Hand Recognition and Gesture Control Using a Laptop Web-camera

Zi Xian Justin Yeo
Exchange Student with
Stanford University
450 Serra Mall, Stanford, CA 94305
yeozx@u.nus.edu

Abstract

Hand gesture recognition is a technology that is becoming increasingly relevant, given the recent growth and popularity of Virtual and Augmented Reality technologies. It is one key aspect to HCI, allowing for two-way interaction in virtual spaces. However, many instances of such interaction are currently limited to specialized uses or more expensive devices such as the Kinect and the Oculus Rift. In this paper we explore the methods for hand gesture recognition using a more common device – the laptop web-camera. Specifically, we explore and test 3 different methods of segmenting the hand, and document the pros and cons of each method. We will also cover one method for hand gesture recognition.

1. Introduction

Computer technology has come a long way in the past two decades. The things that can be done, and the time spent on electronic devices has increased tremendously as well. They have infiltrated every aspect of our lives; from how we learn, to how we share experiences with others. Although the devices have been advancing quickly, the methods of interacting with these devices have been largely neglected – until now. With so many aspects of our lives being affected by computers, people began to wonder if there was a better, more natural way to interact with devices, other than using the traditional keyboard and mouse. This led to the emergence of a new field: Human-Computer Interaction (HCI). The impact of HCI can be seen in the first iPhone, which revolutionized the mobile device industry with its intuitive touch screen. A more recent example of such a revolution is voice commands, through products like Google Now and Apple's Siri, which allowed for hands-free controls. It can be said that both of these methods of interaction succeeded because they were so natural, so familiar to us – it was far more natural to point and tap on an icon using our finger, than to shift a mouse to move a cursor to click on an icon.

As the industry places its focus on Virtual and Augmented Reality, the importance of HCI will grow even more. People will desire a level of interaction that befits the term "virtual reality" – it has to mimic reality, where we use our limbs, five senses and voice to interact with the world around us. Hence, starting with the most basic interaction, we would want to use our hands to move and 'touch' things. In other words, we need hand gesture recognition as the basis of HCI in virtual reality.

This paper aims to explore the existing options for hand gesture recognition in a common context. Most people nowadays own a laptop with a front-facing camera. If we could tap into this, we could possibly bring a more natural method of interaction to the masses. Moreover, as virtual reality devices become more common, the laptop camera may also become a viable complementary interaction device, capturing a field of view separate from the virtual reality device.

Section 2 explores past attempts at hand gesture recognition and Sections 3 to 5 explore the different methods tested.

2. Related Work

Several papers and projects have targeted the issue of hand gesture recognition. Francis et al[1] compared methods for gesture recognition in cars, evaluating accelerometers-based, glove-based, and Kinect-based approaches. Mitra et al[2] analyzed more computationally heavy methods using hidden Markov models and finite state machines. Ghotkar et al[3] presented a novel approach to hand segmentation and gesture recognition using different color spaces.

The methods proposed by Francis et al required additional hardware, while those proposed by Mitra et al were computationally heavy, requiring classification and processing time. Our goal was hence to follow the example of Ghotkar et al, and explore the more basic methods of hand segmentation and gesture recognition available, applying them to execute simple controls on a laptop.

3. Technical Overview

This section highlights the key components that make up a gesture recognition system.

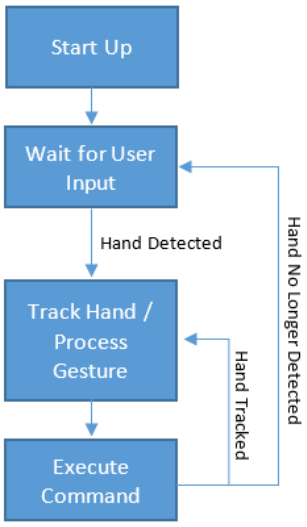

Figure 1: System flow chart.

A simplistic overview of a gesture recognition system is given above. Depending on the method of detection used, there may be a need for an additional calibration step immediately after start-up.

As can be seen from the flow chart, until a hand is detected, the system will constantly be scanning and segmenting, attempting to identify the presence of a hand. The focus of this paper is on this stage; how to clearly identify a hand given a non-simple background, simulating the situations of the everyday laptop user. These methods are

Once a hand is detected, it moves on to gesture recognition. The system tracks the position and state of the hand, and based on how the user moves, determines the command to execute. Following command execution, the system will wait for the next gesture. At any point of this stage, the user can choose to exit gesture control by hiding their hand – the moment the system can longer track or detect the hand, it will return to standby, waiting for the next user input.

4. Hand Segmentation Methods

The basis for recognizing hand gestures is recognizing if there is a hand in the image. Hence, this paper will place substantial emphasis on the methods of segmenting the hand from the background of the camera (or video) input.

The methods below were chosen with two criteria in mind. First, simplicity. From past works, it could be seen that complex algorithms, while more robust, were significantly slower and required a more complicated set-up, such as calibration and classifier training when starting the system in a new environment. The second criterion is flexibility. Our use case involves the average laptop user – who may be sitting in any environment and with a background that may be complex. It is hence desirable to have a segmentation method that would accommodate for varying non-simple backgrounds, despite only using a single RGB camera.

4.1. Canny Edge detection

This method was the first to be tested. Intuitively, human skin color, especially on the surface of the palm, tended to be different from most surfaces one would find in a background. This allowed for color contrasts between the hand and the background, which could be interpreted as edges.

Canny edge detection is done using OpenCV, with the samples[1] as reference.

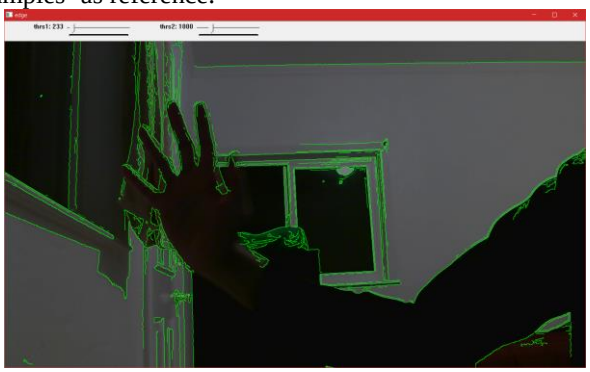

Figure 2: Canny edge gets most of the outline.

As can be seen in the above image, canny edge detection easily forms the outline of the hand. However, there are limitations to the canny edge method.

First, it is prone to noise. It will be difficult to accurately extract the edge contours of the hand from the background, especially if the background has many edges as well. One method to do this would involve some form of shape matching, possibly using a sliding window. However, this is likely to be less rotation- and size-invariant.

A second drawback of the edge detection method is the susceptibility to poor lighting conditions. In the image, it can be seen that some edges are missing, especially since both foreground and background are equally dark due to the lack of lighting.

4.2. Background Subtraction

Background Subtraction involves having the program read multiple frames and subtracting the aggregate values from future frames. In theory this would work in any

[1] See https://github.com/Itseez/opencv/tree/master/samples/python

environment as the program would 'calibrate' itself on start-up. As long as the user is the only moving object in the room, we would be able to detect their hand. There is the possibility of detecting other body parts if the user is far from the camera, but we will leave addressing this issue to a later project.

As an initial investigation, the methods of background subtraction used were the few built-in OpenCV functions such as BackgroundSubtractMOG. It was found that these methods worked fairly well without much customization.

Figure 3: Background subtraction can work well

As can be seen in the above image, background subtraction has the potential to work very well. That said, it is also not an infallible method.

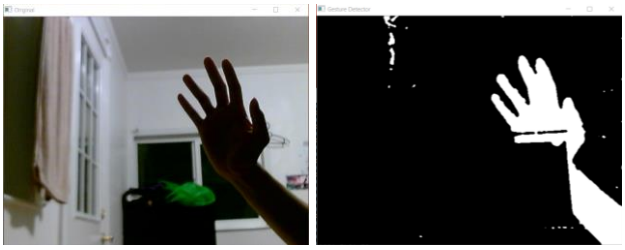

Figure 4: Background subtraction is also susceptible to dark areas and poor lighting

First, background subtraction needs a short calibration period. In this time, the user should not be in the camera view, lest he be read as part of the background. This is a minor issue and can be resolved with careful use.

Second, background subtraction is also susceptible to poor lighting conditions, and is especially bad with dark areas as well, because the lack of light will lead to both foreground and background being similarly dark colors. Given that edge detection also faces this issue, it can be said that dark environments are a key obstacle in using color space to segment the hand. One way of addressing 'dark' areas is to reduce the threshold; but this could introduce noise, and the right threshold may differ with the environment, making this a difficult balance to strike.

4.3. Calibration and Thresholding

The third and last method attempted was calibration and thresholding. This method was inspired by a previous work by Simon Andresen[2]. This method involves having the user place his hand in front of the camera in a way that covers the calibration boxes displayed by the program (see below). The program then gets the mean color values of each box, and saves these for future comparisons. When attempting to segment the hand, we can identify areas that differ from each of these color values by a given threshold. By compiling the segmentation given by each box's color value, we should be able to obtain a more complete segmentation of the hand; one that accounts for the variations in color across different regions of the hand.

Figure 5: RGB thresholding has both false positives and negatives.

The first attempt at this method was using RGB threshold values and OpenCV's inRange() function. This method did not work out well, with many areas of the image being segmented wrongly or poorly.

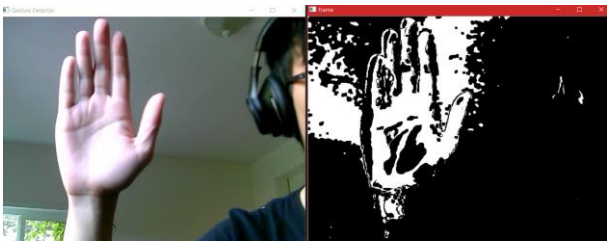

Figure 6: HSV performs better than RGB

The second attempt at this method converted the RGB values to HSV, and applied the thresholds more tightly on the Hue values. The reasoning for this was that the main differentiating factor of the hand was its hue, and the variance in how a hand appears in different environments could be attributed to saturation and value. After some trial-and-error with the threshold values, the result above was obtained.

Although the calibration and thresholding method has the potential to be fairly accurate in segmenting the hand, it turned out to be very sensitive to the environment. In our testing, the threshold values that were required to segment the hand properly varied vastly between environments. There was an environment in which green was mistakenly

[2] See http://simena86.github.io/blog/2013/08/12/hand-tracking-and-recognition-with-opencv/

segmented as part of the hand, and another where parts of the hand were not segmented because of shadowing.

4.4. Method Chosen

From the above investigations, it can be seen that all three methods were susceptible to environmental changes, especially where lighting was involved. This could be viewed as the drawback of using a single RGB camera, which only provides us with the means to differentiate using color. Even if calibration was used to adjust the thresholds somehow, it is worth noting that lighting can differ even within an environment – a hand beside the window would have a different RGB or HSV from a hand in a darker corner of the same room.

Of the three methods, it was determined that background subtraction is the most robust and simplest to implement. The fact that it did not rely on the exact color of the hand meant it was less susceptible to variations in lighting within the environment. It also returned the segmentation in a binary image, which made the gesture recognition steps much simpler. The primary weakness of this method was dealing with dark areas (i.e. the need to light up the hand), which would be the main constraint moving forward.

5. Gesture Recognition

As mentioned earlier, gesture recognition can only come after segmentation is done. Due to the segmentation methods not yielding desired results, this section is not the core focus of this paper.

We will explore one method of identifying simple hand gestures, and implement 2 basic gesture controls: cursor movement and mouse click.

5.1. Convex Hull Method

This method was inspired by an online tutorial by ifheqhar[3].

The convex hull method takes the outline of a shape, and identifies the convex and concave (defect) points along the outline. These points provide us with a rough idea of the shape of the object. In the case of a hand, it should have 5 convex points (one for each finger) and 4 defects (one between two adjacent fingers).

Using this method, we can identify the number of fingers the user is showing to us by the number of convex and defect points. For example, if there are 2 convex and 2 defect points, it is likely that the user is showing us 2 fingers.

In our implementation, we used OpenCV functions such as findContours(), convexHull() and convexityDefects() to obtain the convex hull and defect points. Below is a sample output of the centroid and number of defects for a given detection.

Figure 7: Example of console output. Numbers in brackets are (x, y) coordinates. Number on the right is the number of defects.

Two issues can be seen in this implementation. The first is that the number of defects is higher than expected, and is likely to be due the shape of the hand not being smooth. The presence of noise in the hand's contour led to additional defects. The second issue is that the centroid of the shape is affected by all segmented areas, hand or not. In other words, if the arm is part of the image, it could affect the rate of change of the centroid's location relative to the amount the hand is actually moving. This would impact gesture controls that rely on the hand's position.

5.2. Gesture Controls

The basic gesture controls were implemented using the centroid of the detected hull, and the number of defect points.

The program tracks the centroid from the moment a hand is detected, and then shifts the cursor based on the movement of the hand. As the centroid moves, the cursor is moved the same number of pixels.

Similarly, for the mouse click, the system tracks and stores the number of defect points, compares it with the next number detected. Due to the instability of the defect point detection, the mouse click is set to trigger whenever the number of defects falls below a given threshold – instead of when a given number of fingers are shown.

6. Conclusion

While the system implemented has much room for improvement, it can be said that the investigative process was a success. This paper shows the pros and cons of the simpler methods of hand segmentation and recognition, and the limitations of working with a single-view camera, with a key issue being dealing with lighting.

[3] See http://creat-tabu.blogspot.ro/2013/08/opencv-python-hand-gesture-recognition.html

References

[1] J. Francis and A. B K, "Significance of Hand Gesture Recognition Systems in Vehicular Automation-A Survey", International Journal of Computer Applications, vol. 99, no. 7, pp. 50-55, 2014.

[2] S. Mitra and T. Acharya, "Gesture Recognition: A Survey", IEEE Trans. Syst., Man, Cybern. C, vol. 37, no. 3, pp. 311-324, 2007.

[3] A. S.Ghotkar and G. K. Kharate, " Hand Segmentation Techniques to Hand Gesture Recognition for Natural Human Computer Interaction", International Journal of Human Computer Interaction, vol. 3, no. 1, pp. 15-25, 2012.