen momentarily, the points that were being used to track the eyes were lost. If the eyes were moved back on screen those points would not be detected again by the basic KLT algorithm.

Another issue was that after many frames and movements, some points would begin to be considered outliers based on the calculations of the KLT implementation and then the entirety of the eyes were no longer detected. Therefore, despite the increase in speed of the eye tracking and the decreased computational cost, there was a fundamental issue with this eye tracking method due to the decreased accuracy.

C. Combined Approach

In order to compensate for this issue, we decided to make a method that compromised on speed/computational cost and accuracy. In this method we used the Viola-Jones eye tracking more often than just at the start of the video feed and were able to have the eye tracking and KLT feature tracking work together to achieve better speed than just the Viola-Jones algorithm and better accuracy than when just the KLT algorithm was doing the frame to frame tracking.

After initially finding the eyes and then using KLT to track the salient points in the eye region, the eye detector would once again be run on a single frame in the video. Based on some trial error, we found every 90 frames to be a good frequency of running the eye detector on the video. Once the eyes were detected, the KLT algorithm was run to find salient features in the newly found eye region and then those new points were used to for the next 90 frames before the eye detector was utilized again.

Naturally, this does slightly increase the computational cost since the Viola-Jones eye detector has to be run more than the single time it was used when allowing the KLT tracker to do the brunt of the work. However, it was still much more lightweight than running the Viola-Jones algorithm on every single frame as we initially tried. That method used the eye detector 90 times in three seconds, whereas this combined approach only used the eye detector once in three seconds.

Algorithm 1: Eye Tracking-Combined Method

```
Input: V - A live video stream
Output: BB - A bounding box containing
        eyes (changes every frame)
Initialize Video Stream V
Take a single frame of video
I ← V(current)
Detect the eyes and initialize a bounding box
BB ← detectEyes(I)
Initialize list of salient features (corners) as f
f ← corners(BB → I)
Initialize a list for previous points as prev
prev ← f
Initialize a forward transformation as tf
while V continues do
    if frameCount mod 90 == 0 then
        I ← V(current)
        BB ← detectEyes(I)
        if eyes are detected then
            f ← corners(BB → I)
            prev ← f
        end
    end
    I ← V(current)
    f ← findPoints(I, prev)
    tf ← findTransform(prev, f)
    Moves the bounding box to new eyes
     region.
    BB ← forwardTF(BB, tf)
    prev ← f
    return BB
end
```

Another benefit of this combined method was that it improved the accuracy of our usage of the Viola-Jones algorithm. The Viola-Jones eye detection/tracking returned false negatives when a person's eyes in the video frame were rotated or angled away from the camera. However, with this combined method, if such an issue were to happen when trying to re-detect the eyes, the previously found salient features from the KLT algorithm would continue be used. Thus, if eyes were detected at some point previously, the KLT algorithm still kept track of them and reduced the rate of false negatives. Therefore, this combined approach improved the robustness of the eye tracking.

We also implemented facial overlays as part of the augmented webcam experience and to have a fun feature. Depending on the result of the fingers found/tracked as described in the section below, a different face filter was placed into the video feed. The position was determined based on the bounding box that was continually returned by the eye tracking algorithm with some shifts depending on the type of filter. For example, a beard filter would be below the eye bounding box, while a sunglasses filter would be able to go near or in the bounding box.

III. FINGER TRACKING

We describe here our approach to counting the fingers shown on a hand. On a high level, we first thresholded the hand to isolate it. We then used morphological operators to isolate the palm of the hand and subtracted the result of this from the original thresholded binary image. This resulting binary image contained only the thresholded fingers which were then counted. The subsequent sections will explain in-depth each aspect of our method.

Algorithm 2: Finger Tracking

Input: V - A live video stream
Output: F - Tracked fingers
Initialize Hand Image as H
Initialize Palm Image as P
Initialize Finger Image as $fing$
Initialize Tracked Fingers as F
while V *continues* **do**
  $H \leftarrow thresholdHand(V)$
  $P \leftarrow isolatePalm(H)$
  $fing \leftarrow H - P$
  $F \leftarrow findFingers(fing)$
  return F
end

A. Thresholding Technique and Preprocessing

First, using the bounding box from the Eye Tracking method as described above, we assumed that the area directly below the bounding was the subject's body, so we immediately masked that out in order to decrease the complexity of thresholding the hand cleanly.

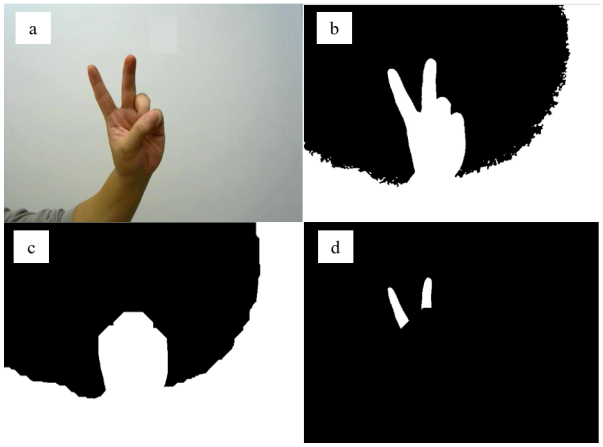

Fig. 3: (a) Image captured from USB-interfaced webcam, (b) Thresholded image, (c) Palm Isolation, (d) Finger Isolation

We experimented with a couple different thresholding techniques to isolate the hand.

We first attempted to utilize a skin-tone based thresholding technique describe by Jusoh et al. which involved thresholding the hue component of an image to get candidate areas of skin in the image [6]. These candidate areas were then used to determine areas to do RGB thresholding. However, with our testing we found this method had a couple of limitations in our application. Firstly, with the steps involved in this thresholding technique, we found that there was a noticeable delay in processing images. In some cases, there was noticeable hanging while performing real-time thresholding. Secondly, there were issues with detecting a range of skin colors. The technique worked best with peach-colored skin in well-lit conditions; without those conditions, the shareholder would not function accurately.

We decided to use a traditional intensity-based thresholding technique using gray-scale images. With stable lighting and a mostly homogeneous background, we were able to threshold hands found in the images quickly and consistently as seen in Figure 3a.

After the hand was thresholded based on intensity, we applied an algorithm that filled holes in binary images (Figure 3b) in order to prepare it for the next steps.

B. Binary Morphological Operators

Once the hand was isolated with thresholding, we used morphological operators to determine the count the hand displayed.

1) Palm Isolation: The first step was to isolate the ”palm” of the hand. In order to do this, we first utilized binary erosion with a large circular structuring element. This allowed us to effectively erode any fingers extending from the palm. We then used binary dilation on the palm in order to smooth its edges, increase its area slightly, and ensure that it acts as a proper mask of the palm for the next step as shown in Figure 3c.

2) Finger Isolation: We subtracted the ”palm mask” from the original image to get a rough binary image of the finger and applied one more binary erosion with a small circular structuring element to eliminate any remaining artifacts and noise in the binary image. Lastly, we thresholded the remaining binary elements by area in order to isolate the final position and count of the fingers as shown in Figure 3d.

IV. MASKS AND IMPLEMENATION

With algorithms for eye tracking and finger tracking, we could properly implement our gesture-based masks.

Masks were overlaid on the frames of video read in. The masks were scaled proportionally to the size of the bounding boxes drawn around the eyes. This ensured the masks scaled with the user properly. We then applied an offset also proportional the bounding box size to ensure the mask stayed in relatively the same position as the user moved.

We combined both algorithms into a script that read in frames from a USB-interfaced webcam and applied the mask in real-time using our eye tracking method. Users were able to put up a number of fingers, from zero to five, in order to switch masks as shown in Figure 4.

One limitation we came across, as described earlier, was significant slowdowns due to the computationally expensive behavior of the Viola-Jones Algorithm. At times, the webcam output would skip frames, and it would sometimes even completely freeze. Our solution to this issue was to use the KLT algorithm in conjunction with the Viola-Jones algorithm. This significantly increased the speed of our operation as described below.

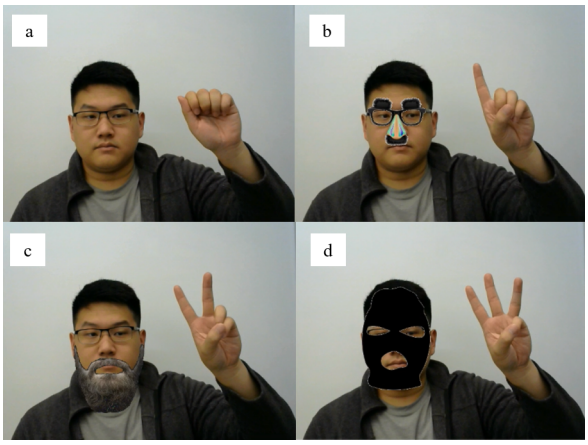

Fig. 4: Output from our implementation. (a) No fingers hid all masks, (b-c) showing different fingers changed the mask being used

V. EXPERIMENTAL RESULTS

While much of our results are qualitative, as in we were observing the output video response to ensure that fingers and eyes are being correctly tracked, there were some attributed that we looked at in determining the speed/accuracy of our system.

The primary accuracy we looked at was the accuracy of eye tracking because in the course of our experimentation we implemented three variations of eye tracking. In our observation, the Viola-Jones Algorithm, when not running alongside the finger tracking ran rather quickly and detected eyes when they came into the image frame. One of the main problems with the accuracy was that too much tilting or rotating of the subject’s head would result in the cascading object detector no longer detecting the person’s eyes, even though they were visible in the image.

The implementation where we used the KLT algorithm to do continuous eye tracking based on feature points found in the eye region from the cascading object detector being run on the video a single time, was also flawed. The problem here was that since feature extraction was based on those points in the eye region from the very first time, after a lot of motion and movement of the head, some of those points were lost, and would not get

re-detected. While the eye detection/tracking with this method was slightly more resistant to rotation than just the cascading-object detector, losing the eyes over time was not useful. Furthermore, the lack of re-detection prevented our design from being robust because a person moving half their head out of the image would only have one eye still tracked even when their entire head moved back into the image.

The final combined method worked best in that regard, because we catered it to meet our design requirements. With the cascading object detector being re-run every 90 frames, the eyes would always be re-detected if lost momentarily, and then the important features extracted by the KLT algorithm were updated when the eyes were re-detected, so if any salient points had somehow been lost that was rectified at a fixed interval.

The primary factor in determining the speed of our system was the frame rate of the webcam video after it was processed and outputted with tracked eyes and fingers. As mentioned previously, when only using a cascading object detector for eye detection in every single frame it resulted in frame-rate drops. Table 1 quantifies the average frame-rate in the different versions of eye tracking combined with finger tracking. Each of these frame-rates is the average of 20 samples. Each sample's frame-rate was the average observed for 3000 frames of video with the roughly the same movement of the test subject's head and hand.

TABLE I: Table of the frame- rates (in fps) in testing with Eye Tracking and Finger Tracking (with differing Eye Tracking Methods

Cascading Object Detector	Viola-Jones + KLT
$12 fps$	$23 fps$

As evident, the combined Viola-Jones and KLT Algorithms approach has a higher frame-rate and was subject to less frame-rate dropping. This was a fast enough frame rate to have good quality video output. The Viola-Jones eye detector alone is fairly quick as well, as the frame-rates would often approach 24 or 25 fps. However, since our end goal was to actually doing finger tracking and eye tracking together in real-time to create our augmented experience, the 12 fps frame-rate when tracking both was too low to have good video output. The combined approach had a solid 23 fps when doing both eye tracking and finger tracking so it was clearly superior.

VI. CONCLUSION

In our project, we were able to develop an augmented webcam experience that was fairly robust for finger tracking and eye tracking. Due our combination of the Viola-Jones Algorithm with the KLT Algorithm and our computational inexpensive method of finger tracking and gesture recognition, we developed a system that works in real-time without noticeable frame-rate dropping or latency. Our next steps would be to expand this environment beyond our novelty application of an augmented webcam experience. Due to its speed and lightweight computational cost, we believe it could be applied to further intuitive control of systems besides the just the webcam. Using the webcam, we could use our finger tracking as an interface for controlling the entire computer. For example, because we can locate the fingers in the image, we can use specific fingers as cursors or even use the webcam as a keyboard input based on where the user move their fingers in the perspective of the video feed. Movement of a finger in specific directions in the webcam view could also be used to scroll in windows or switch windows, meaning the user can control what they see on their screen simply by waving their fingers around. These are just a few possible situation we can apply our system, and our next step would be exploring the implementation of these.

VII. ACKNOWLEDGEMENTS

We thank the TAs and CAs for helping us learn and understand the material. We'd like to especially thank Dr. Austin Reiter for his guidance in the class and on this project.

REFERENCES

[1] Rautaray, Siddharth S., and Anupam Agrawal. "Vision based hand gesture recognition for human computer interaction: a survey." *Artificial Intelligence Review*, 43.1 (2015): 1-54.

[2] Sharp, Toby, et al. "Accurate, robust, and flexible real-time hand tracking." *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, ACM, 2015.

[3] Li, Yi. "Hand gesture recognition using Kinect." *Software Engineering and Service Science (ICSESS)*, 2012 IEEE 3rd International Conference on. IEEE, 2012.

[4] Viola, Paul, and Michael J. Jones. "Robust real-time face detection." *International Journal of Computer Vision*, 57.2 (2004): 137-154.

[5] Bourel, Fabrice, Claude C. Chibelushi, and Adrian A. Low. "Robust Facial Feature Tracking." *BMVC*. 2000.

[6] Jusoh, Rizal Mat, et al. "Skin detection based on thresholding in RGB and hue component." *Industrial Electronics and Applications (ISIEA), 2010 IEEE Symposium on*. IEEE, 2010.