Touchless Fingerprint Capture

Anjali Pankan
Matriculation Number: 11016821
Email Id: 11016821@stud.hochschule-heidelberg.de
Under the Guidance Of:
Prof. Dr.-Ing Christof Joneitz
Professor, SRH Hochschule Heidelberg

***Abstract*—In this project, a mechanism for touchless fingerprint capturing using a camera is explained. The project aims at detecting the palm region including fingers from the captured video. For each collected frame of the video, the fingerprint section is identified and extracted. The palm identification process is based on skin color detection from the collected image frame. In order to realize this, the collected BGR (Blue, Green, Red) image is converted into HSV (Hue, Saturation, Value) image. This HSV image is then converted into a binary image by calculating threshold. The binary image is then used for determining the fingerprint area. In order to implement the project, OpenCV library is being used.**

***Index Terms*—Fingerprint Extraction, OpenCV**

I. Introduction

Due to the persistence and uniqueness nature of fingerprints, they are considered one of the important biometric information worldwide. The application of fingerprint detection and verification is not only limited to forensic and law enforcement sections but also in many other fields such as automatic border control [1] [2]. Most of the currently existing fingerprint detection systems uses a touch based approach where the user is required to touch the surface of the sensor device. One of the drawbacks of touch based systems are ghost fingerprints caused by latent fingerprints of earlier users [1]. Another important aspect of touch based system is the concern of hygiene. Especially during the pandemic situations such as SARS-CoV-2 virus, it is not convenient to use touch based multi-user systems for fingerprint capturing.

In order to overcome the limitations of touch based detection systems, touchless fingerprint detection systems has been researched for a while [3]. Touchless detection systems mainly uses image processing techniques to capture and extract fingerprints from the captured image. In this project a touchless fingerprint detection scheme is proposed. The implementation of the project largely depends on OpenCV library functions. The video captured using the camera is processed frame by frame to segment hand/palm section based on skin color. A binary image is produced using thresholding functions. By applying palm removal and finger area detection logic the fingerprint area is extracted.

II. Design

In this section, the overall design of the fingerprint detection and extraction mechanism implemented in this project is discussed. The process is divided into two parts. The first part involves identifying the hand/palm section based on skin color. From the obtained output image, the hand/palm section is segmented. The second part involves detecting fingers from the obtained image by separating the palm section. Finally, the fingerprint section is determined and extracted. Figure 1 illustrates the sequence of steps carried out in the fingerprint extraction process.

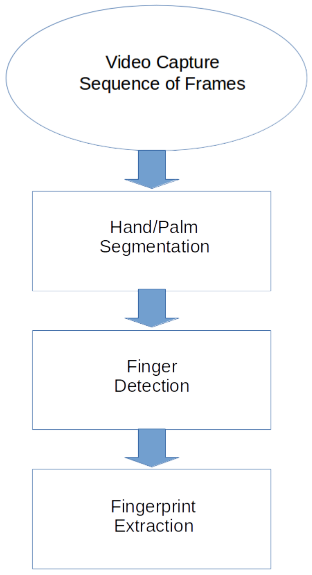


Fig. 1: Fingerprint extraction procedure

The fingerprint extraction scheme works by first capturing the video of the hand/palm portion using a camera. Each frame of this video is then processed in iteration to identify the hand/palm portion compared to the background. One thing to be noted here is that the detection mechanism also considers palm section in the obtained video frame. As a result, determining the finger section is necessary to identify exact location of the fingerprint area in each finger [2].

III. Hand/Palm Segmentation

In this section, the implementation of the hand/palm segmentation procedure is explained. Figure 2 represents the steps

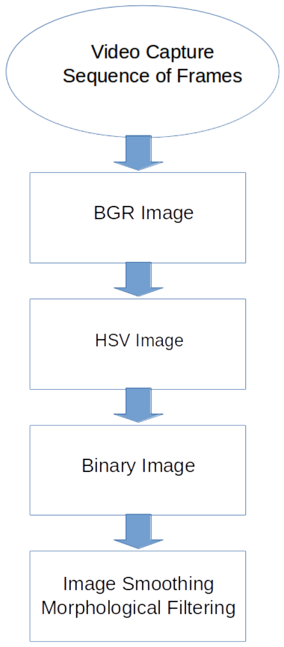


Fig. 2: Hand/palm detection and segmentation

involved in this procedure.

One of the well known way to segmenting an image is based on color. Human hand can be recognized from the input image by considering skin color [4]. In [5] HSV color space is considered for skin color identification. In this project, hand segmentation is carried out using HSV color space which is explained in following subsections.

A. Video Capture

The first step towards hand/palm segmentation is capturing hand/palm image using a video camera. This process can be easily realized using OpenCV built-in class VideoCapture [6]. This class provides a C++ API for reading video files or capturing video from a camera. The default camera can be accessed by passing value ”0” to the API call. If the camera can be opened successfully, then the video capture can proceed and can capture video frames. The frames collected are processed individually, that means, frames are collected in iterations and processed one at a time. The video captured by the above procedure is either in BGR (Blue, Green, Red) or RGB (Red, Green, Blue) format as electronic equipments usually supports this color space. Figure 3 shows a BGR image of hand collected using video camera.

B. BGR to HSV Color Space Transformation

Once the BGR video frame of the hand is obtained, the corresponding HSV color spaced image can be obtained by using OpenCV built-in function cvtColor(). Listing 1 shows the code for converting the BGR image to HSV image where, frameOrg is the input BGR image, frameHSV is the output HSV image and $COLOR\_BGR2HSV$ is the BGR to HSV color conversion code. Figure 4 shows the transformed HSV image.

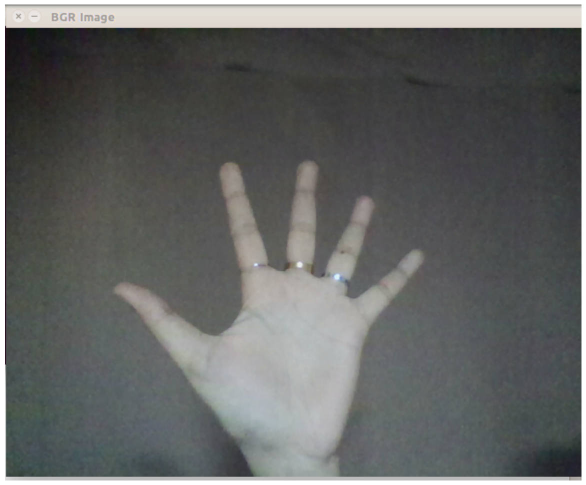


Fig. 3: BGR image captured using video camera

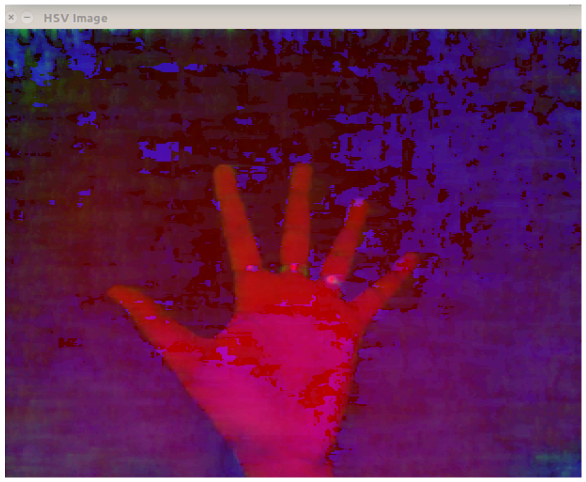


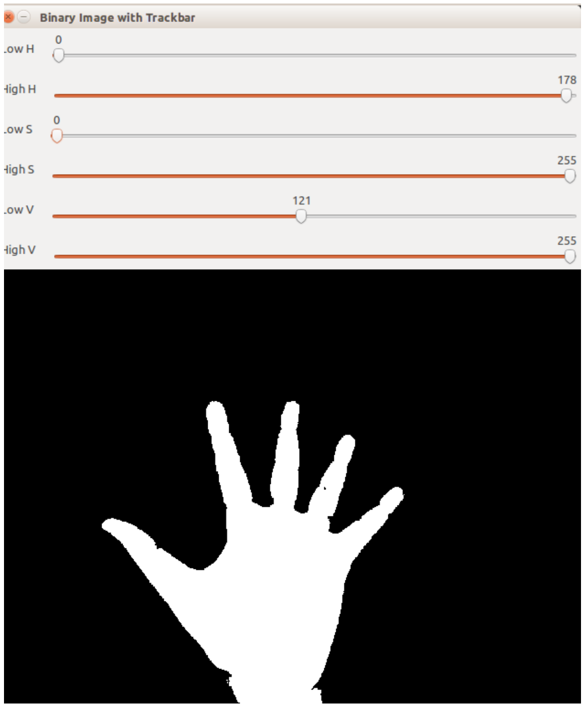
Fig. 4: Transformed HSV image

Listing 1: Convert from BGR to HSV color space

```
// Convert from BGR to HSV color space.
Mat frameHSV;
cvtColor(frameOrg, frameHSV, COLOR_BGR2HSV);
```

C. Binary Image Formation

A segmentation procedure using color searches for pixels with a specific color. The HSV color space comprises of three components: Hue, Saturation and Value. The Hue component represents the color whereas the Saturation component gives the dominance of that color. The Value component gives the measure of brightness. Hence, for searching a specific color in the image, it can search for color $position$ and color $purity$. For detecting and segmenting skin color from the input HSV image, it requires to define some tolerance values in each component. This can be accomplished by using OpenCV built-in function inRange() as shown in Listing 2. Here,


Binary Image with Trackbar

ow H 0

igh H 178

ow S 0

igh S 255

ow V 121

igh V 255

Fig. 5: Binary image with trackbar

frameHSV is the input HSV image, the next two arrays are the inclusive lower and upper boundaries of HSV components and frameThresh is the binary output image. Figure 5 shows the final binary image obtained along with trackbar.

Listing 2: Performing thresholding to obtain binary image

```
// Perform thresholding for HSV frame, to
    obtain binary image.
Mat frameThresh;
inRange(frameHSV, Scalar(ilowH, ilowS,
    ilowV), Scalar(ihighH, ihighS, ihighV),
    frameThresh);
```

D. *Image Smoothing, Morphological Filtering*

The image improvement process comprises of three tasks, which are listed below:

- Image Smoothing: Using the image smoothing procedure the noise in the image can be reduced. In order to perform the smoothing, OpenCV function medianBlur() is used. This function blurs an image using the median filter. Median filter is especially good for removing salt-and-pepper noise. It is a non-linear filter which takes the median of the kernal area for determining the pixel.
- Morphological Opening: It is used for removing small white objects from the image. This process consists of erosion followed by dilation.
  - Erosion: It is used to erode away the boundaries of a foreground white object of an image [7]. Here, a pixel in the original image is taken as 1 (white) only when all the pixels under the kernal is 1, otherwise it is set to 0 (black). As as result, the thickness of the foreground white image gets reduced.

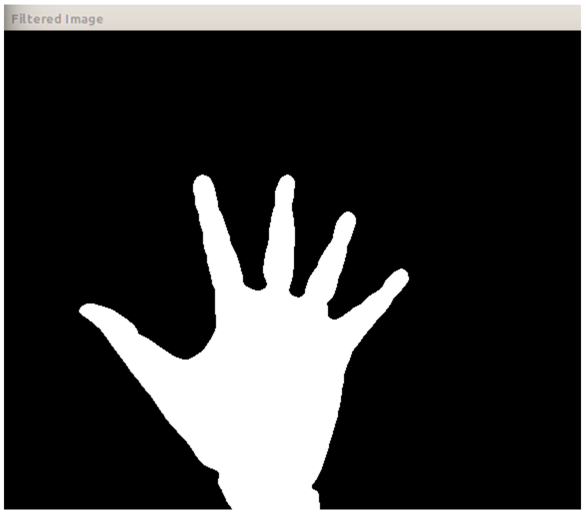


Fig. 6: Image smoothing, morphological filtering

  - Dilation: It is used to remove black holes in a foreground white object of an image. Here, a pixel in the original image is taken as 1 (white) if at least one pixel in the kernal is 1.

  Hence, by applying erosion followed by dilation, the small white noises are removed from the image while keeping the thickness of the foreground object unchanged.
- Morphological Closing: It is used for removing small black holes from the image. This process consists of dilation followed by erosion.

Figure 6 shows the final improved binary image after applying smoothing and morphological filtering.

IV. Finger detection and Fingerprint extraction

A. *Palm Section Removal*

Since the video captured may contain not only the finger part but also the full hand/palm section, it is required to exclude the palm section from the image. There can be possibilities that a user can show only his/her fingers or can show the whole hand. The finger detection procedure should thus consider both the cases.

For a given binary image it is possible to find the contours by using OpenCV built-in function findContours(). The contours are useful for analyzing shapes and determining objects in an image. Palm section removal procedure utilizes contour calculation to find finger position.

In order to remove palm section from the binary image, a black line is drawn starting from the bottom of the image up to the top, row by row. This process is performed in a loop and the number of contours are determined in each iteration. When the counted number of contours is more than the previously counted maximum contours, then the current value is saved as maximum contours along with the row index value. After completing the above steps, the stored maximum contour count gives the maximum number of contour possible in the binary image as well as the corresponding row index where it happened at the first time. Once these values are available,

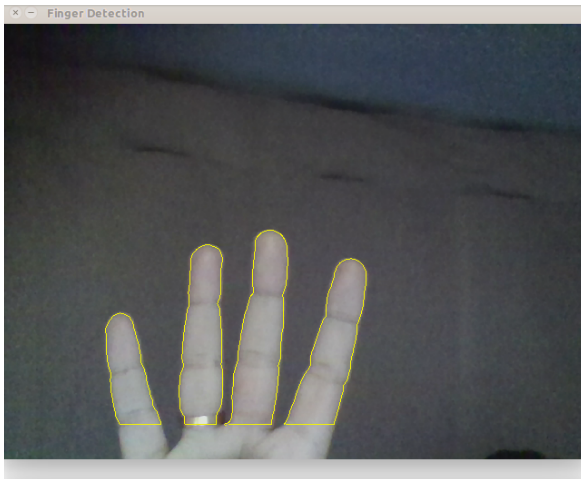


Fig. 7: Finger detection

it is possible to make bottom part of the image black until the obtained row index. In other words, the above procedure removes the palm section from the image by only retaining the finger portion. Algorithm 1 shows the palm removal procedure.

Algorithm 1 Palm section removal

```
Input: $inImage$                                ▷ Input binary image
Output: $outImage$                              ▷ Output binary image
  $outImage \leftarrow inImage.clone()$
  $rows \leftarrow inImage.rows$
  $maxContours \leftarrow 0$
  $index \leftarrow inImage.rows$
  for $i = rows; i > 0; i--$ do
      $draw\ black\ line\ at\ row\ i\ of\ inImage$
      $contourCount \leftarrow find\ contours\ of\ inImage$
      if $contourCount$ greater than $maxContours$ then
          $maxContours \leftarrow contourCount$
          $index \leftarrow i$
      end if
  end for
  for $j = rows; j > index; j--$ do
      $draw\ black\ line\ at\ row\ j\ of\ outImage$
  end for
```

Considerations can be made when counting the number of contours in each loop by selecting only bigger enough contours. Finally, after palm removal procedure a binary image with only fingers are obtained.

B. Fingerprint Extraction

Once the palm section is removed from the hand portion, the output binary image contains white objects corresponds to only fingers. The edges of this image can be computed using OpenCV built-in function Canny(). Listing 3 shows the Canny() function applied in this project, where frameThresh is the input binary image, frameDetect is the output image, the next two arguments are thresholds for the hysteresis procedure and last argument is the aperture size [8].

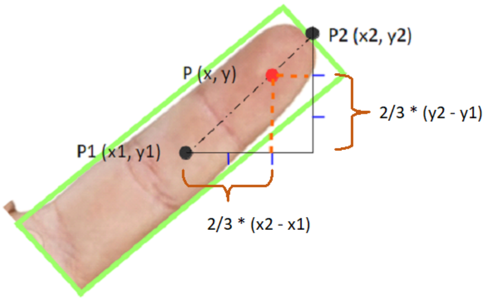


Fig. 8: Fingerprint area center point determination

Listing 3: Edge detection using Canny

```
// Determine edges in the frame.
Mat frameDetect;
int thresh = 100;
Canny(frameThresh, frameDetect, thresh,
    thresh * 2, 3);
```

Once the edge detected image is available, the next step is to find the contours of the above produced image by using the OpenCV built-in function findContours(). The function used $CV\_RETR\_EXTERNAL$ mode to obtain only the outer contour from the image. As a result, the contours for the fingers are generated. Figure 7 shows a BGR image of a hand with contours produced for its fingers using the above steps.

The final step of fingerprint capture procedure is to identify the fingerprint area of each finger. Once the contour of a finger is available, it is possible to find a minimum rectangle enclosing the contour points using OpenCV built-in function minAreaRect(). The produced rotated rectangle contains properties such as, center, height, width and angle. With this information a new rotated rectangle around the fingerprint can be drawn. For doing this, the center of the new rotated rectangle needs to be found first. Figure 8 shows the method employed in determining the new center point. Given $P1(x1, y1)$ the center point and $P2(x2, y2)$ the top point of the rotated rectangle of the finger contour, then the fingerprint area center point $P(x, y)$ is determined using the Equation 1.

$$\begin{aligned} x &= x1 + \frac{2}{3} \times (x2 - x1) \\ y &= y1 + \frac{2}{3} \times (y2 - y1) \end{aligned} \qquad (1)$$

For the the rotated rectangle of the finger, the height may be higher or lower than the width based on its orientation. If the height is greater, then the height of the new fingerprint area rectangle is taken 1.6 times the width. If the width is greater then the new width is taken 1.6 times the height. Since the angle of finger area rectangle and fingerprint area rectangle are the same, it is unchanged.

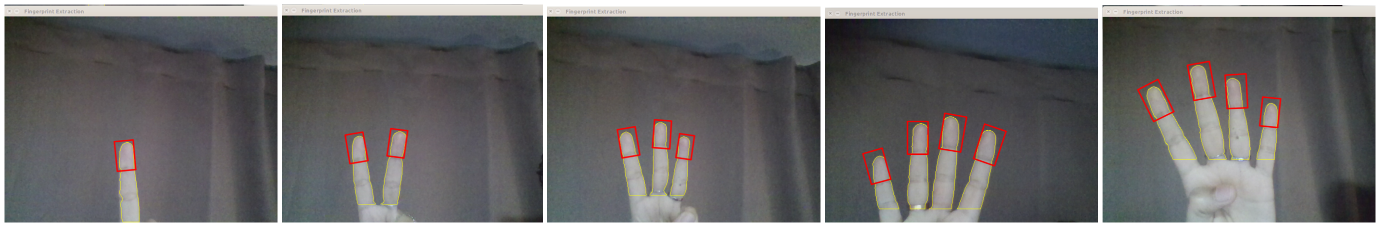


Fig. 9: Fingerprint extraction for (1) one finer, (2) two fingers, (3) three fingers, (4) four fingers, (5) four fingers with palm

V. Results

Figures 9 shows the final result of fingerprint extraction procedure. From the figures it is clear that the implementation could extract fingerprint area for one finer, two fingers, three fingers, four fingers and four fingers with palm area.

VI. Conclusion

In this project, a touchless fingerprint capturing mechanism is successfully implemented. The hand/palm segmentation procedure employed could properly identify and segment the hand/palm section using skin color. The finger detection procedure could separate the finger section from the hand/palm image using palm removal logic. Finally, the fingerprint area is determined from the rotated rectangle of the finger contours by calculating the center point of the fingerprint area. A limitation of this project is that the finger detection procedure works only when the hand is positioned vertically with fingers at the top.

References

[1] Priesnitz, J.; Huesmann, R.; Rathgeb, C.; Buchmann, N.; Busch, C. *Mobile Contactless Fingerprint Recognition: Implementation, Performance and Usability Aspects*. Sensors 2022, 22, 792. https://doi.org/10.3390/s22030792

[2] Christof Jonietz, Eduardo Monari, Chengchao Qu. *Towards Touchless Palm and Finger Detection for Fingerprint Extraction with Mobile Devices*. Vision and Fusion Laboratory (IES), Karlsruhe Institute of Technology (KIT)

[3] Chulhan Lee, Sanghoon Lee, and Jaihie Kim. *A Study of Touchless Fingerprint Recognition System*. Department of Electrical and Electronic Engineering, Yonsei University, Biometrics Engineering Research Center (BERC)

[4] Son Lam Phung, Abdesselam Bouzerdoum, and Douglas Chai. *A NOVEL SKIN COLOR MODEL IN YCBCR COLOR SPACE AND ITS APPLICATION TO HUMAN FACE DETECTION*. Visual Information Processing Research Group Edith Cowan University, Western Australia

[5] Alberto Albiol, Luis Torres, and Edward J. Delp. *Optimum color spaces for skin detection*. Image Processing, 2001. Proceedings. 2001 International Conference on. Volume: 1, DOI:10.1109/ICIP.2001.958968

[6] *cv::VideoCapture Class Reference*. docs.opencv.org. [Online]. Available: https://docs.opencv.org/3.4/d8/dfe/classcv_1_1VideoCapture.html

[7] *Morphological Transformations*. docs.opencv.org. [Online]. Available: https://docs.opencv.org/3.4/d9/d61/tutorial_py_morphological_ops.html

[8] *Canny Edge Detection*. docs.opencv.org. [Online]. Available: https://docs.opencv.org/3.4/da/d22/tutorial_py_canny.html