Fingerprint Detection and Tracking Using OpenCV

Aswin Chand
Dept. of Information Technology
SRH Hochschule Heidelberg
Heidelberg, Germany

Christof Jonietz
Dept. of Information Technology
SRH Hochschule Heidelberg
Heidelberg, Germany

Abstract— **The computer industry is developing at a fast pace. With this development almost all of the fields under computers have advanced in the past couple of decades. But the same technology is being used for human computer interaction that was used in 1970's. Even today the same type of keyboard and mouse is used for interacting with computer systems[1]. Recently with the introduction of touch screen in smart phones and the emergence of Augmented Reality and Virtual Reality devices we are taking a leap into the future of human computer interaction. Many vision-based applications have used fingertips to track or manipulate gestures in their applications. Gesture identification is a natural way to pass the signals to the machine, as the human express its feelings most of the time with hand expressions. Here an efficient algorithm has been described for fingerprint detection. This method detects the four fingers excluding the thumb and in preprocessing it cuts only hand part from the full image, hence further computation would be much faster than processing full image. It is done with simple signal processing performed on images obtained from a regular laptop web-camera.**
.

Keywords- OpenCV, Colorspaces,Thresholding, Contour

I. INTRODUCTION

In today's age a lot of research is done for finding effective techniques and methods to make existing systems more reliable and effective. One of the most important parameter to make system efficient and reliable is Human Computer Interaction (HCI). Many systems provide simple techniques for HCI, most common techniques for input to the systems include use of mouse, keyboard etc. These are physically in contact with the system. Recently new techniques have been developed to make interactions with the system more efficient [1]. We apply background elimination using simple techniques and apply a set of filters to get the finger's contour, on which we find the minimum rectangles and draw rectangles on the finger prints which is a challenge [2]. Modelling skin colour implies the identification of a suitable colour space and the careful setting of rules for cropping clusters associated with skin colour. Unfortunately, most approaches to date tend to put the illumination channel in the "non-useful" zone and therefore act instead on colour transformation spaces that de-correlate luminance and chrominance components from an RGB image. It is important to note that illumination and luminance are defined slightly differently as they depend on each other. As this may cause confusion, for simplicity, here we will refer to both as the function of response to incident light flux or the brightness [3]. This paper presents a method to perform fingerprint tracking in a mostly static environment with only the fingers in the range of visibility using OpenCV.

II. COLORSPACES USED FOR SKIN MODELLING

Colorimetry, computer graphics and video signal transmission standards have given birth to many colorspaces with different properties. A wide variety of them have been applied to the problem of skin color modelling. We will briefly review the most popular colorspaces and their properties [4].

A. OpenCV

OpenCV (Open Source Computer Vision Library) is a library mainly aims at real-time computer vision systems, developed by Intel. The library is platform independent. It mainly focuses on real time Image Processing and Computer Vision. Computer Vision is a science and technology that is used to process images to obtain relevant information. OpenCV library is written in C language with its primary application interface available in C. It provides wrapper classes for use in C++ language as well. Also, there are full interfaces available for various other languages such as Python, Java, MATLAB and Octave. It provides basic data structures for Matrix operations and Image Processing with efficient optimizations. It provides fast and efficient data structures and algorithms which can be used for complex real time Image Processing and Computer Vision applications [5].

B. C++

C++ (pronounced as see plus plus) is a multi-paradigm, general-purpose and compiled programming language. It comprises of a combination of high-level language features like OOPS concepts and low-level language features like memory access using pointers etc. It added object-oriented concepts like classes to the C language. We use the C++ interface of OpenCV for the implementation. The C++ interface provides a powerful data structure Mat (short for Matrix) which is efficient, portable and easy to use. Mat is basically just a multi-channel and multi-dimensional array which can be used to store images, intermediate image transformations, values just like a normal Array.

III. COLORSPACES USED FOR SKIN MODELLING

A. *RGB*

RGB is a colorspace originated from CRT (or similar) display applications, when it was convenient to describe color as a combination of three colored rays (red, green and blue). It is one of the most widely used colorspaces for processing and storing of digital image data. However, high correlation between channels, significant perceptual non-uniformity, mixing of chrominance and luminance data make RGB not a very favorable choice for color analysis and colorbased recognition algorithms.

B. *HSI, HSV, HSL - Hue Saturation Intensity (Value, Lightness)*

Hue-saturation based colorspaces were introduced when there was a need for the user to specify color properties numerically. They describe color with intuitive values, based on the artist’s idea of tint,saturation and tone. Hue defines the dominant color (such as red, green, purple and yellow) of an area, saturation measures the colorfulness of an area in proportion to its brightness [Poynton 1995]. The “intensity”, “lightness” or” value” is related to the color luminance. The intuitiveness of the colorspace components and explicit discrimination between luminance and chrominance properties made these colorspaces popular in the works on skin color segmentation.

An alternative way of hue and saturation computation using log opponent values was introduced in [Fleck et al. 1996], where additional logarithmic transformation of RGB values aimed to reduce the dependence of chrominance on the illumination level.

$$H = \arccos \frac{1}{2} \frac{((R-G) + (R-B))}{\sqrt{((R-G)^2 - (R-B)(G-B))}} \qquad (1)$$

$$S = 1 - 3\frac{\min(R, G, B)}{R+G+B} \qquad (2)$$

$$V = \frac{1}{3}(R+G+B) \qquad (3)$$

C. *YCrCb*

YCrCb is an encoded nonlinear RGB signal, commonly used by European television studios and for image compression work. Color is represented by luma (which is luminance, computed from nonlinear RGB [Poynton 1995]), constructed as a weighted sum of the RGB values, and two-color difference values Cr and Cb that are formed by subtracting luma from RGB red and blue components.

$$\begin{aligned} Y &= 0.299R + 0.587G + 0.114B \\ Cr &= R - Y \\ Cb &= B - Y \end{aligned} \qquad (4)$$

IV. IMPLEMENTATION

The following steps were taken to detect the fingerprint

A. *Capturing the hand*

To capture the image, we use the “VideoCapture” object and we obtain the index of camera that we want to access through the arguments passed to the object. In a loop, we obtain a frame from the camera (through the >> operator) for the specified time period (the argument of “waitKey” function, which waits x milliseconds until a key has been pressed and returns -1 if none was pressed). To display the captured image frame we use the “imshow” function).

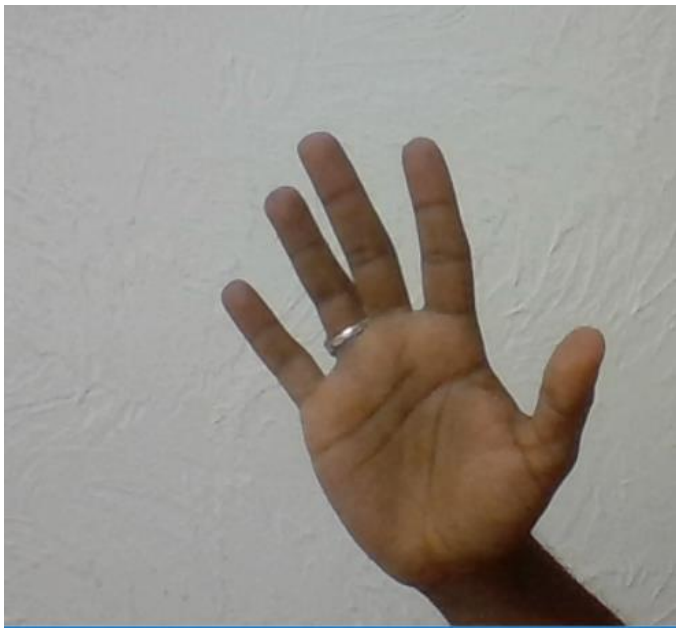

Figure 1.0 Input image frame

B. *Hand Segmentation*

With segmentation we refer to the process of extracting objects of interest from an image. In our case, the object of interest is the finger of the user. There are many possible approaches to solve this problem, each with different complexity and accuracy. Due to the lack of strong constraints on the scene's composition and illumination, we had to exclude segmentation techniques based on thresholding grayscale images in the initial phase even though this method is generally fast and reliable (with the right images), our typical image histogram doesn't have any recognizable separation of modes, therefore grayscale analysis is not a viable option. In terms of complexity and reliability, the best approach to begin with is to segment the skin colour.

The first step for hand segmentation is the conversion of the captured frame to HSV format. HSV colour space is much better, because in there the information of colour is dissociated from the information of illumination.

```
cvtColor(frame, dst, CV_BGR2HSV);
```

- frame - is of data type InputArray and contains the source image or frame.
- dst - is of data type OutputArray and is the destination space for the HSV converted frame.
- cvtColor - Converts an image from one color space to another. In this case BGR is converted to HSV using the CV_BGR2HSV method.

The next step is to get the skin colour from the image. For normal human skin, the skin tone is usually in the range of $0<H<0.25$ - $0.15<S<0.9$ - $0.2<V<0.95$.

The extracted skin colour segmented image is then undergone binary conversion. For this we are firstly converting the resulting image after applying the skin tone factors into a greyscale image. The HSV converted image which was undergone skin tone segmentation can be easily converted to binary, but in this case I firstly converted it to greyscale image and then undergone binarization as the clarity of the later was much better compared to the direct conversion of the HSV.

C. *Thresholding the image*

The image which is converted into grayscale is then either a static threshold value is used or a threshold value is selected from 0 to 255 according to the user specification which acts as the threshold value. This threshold value should be chosen by the user in such a way that the white blob of the hand and mainly the fingers are segmented with minimum noise possible. For every usage, either the thresholding value is static that is each time same value is used or the user is required to set the threshold value to ensure good level of hand segmentation. Thus, this method is not used since it puts the systems success or failure dependent on the user setting a proper threshold value or on the quality of the static threshold value. This method is useful where the intensity of the hand is almost similar whenever the system is used. Also the background intensity should be similar every time the system is used. But even in constant lighting conditions during every system use, the system might fail depending on the user's hand color. If the user's hand is also darker in color, the system might not be able to separate the user's hands and the dark background [1].

```
threshold(frame_gray, frame_gray, 60,
255, CV_THRESH_BINARY);
```

- Frame_gray - is of data type grayscale converted frame

A binary threshold is a simple "either or" threshold, where the pixels are either 255 or 0. The first parameter here is the image. The next parameter is the threshold, we are choosing 60. The next is the maximum value, which we're choosing as 255. Next and finally we have the type of threshold, which we've chosen as THRESH_BINARY. Normally, a threshold of 10 would be somewhat poor of a choice. We are choosing 60, because this is a low-light picture, so we choose a low number. Normally something about 60-150 would probably work best.

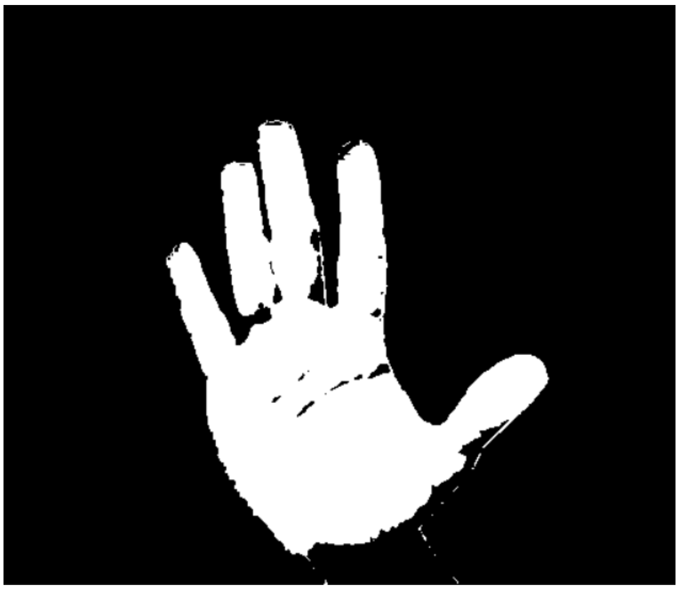
Figure 1.1

After binarization the image resulted a bit noisy, because of false positives. To clean the image and reduce the noise we must filter the image using blurring effect and various morphological operators.
When smoothing or blurring images (the most popular goal of smoothing is to reduce noise), we can use diverse linear filters, because linear filters are easy to achieve, and are kind of fast, the most used ones are Homogeneous filter, Gaussian filter, and Median filter. Homogeneous filter is the simplest filter, each output pixel is the mean of its kernel neighbors (all of them contribute with equal weights). Gaussian filter is nothing but using different-weight-kernel, in both x and y direction, pixels located in the middle would have bigger weight, and the weights decrease with distance from the neighborhood center, so pixels located on sides have smaller weight. In this case , I have used the Gaussian blur to filter the binary image as I need to focus on the middle portion. After blurring we perform morphological operations to the image mainly erosion and dilution.
As with all other morphological filters, the two filters of this recipe operate on the set of pixels (or neighborhood) around each pixel, as defined by the structuring element. Recall that when applied to a given pixel, the anchor point of the structuring element is aligned with this pixel location, and all pixels intersecting the structuring element are included in the current set. Erosion replaces the current pixel with the minimum pixel value found in the defined pixel set. Dilation is the complementary operator, and it replaces the current pixel with the maximum pixel value found in the defined pixel set. Since the input binary image contains only black (0) and white (255) pixels, each pixel is replaced by either a white or black pixel[6]. A good way to picture the effect of these two operators is to think in terms of background (black) and foreground (white) objects. With erosion, if the structuring element when placed at a given pixel location touches the background (that is, one of the pixels in the intersecting set is black), then this pixel will be sent to background. While in the case of dilation, if the structuring element on a background pixel touches a foreground object, then this pixel will be assigned a white value. This explains why in the eroded image; the size of the objects has been reduced. Observe how some of the very small objects (that can be considered as “noisy” background pixels) have also been eliminated. Similarly, the dilated objects are now larger and some of the “holes” inside of them have been filled [2].

```
dilate(frame_gray, frame_gray,
getStructuringElement(MORPH_ELLIPSE,
Size(7, 7)));
erode(frame_gray, frame_gray,
getStructuringElement(MORPH_ELLIPSE,
Size(7, 7)));
erode(frame_gray, frame_gray,
getStructuringElement(MORPH_ELLIPSE,
Size(7, 7)));
dilate(frame_gray,frame_gray,
getStructuringElement(MORPH_ELLIPSE,
Size(7, 7)));
```

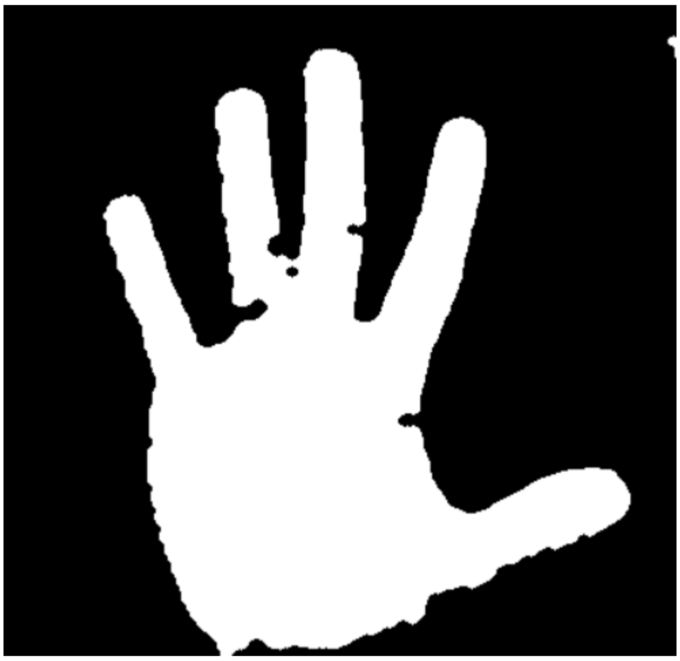
Figure 1.2

After this we find the canny function to display the edges of the binary image. The code the Canny Detector and generates a mask (bright lines representing the edges on a black background). This masked image can be used for finding the contours in the later part.

```
Canny(detected_edges,detected_edges,
lowThreshold,lowThreshold*ratio,
kernel_size);
```

- detected_edges – input array to the canny
- lowThreshold: The value entered by the user
- kernel_size: We defined it to be 3 (the size of the Sobel kernel to be used internally.

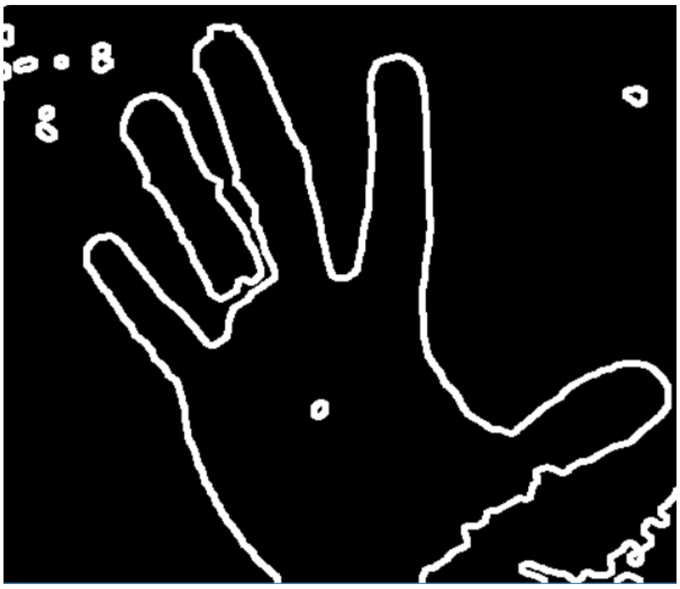

Figure 1.3

D. Contour Detection

A contour is the curve for two variables function along which the function has a constant value. A contour joins points above a given level and of equal elevation. A contour map illustrates the contour using contour lines, which shows the steepness of slopes and valleys and hills. When the lines are close together, the magnitude of the gradient is usually very large. Contours are straight lines or curves describing the intersection of one or more horizontal planes with a real or hypothetical surface with. The contour is drawn around the white blob of the hand that is found out by thresholding the input image. There can be possibilities that more than one blob will be formed in the image due to noise in the background. So, the contours are drawn on such smaller white blobs too. Considering all blobs formed due to noise are small, thus the large contour is considered for further processing specifying it is contour of hand [1].

Now that we have the binary image, we use OpenCV's function findContours to get the contours of all objects in the image. From these we select the contour with the biggest area. If the binarization of the image has been done correctly, this contour should be the one of our fingers.

```
std::vector<std::vector<cv::Point> >
contours;
std::vector<cv::Vec4i> hierarchy;
cv::findContours(detected_edges,
contours, hierarchy, CV_RETR_EXTERNAL,
CV_CHAIN_APPROX_SIMPLE, cv::Point(0,
0));
```

- detected_edges – input array
- contours– is a vector of vector of Point data structure defined by OpenCV
- mode – is the contour retrieval mode. We use CV_RETR_EXTERNAL which retrieves only the extreme outer contours. Other modes are also available like CV_RETR_TREE, CV_RETR_CCOMP, CV_RETR_LIST
- method– is to select the contour approximation method. We use CV_CHAIN_APPROX_SIMPLE which can compress vertical, horizontal and diagonal segments and stores only their end points. This provides sufficient accuracy with lesser storage requirement.

E. Fingerprint Detection

After detecting the contours, we detect the minimum rectangles that are enclosed in the contours. The bounding rectangle which in this case is the fingerprints is drawn with minimum area, so it considers the rotation also. The function used is cv2.minAreaRect(). It returns a Box2D structure which contains following details - (center (x,y), (width, height), angle of rotation). But to draw this rectangle, we need 4 corners of the rectangle. Here we find the minimum rectangles enclosed only in the larger contours.

```
std::vector<RotatedRect> minRect(
contours.size() );
for(size_t i=0;i<contours.size(); i++
){
 minRect[i] =
minAreaRect(Mat(contours[i]) );
if(contours[i].size()> 120) {
Point2f rect_points[4];
minRect[i].points( rect_points );

if(minRect[i].size.height>minRect[i].s
ize.width){
minRect[i].size.height=(float)(1.5)*mi
nRect[i].size.width;
minRect[i].center
=(rect_points[1]+rect_points[2])/2 +
(rect_points[0]-rect_points[1])/6;
 }
else {
minRect[i].size.width
=(float)(1.5)*minRect[i].size.height;
minRect[i].center =
(rect_points[2]+rect_points[3])/2 +
(rect_points[0]-rect_points[3])/6;
 }
```

Inorder to get the rectangle only on the fingerprints, we can set the height and width of the minRect[]. std::vector<cv::Point2f> points[2] declares an array of two std::vector<cv::Point2f> elements here in this case it is rect_points. Hence the minimum rectangles are calculated. Now the minimum rectangles have to be ddrawn to the original frame in order to display the fingerprints. We use the below code to draw the contours

```
drawContours(
InputOutputArray image,
InputArrayOfArrays contours,
intcontourIdx,
const Scalar&color,
int thickness=1,
intlineType=8,
InputArray hierarchy=noArray(),
intmaxLevel=INT_MAX,
Point offset=Point() );
```

- image – is the input image on which the contours
  should be drawn. In our case this is the input frame
- contours– is a vector of vector of Point data structure defined by OpenCV. These are the contours that we want to draw. In this case we use the minimum rectangles as the contours for drawing.
- contourIdx – index of the contour to be drawn from
  the contours vector. If this value is negative then all the contours in the parameter contours are drawn
- color – is of the data type Scalar and it is a constant value. This is the color to be used to draw the contours
- thickness – defines the thickness of the line to be used to draw the contours. If this value is negative, then the contours are filled with the color specified. Default value is set to 1, in this case I had given 2.
- lineType – is to select how the lines are drawn whether they are 8-connected line, 4-connected line and CV_AA for antialiased line
- hierarchy – this is optional information only required when only some of the contours are required to be drawn
- offset – optional offset to add to contour points while drawing

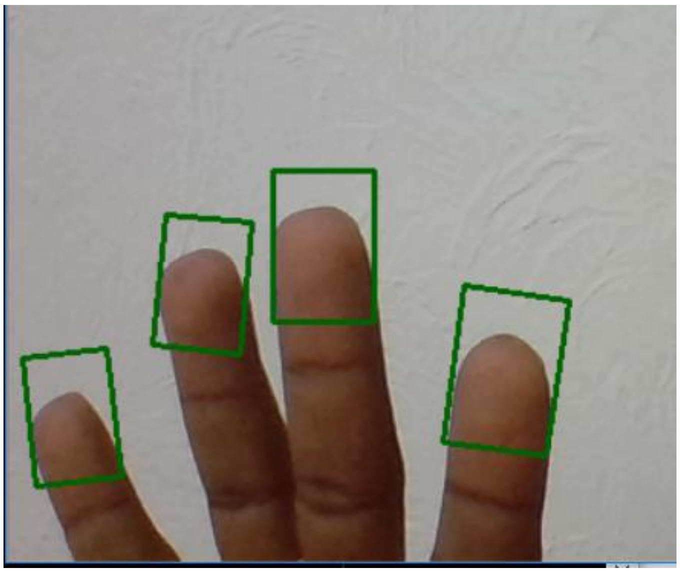

Figure 1.4

V. LIMITATIONS AND IMPROVEMENTS

- Initially we have to assume the fact that the camera is static here and the user is stationary. Improvements can be made to provide a more real time-based system.
- The accuracy of this solution is not the always the best and the program may need to be tuned differently for different environments.
- This software has been written with limited knowledge of both OpenCV and C++, there’s probably plenty of room for optimizations and performance improvements.
- The samples of the user’s skins are collected in an extremely simplistic way. There’s a lot of room for improvements here.

VI. FUTURE WORKS

We can expand on the system in the future to improve detection and tracking to overcome the limitations, for example apart from using just skin colour for detection. Also, here just the fingerprint is detected we can implement a much deeper software to identify the fingerprints in order to identify a person that can be used for personal identification and security. Similarly, we can use other techniques for tracking and especially we can work on making the recognition phase more autonomous and recognize the gestures in real time as well.

ACKNOWLEDGMENT

I would like to express my gratitude to Professor Christof Jonietz, for his support and guidance provided during the work on the implementation of these techniques and while writing this paper. Also I would also want to thank my friends for helping me completing this project.

REFERENCES

[1] Amiraj Dhawan, Vipul Honrao,Implementation of Hand Detection based Techniques for Human Computer Interaction, Fr. Rodrigues Institute of Technology, VashiNavi Mumbai

[2] Lakshminarayanan(s-ln.in), Hand Tracking And Guesture Recognition

[3] Abbas Cheddad, Joan Condell, Kevin Curran and Paul Mc Kevitt, A Skin Tone Detection Algorithm for an Adaptive Approach to Steganography, pp. 2.

[4] Vladimir Vezhnevets ∗ Vassili Sazonov Alla Andreeva, A Survey on Pixel-Based Skin Color Detection Techniques

[5] O'reilly, Learning OpenCV, Computer Vision in C++ with the OpenCV Library, Adrian Kaebler and Gary Bradski.

[6] Philip Krejov and Richard Bowden, Multi-touchless: Real-Time Fingertip Detection and Tracking Using Geodesic Maxima

[7] https://hub.packtpub.com/opencv-image-processing-using-morphological-filters/