Contactless Fingerprint Detection Using OpenCV

Shubham Agarwal
Dept. of Information Technology
SRH Hochschule
Heidelberg, Germany
Shubham.aga26@gmail.com

Abstract— **The purpose of this project is to design a software to detect fingerprints, which can be used for biometric applications. Fingerprints will be detected contactless using a camera. The complete project is implemented using Open CV libraries for Image Processing, Visual Studio as compiler and QT Creator as IDE.**

Keywords – skin segmentation, contour, minArearect, Rotated rectangle, video capture.

I. Introduction

Past a decade biometric verification has become mandatory in many places such as offices, banks, airports etc. Scanners used for biometric verification needs finger to be placed on it and is time consuming. In this paper we will be detecting fingerprints with the help of a camera which further can use this fingerprint to verify the biometrics of a person. As a camera will be capturing the fingerprints, it will be contactless and all the fingerprints can be taken at once.

We will be using Open CV libraries for real time computer vision. We will first separate the complete palm by using skin segmentation methodology, then we will draw a contour around the palm and divide it such that we can get separate contours around the fingers, then we will draw a rotated rectangle around contour, further we will reduce this rectangle to only our finger tips by implementing some calculations.

The overall goal of this project is to extract the image of the tips of the fingers from our palm following the process shown below in fig 1.

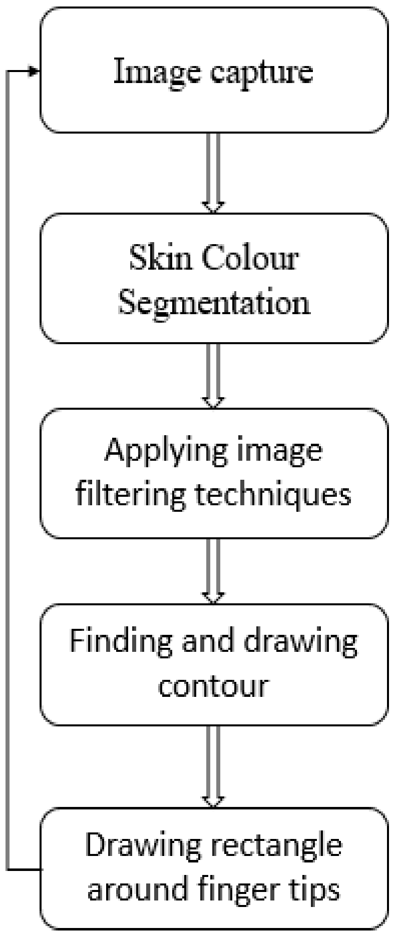
Image capture
Skin Colour Segmentation
Applying image filtering techniques
Finding and drawing contour
Drawing rectangle around finger tips

Fig 1. Architecture

II. Methodology

A. Skin Segmentation

The first step is to segment the skin color from each frame of the video. In order to do this first we will need to convert the read RGB frame to HSV frame. A crucial point to be noted is that the captured frames of video are represented in the RGB color format. When using the

RGB format, it is quite difficult to segment the object based on its color because colors in RGB are coded using three channels [1]. Using the HSV color format is a much better option because in HSV (hue, saturation, value), the color space acts similar to the way the human eye perceives colors. The hue channel represents the color, with hue being an angle from 0 -360 degrees. Saturation represents the grey portion in the color space, going to ranges from unsaturated shades of gray to fully saturated. It's value ranges from 0-100%. The value 0 denotes that the color is gray and if value is 100 then the color is primary color. The color fades with the decreasing saturation value. [2]. Value represents the brightness or the intensity of the color that also ranges from 0 – 100 %. At the lowest value i.e. 0 the color space will be completely black, and with the increasing value the color brightens up, thus enabling us to see different colors. The overall design HSV color space can be represented as shown in fig 2.

After the color conversion from RGB to HSV we threshold the frame using inRange(). In this we give the highest value and the lowest value of HSV range in order to extract skin color from our frame. The value for human skin is in between [H = 0, S = 58.65, V = 50] and [H = 50, S = 173.4, V = 255]. It returns us binary image of the frame where the area with white color pixels are segmented skin color area. With this we are successful in segmenting skin color from the frame. In case of our project we will get frame of segmented palm as shown in fig. 3.

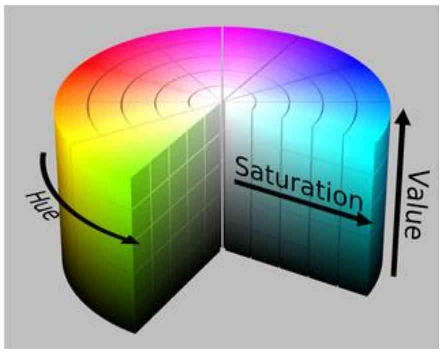

Fig 2. HSV representation [1]

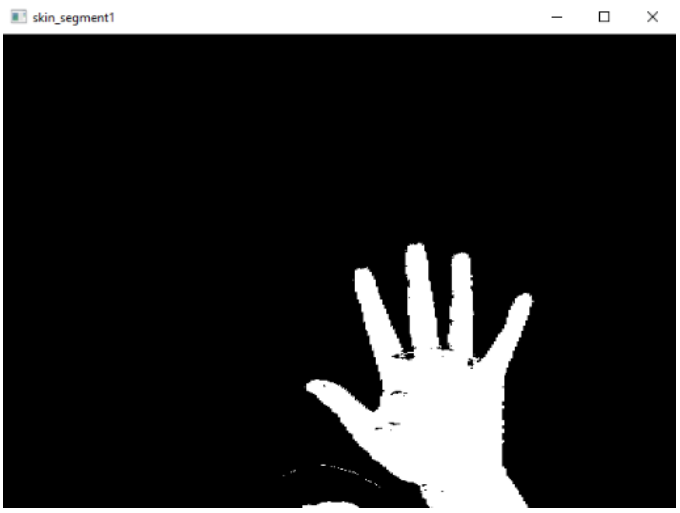

Fig 3. Image after hand segmentation

B. Image Enhancement

The image obtained after segmentation probably contains a lot of noise. We implement a few techniques to reduce the noise. Blurring the image to smoothen the image overall is the first step. This can be done with the help of an inbuilt Open CV function median Blur. This function works with the principle of going through every element in the image and replacing each pixel with the median of its neighboring pixels. This provides us with a smoothened image frame. Further, dilation or erosion techniques are made to use to enhance the image frame. This operation consists of convoluting an image with some kernel, having any shape or size, mostly a square or a circle. The kernel has a defined anchor point, which is usually the center of the kernel. As the kernel is scanned over the image, the maximal pixel value overlapped by is calculated and the image pixel in the anchor point position is replaced with that maximal value. As we can notice, this maximizing operation causes bright regions within an image to grow. The object in white gets bigger with dilation. An example of dilation is shown in fig 5. Whereas erosion is like the opposite of dilation which makes the object in white smaller as shown in fig 7 [3]. Fig 9 shows image of hand after applying both dilation and erosion.

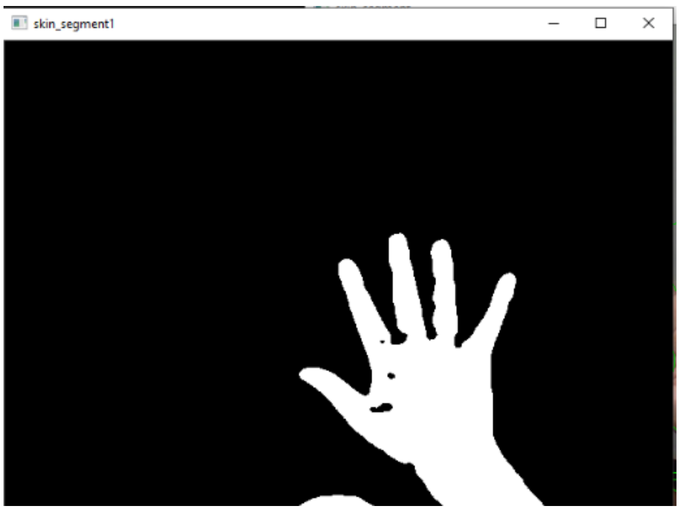

Fig 4. Image after Median Blur

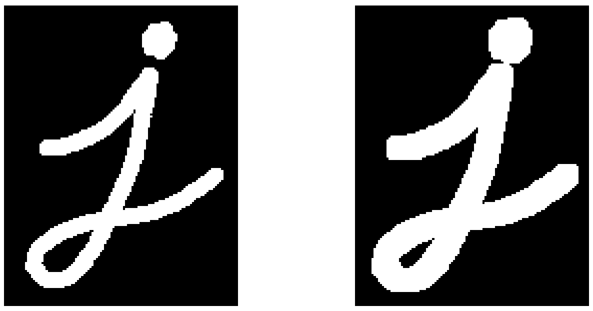

Fig 5. Example of Dilation [3]

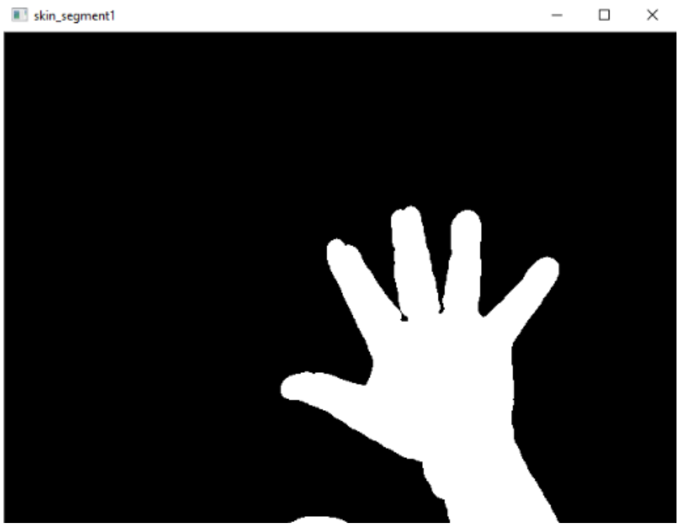

Fig 6. Hand Dilation

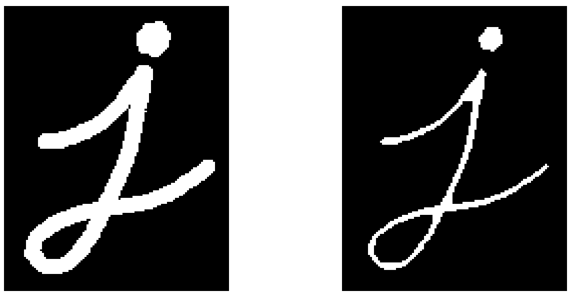

Fig. 7 Example of Erosion [3]

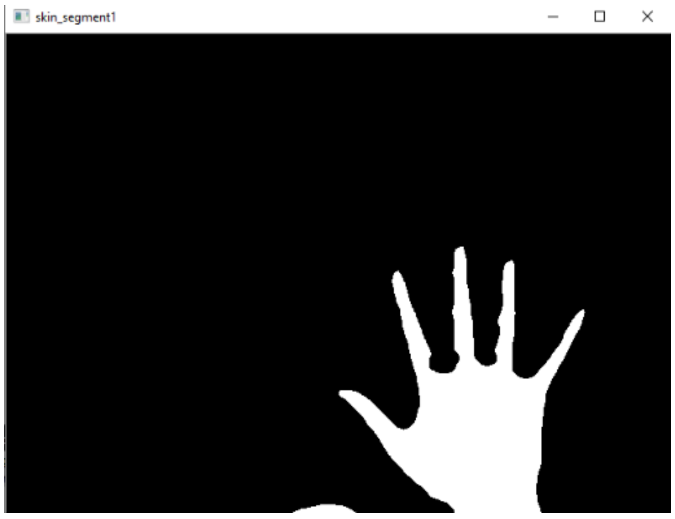

Fig. 8 Hand erosion

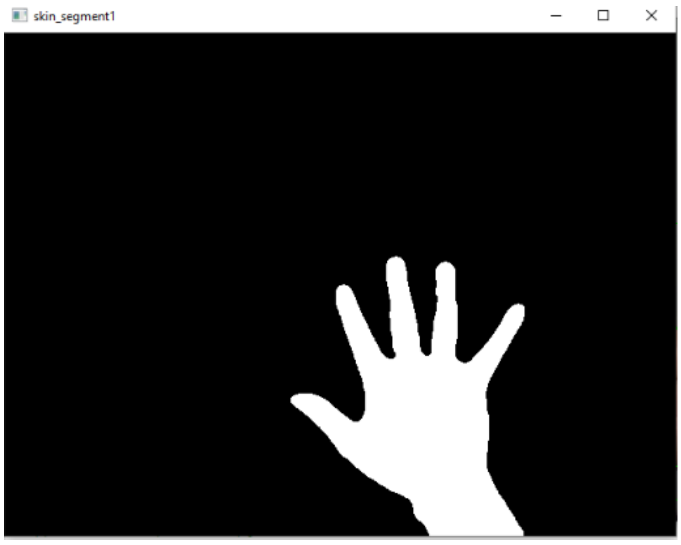

Fig. 9 Image of hand after Dilation and Erosion

C. Finding and Drawing Contour

Contour acts as a boundary line that joins all the pixels of same color or intensity. The contours prove to be a useful tool for shape analysis and object detection & recognition. Binary images are used for better accuracy. Applying threshold or canny edge detection is important before finding contours. In Open CV, finding contours is like finding white object from black background. So, the object to be found has to be white and background should be black. A few parameters need to be taken into consideration while applying contour. The first one is contour retrieval mode. We use RETR_EXTERNAL for our algorithm for retrieving only the extreme outer contours. The second parameter is contour approximation method. We use CHAIN_APPROX_SIMPLE in our algorithm as it compresses horizontal, vertical, and diagonal segments and leaves only their end points [4]. Fig. 10 shows a contour around hand palm. In order to overcome disturbances i.e. small contours, we can find area of contour and process further operations only when area is greater than certain value. With this all the small contours gets eliminated.

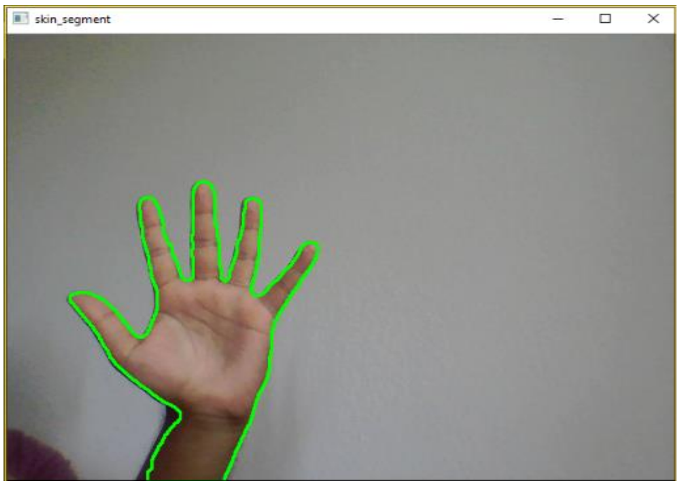

Fig. 10 Contour around hand palm

D. Drawing rectangle around region of interest

First task is to draw rotated rectangle around the obtained contour. We achieve this by using minArearect function, with this we get the center of contour, four coordinates around it, height, angle and width [10]. Using the four coordinates we can draw a line joining these coordinates to obtain a rectangle. This will give us a rectangle around our whole palm. But we need to obtain rectangle around our fingers i.e. we have to divide this contour. In order to achieve this we draw horizontal black lines starting from bottom and cover around ¾ of our frame. The lines are drawn before finding contours such that the line divides the fingers in different blobs. We choose ¾ of frame because according to analysis the length up to lower part of the finger covers ¾ of the frame. By doing this we get the contours around our fingers and rectangle around each of them as shown in fig. 11.

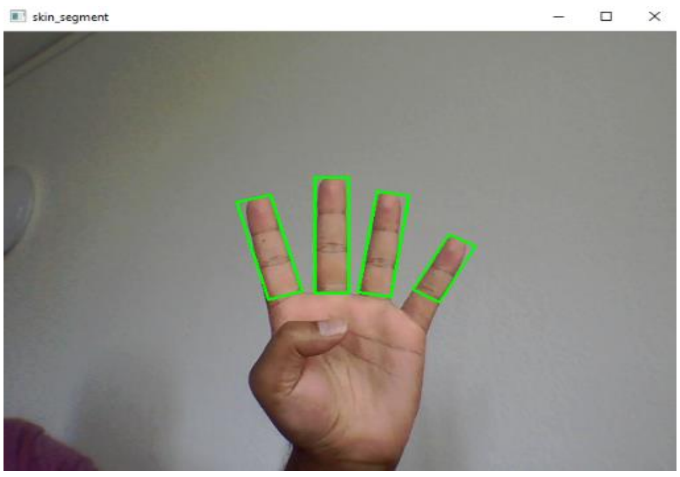

Fig. 11 Rotated Rectangle around finger

The next task is to reduce the obtained rectangle to our region of interest which is our finger tips. This is achieved by manipulating the center points of obtained rectangle around the contour. We need to shift the center point upwards which is obtained by basic mathematical calculations on obtained 4 coordinates. If height of rectangle is more than width then we reduce the rectangle height by 33% and manipulate the center point by using formula (coordinates [1] + coordinates [2])/2 + (coordinates [0] - coordinates [1])/6. Else if width is more than height than we reduce the width by 33% and manipulate the center point by using formula coordinates [2] + coordinates [3])/2 + (coordinates [0]- coordinates [3])/6. The first half part of the formula will retain the center position of the center point, second half of the formula shifts the center point upwards by 6 times. With this we achieve new set of coordinate points using which we draw a connecting lines through all these coordinate points and obtain a rectangle around our fingertip as shown in fig 12 and fig. 13.

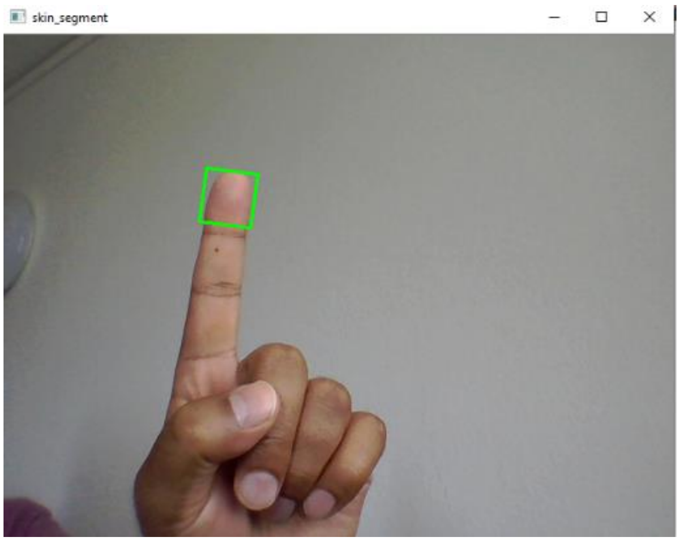

Fig. 12 Rectangle around one finger tip

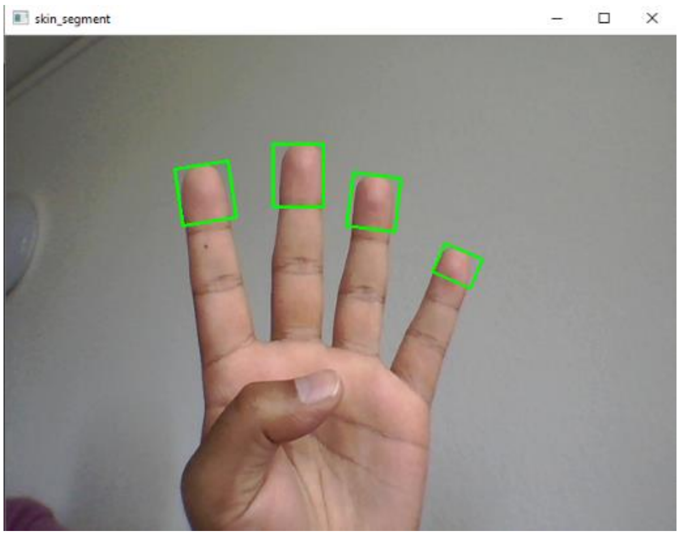

Fig 13.Rectangle around four finger tips

III. Implementation Challenges

Challenges faced for implementation of contactless fingerprint detection are as follows:

1) The lighting makes a lot of affect in detecting the skin color. Hence with improper lighting skin color is not detected hence hand palm is not segmented.

2) Dividing the contour in order to achieve separate contours around the fingers is significant. Palm should be shown to the camera and adjusted accordingly.

3) Manipulation of the rotated rectangle center point is very important and a great challenge to shift upwards to a precise point so that we obtain rectangle around our require region of interest.

4) The background also plays an important role, it should be plane and ideally completely different color from our skin.

IV. Results and Analysis

First the test was conducted on only one fingertip which was successful and then gradually we increase the number of fingers and got our region of interest on tip of four fingers, thumb excluded.

Lighting on palm plays a significant role, hence the test was conducted in various different lightings ideally indoors. The algorithm works perfectly fine in indoors with unnatural lighting. Extreme or very dull lighting misleads the algorithm in detecting the skin color.

The algorithm was tested on different backgrounds, plane background gives best results for segmenting image, background with few disturbance in background also gave accurate results as we are eliminating it in our algorithm but background with a lot of disturbance i.e. background having bigger skin color range objects lead to confusion, as it detects those objects also and draws rectangle around it.

Fig. 12 and fig. 13 shows the final result of contactless fingerprint of one and four fingers with region of interest being rectangle box in green color.

V. Conclusion

This paper presents method of contactless fingerprint detection which is implemented using Open CV libraries and it was tested and verified in different environment. We can conclude that this prototype can be effectively used for various applications such as biometric verification in Visa offices, and airports, attendance recording in offices, biometric lock system in gadgets. In crisis of CORONA this project can present a very powerful source in above mentioned applications and many more as it is contactless.

VI. Future Scope

As we have seen, this algorithm is limited to certain backgrounds and lighting. Also, dividing the contour can be considered dynamically. It can be further improved and more robust algorithm can be developed overcoming all these limitations. With this we will be able to use this system in more number of applications with more accuracy.

VII. References

[1]https://docs.opencv.org/3.4/da/d97/tutorial_threshold_nRange.html

[2] https://www.tech-faq.com/hsv.html#

[3]https://docs.opencv.org/2.4/doc/tutorials/imgproc/erosion_dilatation/erosion_dilatation.html

[4]https://docs.opencv.org/master/d5/daa/tutorial_js_contours_begin.html

[5] https://answers.opencv.org/question/128190/detect-hsv-color-for-skin-color-and-extract-the-color-in-that-region/

[6]https://docs.opencv.org/3.1.0/d2/de8/group__core__array.html#ga60b4d04b251ba5eb1392c34425497e14

[7] http://creat-tabu.blogspot.com/2013/08/opencv-python-hand-gesture-recognition.html

[8]https://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html?highlight=minarearect

[9]https://docs.opencv.org/3.4/db/dd6/classcv_1_1RotatedRect.html

[10]https://stackoverflow.com/questions/15956124/minarearect-angles-unsure-about-the-angle-returned

[11]https://stackoverflow.com/questions/42449090/how-does-one-use-cvboxpointsrotatedrect-box-outputarray-points

[12]https://www.researchgate.net/publication/327733447_Contactless_Fingerprint_Capture_Fingerprint_extraction_from_segmented_palm

[13] https://github.com/lzane/Fingers-Detection-using-OpenCV-and-Python