Contactless Fingerprint Detection

Rakshit Adderi Rohit
M.E. in Information Technology
Department of Engineering and Architecture
SRH Hochschule
Heidelberg, Germany
rakshitar29@gmail.com

Abstract—**With the advances made in image processing and computer vision and also taking clue from COVID-19 pandemic, the future of biometric authentication is going to be contactless. The project aims to reduce human contact during authentication using biometrics. The project is implemented to detect fingerprints when the fame consists of only hand of a human. The project generates a segmented image of human hand and then retains only finger data in the image by deleting rest of the data regarding human hand. Then the ROIs are drawn on the fingerprints by detecting contours in the image. The extracted finger prints can be used for authentication.**

Keywords-median blur, skin tone detection, segmentation, finger retention, region of interest (ROI), fingerprints

I. Introduction

Authentication has been in use since ages. Authentication has evolved from primitive techniques such as piece of paper or secret codes to modern techniques which use finger prints, face features and blood vessel networks.

Fingerprint authentication is being widely carried out with a human contact on the finger print sensor. Although this technique seems user friendly and safe, it may not be the case going forward as we're already witnessing due to the COVID-19 pandemic. The future of fingerprint authentication is contactless. Here I'm trying to move towards the future by extracting the fingerprints.

II. Overview Of The System

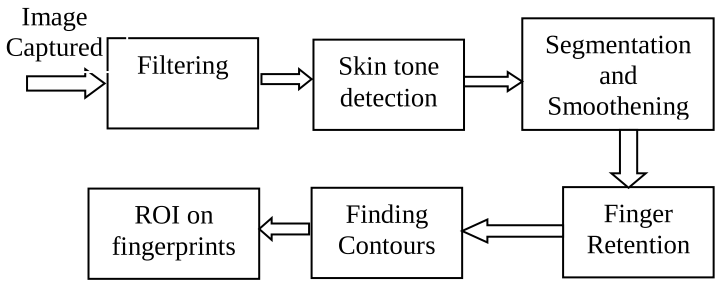

Figure 1: Block Diagram

The system takes input from the live feed of the camera which is an RGB image frame which consists of only human hand in the foreground at any given time. Firstly, the captured image frame is filtered to remove any noise or distortions in the frame. The filtered image is then taken up for skin tone detection. A technique which uses a combination of RGB and gray spectrum to detect skin tone is used. Thereafter, the segmented image is operated to remove the palm from it so that only fingers are retained in the image frame. This image frame makes the identification of contours vary easy as only all the fingers are detected as contours. Such found contours are manipulated to adjust the region of interest (ROI) over the fingerprints in the image frame.

Programming language: C++
Tools: OpenCV and Microsoft Visual Studio.
See 'Appendix' for more information.

III. Detailed Description Of The System

A. Filtering

Image filtering involves the application of window operations that perform useful functions, such as noise removal and image enhancement [1]. Here we are using filters to remove any additional noise in the image frame captured.

The filter that is being used is median filter. Median filter uses the pixel values of the neighboring pixels to reduce the noise by blurring the image. Intensity of the blurring depends on the size of the window that is being used. The window size represents the area of neighborhood pixels in consideration. Median filter operates by replacing the pixel that need to be filtered with the mean value of the neighborhood. Median filter mainly operates on a single channel. For multiple channel images, respective channels of each pixel in the neighborhood is considered for the operation.

In the project, blurring using median filter is being carried out with an operating window size of 5*5. For more information on median blur filter used please refer OpenCV library [5].

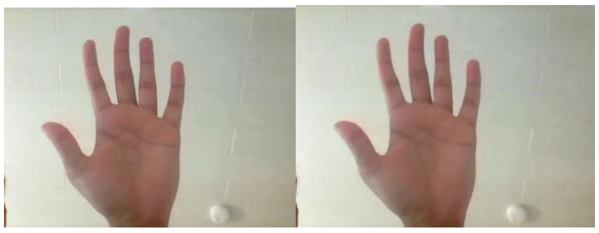

Figure 2: Input image (Left image) and Filtered image (Right image)

B. Skin Tone Detection

There are several ways to detect the skin tone in an image frame. Skin tone can be detected using various color spaces such as RGB, HSV and YC_bC_r.

The methodology used here focuses on illumination and luminance of an image in RGB color space. Illumination is nicely smeared along RGB colors in any given color image. Hence, its effect is scarcely distinguished here. There are different approaches to segregate such illumination. The utilized transformation matrix is:
$\alpha = [0.298936021293775390, 0.587043074451121360, 0.140209042551032500]^T$
, where the superscript T denotes the transpose operator to allow for matrix multiplication. Let ψ denote the 3D matrix containing the RGB vectors of the host image and let $x \in [1, 2, ..., n]$, $n=W*H$, W, H denote the width and height of the image, respectively. Note that the proposed method acts on the RGB colors stored in double precision, i.e., linearly scaled to the interval [0 1]. The initial color transformation is expressed as in Eq.1.

$$I(x) = (\psi(r(x),g(x),b(x)) * \alpha) \quad (1)$$

Where * represents the product operation. This reduces RGB color representation from 3D to 1D color space. The vector I(x) eliminates the hue and saturation information while retaining the luminance. It is therefore regarded formally as a grayscale color. Next, the algorithm tries to obtain another version of the luminance but this time without taking the R vector into account (most of skin color tends to cluster in the red channel). The discarding of red color is deliberate, as in the final stage it will help calculating the error signal. Therefore, the new vector will have the largest elements taken from G or B:

$$I\_hat(x) = \max(G(x), B(x)) \quad (2)$$

Eq. 2 is actually a modification of the way HSV (Hue, Saturation and Value) computes the V values. The only difference is that the method does not include in this case the red component in the calculation. Then for any value of x, the error signal is derived from the calculation of element-wise subtraction of the matrices generated by Eq. 1 and Eq. 2 which can be defined as:

$$e(x) = I(x) - I\_hat(x) \quad (3)$$

e(x) represents skin probability map (SPM) that uses an explicit threshold based skin cluster classifier which defines the lower and upper boundaries of the skin cluster. According to the skin probability map, the boundaries of skin color is between 0.02511 and 0.1177.

C. Segmentation

Image segmentation is the process of partitioning an image into multiple segments. Image segmentation is typically used to locate objects and boundaries in images [3]. The output of segmentation is the binary image.

The generated skin probability map e(x) represented by Eq. 3 need to be segmented for skin tone. The boundaries of skin color lies between 0.02511 and 0.1177 which translates to 4 and 30 in unsigned integer type.

Segmentation is done using “inRange()” function in OpenCV. For more information on “inRange()” function used please refer OpenCV library [5].

In order to smoothen the image after segmentation to fill in gaps if present, median blurring is done with a window size of 15*15. For more information on median blur filter used please refer OpenCV library [5].

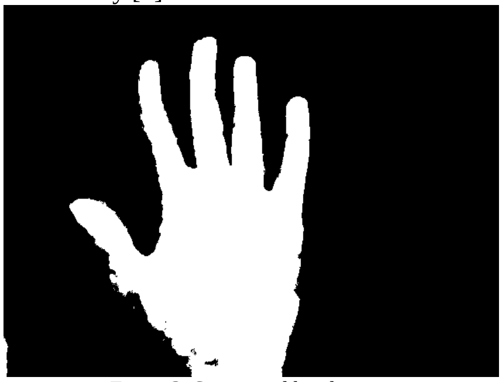
Figure 3: Segmented hand image

D. Finger Retention

The segmented image consists information about human hand in the image frame. In order to detect the fingerprints, fingers of the hand need to be found. One of the efficient way to do this is to delete other parts of the hand except fingers.

The algorithm to retain only fingers in the image frame focuses on deletion of other parts of the human hand from the image frame. This algorithm takes the segmented image of human hand as an input. Entire image frame is scanned starting from pixel location (0, 0). All of the pixels representing human hand is identified in each row and width of each stretch of human hand is calculated. The width of each stretch of human hand in a row is compared against a pre-defined width of a human hand finger which is calculated by conducting test runs on various test samples of the human hand fingers. If the width of a stretch of human hand is greater than the pre-defined width of the human hand finger, then the stretch of pixels in question is deleted by making the pixel values 0. By applying this to each row of the image, information about only fingers in the image frame is retained by deleting information about rest of the human hand.

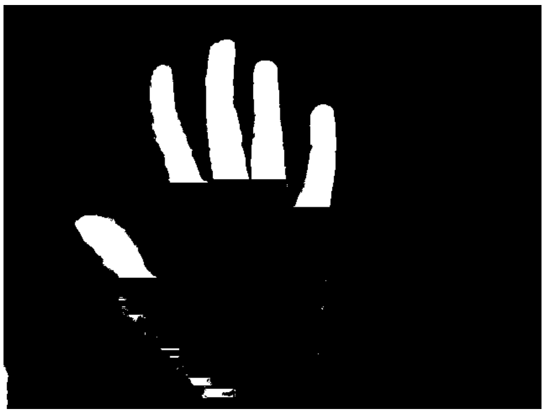

Figure 4: Binary image with only fingers data

E. Finding Contours

The contours in the binary image where only the finger information is retained is found using Suzuki85 algorithm [4].

This algorithm is an extended version of the border following algorithm which discriminates between outer borders and hole borders. The border following algorithm derives a sequence of the coordinates or the chain codes from the border between a connected component of l-pixels (l-component)' and a connected component of 0-pixels (background or hole) [4, 5]. The extensions applied to border following algorithm are:

1. To put a unique mark on each border rather than to adopt the same marking procedure for every border (border labeling)
2. To add a procedure for obtaining the parent border of the currently followed border

With this algorithm we can extract the surroundness relation among the borders, which corresponds to the surroundness relation among the connected components. If a binary image is stored in the form of the borders and the surroundness relation is extracted by this algorithm [4].

The Suzuki85 algorithm is implemented using “findContours()” function in OpenCV. For more information on “findContours()” function used please refer OpenCV library [5].

F. ROI On Fingerprints

Once the contours are found, the final stage of the project is to draw ROI on the fingerprints. This is done as following.

1) Rotated Rectangle On Fingers

The binary image that contains only human hand finger information needs to be operated to retrieve data regarding rectangles that can be drawn on the human hand fingers with the help of contours found. But each human hand finger is usually oriented at a particular angle and this data also needs to be recorded. To filter out some unwanted noise in the image, we consider contours whose size is greater than 200.

To carry out this step, there is a pre-defined function in OpenCV named “minAreaRect()” which give data of rotated rectangles on contours found. The data retrieved are centroid, height, width, angle and vertices of the rectangle drawn. For more information on “minAreaRect()” function used please refer OpenCV library [5].

2) Rotated Rectangles On Fingerprints

The rectangles drawn on the fingers of the human hand needs to be modified to draw the rectangles on the fingerprints of the human hand. This is done by shrinking the rectangle on covering the finger to cover just the fingerprints.

The shrinking of the rotated rectangle is done by modifying the centroid and height or width of the rectangle depending upon whether height is greater or the width of the rectangle. The centroid of the rectangle is moved to the top part of the rectangle using the vertices data of the rectangle. When width is greater than height of the rectangle then width is updated to be 150% of the height and when height is greater than or equal to width of the rectangle then height is updated to be 150% of the width. With this we find the new vertices of the rectangle with the angle data retained and the rotated rectangle is drawn to generate ROIs on the all of the fingerprints.

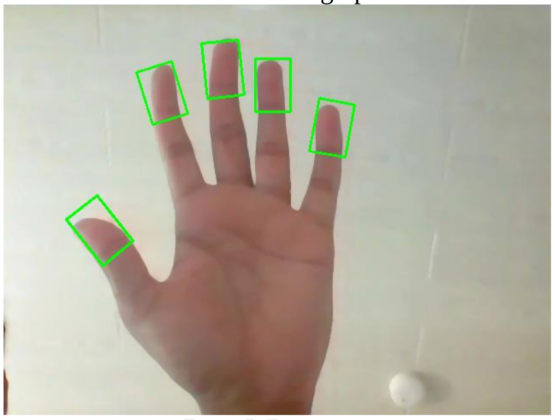

Figure 5: Fingerprints

IV. FUTURE IMPROVEMENTS

Future improvements are aimed at resolving some of the drawbacks in the system.

1. Dynamic finger width

   Issue: The algorithm to delete other parts of human hand except fingers is static i.e. it uses static width of fingers to delete other parts.

   Possible solution: The fingerprints found can be used as an input to change the finger width. If the fingerprints are less than expected then, the finger width can be decreased and vice versa.

2. Detect fingerprints when the human hand is upside down.

   Possible solution: Change the algorithm that shrinks the rectangle on the fingers by detecting the hand to be upside down.

3. Detect fingerprints when human hand is horizontally oriented.

   Issue: The current system doesn't detect fingerprints when the human hand/fingers are oriented at 180 degrees.

   Possible solution: To resolve this, detect the hand orientation to be 180 degrees. In such case scan each column of the image frame rather than scanning each row.

4. Detect only human hand when other parts of human body are visible.

V. Appendix

A. *C++*

C++ is a general-purpose programming language created by Bjarne Stroustrup as an extension of the C programming language, or "C with Classes". The language has expanded significantly over time, and modern C++ now has object-oriented, generic, and functional features in addition to facilities for low-level memory manipulation [6].

B. *OpenCV*

OpenCV (Open Source Computer Vision Library) is a library of programming functions mainly aimed at real-time computer vision. Originally developed by Intel, it was later supported by Willow Garage then Itseez (which was later acquired by Intel). The library is cross-platform and free for use under the open-source BSD license [7].

OpenCV is written in C++ and its primary interface is in C++, but it still retains a less comprehensive though extensive older C interface. All of the new developments and algorithms appear in the C++ interface. There are bindings in Python, Java and MATLAB/OCTAVE. The API for these interfaces can be found in the online documentation. Wrappers in other languages such as C#, Perl, Ch, Haskell, and Ruby have been developed to encourage adoption by a wider audience. Since version 3.4, OpenCV.js is a JavaScript binding for selected subset of OpenCV functions for the web platform [7].

C. *Visual Studio*

Microsoft Visual Studio is an integrated development environment (IDE) from Microsoft. It is used to develop computer programs, as well as websites, web apps, web services and mobile apps [8].

Visual Studio supports 36 different programming languages and allows the code editor and debugger to support (to varying degrees) nearly any programming language, provided a language-specific service exists. Built-in languages include C, C++, C++/CLI, Visual Basic .NET, C#, F#, JavaScript, TypeScript, XML, XSLT, HTML, and CSS [8]. For image processing using OpenCV in C++, appropriate OpenCV library is integrated into Microsoft visual studio.

VI. References

[1] E.R. Davies, Computer Vision (Fifth Edition), Elsevier, 125 London Wall, London, United Kingdom, 2018, Pages 39-92

[2] A. Cheddad , J. Condell, K. Curran, and P. McKevitt . “A new colour space for skin tone detection.” IEEE International Conference on Image Processing (ICIP) ICIP), 2009 16th, pages 497 500, Nov 2009.

[3] Ying Tan, GPU-based Parallel Implementation of Swarm Intelligence Algorithms, Elsevier, 50 Hampshire Street, 5th Floor, Cambridge, MA, USA, 2016, Page 167

[4] I Suzuki, S. and Abe, K., Topological Structural Analysis of Digitized Binary Images by Border Following. CVGIP 30 1, pp 32-46 (1985)

[5] Bradski, G., 2000. The OpenCV Library. Dr. Dobb&#39;s Journal of Software Tools.

[6] Wikipedia contributors, C++, Wikipedia, The Free Encyclopedia, 30 June 2020 06:51 UTC (last modified), 6 July 2020 12:44 UTC (Retrieved)

[7] Wikipedia contributors, OpenCV, Wikipedia, The Free Encyclopedia, 10 June 2020 20:27 UTC (last modified), 6 July 2020 12:51 UTC (Retrieved)

[8] Wikipedia contributors, Microsoft Visual Studio, Wikipedia, The Free Encyclopedia, 27 June 2020 05:47 UTC (last modified), 6 July 2020 12:55 UTC (Retrieved)