College of Engineering

COMP 408 – Term Project Report

Gesture Recognition by Fingertip Calculation and Hand Tracking

Participant information:

Ali Özkaya 50538
Sidem Işıl Tuncer 39170

Instructor:
Yücel Yemez

13.01.2019

Abstract

In our project we wanted to create a tool for applications that can be controlled by anyone without requiring an extra tool but the user's hand. Developing such systems with high accuracy and smoothness is a challenging topic and our aim was to develop a system that supports both simplicity and efficiency. In our system we preferred using a Webcam for image acquisition since it can be found in every home and we converted its output to HSL color model based on hue, saturation and lightness. Changing environment is an important issue in hand gesture recognition and external factors such as background, lightning and skin color changes from case to case. We got the best output from HSL color model after trying also other color models like RGB and HSV. In the application the most critical part was to collect color samples from the user's hand at the beginning of the program and use them to create filters to separate the hand from the background and create a binary image. Sometimes there were so much noise around the hand area, so we decided to subtract canny edges from the binary image and break the connection between the noise and the area of the hand. Afterwards we calculated the contour of the biggest white area with OpenCV methods. However, it was not enough to eliminate the noise and the objects of the colors similar to the skin color of the hand such as user's face. Therefore, we decided to add the method of background subtraction to our program. We used skin color + canny edge detection for the hand segmentation to calculate the contours and bounding box, then we applied background subtraction in that bounding box area and made everywhere else black. Consequently, we obtained a noiseless hand area as only the hand was painted white. After that procedure, extracting convex hull of the hand and calculating the cavities between the fingers helped us with the hand tracking and gesture recognition.

With a plain or complex background, a fixed camera and under stable lightning we achieved a good accuracy in hand gesture recognition and we also provided a smooth hand tracking.

TABLE OF CONTENTS

Section 1 Introduction

1.1 Objectives

Expected outcome of this project was to create a tool that enables input recognition with hand gestures. For this purpose, gesture recognition and hand tracking had to be real time and highly accurate. Because we chose to use a webcam for hand detection, our program does not support all backgrounds very well, but it has good performance when the lighting of the room is stable, and the camera is immotile.

1.2 Background

This application presents the experience of giving inputs to the computer with hand gestures, which is a recent UI implementation. Controlling applications with hand gestures removes the necessity of contact with an input device and feels more natural for the user. We chose webcam instead of more precise tools such as Kinect or Leap because we wanted our program to be more accessible, as the ownership of these devices are rare.

We have used one of the most common cues for segmenting the hand which is skin color. It is easier to implement when compared to machine learning methods and it is invariant to scale, translation and rotation changes. This is done by collecting skin color data and specifying upper and lower limits of the RGB and changing it to another color model. These limits are used in the elimination of the other pixels out of this range, so it leaves with the pixels in the hand area [2]. Other methods suggest using HSL color model as the base of segmentation. This method collects data from different regions of the hand and takes an average of upper limit HSL and lower limit HSL values of the skin pigment. Again, this range specifies the pixels that are going to be marked and the rest will be removed. This method may need readjustment for the noise due to the lighting and background [3]. It can be seen that color models are very sensitive to illumination changes so using only this method is not trustable. Background subtraction is an effective method and OpenCV provides a Gaussian Mixture Based Background Segmentation Algorithm named BackgroundSubtractorMOG2. Its adaptability to varying illumination is well enough and it is better for real time processing when compared to other methods such as GMG, MOG and KNN [1].

Section 2 Problem Analysis and Solution Design

Accuracy is affected by the number of light sources in the room, especially if the light source is present in the camera view. Similar colors being present on the background generally does not

interfere with segmentation, unless they overlap with the hand or take larger space than the hand. Most commonly the user's arms and face were also segmented as they share a similar color pattern with the hand. In Figure 1 we can see the noise around the hand area when we segment the hand using only skin colors. User's face and neck are also included in the white area which is supposed to represent only the hand. We have overcome this problem by subtracting the canny edges from the image. In the figure it can be observed that canny edges separate the hand from the noise, therefore the biggest white area becomes the hand itself. There is one issue that must be clarified which is user's face should not take more space than the hand, namely the hand should be closer to the camera so that the program works properly.

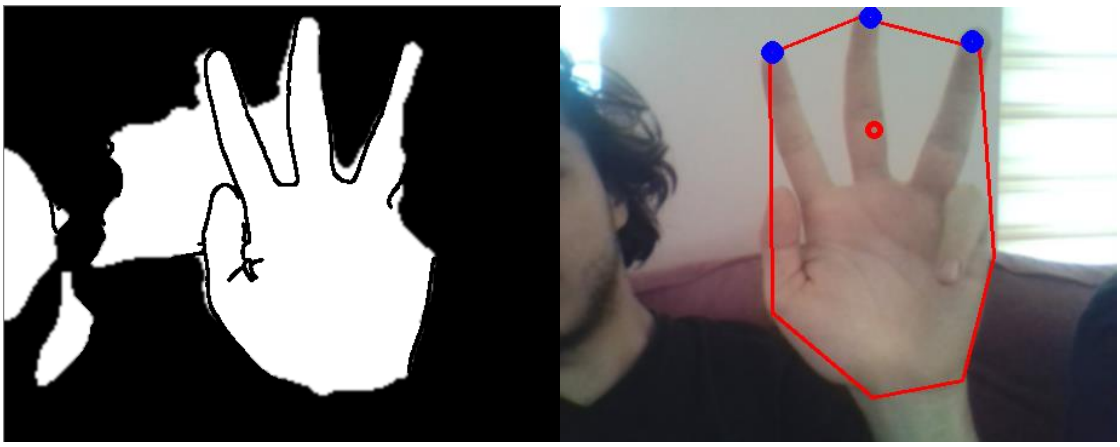

Figure 1: Noisy hand segmentation using skin color and subtraction of canny edges

The algorithm we used in our solution is as follows;

1. Take background and hand color samples for a certain time interval
2. Segment the hand using filters created from the color samples
3. Take canny edges of the original image
4. Subtract canny edges from the segmentation result
5. Calculate the inverse of the bounding box of the biggest contour
6. Segment the hand using background mask
7. Subtract bounding box from background subtraction's result
8. Take convex hull and calculate fingertip number using convexity defects
9. Track the hand using the convex hull

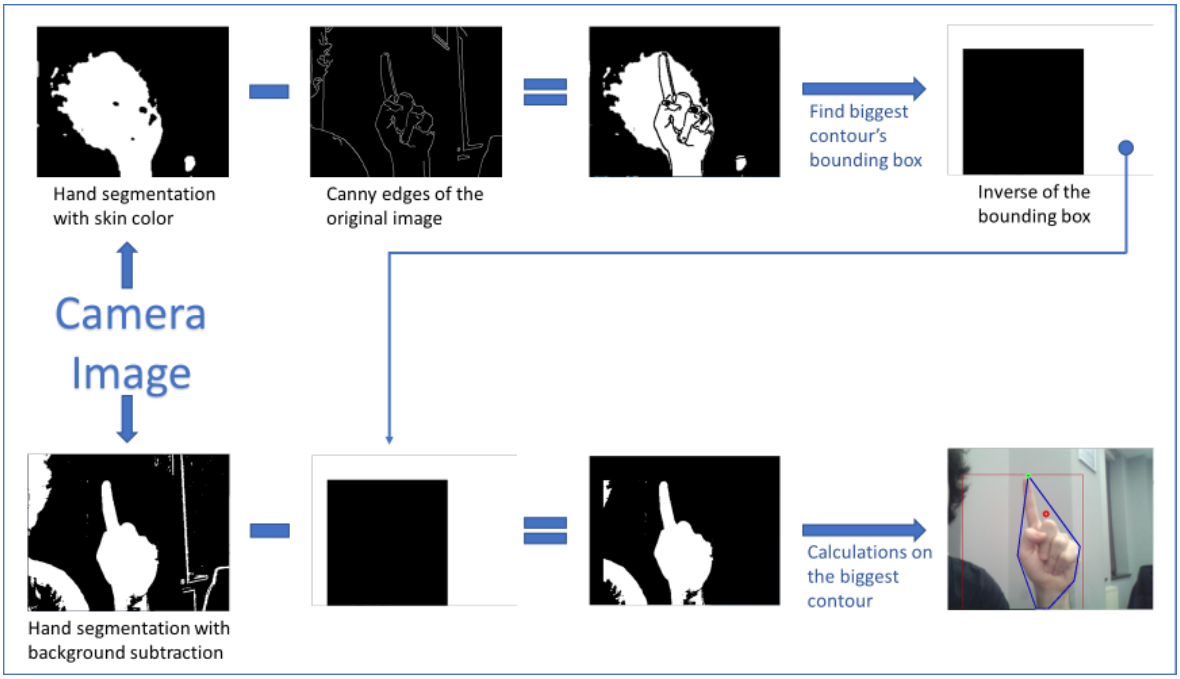

Figure 2: System Design

Section 3 Implementation and Testing

3.1 Implementation

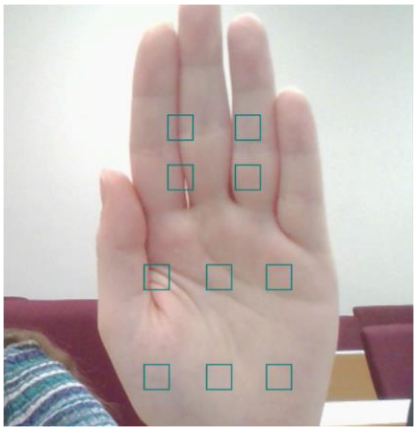

Figure 3: Color Sampling

The OpenCV plugin initially converts the visual input it receives from the webcam to an RGB matrix with width 540 and height 360. The resolution is low to achieve fast processing, because the application requires real time gesture calculation and tracking. Camera image is then flipped to give a mirror effect to the user, which makes the movement on the screen in the same direction with the movement of the user. Later a separate version of the matrix is created and converted to HSL color domain and all calculations for hand recognition are done on this matrix. HSL domain is preferred over RGB because it separates color components from intensity, which makes the lighting changes less problematic.

To segment the hand from the background, we calibrate the background by taking background samples from approximately 100 frames and using them for BackgroundSubtractorMOG2 model training. Then HSL color samples of the user's hand are taken. Because each user may have a different

skin color, a separate sampling is necessary for each execution. Color sampling is achieved by taking color samples from certain parts of the image matrix and creating a filter depending on these samples. These parts of the webcam image are displayed to the user as rectangles in the beginning of execution, the user is required to cover these rectangles with their hand as shown in Figure 3. 10 different samples are taken from the hand, then each sample is separately averaged by calculating the mean HSL values. 10 different filters are created by using these mean values.

The HSL matrix is further reduced in resolution and blurred for better segmentation. Then, the matrix is converted to black and white by applying each of the 10 filters and summing their filtration results. Because different locations of the hand, especially palm and finger tips may have separate colors, the summation of different filter results helps in better detecting the hand. The resulting binary image's noise is lessened with median blurring. Canny edge detection is applied to the main image and the result is subtracted from this binary image to better separate edges. The hand is white colored in the image at this point as seen in Figure 4, and it is separated from noise by selecting the largest white contour. This contour's bounding box is taken and stored to assist in background segmentation. We increased the thresholds from our previous implementation because in this implementation this part is meant to find the general location of the hand, it is not supposed to completely segment the hand.

Figure 4: Hand Segmentation

For background segmentation each new frame is compared with BackgroundSubtractorMOG2's background mask with a certain threshold. The result is a binary image where 0 represents background and 1 foreground. We subtract the bounding box which was calculated during color segmentation from this binary image. The color segmentation represents the location of the hand better than the background segmentation, because background segmentation considers all new objects as hand, while background subtractor distinguishes hand features better. By subtracting the bounding box, we eliminate other objects which are not the hand from the background segmentation, then we calculate the results biggest contour which represents the hand. This contour's convex hull and bounding rectangle are calculated, which are used for tracking. Hand tracking is done by taking the middle point between the center of the contour and the center of the bounding rectangle's upper side, this is because segmentation is likely to count the user's arm as part of the hand, which makes it hard for the tracked point to go higher than half a screen when only the center of the contour is used for tracking.

The convex hull is also used for gesture detection. Cavities between the fingers are used to detect numbers from 2 to 5, and these cavities are detected using convexity defects. Convexity defects are calculated with the convex hull and defects not belonging to the cavities between the fingers are eliminated based on angles and length. Endpoints falling on the same fingertip are removed by iterating over the points and comparing distances, so each finger cavity is only represented once. Endpoints of the cavities are used to mark the fingertips on the screen. Number 1 is calculated separately because no finger cavity is present in this gesture. This is done by calculating the corner of the contour with the lowest y value and comparing it with the height of other corners. In OpenCV a lower y value means the point is higher on the screen, because the origin is the left top corner of the image matrix.

The hand gesture is determined by storing the fingertip count results for a fix amount of time and selecting the one with the most occurrences. The camera image displayed to the user includes the tracking point, fingertip locations, bounding rectangle and the convex hull as seen in Figure 5.

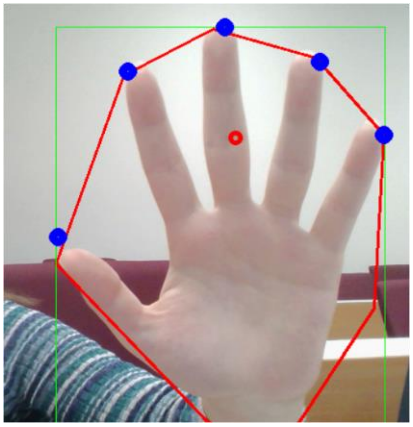

Figure 5: Displayed Image

3.2 Testing

The program works better in environments which are illuminated with white light when compared to yellow light. In a well-lit environment with white light the program works with 90% accuracy while in a poor-lit environment with yellow light it can go as low as 20% detection rate. In a suitable environment with good lighting the algorithm was tested with an easy background and hard background. The results for number five are given below. Color segmentation works noticeably better in the easy case, but the integration of background segmentation allows the gesture to be recognized in the hard case. Canny edge detection improves the result of color segmentation in the hard case by adding the edges of the hand. As better seen in the easy case, subtraction of the bounding box removes the user's body from background segmentation's result and distinguishes the hand.

color segmentation *canny edge detection* *color – canny* *background segmentation* *bounding box* *background – bounding box* *result*

Figure 6: Easy Case

color segmentation *canny edge detection* *color – canny* *background segmentation* *bounding box* *background – bounding box* *result*

Figure 7: Hard Case

Section 4 The Evaluation

Initial hand segmentation process had many performance issues depending on the surroundings of the user. We tried to minimize them suggesting new ideas beside the skin color method. We have seen that skin color method performed poorly when the colors in the environment are similar to user's hand an decided not to use this method as our main method to take the contour of the hand. With background subtraction we realized that user's hand was drawn much more precisely so we combined them using their strong sides.

Before we combined the strengths of 2 methods, we had designed our system according to 2 conditions. We were using skin color + canny edge detection method for mobile background and stable luminance. And we were using background segmentation method for static background and changing luminance. Of course, these changes in the environment were slight fluctuations. After seeing the results, we were not convinced because both of them had performed below expectations. So, we have changed our design such that it would work better than those 2 methods under certain conditions. These conditions are there should not be any movement in the background around the area of the hand and the luminance should change only slightly.

Section 5 Conclusion

We used OpenCV and C++ to create a program which takes camera view as input and segments the user's hand from this image to calculate number gestures. Initially the background and the user's hand color is sampled. The hand segmentation's result is improved by using canny edge detection and is then used to create a bounding box. This bounding box is subtracted from the result of the background segmentation to eliminate non-hand objects. The user's number gestures up to five can be recognized by mainly calculating and counting the cavities between fingers, and the location of the hand is tracked. We used low resolutions in hand recognition to increase the execution speed of the hand detection system, it performs with above 15 frames per second which can be considered

real-time. Overall if the segmentation process is successful the program works as desired, but the user's surrounding can greatly reduce the accuracy of segmentation. The result can be improved by using Leap or Kinect instead of a webcam. We did not want to develop our program using these tools because we wanted to write our own segmentation algorithm and webcams are abundant when compared to these tools.

Section 6 References

[1] Tibor Trnovszký, Peter Sýkora, Róbert Hudec. 2017. Comparison of background subtraction methods on near Infra-Red spectrum video sequences. In *Proceedings of the International scientific conference on sustainable, modern and safe transport*.

[2] M. K. Ahuja and A. Singh, "Static Vision Based Hand Gesture Recognition Using Principal Component Analysis, " pp. 402-406, 2015.

[3] Bunke, Horst. Progress in Computer Vision and Image Analysis. World Scientific, 2010. Chapter 22.

Section 7 Appendix