國 立 台 灣 科 技 大 學

資 訊 工 程 系

碩士學位論文

基於前臂輪廓的即時掌心追蹤以及手勢判斷

Real-Time Palm Tracking and Hand Gesture Estimation Based on Fore-Arm Contour

陳威詔

M9815024

指導教授：范欽雄　博士

中華民國 一百 年 七 月　　日

中文摘要

在人機介面發展的歷程中，一直朝著人性化與簡單化的方式不斷的持續成長。尤其是近幾年更是有爆炸性的突破，隨著各種新方式的出現，傳統的硬體按鍵輸入已逐漸被取代。觸控面板從單點控制進入到多點控制，體感技術的出現更擺脫了按鍵的束縛，讓人機介面更貼近人類的行為模式。體感中最具重要性的部位便是手部的控制。鑒於此，我們希望提出對手部重要資訊能夠更精確定位的系統。

本論文提供一個使用視訊攝影機之影像處理系統。和以往手勢辨識不同的地方在於，我們並不是將手勢化成數種指定的指令，而是藉由計算幾何的方式將手部的重要資訊：手指、手掌的位置準確的標出，提供資訊讓手部與系統做即時的互動。利用計算幾何帶來的優點，本系統可以在包含前臂的情況下準確的判斷出手心的位置，並且容許手掌和手臂一定程度的翻轉。大大的提高了掌心辨識所能掌握的自由度。

實驗結果顯示指尖的辨識準確度為 99.1%，在手掌手臂不旋轉傾斜下的掌心辨識率為 99.91%，在手臂傾斜下的掌心辨識率為 99.53%，在手掌傾斜下的掌心辨識率為 93.57%，在手掌手臂旋轉下的掌心辨識率為 90.69%。

關鍵字：　人機互動、手勢辨識、指尖偵測、掌心偵測、計算幾何、電腦視覺

Abstract

The development of HCI (human computer interface) is continuously pursuing an user-friendly interface and simplification system. There are some innovated breakthroughs recently. The traditional keyboard-input has been replaced gradually since many new methods had been invented. The touch panel evolves from single-touch to a multi-touch interface. The body sense technology even gets rid of restriction of input device; make the HCI closer to human's nature action. The most important part of it will be manipulation using hand. Hence, we propose a system which is accurate in locating some important features of hand.

In this thesis, we proposed an image processing system using a web camera. Differently from other hand recognition method, we are not trying to transfer the gesture to some certain instructions. We mark up the important features of hand: fingertips, palm center by computation geometry calculation, provide real-time interaction between gesture and the system. Within the advantages brought by computation geometry method, our system can accurately locate the palm center even when the fore-arm is involved. And the system tolerates a certain rotation of palm and fore-arm, which enhances the freedom of use in palm center estimation.

The experiment result shows that the accuracy is 99.1% for fingertip detection. Accuracy for palm position estimation will be 99.91% when the arm and hand are not tilt and rotate. Accuracy for palm position estimation will be 99.53% when the arm is tilt. Accuracy for palm position estimation will be 93.57% when the hand is tilt. Accuracy will be 90.69% for palm position estimation when the arm and hand are rotated.

Keyword: HCI, Gesture Recognition, Fingertip Detection, Palm Detection, Computation Geometry, Computer Vision.

致謝

Table of Contents

List of Figures

List of Tables

Chapter1 Introduction

1.1 Overview

In recent years, the HCI has become more and more widespread and diversified. Digital devices nowadays are transforming into many different types, smart phones and tablet computer are the current trend. They change the way how people use computer. Also the appearance of Microsoft Kinect and Wii redefined our knowledge of manipulation interface. The traditional input devices like keyboard and mouse are being replaced.

In the future, computers won't be just computers. Many daily devices will be integrated with computers, like cars, television, household appliances, etc. Since so many types of machines will come up, we need various HCIs to cope with different type of machines. The hand gesture will be the most popular one. It is the most intuitive way for human to communicate with their body.

With the communication by hand, people will no longer need to use computer devices in a certain location. For example, if a TV is integrated with gesture recognition function, people can manipulate it everywhere in their house. A remote control will be needless. The gap between human and machines will be closer. It increases the freedom of usability. Another example will be the outdoor interactive advertising board. More and more related products will appear and the gesture interface will be more important.

1.2 Motivation

As previous section described, we believe the hand gesture interface will be a future trend. It breaks the restriction of devices and space. The most important features of hand gesture will be fingertips and palm. Most research in hand gesture recognition try to convert a certain hand gesture into a particular symbol. In those cases, hands can only perform several simple instructions.

If we want to implement a gesture interface which is more interactive, the position locating of important features of hand will be the most important task to do. Unlike fingertips, palm center doesn't have apparent characteristic. Many researches have defined the palm center in many different ways, but they can only be effective in certain conditions. Some will be ineffective if the fore-arm is included and some will be ineffective when the palm is tilted or rotated.

Our goal is to locate those important features of hand more accurately and increase the toleration of different conditions, like fore-arm participation, rotation and tilt. The gesture interface can be more widely applied if the fingertips and palm can be located more accurately and the freedom of usability is increased.

1.3 System Description

Our system and its computation are based on the contour of hand and fore-arm. In our system, we assume that the input contour is the contour of hand and fore-arm. We are not performing any hand or arm recognition. The original input data will be the RGB image captured by a web camera. We perform some preliminary processing to generate a binary image which provides enough information of hand contour. The binary image will be used to calculate the contour and the convex hull of the contour. The palm position can be initially estimated by the information we extracted from

convex hull, the fingertips position can be detected from the hand contour. Then the system will make some modification of the initially estimation to make our results more accurately. The computation cost of our system is low since only computation geometry algorithms are performed; and there's no need to apply classifier and no pre-trained data base is needed.

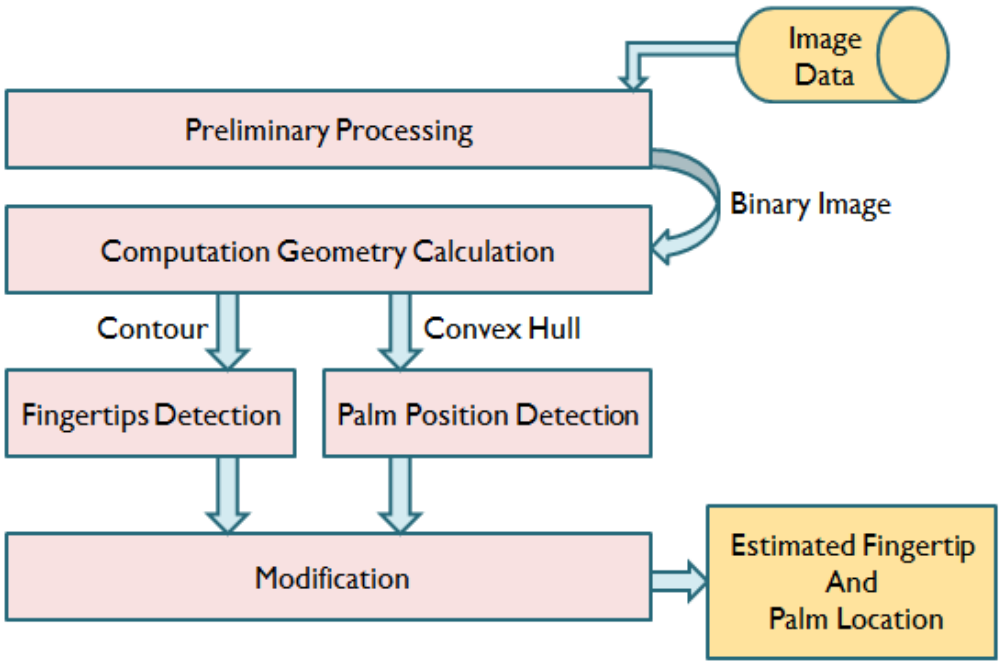


Figure 1.1 System architechture overview

1.4 Thesis Organization

The thesis is consisted of six chapters. In chapter 2, some important related works will be reviewed. Chapter 3 will discuss our methods to obtain the binary image and calculate the contour and convex hull. Chapter 4 focuses on palm and fingertip position detection. Chapter five shows some experimental results. In chapter six, we will make a conclusion and discuss about future works of our system.

Chapter 2 Related Works

In this chapter we will discuss the related works, which contain palm and fingertip detection, hand segmentation, gesture recognition based on contour. Later in the experiment result section, we will make some comparisons between our method and the related works. H.C. Xu [1] detects fingertips by wearing a special glove with LED on the fingertips. Different fingers are equipped with LED of different colors. The restrictions of the method are: must wear the glove and be under dark environment.

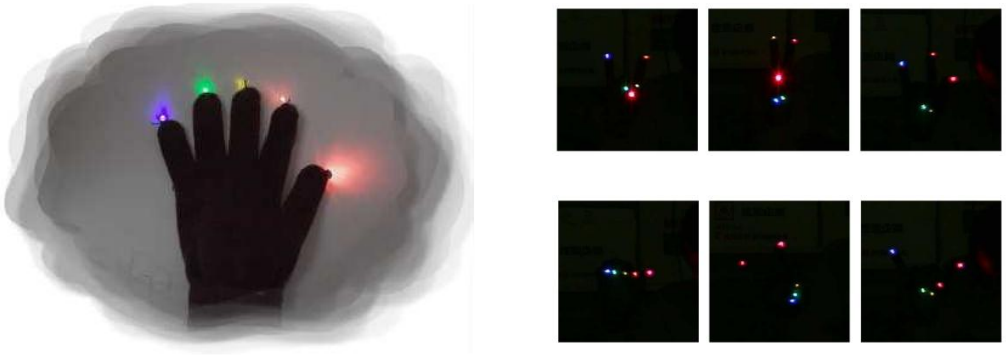


Figure 2.1 Special type of glove with each fingertip equipped with different color LED lamp

Anagnostopoulos and Pnevmatikakis [2] find the palm center by calculating the distance between a skin color point and the nearest point on the contour. Every point will have its own distance; they assume the point which has the largest distance to be the palm center.

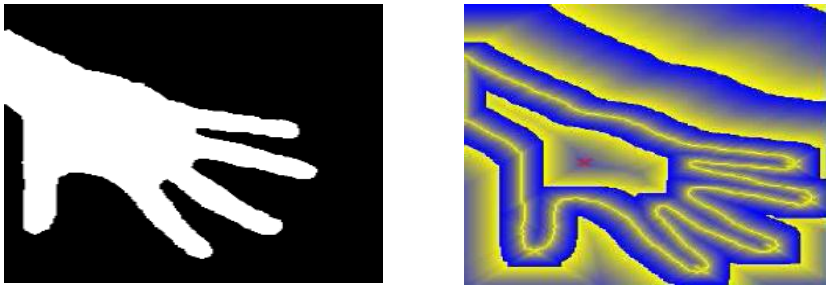


Figure 2.2 Red dot represents the point which has the largest distance

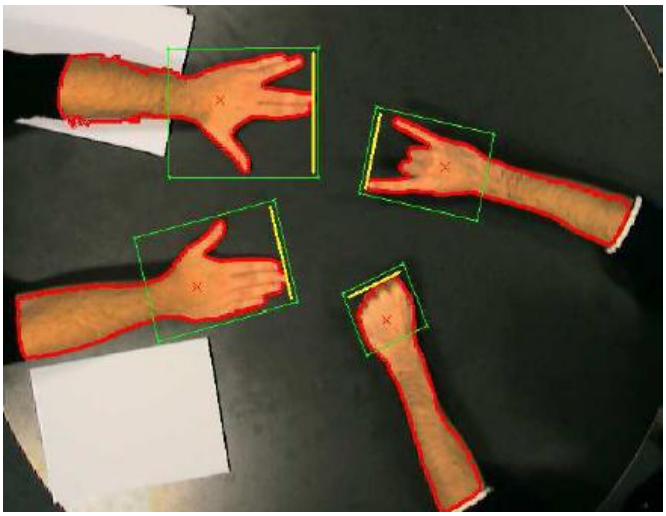


Figure 2.3 Estimated palm center from the contour

G. Amayeh et al. [3] proposed a human hand verification system. In their system, they separate each fingertip and the palm to perform further measurement and comparison. They separate the wrist and arm by finding the biggest circle inside the hand region. When the arm area is removed, they use their pre-defined structure element to filter out the fingers.

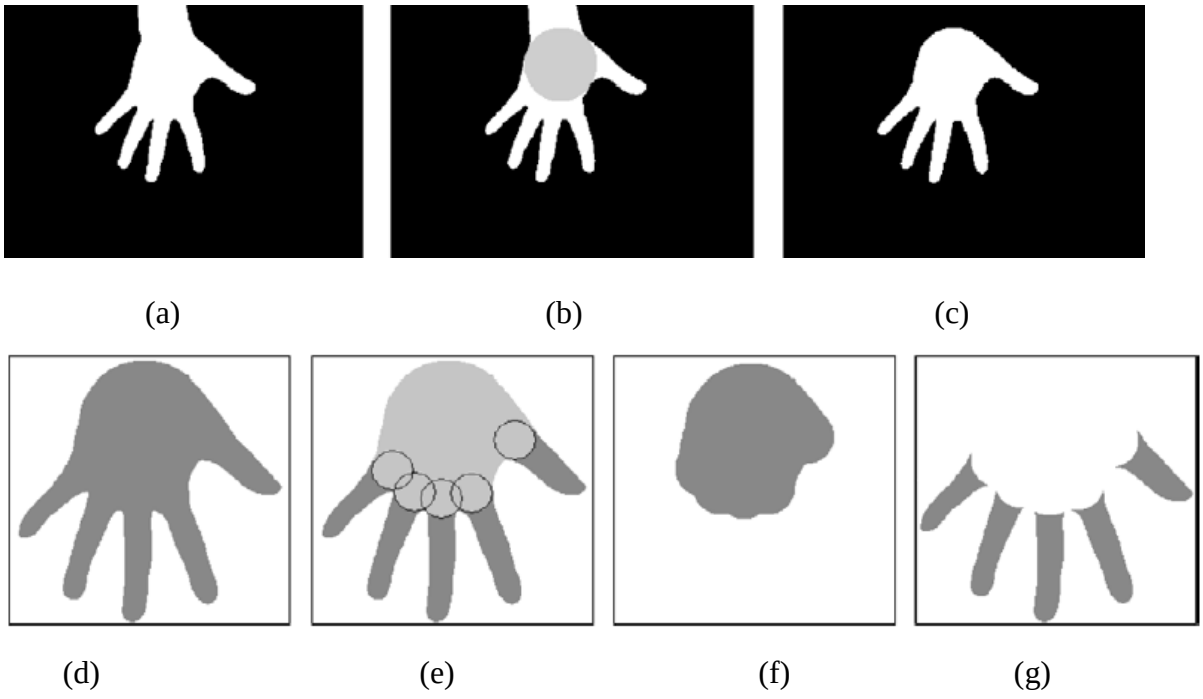


Figure 2.4 (a) binary image (b) the biggest circle(c) remove arm (d-g) palm and finger segmentation

Fujiki and Arita [4] proposed a method to estimate 3D hand gesture from a 2D hand image. They cut the wrist and the arm by searching for the minimum number of skin-color pixels projected on x-axis. When the arm is cut off, they use the same method as [] to find the palm center.

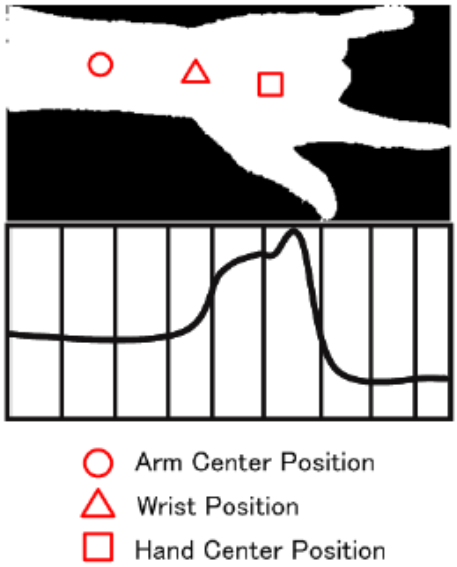


Figure 2.5 Number of skin-color pixel projected onto the x-axis

Chang [5] proposed a method to trace hand-pose trajectory. When to contour of hand has been extracted, they use least-square approximation to find the lest-square line. Then they search contour vertically along the lease-square line from the bottom to the top.

Figure 2.6 Width search vertically along the least-square line

When the increased size of the vertical line is larger than a certain threshold compare to several previous lines, they will cut off the hand regions below the line and take the remained part as palm and fingers.

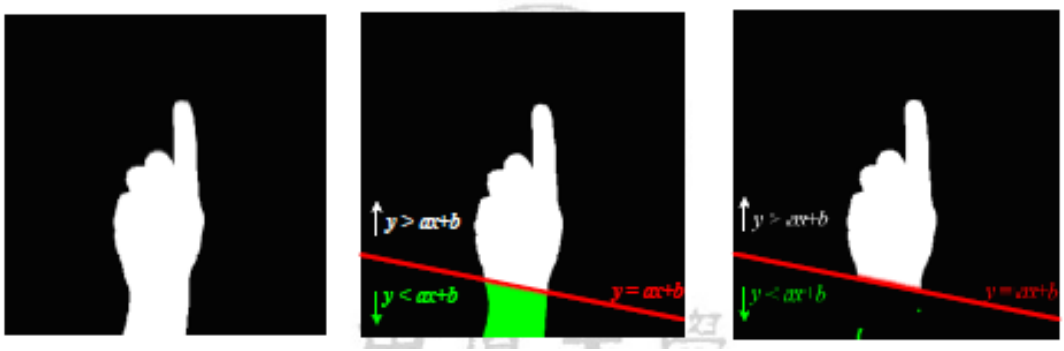


Figure 2.7 Wrist cut off

And the estimated radius in each corner is calculated. The region outside of the radius is considered the finger part. A lot of modifications need to be applied then.

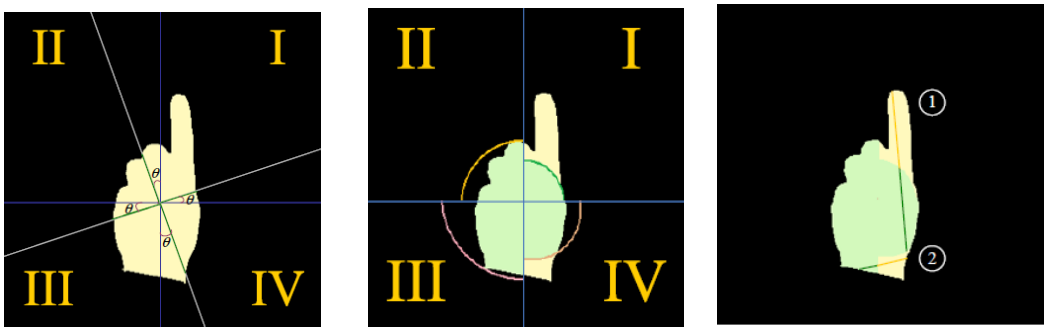
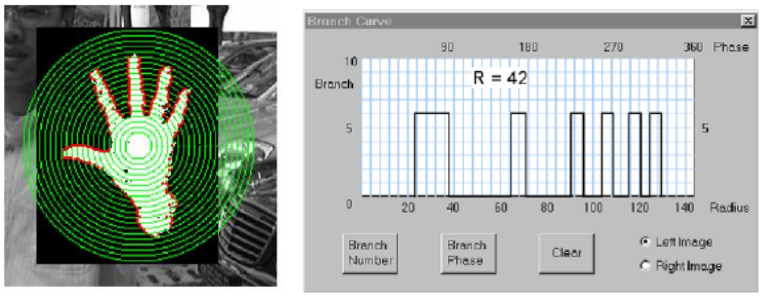


Figure 2.8 Separating the finger and palm by calculating the desirable radius of each corner

Xiaoming Yina and Ming Xi [20] proposed another method to do the finger detection. First locate the center of the palm; then establish several different radiuses from the center in order to search for the fingers. The circle with the largest number of fingers covered is chosen. By employing the decision tree for various hand gestures, we can complete the identification.



(a) (b)

Figure 0.9 (a) feature points extracted from the binary image of the segmented hand region, (d) plot of branch phase of the hand on the selected search circle.

Chapter 3 Pre-Processing and Convex-Hull Method

In this chapter, we will introduce how to extract the interested area from color image. We use a single web camera to capture a series of images. After transforming the image into HSV color space, we might be able to extract the interested area by defining the range of hue, saturation and value. A binary image will be produced and morphology processing will be performed. Erosion will eliminate the noises while the dilation will fill up the defects to smooth the contour of interested area.

Theo Pavlidi's algorithm [7] is used to obtain the contour sequence. Instead of using connected-component, we can determine the contour we are interested in by choosing the longest sequence. The convex-hull can be found by the Three-Coins algorithm [8, 9] , it's a set of points which can wrap the contour like a rubber band. By comparing the convex-hull area and contour area, convexity defect area can be calculated. Convexity defect means the area which is in the convex-hull area but not in the contour area. Convexity defects help us to find the features of a contour. The depth of a defect area is the most important feature in this thesis. Palm position can be detected by a set of defection depths. The start points and end points and also help us to find the convex points in the contour. But it would be invalid in some conditions in hand gesture estimation. We will discuss this problem in this chapter and another algorithm will be applied in the next chapter to solve the problem.

3.1 Preliminary Processing

The color model captured from a web camera is composed of RGB values, but it will be influenced by the light very easily, we must convert the RGB color space into another color space which is not sensitive to light variation. In addition to RGB, there are some other commonly used color spaces such as Normalized RGB [10], HSV [11], YCbCr [12], and so forth. In order to make the system achieve real-time processing and adapt to most of the environments, we choose the color model with simple converting formulas and low sensitivity. In this thesis, we choose HSV color space to extract human skin areas. The skin color areas will be represented in binary image. Hence we will perform morphology processing including erosion and dilation. Noises will be eliminated by erosion and the contour of skin color area will be smoothed by dilation.

3.1.1 Skin color detection using HSV color space

The HSV model comprises three components of a color: hue, saturation, and value. It can distinguish the value from the hue and saturation. A hue represents the gradation of a color within the optical spectrum, or visible spectrum of light. Saturation is the intensity of a specific hue, which is based on the purity of a color. The practicability of the HSV model can be referred to two main factors: 1) the value separated from the color image, and 2) both the hue and saturation related to human vision. Hence, for developing a human vision-based system, the HSV model is an ideal tool [11]. The chroma component (hue and saturation) and the luminance component (value) of the HSV color model are defined by the stereo picture as shown in Figure 3.2. All colors composed of three so-called primary colors (red, green, and

blue) are inside the chromatic circles. The hue H is the angle of a vector with respect to the red axis. When $H = 0^\circ$, the color is red. The saturation S is the degree of a color that has not been diluted with white. It is proportional to the distance from the location to the center of a circle. The longer distances are away from the center of a chromatic circle, the more saturation the colors will be. The value V is measured by the line which is orthogonal to the chromatic circle and passes through the center of the circle. It tends to be white along this center line to the apex.

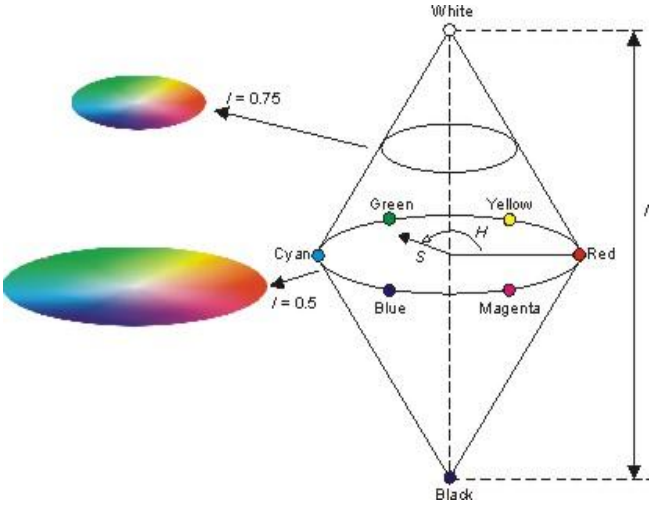


Figure3.1 The stereo picture of the HSV color model.

The relationship between the HSV and RGB models is expressed below:

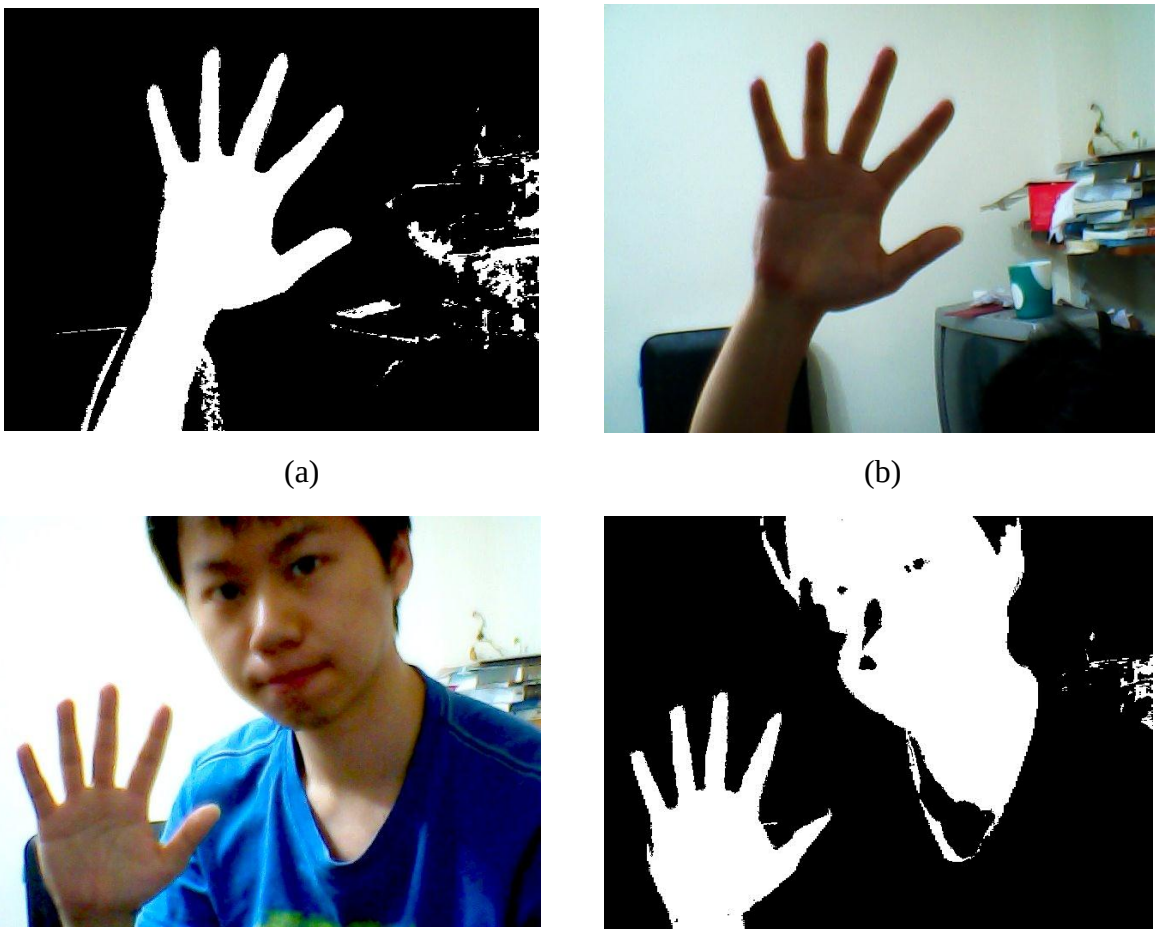
$$H = \begin{cases} \theta & \text{if } B \le G \\ 360^\circ - \theta & \text{if } B > G \end{cases} \tag{3.1}$$

$$\text{where} \quad \theta = \cos^{-1}\left\{ \frac{\frac{1}{2}\left[(R-G)+(R-B)\right]}{\left[(R-G)^2+(R-B)(G-B)\right]^{\frac{1}{2}}} \right\} \tag{3.2}$$

$$S = 1 - \frac{3}{(R+G+B)}\left[\min(R,G,B)\right] \tag{3.3}$$

and $$V = \max(R,G,B) \tag{3.4}$$

We distinguish the skin color from non-skin color regions by setting upper and lower bound thresholds. In our experimental environment, we choose the H value from 2 to 39 and from 300 to 359; the S value between 0.1 and 0.9 for the range of skin colors. Some of skin color detection results are shown in Figure 3.2.



(a) (b)

(c) (d)

Figure 3.2 (a) Binary image (b) Original image (c) Original image (d) binary image

3.1.2 Morphology Processing

After the generating of the binary image, we further perform morphological operations on them to smooth hand segments and it also can eliminate some noises. Two basic morphological operations are dilation and erosion. These two morphological operations are described below [11].

For two sets in Z^2, A and B, the dilation of A by B denoted $A \oplus B$ is defined as

$$A \oplus B = \left\{ z \,|\, (\hat{B})_z \cap A \neq \phi \right\} \tag{3-5}$$

This equation is based on obtaining the reflection of B about its origin and shifting this reflection by z. Then the dilation of A by B yields the set of all displacements, z, such that $\hat{B}$ and A overlap with at least one element. Therefore, Eq.(3-5) may be rewritten as

$$A \oplus B = \left\{ z \,\middle|\, [(\hat{B})_z \cap A] \subseteq A \right\} \tag{3-6}$$

Set B is commonly referred to a structuring element in the dilation operation as well as in other morphological ones. Figure 3.3 illustrates a simple result of dilation. Figure 3.3 (a) shows a simple set, and Figure 3.3 (b) shows a structuring element and its reflection. Figure 3.3 (c) shows the original set for reference and the solid line indicates the limit beyond which any further displacements. Therefore, all points inside this boundary constitute the dilation of A by B.

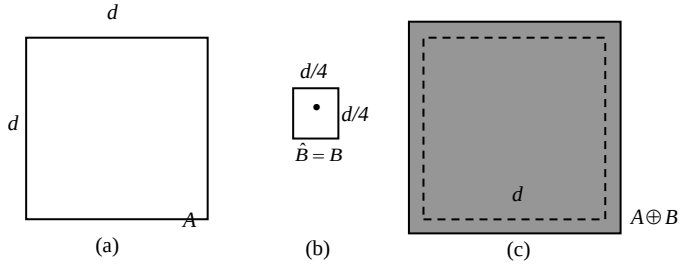


Figure 3.3 Dilation operation; (a) set A; (b) square structuring element B; (c) dilation of A by B.

Given two sets A and B in Z^2, the erosion of A by B denoted $A \ominus B$ is defined as

$$A \ominus B = \left\{ z \middle| (B)_z \subseteq A \right\} \tag{3-7}$$

In words, this equation manifests that the erosion of A by B is the set of all points z such that B, translated by z, is contained in A. As in the case of dilation, Eq. (3-7) is not the unique definition of erosion. However, Eq. (3-7) is usually favored in practical implementations of mathematical morphology for the same reasons stated earlier in connection with Eq. (3-5). Figure 3.4 demonstrates a simple result of erosion where, Figure 3.4 (c) shows the original set for reference, and the solid line stands for the limit beyond which any further displacements. In consequence, all points inside this boundary constitute the erosion of A by B.

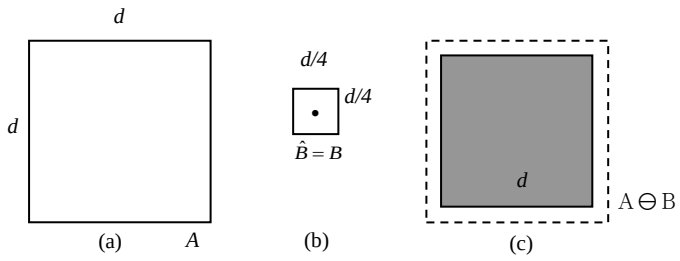


Figure 3.4 Erosion operation; (a) set A; (b) square structuring element B; (c) erosion of A by B.

The erosion and dilation operations are both the conventional and popular methods used to clean up anomalies in the objects. The combination of these two operations can also achieve good performance. Therefore, a moving object region is morphologically eroded (once) then dilated (twice). Some examples of morphological operation are demonstrated in Figure 3.5

(a) (b)

(c)

Figure 3.5 (a) input binary image (b) Erosion (c) Dilation

3.1.3 Contour Finding

After the previous processing, we should be able to obtain a binary image. The white pixels represent for skin color regions. The next step will be contour finding. A contour is a sequence of points which are the boundary pixels of a region. The contour of those regions will be found so that we can disregard those small areas and focus on the fore-arm area we are interested in. It can be done by comparing the length of their contour. The longest one is the one we are looking for. Unlike connected-component,

contour finding doesn't need to trace the whole pixels in an area. Only boundary pixels will be visited. It brings not only lower computation cost but also comes with the contour information we need. In this thesis, we use Theo Pavlidis' Algorithm [7] to find contours. It works very well on 4-connected patterns which the fore-arm contours are always tending to be.

The algorithm begins with a start point. Locating a start point can be done by scanning each row of pixels from the button left corner. The scan of each row starts from the leftmost pixel proceeding to the right. When a white pixel is encountered, declare that pixel as start pixel. After deciding the start point, there are 3 pixels which we are interested in. That will be P1, P2, P3 shown in Figure 3.6; P1 is the upper-left pixel with respect to the start. P2 is the upper pixel with respect to the start. And P3 is the upper-right pixel with respect to the start.

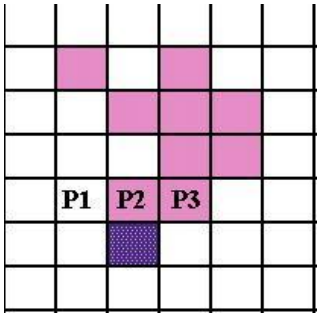


Figure 3.6 Position of p1, p2, p3 relative to start point

When locating the start pixel, there are four cases. The first thing to do is to check P1. If P1 is a white pixel, declare P1 to be your next start pixel and move one step forward followed by one step to your current left to land on P1 as shown in Figure 3.7. The next P1, P2 and P3 will be changed according to the location and change of orientation.

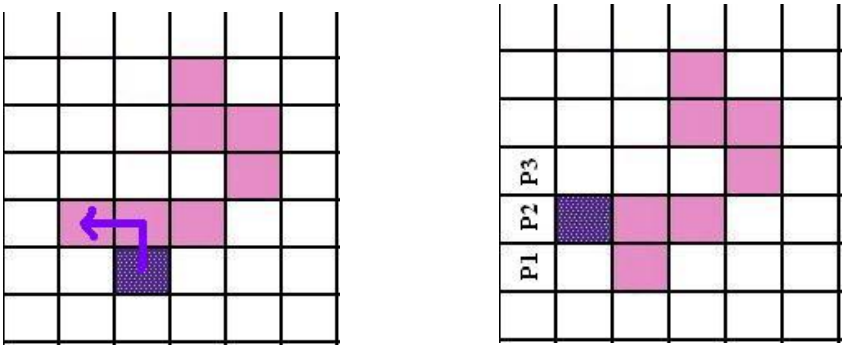


Figure 3.7 Move to p1 and change direction

Only if P1 is a black pixel, then we check P2. If P2 is a white pixel, we declare P2 to be the next start pixel and move one step forward to land on P2 as shown in Figure 3.8. The next P1, P2 and P3 will be changed according to the location and change of orientation.

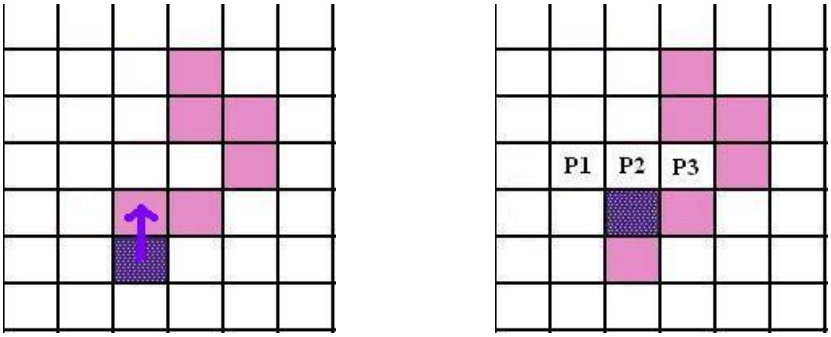


Figure 3.8 Move to p2

If P1 is black pixel and P2 as well, we check P3. If P3 is a white pixel, we declare P3 to be the next start pixel and move one step to your right followed by one step to your current left to land on P2 as shown in Figure 3.9. The next P1, P2 and P3 will be changed according to the location and change of orientation.

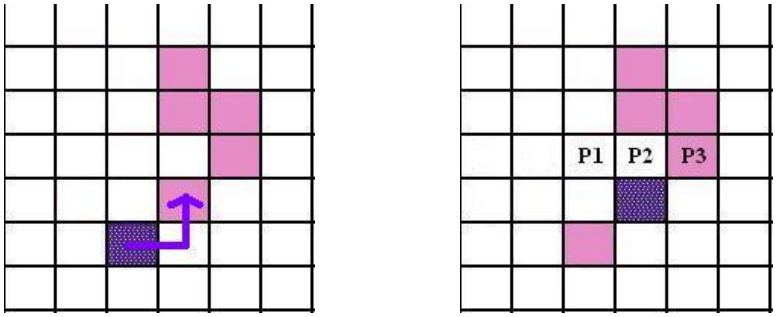


Figure 3.9 Move to P3

If none of P1, P2, P3 are white pixel, the orientation will be rotated 90 degrees clockwise while standing on the same pixel as shown in Figure 3.10. Afterwards we do the same procedure all over again.

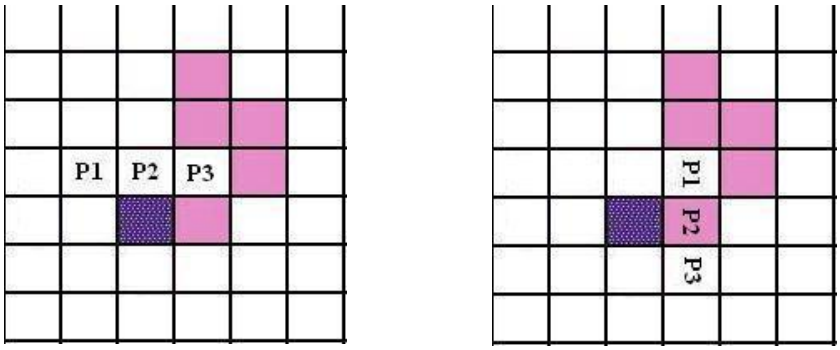
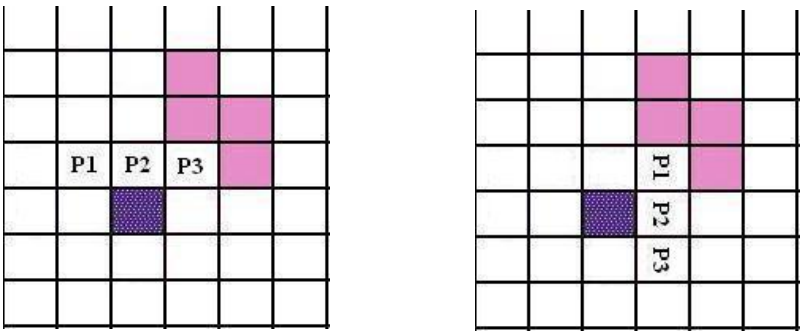
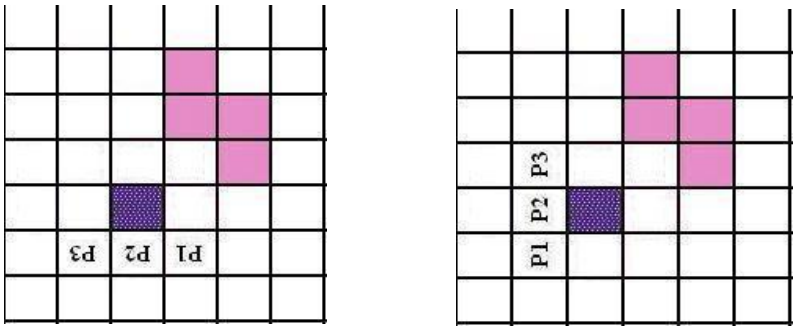


Figure 3.10 Rotate when p1, p2 and p3 are none

If the orientation has been rotated 3 times continuously without finding any white pixel in P1, P2 or P3, it means that we are locating on an isolated pixel. The pixel is not connected to any white pixels. This situation will cause the algorithm to terminate as shown in Figure 3.11



(a) (b)



(c) (d)

Figure 3.11 (a) p1, p2, p3 are none (b) first rotation (c) second rotation (d) third rotation

Formal Description of Pavlidis' Algorithm
Input: A binary image containing connected group of white pixel Output: A sequence **B (b_1, b_2 ,..., b_k)** of boundary pixels as contour.

Begin

- Scan each pixel in rows from the bottom-left until you find a white pixel **s**
- Insert **s** in **B** and set **s** to be the starting pixel
- While (**s** haven't been visited twice)

  If pixel **P1** is white
  - Insert **P1** in **B**
  - Set **s=P1**
  - Move one step forward followed by one step to your current left

  else if **P2** is white
  - Insert **P2** in **B**
  - Set **s=P2**
  - Move one step forward

  else if **P3** is white
  - Insert **P3** in **B**
  - Set **s=P3**
  - Move one step to the right, update your position and move one step to your current left

  else if s has been rotated 3 times
  - terminate the process and declare **s** as an **isolated** pixel

  else
  - rotate 90 degrees clockwise while standing on the current pixel **p**

End

Table 3.1 Pavlidis' Algorithm

When the search is finished, couple of contours could be found in the image. If the longest contour in the image is larger than a certain threshold, we choose the contour to be the one we are interested in. Usually there will be only one or zero large contour in the image otherwise the skin color segmentation might not be effective. By performing the operation, noises in the image will be ignored. We can then focus on the fore-arm contour.

(a)

(b)

(c)

Figure 3.12 (a) input binary image (b) Contours (c) select the longest contour

3.2 Convex Hull

Finding a convex hull is a well-known problem in computation geometry. Let us imagine that there are several nails on the wall, a rubber band is used to surround those nails. Only the peripheries will touch the rubber band while the nails inside won't be able to affect the rubber band. The shape of the rubber band is probably how the convex hull of the nails will be looked like. It is obviously not difficult for a human to understand the shape of convex hull just in a glance. But as for machines, an algorithm is needed.

We calculate the convex hull of the fore-arm contour in order to find the desired information. Usually the arm part has a smooth contour and doesn't contain any important information. The hand part has more convex and concave contours and it usually contains the information we want. After the comparison of a fore-arm contour and its convex hull, we find out that the convexity defects are around the palm area. Hence we might be able to find the points which have the longest distance to the convex hull in each convexity defect. Those points are on the edge of palm section since the convexity defects are around the palm area. Since the fore-arm is relatively smooth, so the neighbor defect usually won't create a point which has the longest distance to the convex hull on the arm contour. By these points, we can determine the position of palm even the fore-arm contour is included in the contour.

The following section will introduce the convex polygon, and the 3-coin algorithm which is applied to find the convex hull of a set of points. After the convexity defects are found, we will discuss about the features we can find through it.

3.2.1 Convex Polygon

Before we begin to explain convex hull, the concept of convex polygon needs to be introduced first. A convex polygon is a polygon which does not contain any concave part. In Figure 3.13, the arrow is pointing to a concave part of a polygon. Although this character means “convex” in Chinese words, the character itself is not a convex polygon.

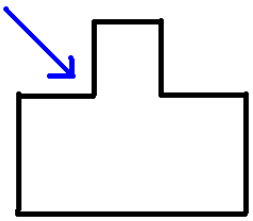
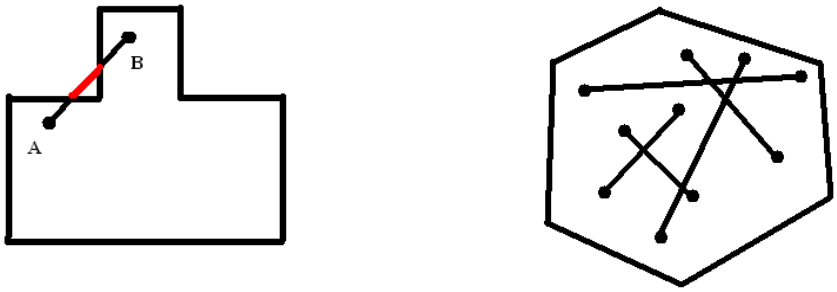


Figure 3.13 Not a convex polygon

In mathematics, there is a strictly definition of a convex polygon. Assume that we can choose any two points in a polygon (includes the boundaries and the area which the boundaries are covering) and connect the two points together within a straight line. If all the straight lines that any two points inside the polygon can form doesn’t exceed the boundary of the polygon, we can say that the polygon is a convex polygon. Figure 3.14 shows that we can find two points (A and B) and connect them within a straight line. We can obviously see that part of the line (which is represented in red color) exceed the boundary, so we can say that the polygon is not a convex polygon.



(a) (b)

Figure 3.14 An example of a convex polygon, all the connected line of two points will never exceed the boundary. (a) Not a convex polygon (b) Convex polygon

3.2.2 Definition of Convex Hull

Once we know the definition of a convex polygon, we might be able to learn the concept of convex hull. For a given nonempty set of points in a plane, the convex hull of the set is the smallest convex polygon which covers all the points in the set. For example, in Figure 3.15 there are 10 points. The hexagon in the figure is the convex hull of the set. The six points which compose the hexagon are called “hull points”.

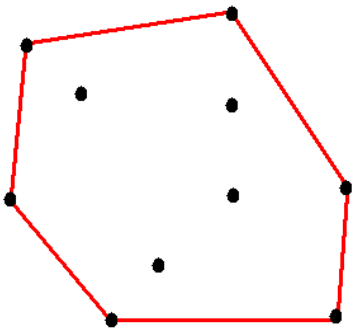


Figure 3.15 An example of convex hull

We know that a set S in a plane or in space is a convex polygon (or convex set) if and only if whenever points X and Y are in S, the line segment XY must be contained

in S. The intersection of any collection of convex sets is also convex, as shown in Figure 3.16. For an arbitrary set W, the convex hull of W is the intersection of all convex sets containing W. The boundary of the convex hull is a polygon with all the vertices in W.

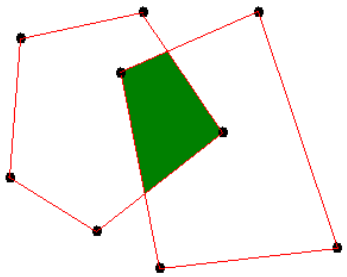
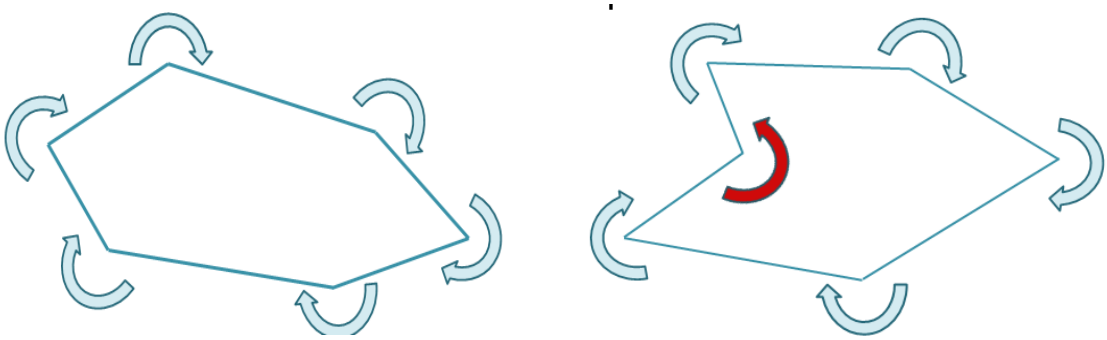


Figure 3.16 Any intersection of convex set is also a convex set.

3.2.3 Three-Coin Algorithm

The three-coin algorithm was developed by Graham and Sklansky [8, 9], trying to find the convex hull of a given set of points. To understand the algorithm, there's one thing we need to realize. Suppose there is a convex polygon, if we take a walk along the polygon's edge, we would only make right turns (or left turns). If it is not a convex polygon, a turn in an opposite direction will be expected.



(a) (b)

Figure 3.17 (a) convex polygon (b) Not a convex polygon

Graham's algorithm is slightly different than Sklansky's algorithm. In Graham's algorithm, the input will be a set of points in a plane. Sklansky's algorithm was designed for a simple polygon in a plane. Simple polygon means the polygon which doesn't have any intersected edges. Graham's algorithm needs to do a sort for all the points as show in figure 3.18. The sort gives number to each point, and the algorithm processes according to the order. Sklansky assumes the input to be a polygon. Once the starting point is decided, the algorithm follows the edges to process, which is just like following the orders. Since the fore-arm contour we found is a series of points, all of the points will be stored in a sequence structure, we obviously have the order of all the points. So we can skip the Graham's sorting. Both algorithms are the same except the sorting.

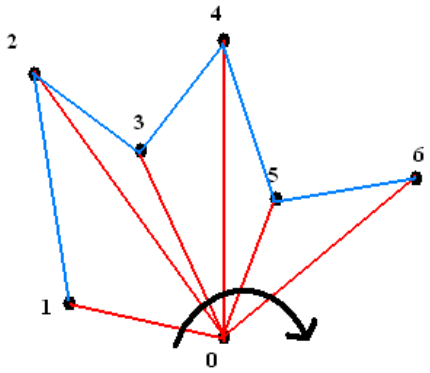


Figure 3.18 Graham's sorting

The fore-arm contour will be the input for the Sklansky's algorithm. The three-coin algorithm means that 3 points will be labeled each time. They will be represented in different colors (for example, black, red and grey). The following figures will illustrated the algorithm. Let's take the polygon in figure 3.19 for example.

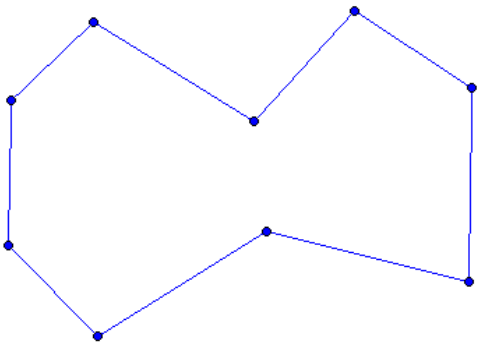


Figure 3.19 An example of input points

First of all, we choose a point to be the starting point, mark this point as a black coin. The starting point must be a convex vertex. We can choose the point which is the left most. The point after black point will be marked as a red coin. The point after the red coin will be marked as grey point. We also called them end coin, middle coin and front coin according to their order and regardless of their color.

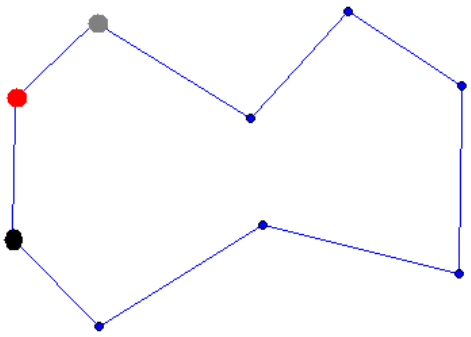


Figure 3.20 Place three coins

Let's check the path from the end coin (currently the black coin) to the middle coin (currently the red coin), then arrive the front coin (currently the grey coin). The path forms a right turn. Whenever encounter a right turn, move the coin at the end coin to the point next to the front point. So we have the black coin as front coin while the grey coin is the middle coin and red coin turns out to be the end coin.

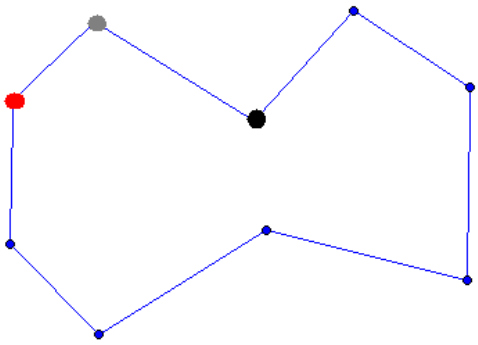


Figure 3.21 Move the end coin next to front coin

Once again, we check the path of the three coins. It forms a right turn again. So we move the end coin to the point next to the front coin. The front coin is now the red coin while the middle is the black point and the grey point become the end coin.

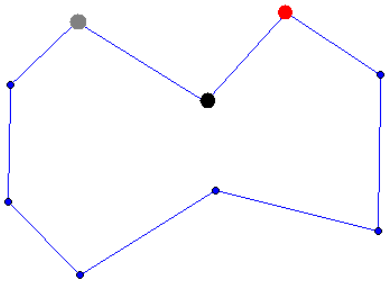


Figure 3.22 Move end coin next to front coin

We check the path of the three coins, this time it forms a left turn. Whenever we encounter a left turn, we should delete the point where the current middle coin stands on, and then move the middle coin to the point before the end point (currently the grey point). Right now, our front coin remains the same (the red coin). But the middle coin has been changed to the grey coin and the black coin is the end coin.

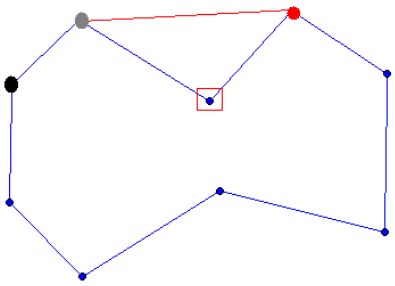


Figure 3.23 Remove middle point

Let’s check the path again. It’s a right turn this time. So we move the end coin to the point next to the front coin.

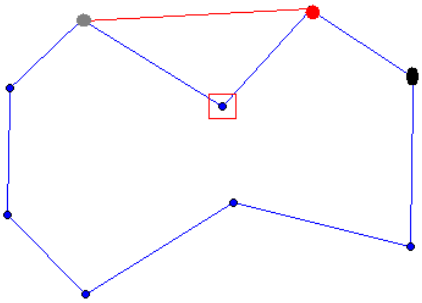


Figure 3.24 move end point next to front point

It forms a right turn again. So we move the end coin to the point next to the front coin. Previous step still forms a right turn, so we do the same procedure again. Still makes a right turn, so we do the same procedure.

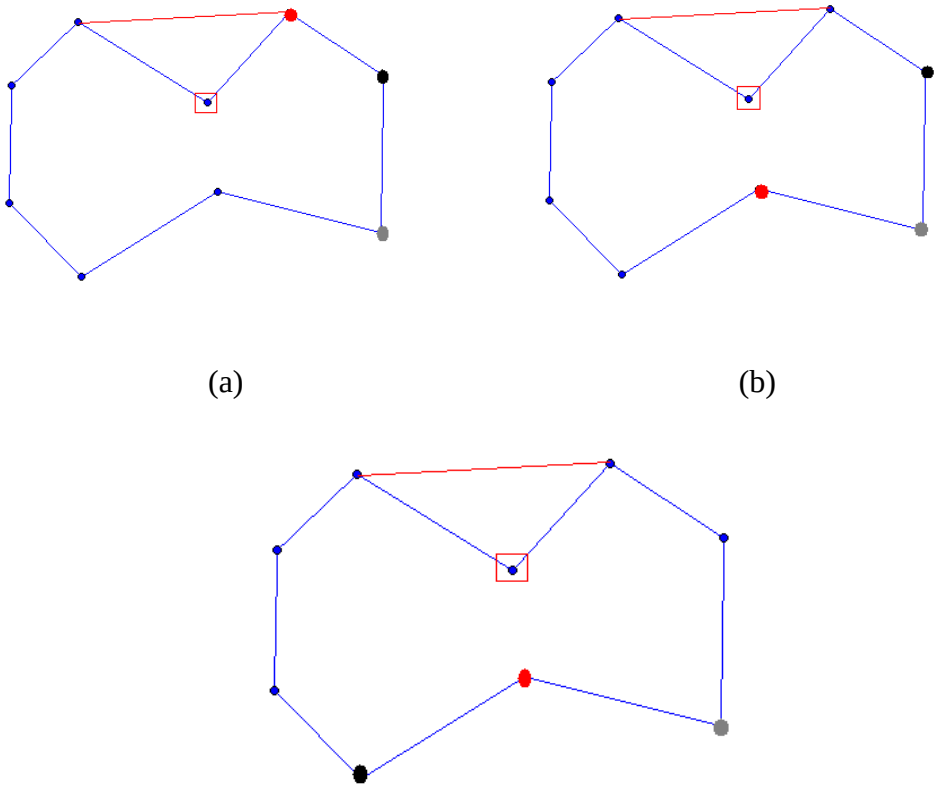


Figure 3.25 (a-c) move end point next to front point

Finally the coins form a left turn. We delete the point where the middle coin stands on right now (currently the red coin). And we move the middle coin to the point before the end coin (currently the grey coin).

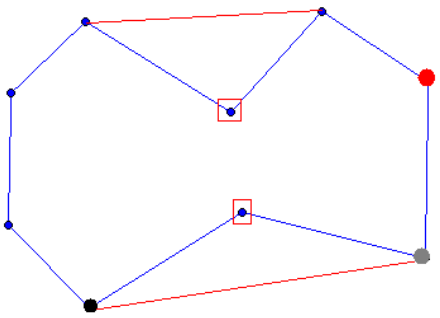


Figure 3.26 Remove middle point

It forms a right turn. As we move the end coin to the point next to the front coin, we find out that it comes back to the starting point. Therefore, the whole procedure will be terminated.

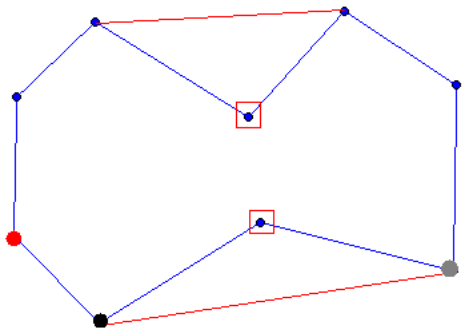


Figure 3.27 Process terminates when visit start point again and make a right turn

Link all the remained points. We got the convex hull of the set of points as shown in figure 3.28.

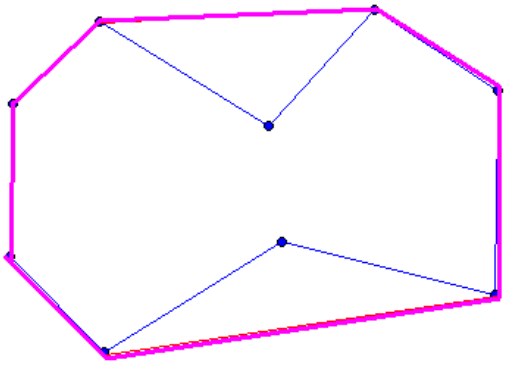


Figure 3.28 Convex hull of the input points

We can successfully generate the convex hull of the fore-arm contour by applying the three-coin algorithm as shown in figure 3.29. By observing the contours and its convex hull, we notice that the areas which the convex hull contains but is not included in the contour are called the convexity defect s. Convexity defect s give us useful

information about the shape of a contour. In the case of fore-arm contour, we find out that the convexity defects are often being around the palm. We will discuss about convexity defect in the next section.

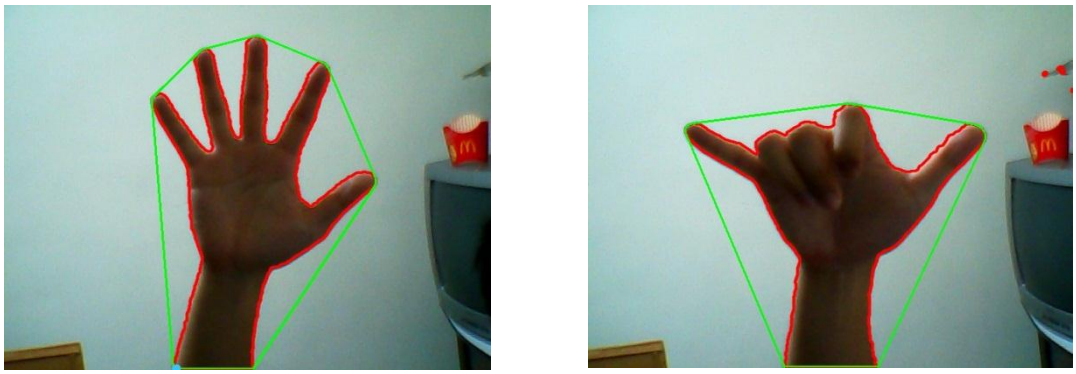


Figure 3.29 Convex hull of hand contour

Formal Description of Three-Coin Algorithm
Input: A set of N points which form a polygon in a plane. Output: A set of points which form the convex hull of the input.
Begin ● Find the left most point as a starting point. Label it p0. ● Label the remained N-1 points in a clockwise order. ● Place three coins on p0, p1, p2 and label them “end coin”, “middle coin”, “front coin” respectively. **Do Until:** the “front coin” is on the starting point and they form a right turn **If** the 3 coins form a right turn (or the 3 coins lie on collinear vertices) ● Take “end coin”, put it on the point next to “front point”. ● “end coin” become “front coin”, “front coin” become “middle coin”, “middle coin” become “end coin”. **Else** the coins form a left turn ● Remove the point (and associated edges) that "middle coin" is on ● Take "middle coin", put it on the point before "end point” ● “middle coin” become “end coin”, “end coin” become “middle coin” **End** The remained points form a convex hull when connecting them in order.

Table 3.2 Three-Coin algorithm

3.3 Convexity Defect

We calculate the convex hull of the fore-arm contour in order to get the convexity defect of the contour. Convexity defect provides us very useful information to understand the shape of a contour. Many characteristic of complicated contours can be represented by convexity defects.

Figure 3.30 illustrates the convexity defect of a star-like shape. The green lines represent the convex hull of the star-like shape. As you can see in the figure, the areas in yellow are contained in the convex hull. But they are not contained in the star. Those areas are so called convexity defects.

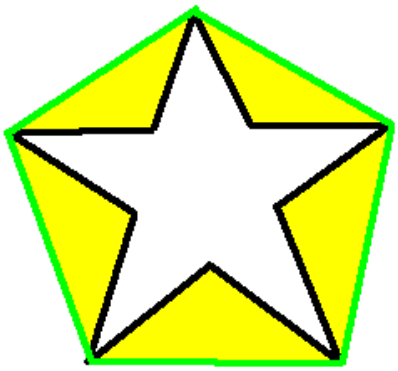


Figure 3.30 Yellow parts represent the convexity defects

The previous section tells us that the points which form the convex hull must be part of the contour. Our first step of searching a convexity defect will be finding the starting point of a convexity defect on the contour. The starting point of a convexity defect means a point on the contour which is also included in the convex hull points, but the next point on the contour is not included in the convex hull points. The figure 3.31 illustrates the starting point of a convexity defect. We search the contour clockwise. The red point is the first point which is included in the convex hull, but the next point is not included in the convex hull. We mark the point as a starting point.

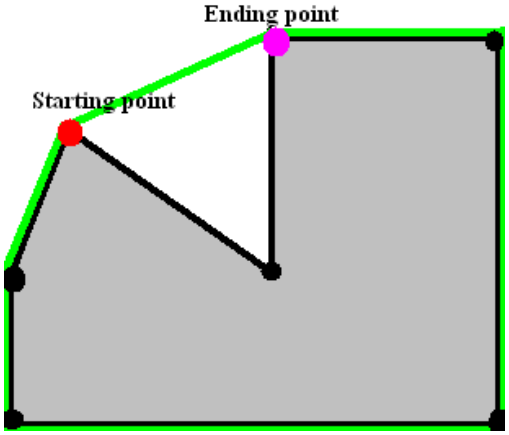


Figure 3.31 Starting and ending points of a convexity defect

Once we know about the starting point, the ending point will be similar. We define the ending point to be the point in the contour which is included in the convex hull points, but the point before it is not included in the convex hull points. As we can see in the figure 3.31, the purple point is the ending point of a convexity defect.

By connecting the starting point, ending point and the points on the contour between the starting point and the ending point, we get a convexity defect area as shown in figure 3.32.

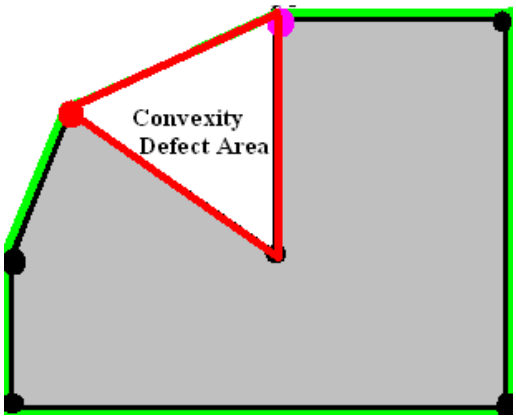


Figure 3.32 Convexity defect area

When all the points in the contour have been searched, we may find various convexity defects as shown in figure 3.33. Each convexity defect is composed by a starting point, ending point, and the points between them.

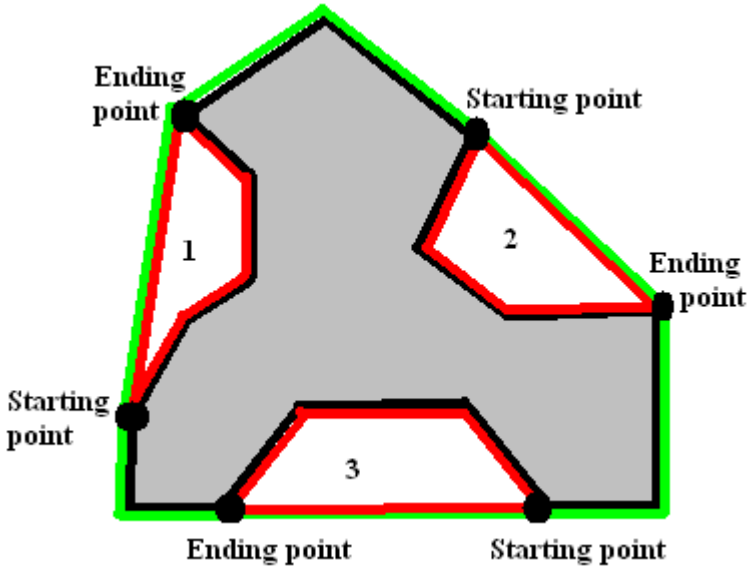


Figure 3.33 there are three convexity defects in the figure

Except the starting point and ending point, other useful information we can obtain from a convexity defect will be the depth of the defect and the depth point. The depth of the defect is the longest distance of all points in the defect to the convex hull edge of the defect. The point in the defect which has the longest distance to the convex hull edge of the defect will be the depth point.

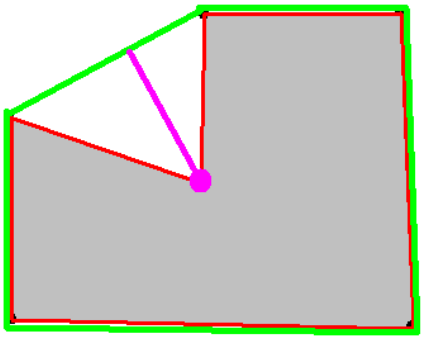


Figure 3.34 the depth point of a convexity defect

As shown in figure 3.34, there is only one convexity defect. The purple point is obviously the point with the longest distance to the defect's convex hull edge. And the length of the purple line is the depth of the purple point. Since the depth of the purple is the longest, it will also be the depth of the convexity defect. And the purple point is the depth point. Figure 3.35 shows all the depth points of a fore-arm contour.

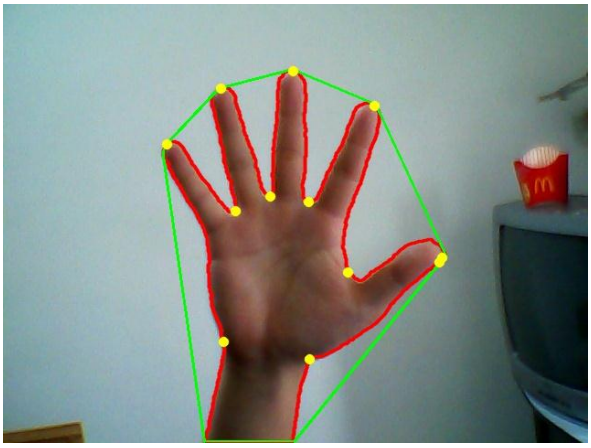


Figure 3.35 All depth points of a hand contour

The depth points are represented in yellow. Since there are 12 depth points in the figure, we know that there are 12 convexity defects in the fore-arm contour. Only six out of 12 convexity defects can be seen with human eyes while the rest of them are too small to be seen. Since the fingertips connect the convex hull edge, it would be easily to create a very small convexity defect if there is a concave contour. We should regardless such convexity defects and take the convexity defects with large area as what we want. The large convexity defects also have long depth. So we need to check the depth of all the convexity defects. If the depth is longer than a certain threshold, we draw the depth point of it. If it's not longer than a certain threshold, the defect will be ignored. Figure 3.36 shows the result when the small convexity defects are ignored and only the depth points of large convexity defects will be shown.

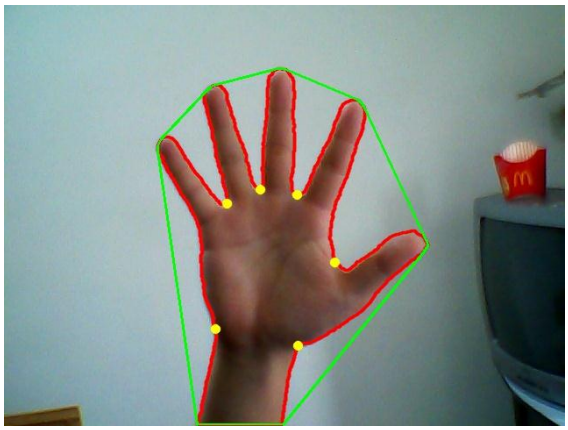


Figure 3.36 Remove depth points which have small depth

An open hand gesture is a perfect example to show that the depth points of large convexity defects are around the hand palm. But it also works very well in other gestures. Figure 3.37 shows the distribution of depth points in different hand gestures.

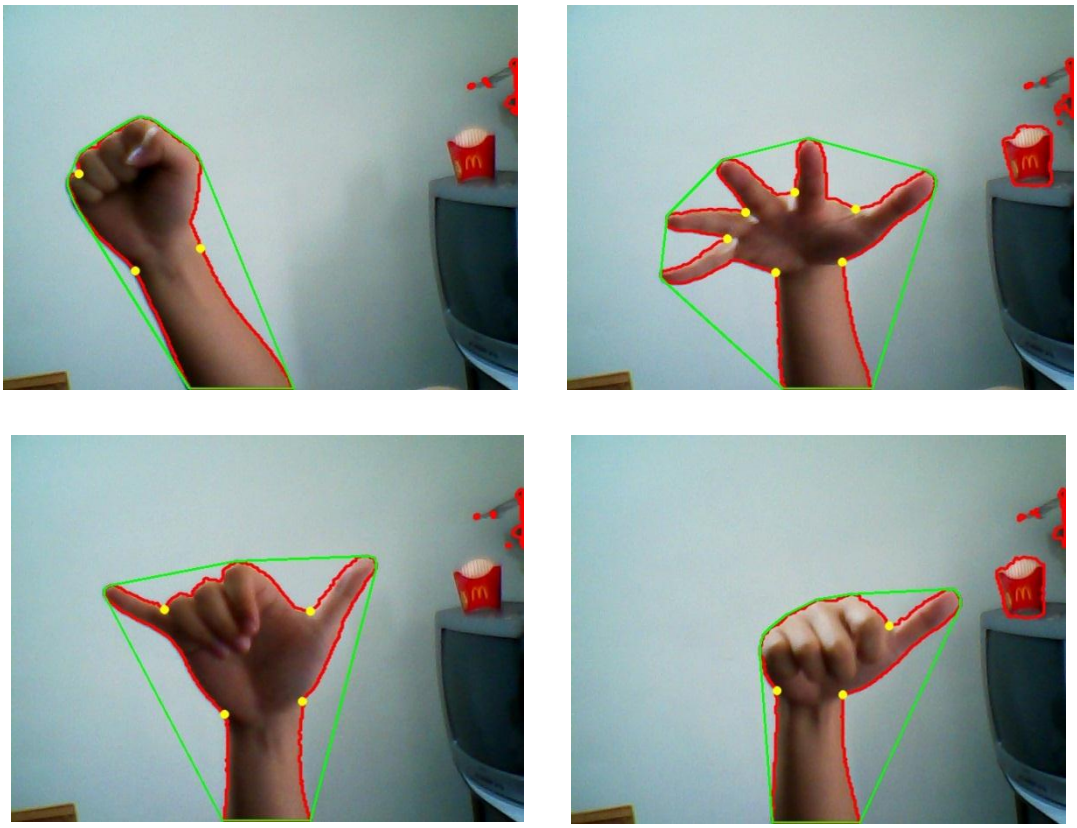


Figure 3.37 Examples of depth points of hand contour

The palm is wider than the wrist, so we usually have a depth point in the wrist part. That enables us to have an accurate locating of palm since the depth points in the wrist may separate the hand and the arm in the fore-arm contour. In most cases, we have two depth points around the wrist. Sometimes when the hand has a certain level of rotation, it may cause the depth of the convexity defect in one side of the wrist to be reduced. In the meanwhile, it also increases the depth of the convexity defect in the other side of the wrist. So there's always at least one depth point at the wrist. By this characteristic, we may separate the palm and the arm very well. Figure 3.38 shows some situation of rotated hand.

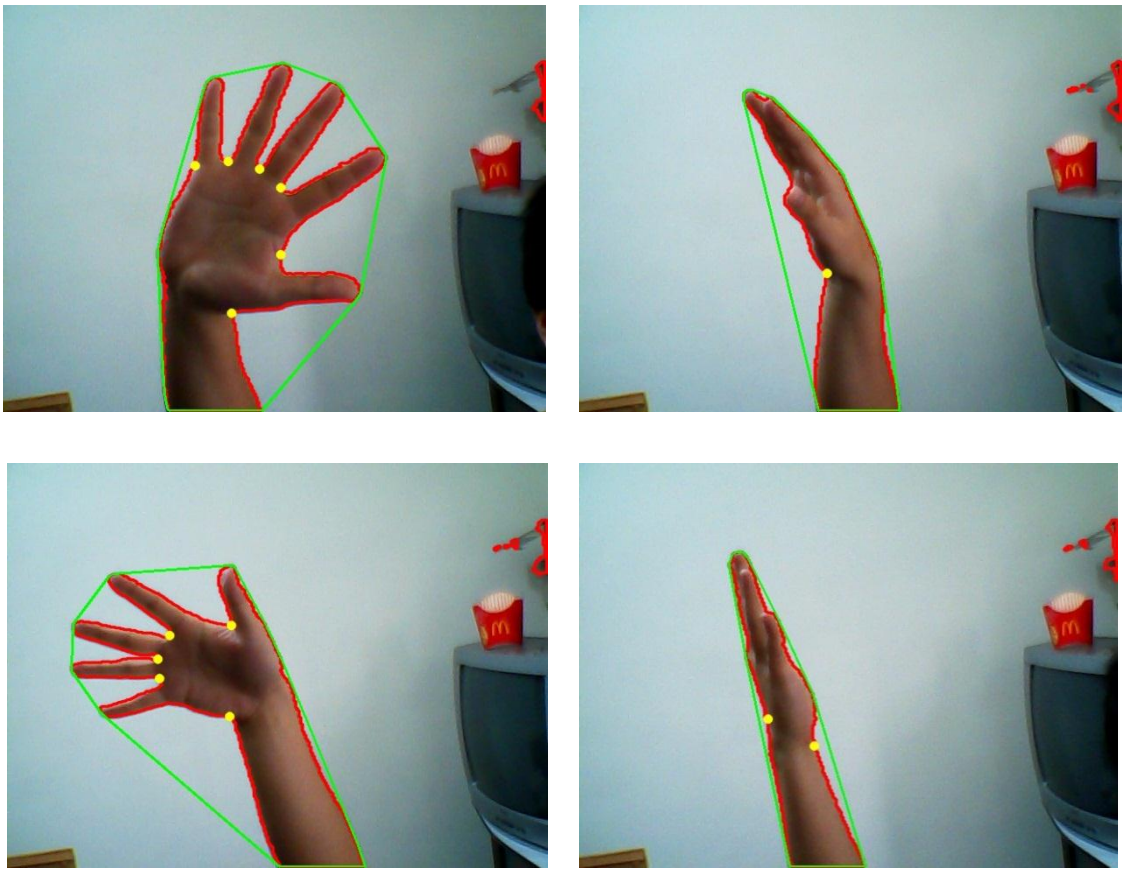


Figure 3.38 Rotated hands and its depth points

Chapter 4 Palm Tracking and Fingertip Detection

In the chapter, we will focus on fingertip and palm detection. We are able to extract the point which has the longest depth in a convexity defect. The convexity defects with a depth larger than a certain threshold tend to be appeared around the palm. That is because the shape of fore-arm part is smooth. It's not easy to generate a convexity defect with a long depth. Meanwhile, the space between fingers will tend to be in a long shape and the depth point will be on the edge of palm. Also the convexity defects beside the hand wrist are more likely to have their depth point on the wrist. The reason why is because the palm is usually having more width than the arm.

Gathering the depth point gives us useful information to decide where the palm position is. In this thesis, there are two ways to determine the palm center according to the depth points. One is to enclose all the depth points within a minimum enclosing circle. It calculates the smallest circle which covers all the depth points. The other way is to calculate the average position of all the depth points. Both methods have their own advantages and disadvantages. We decide to use both methods to mix up their advantages also to reduce their disadvantages.

Another issue is about finding the fingertips. Since the fore arm contour that we extracts is actually a series of points, we need a definition of fingertip in the contour. That will be the points in the end of a sharp shape. We applied an algorithm proposed by [13] to distinguish the fingertip points from a series of contour points.

When the all fingertips have been extracted, we are now able to know the palm position and the fingertip positions. By knowing how many fingers we have in the image right now, we then know when to have some mechanism to handle exceptions.

When our gesture is a fist, there will be no fingers. The depth points will sometimes be the two points on the left and right side of the wrist. This will cause the wrong palm position estimation. We will need to add extra point to fix that issue.

Also the palm position we estimates will be shaking due to the constantly changing of hand contour. Even if the contour only changes slightly, it will sometimes affect some depth point positions. Since there are not much depth points, changes from only a few of them will still cause the estimated palm position to be changed. To keep it from shaking, we apply two buffers for palm center and the radius of palm circle. The buffers make their movement more smoothly.

The depth point beside the wrist may slip away when the fore-arm part is large in the image. One slipped depth point may enlarge the size of the palm circle. We calculate the average position of current depth point, and mix it with the palm circle position to reduce the effect of slipped depth point.

4.1 Minimum Enclosing Circle

The minimum enclosing circle problem is to find the smallest circle that completely contains a set of points. Formally, given a set of points S in a plane, find the circle C with smallest radius which that all points in S are contained in C or its boundary.

The minimum enclosing circle is often used in planning the location of a shared facility. For example, a hospital or a post office will be a good shared facility. If we consider each home in the community as points in a plane, the center of the minimum enclosing circle will be a good place for the shared facility. We apply this idea in finding the palm position because the ideas are similar. Since all the depth points are around the hand palm and they share the same palm center, the minimum enclosing circle of the depth points will be a good palm position estimation.

The fastest algorithm of finding the minimum enclosing circle is proposed by Nimrod Megiddo in 1983[14]. The minimum enclosing circle can be found in O(n) time using the prune-and-search techniques. This is one of the most impressive works in the field of computational geometry. Assumed the total amount of points to be N, the Megiddo's algorithm discards at least N/16 points at each iteration without affecting the result of minimum enclosing circle. This procedure can be repeatedly applied until a basic case is reached. The total time will be 16N (N + 15/16N + 225/256N + …) which is linear time.

But usually the total amount of depth point won't exceed six points, so another algorithm which is relatively simple would be desirable. It's proposed by Skyum in 1990[15]. According to [15], the algorithm which takes O(nlogn) time, would be easy to implement than Megiddo's algorithm. It's a good alternative method if the total amount of points is not large.

Assume that there is a set of points S in a plane, and the convex hull of S is C. It is obviously that the minimum enclosing circle of S equals to the minimum enclosing circle of C. Since the points inside the convex hull won't be able to affect the result of convex hull, those points can be eliminated. We can apply the three-coin algorithm we introduce in the last chapter to find the convex hull. In Skyum's algorithm, the input must be a convex polygon.

Radius (p,q,r) denotes the radius of the circle through the three points p, q and r if they are different. If two of them are identical, then the radius will be half the distance between the two points. Angle(p,q,r) denotes the angle between the line from p to q and the line from q to p. It is mandatory true that q is not identical to p as well as identical to r. But p and r can be identical. The above two routines will be reused a lot in the algorithm.

4.1.1 Find Circle through Three Points

It is obviously that three different points in a plane can form a unique circle. The equation of the circle in a plane has three unknown variables.

$$(\mathrm{x} - \mathrm{a})^2 + (y - b)^2 = r^2 \qquad (4\text{-}1)$$

When there are three points in a plane, we might be able to input their x and y coordinate into the equation to generate three equations with three variables. That makes unique solution. It also proves that three points can form a unique circle. Instead of solving the simultaneous equations, we can first calculate the center of the circle according to geometric property. When the center of the circle is calculated, it will be easy to get the radius.

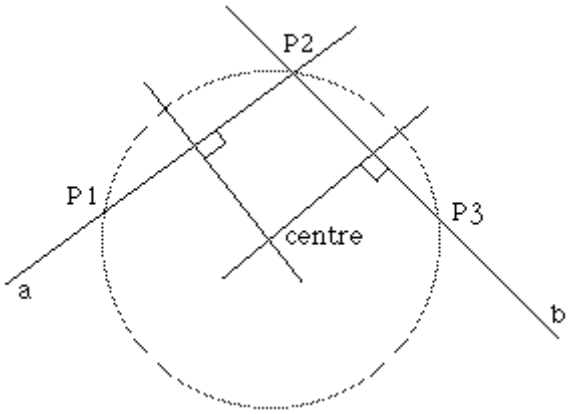


Figure 4.1 Center of the circle formed by three points

In figure 4.1, the circle is formed by three points P1, P2 and P3. Two lines can also be formed through two pairs of three points. Line A passes through P1 and P2. Line B passes through P2 and P3. The equations of these two lines are:

$$y_a = m_a(x - x_1) + y_1 \qquad (4\text{-}2)$$

$$y_b = m_b(x - x_2) + y_2 \qquad (4\text{-}3)$$

Where m is the slope of the line given by:

$$m_a = \frac{(y_2 - y_1)}{(x_2 - x_1)} \tag{4-4}$$

$$m_b = \frac{(y_3 - y_2)}{(x_3 - x_2)} \tag{4-5}$$

The center of the circle is the intersection of the two lines which are perpendicular to line A and line B and also pass the midpoints of P1P2 and P2P3. The perpendicular of a line with slope m has slope -1/m, thus the equations of the lines perpendicular to line A and line B which pass the midpoints of P1P2 and P2P3 are:

$$y'_a = -\frac{1}{m_a}\left(x - \frac{x_1 + x_2}{2}\right) + \left(\frac{y_1 + y_2}{2}\right) \tag{4-6}$$

$$y'_b = -\frac{1}{m_b}\left(x - \frac{x_2 + x_3}{2}\right) + \left(\frac{y_2 + y_3}{2}\right) \tag{4-7}$$

The two lines of the equations above intersect at the center point, so the coordinate of the center point will be the common solution for the two equations above. The equation below solves the x-coordinate of the center point:

$$x = \frac{m_a m_b (y_1 - y_3) + m_b (x_1 + x_2) - m_a (x_2 + x_3)}{2(m_b - m_a)} \tag{4-8}$$

$(m_b - m_a)$ will be zero only when the two lines are parallel, that implies that they must be coincident. But we apply the calculation only when the three input points are different, so we will avoid that situation. By substituting the x value into Eq. (4-2), we obtain the y-coordinate of the center point. Since the center point has been calculated, the calculation of the radius would be simple. Just compute the distance between the center point and P1, P2 or P3 and we will get the radius.

4.1.2 Angle Calculation by Dot Product

Dot product is an algebraic operation which takes two equal-length sequences of number and returns a single number by multiplying the corresponding entries and sum up those products. The basic use of dot product is to find the cosine value between two vectors. The dot product of two vectors a= [a1, a2, a3, …, an] and b=[b1, b2, b3, …, bn] is defined as below:

$$\mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n \tag{4-10}$$

In our system, the dimension is two. So the dot product of two vectors [a, b] and [c, d] will be ac + bd. In Euclidean geometry, the dot product $\mathbf{a} \cdot \mathbf{a}$ is the square of the length of $\mathbf{a}$, or

$$\mathbf{a} \cdot \mathbf{a} = \|\mathbf{a}\|^2 \tag{4-11}$$

The dot product of two vectors in Euclidean space can also be defined as:

$$\mathbf{a} \cdot \mathbf{b} = \|\mathbf{a}\| \, \|\mathbf{b}\| \cos\theta \tag{4-12}$$

So the angle between two vectors can be:

$$\theta = \arccos\left(\frac{\mathbf{a} \cdot \mathbf{b}}{\|\mathbf{a}\| \, \|\mathbf{b}\|}\right) \tag{4-13}$$

Since the cosine value changes with the angle, we don’t need to calculate the exact angle. Cosine value changes from 0 to 1 when angle changes from 90 degree to 0 degree. Cosine value changes from 0 to -1 when the angle changes from 90 degree to 180 degree. So we can compare two angles directly by their cosine value.

4.1.3 Skyum's algorithm

When the two subroutine radius(p,q,r) and angle(p,q,r) has been defined, the Skyum's algorithm can be used to find the minimum enclosing circle. The first step of the algorithm is to check the amount of points in the input is more than 1 point. If not, the algorithm terminates. Then, sort all the point in lexicographic order within the following rule: (radius(before(p), p, next(p)), angle(before(p), p, next(p))). Find M as the maximum point in the lexicographic order. If angle(before(M), M, next(M)) > $\pi/2$, M will be removed and the remained points will be sorted again in the same lexicographical order. The maximum point will be picked up again to check its angle. The process will be repeated until the angle(before(M), M, next(M)) < $\pi/2$ or there are only two points left.

We take those four points as shown in figure 4.2 as an example. Those points are labeled in a counter-clockwise order. The first step is to find the circles formed by 123, 234, 341, 412, as shown in figure 4.3.

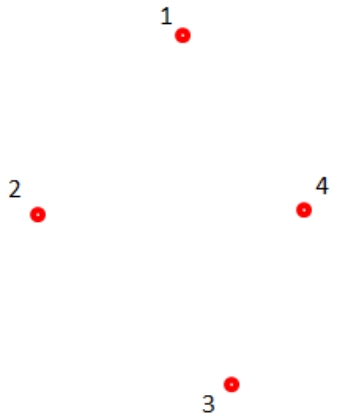


Figure 4.2 Example of input points for Skyum's algorithm

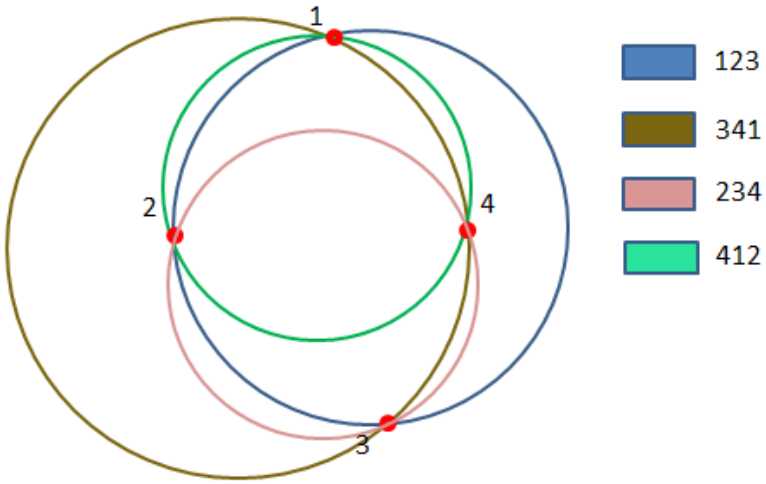


Figure 4.3 All the circles formed by the input points

Four circles can be formed by four points as figure 4.3 shown. We can now sort those points in lexicographical order. It is obviously that circle 341 is the largest; circle 123 is the second largest. Circle 234 and circle 412 are the smallest, and they also have the same radius. Since circle 234 and circle 412 has the same size, we need to compare the angle of them. ∠412 is 73 degree while ∠234 is 71 degree. The lexicographical order of those four points will be 4, 2, 1, and 3. We pick up the maximum, which is 4, and then we check ∠341. ∠341 is obviously larger than π/2, so we discard 4 and draw the circles formed by the remained points as figure 4.4 shown.

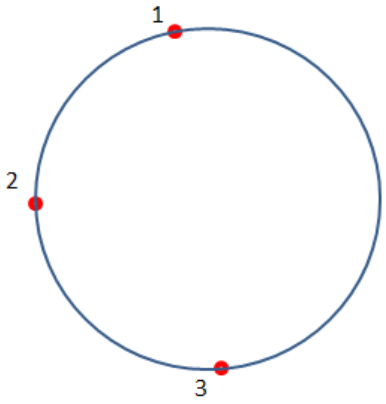


Figure 4.4 Remove 4 and the rest of the points form a circle

There are only three points left, so they can only form 1 circle. Since the circle they form has the same radius, we sort those points according to its angle, which is the second parameter in their lexicographical order.

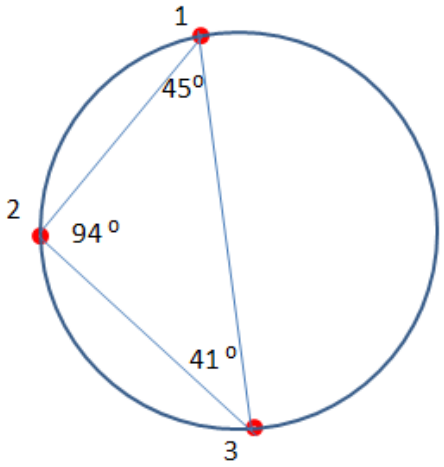


Figure 4.5 all the angles inside the triangle

Figure 4.5 shows the degree of those angles. According to the definition of its lexicographical order, we pick up the point which has the largest angle. That will be 2 which is 94 degree. ∠123 is larger than π/2, so we will discard the point. There are only two points left after the removal of 2, so the minimum enclosing circle will be the circle which is formed by 1 and 3. Figure 4.6 shows the final result.

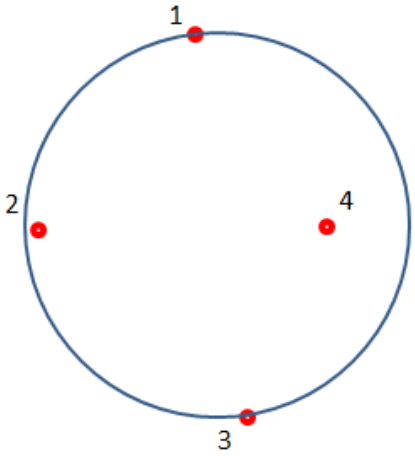


Figure 4.6 the minimum enclosing circle of the input points

We can notice than point 2 is very close to the boundary. Because ∠123 is very close to π/2. If ∠123 is equal to π/2, then the point 2 will be exactly located on the boundary of the circle which point 1 and point 3 create. If ∠123 is smaller than π/2 and ∠341 is still the largest among all the angles, the minimum enclosing circle can be formed by 1, 2 and 3.

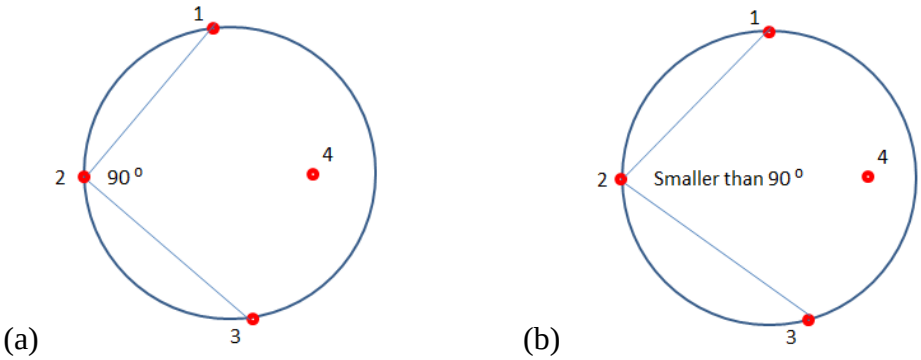


Figure 4.7 (a) if ∠123 is 90 degree (b) if ∠123 is smaller than 90 degree

The following figure 4.8 shows the result of our estimated palm location calculated by depth points and its minimum enclosing circle.

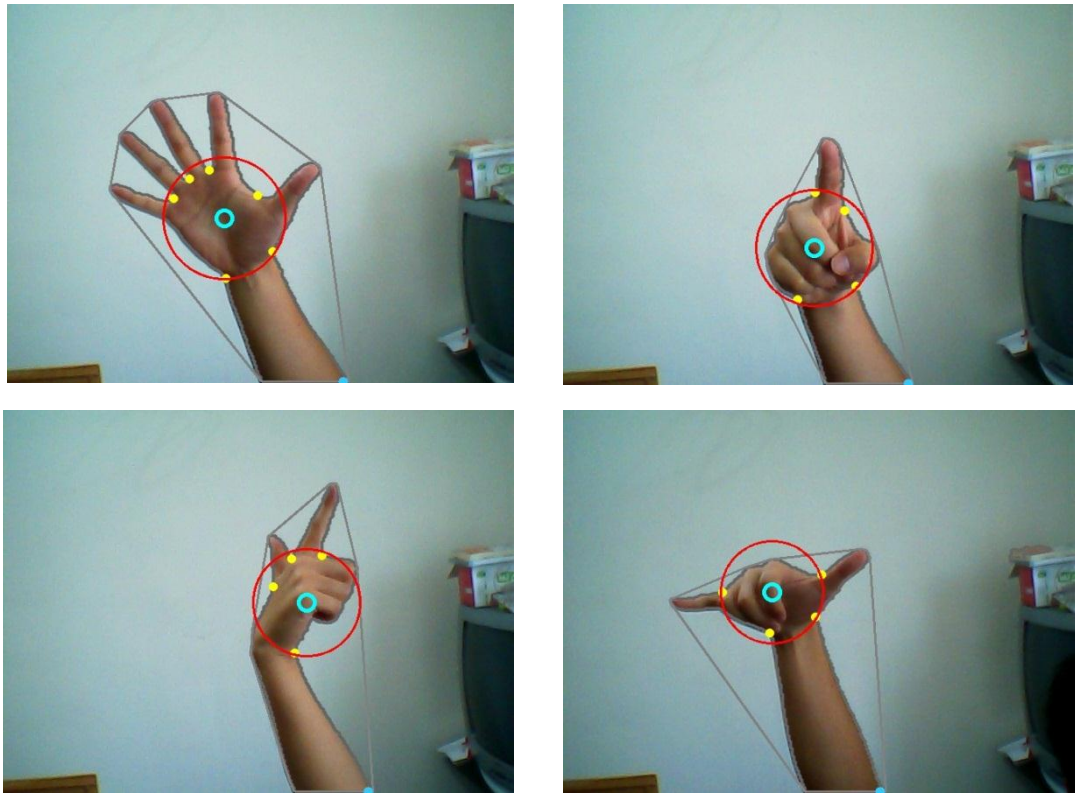


Figure 4.8 Examples of minimum enclosing circle formed by the depth points

4.2 Fingertip Detection

In this thesis, we apply the method which is proposed by Wen and Niu in 2010[13]. Their algorithm marks up the fingertips on the contour by computing the angle of each point. In mathematic view, the problem is to find out the most aculeate point in a contour, which might be the fingertips. In their proposal, the algorithm is experimented within static images instead of real-time video. Also the hand contour includes only palm and fingers. The fore-arm part is removed by wearing clothe to cover it up. We apply their method in our real-time system and the contour of fore-arm is included.

To identify the most aculeate points in a contour, the angle of a point needs to be defined. This can be done by the following equation Eq. (4-14):

$$\text{Fingertip-Angle} \langle X \rangle = (X\text{-}X1) \bullet (X\text{-}X2) / |X\text{-}X1 \| X\text{-}X2| \qquad (4\text{-}14)$$

X1 and X2 are separated by X within the same distance. X1 is the previous point with a certain number of points before X. X2 is the following point with a certain number

of points after X. The distance is 20 points in our experiment. The symbol (•) means the dot product. The value of Fingertip-Angle <X> is between 0 and 1, because it's the cosine value of X_1XX_2. Figure 4.9 illustrates the angle of a point.

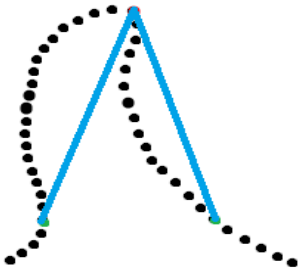


Figure 4.9 angle of a contour point

All the angle of each point in the contour will be calculated. We can notice that the more aculeate the angle is, the larger the cosine value is. So we can set a threshold for the cosine value, 0.6 is set in our experiment. As we can see in the figure 4.10, Those points which has their angle value larger than the threshold are more likely to be in the fingertip position as well as the interval of two fingers. These points are the candidates of fingertips.

Figure 4.10 White points are which their cosine value is larger than threshold

Our next step is to choose a point in a group of continuous points. This can be done by choosing the point which is the local maximum of the group. As in figure 4.11, the point which is the maximum would be chosen.

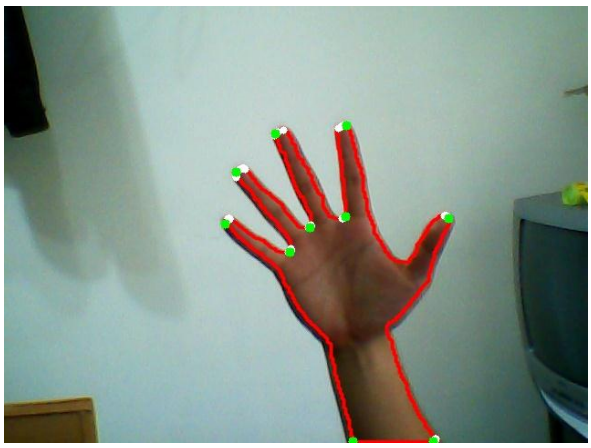
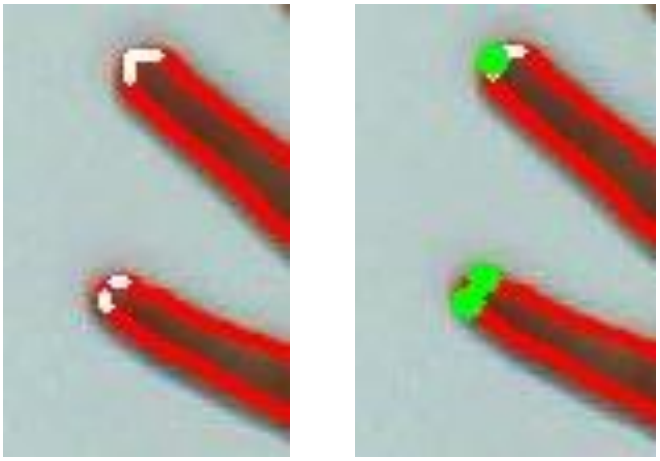


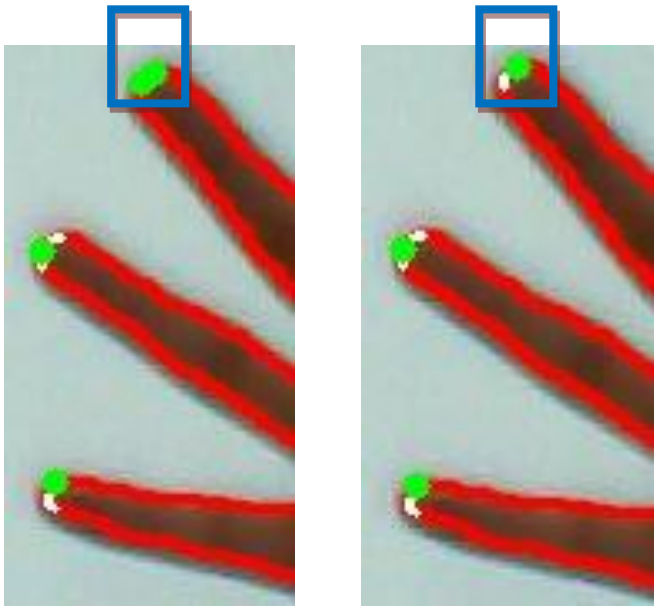
Figure 4.11 green points are the local maximum

There are also some exceptional conditions which we might need to take care of. Sometimes due to unsmooth contour, a group of fingertip candidates is not continuous. One or more points which are not candidate points are in-between the groups, those points separate the group to be 2 or more minor groups. Hence, 2 fingertip points might be marked up in one fingertip. When two fingertip points are too close to each other, we might need to discard one of them. We set a threshold of 50 points. When the interval points between two fingertip points are less than 50 points, the latter fingertip point will be discarded. Wen and Niu's proposal didn't cover this part. This is our modification of their algorithm. Figure 4.12 shows that a group of points sometimes might be a combination of some smaller groups. We need to discard the redundant fingertip points by our modification of the algorithm as shown in Figure 4.13.



(a) (b)

Figure 4.12 (a) two groups are on the same fingertip (b) two fingertip points are on the same fingertip



(a) (b)

Figure 4.13 (a) two fingertip points on the same fingertip (b) remove one of them

The last step is to eliminate the points in the interval of two fingertips, which would be recognized as fingertip points by the previous method. Those points also have aculeate angle but in the opposite direction than the fingertips. To define the direction of aculeate angles, we need an extra point to measure. A minimum bounding box will be applied to

the contour. The box can be found by choosing the contour's maximum value of x-axis, maximum value of y-axis, minimum value of x-axis and minimum value of y axis. So the four corner of the minimum bounding box are (minimum x-axis, minimum y-axis), (minimum x-axis, maximum y axis), (maximum x-axis, minimum y-axis) and (maximum x-axis, maximum y-axis). The midpoint of the minimum bounding box is our desirable point of measurement. Figure 4.14 shows a minimum bounding box of a contour and the midpoint of the box.

Figure 4.14 minimum bounding box and its center

To distinguish the fingertip points and the points in the interval of two fingers, we need to calculate the angle PXX1 or PXX2. P is the midpoint of minimum bounding box. If the angle is larger than 90 degree, we discard the point because it's the point in the interval of two fingers. If the angle is smaller than 90 degree, we choose the point to be a fingertip point.

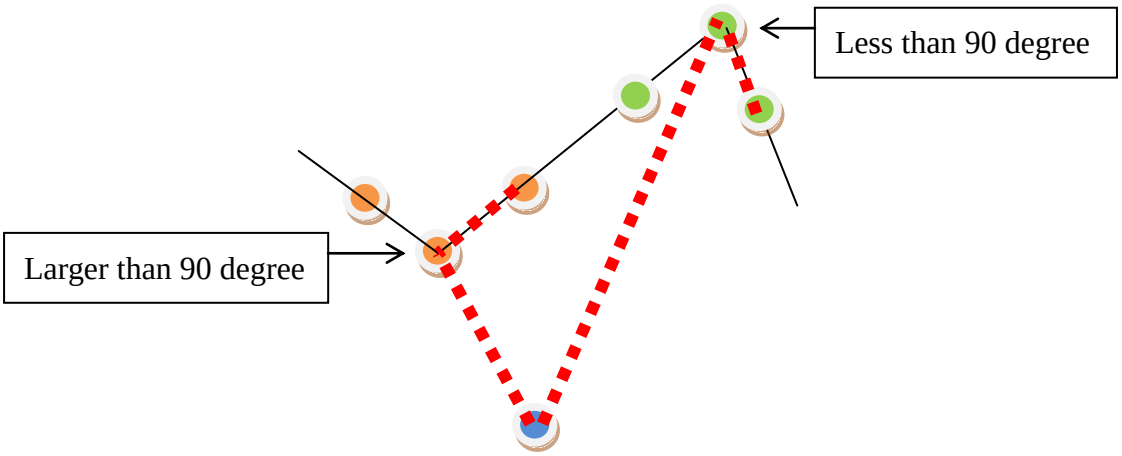


Figure 4.15 Identify different directions according to the center

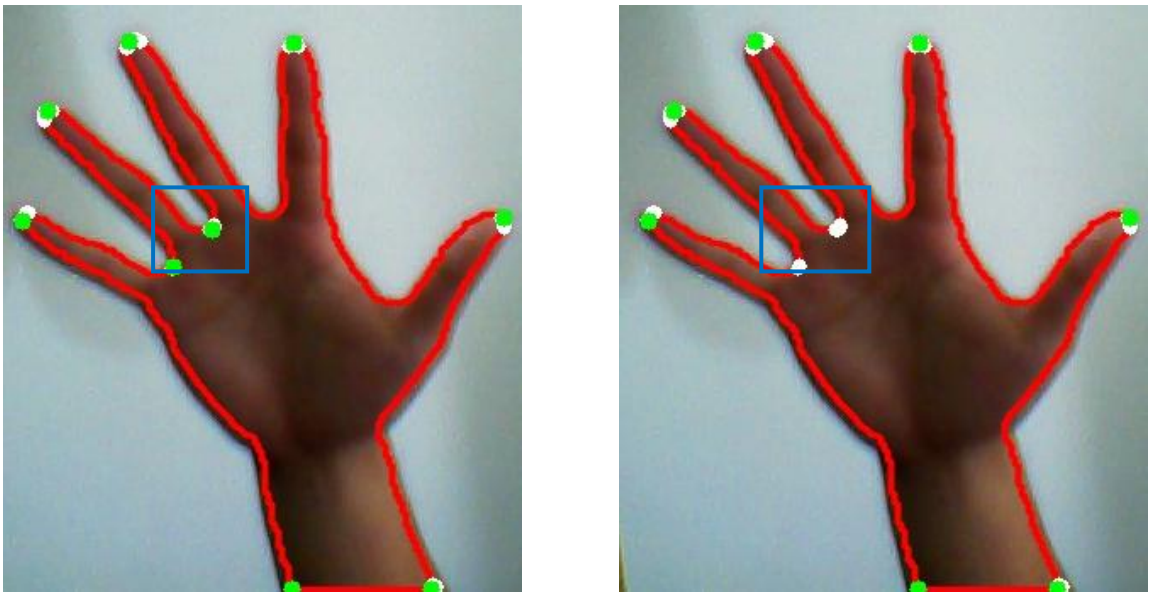


Figure 4.16 Remove points with opposite direction

When the fore-arm contour is included, some points at the boundary of the image might be recognized as fingertips point. Those points would also need to be removed according to its position. If it is at the boundary if the image, we remove the point. Figure 4.17 illustrates this situation.

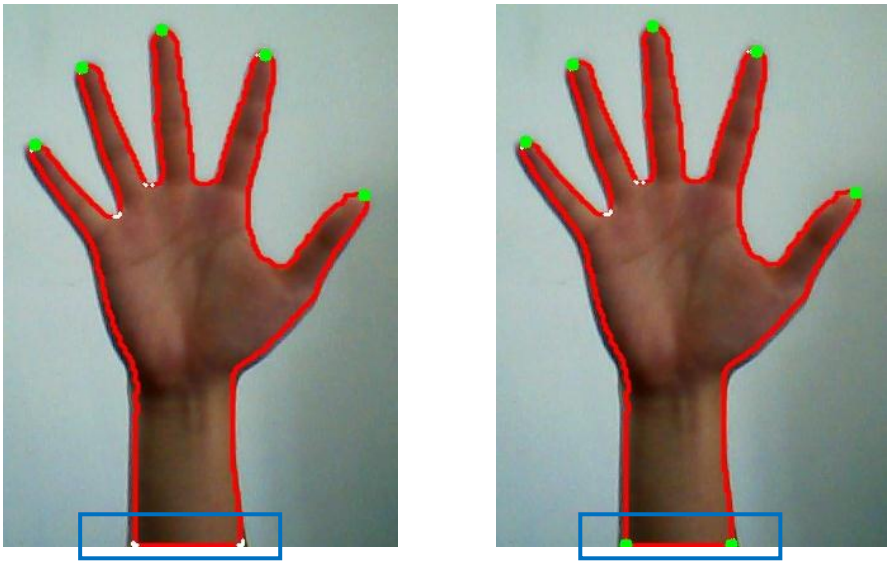


Figure 4.17 remove the points at the image boundary

By applying the above methods, the fingertips of a hand contour can successfully be recognized. Figure 4.18 shows some result of the algorithm.

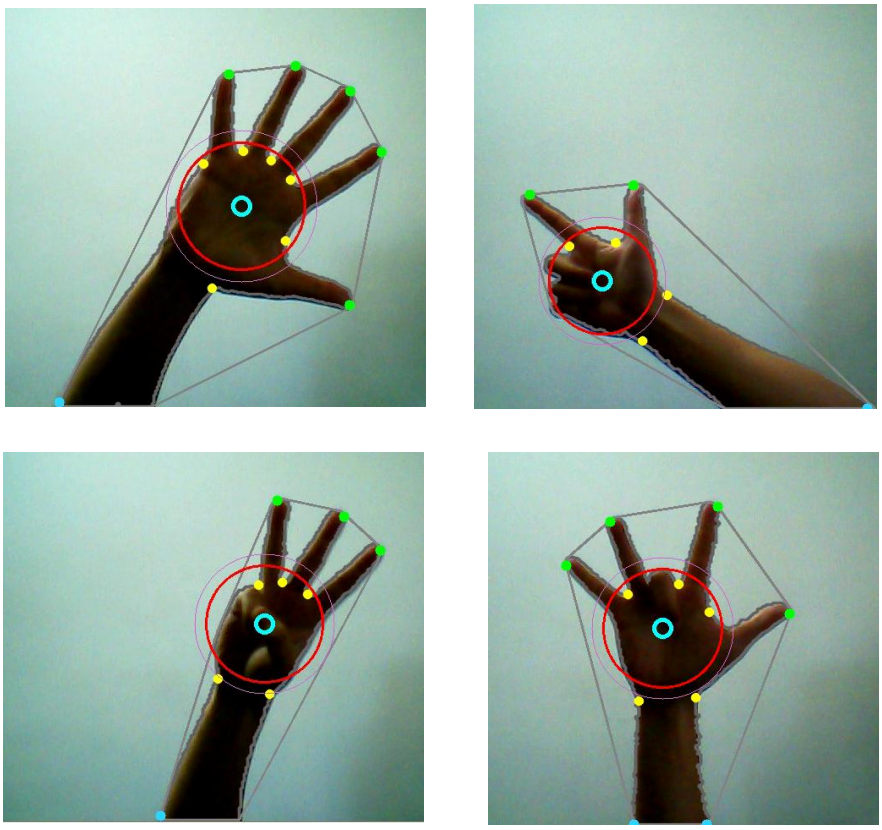


Figure 4.18 Some results of fingertip detection

Formal Description of Fingertip detection algorithm
Input: closed contour Output: A sequence of detected fingertip points
Find the center C of minimum bounding box **Set Group = false** **For** (all the points X in the contour) ● Get X1 and X2, X1 and X2 are separated by X with M points ● Compute P, the cosine value of ∠X1, X, X2 ➢ If (cosine value < threshold && !group) Discard X; Set Group = False; ➢ Else if (P > threshold && !group) Group = true; Set max = P; ➢ Else if (P > threshold && group && P > max) max = P; ➢ Else if (P < threshold && group) Push max into T; group = false; **End** **For(**all points W in T**)** ● If (∠CXX1 > 90 && ∠CXX2 > 90) Discard W ● Else if (∠CXX1 < 90 && ∠CXX2 < 90 && W is not at the image boarder) ➢ If (W is M points away from previous point in T) Mark W as fingertip points **End**

Table 4.1 Fingertip detection algorithm

4.3 Buffer for palm position

Since the contour we extract can have some small changes from frame to frame, it may cause the palm position to move a little bit in every frame. So the palm position might be shaking. To reduce this effect, we apply a buffer for the palm position.

The buffer will record the position and radius of palm circles in the past 3 frames and the current frame. The palm circle we are going to draw in the current frame will be the average of the buffer. In our experiment, the palm position will be a lot more stable when the buffer is applied.

4.4 Mix of palm position and average depth point position

When the fore-arm part is large in the image, the depth point at the wrist part might slip away. This will enlarge the size of the palm circle and the estimation of palm will be wrong. Figure 4.19 illustrates this kind of situation.

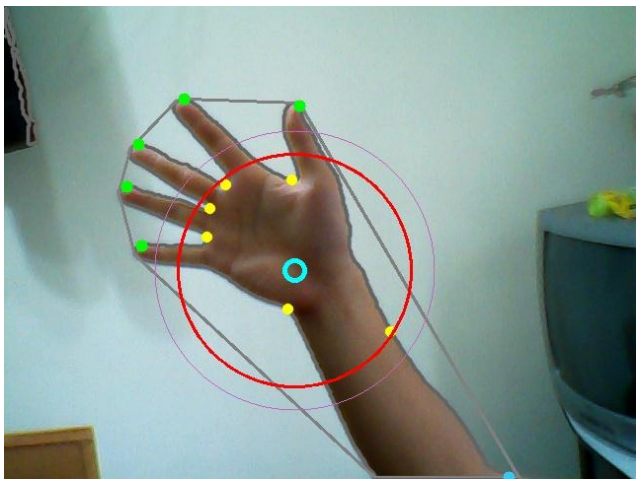


Figure 4.19 Depth point at the wrist slips away

We calculate the average position of current depth points and so that the slipped depth point can only affect 1/(amount of depth points) of the results. Figure illustrates the result of the average of depth point positions.

The reason we are not using the average depth points position as the palm position estimation is because that when the hand is open, four depth points might be at the top of the palm while two might be at the bottom. The average position will be closer to the top and the estimation is not desirable. The minimum enclosing circle behaves more like the relation between depth points and the hand palm center. Figure 4.20 illustrates the situation.

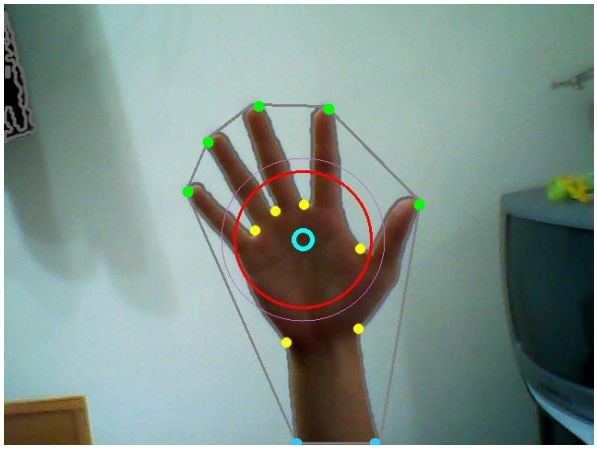
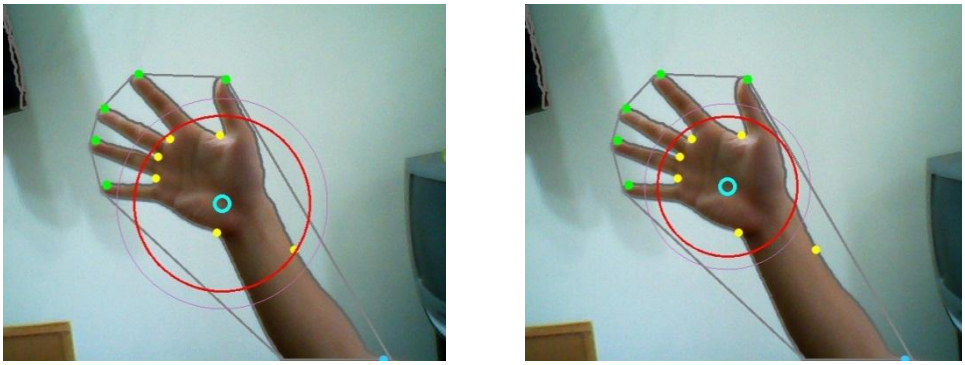


Figure 4.20 the average position and the radius of depth points

So we calculate the average of average depth points positions and the palm circle. The effect of slipped depth point will be reduced and the estimation of palm position will still be accurate. Figure 4.21 shows the result.



(a) (b)

Figure 4.21 (a) minimum enclosing circle of depth points (b) mix of minimum enclosing circle and average depth point position

4.5 Add an Extra Point when there is less Than Two Depth Points

Another condition we need to take care of often happened when the hand is closed; all the depth of convexity defects could be smaller than the threshold except the two convexity defects beside the wrist. This kind of situation will cause the estimation of palm center to be incorrect. Figure 4.22 show this kind of situation.

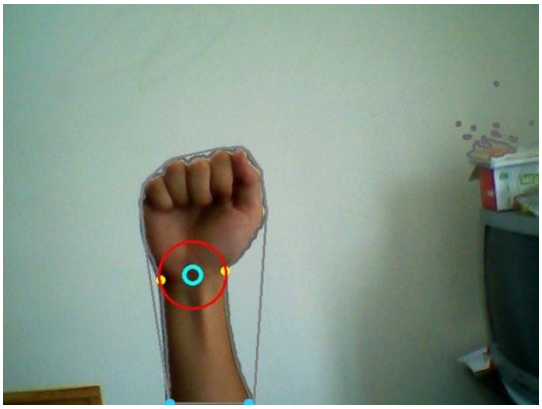


Figure 4.22 Wrong estimation when only 2 depth points

To avoid this problem, when the total amount of depth points is 2 or less, the system will add an extra point for minimum enclosing circle. The extra point can be obtained by finding the point which is at the top of the contour. Figure 4.23 shows the difference when the extra point is added.

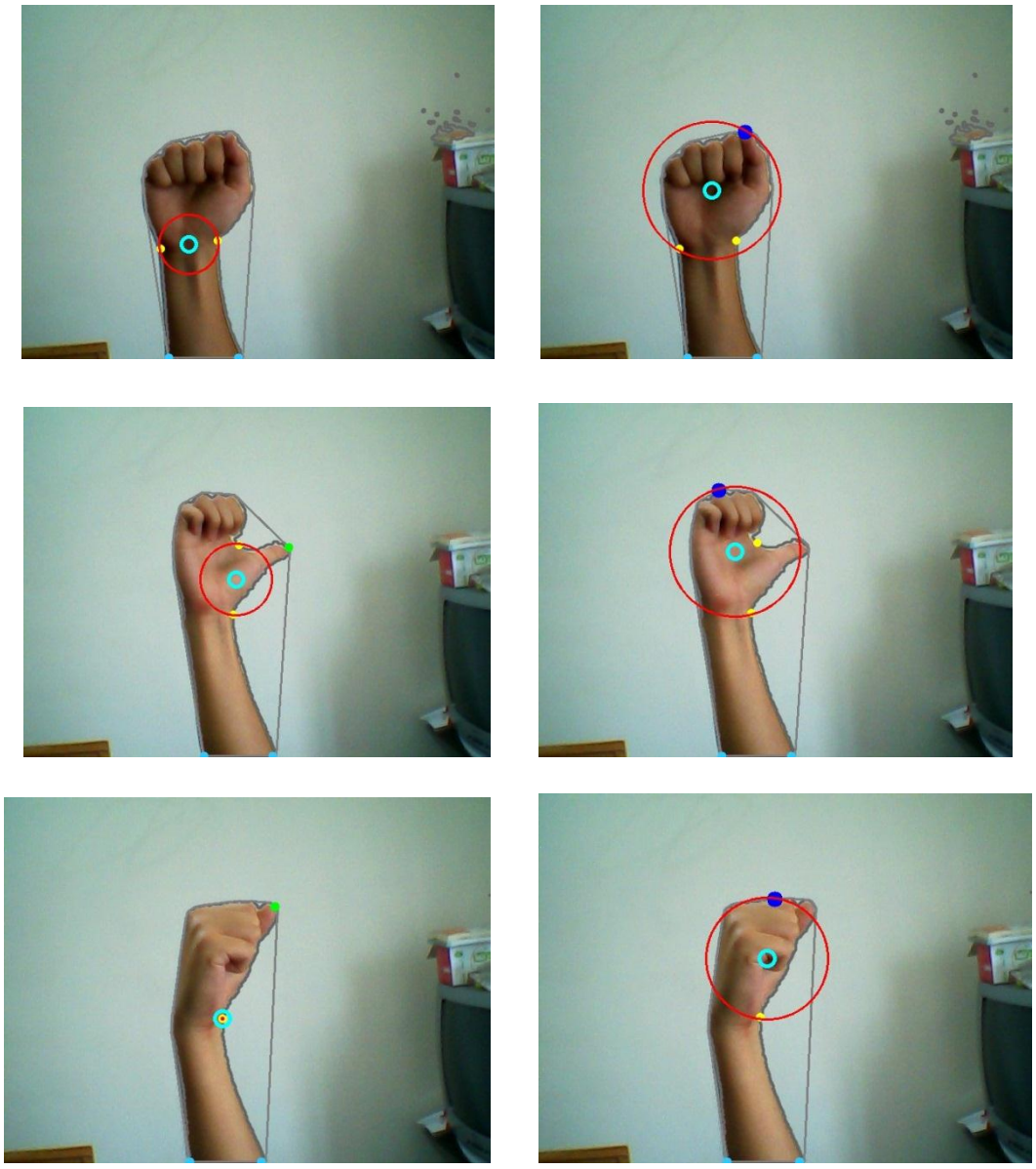


Figure 4.23 Results of adding an extra point

4.6 Distance Threshold for Fingertips

When the fingers are bending or when the gesture is a close hand, the contour can be unsmooth so that some unexpected fingertips will be marked up. Figure 4.24 illustrates this kind of situation.

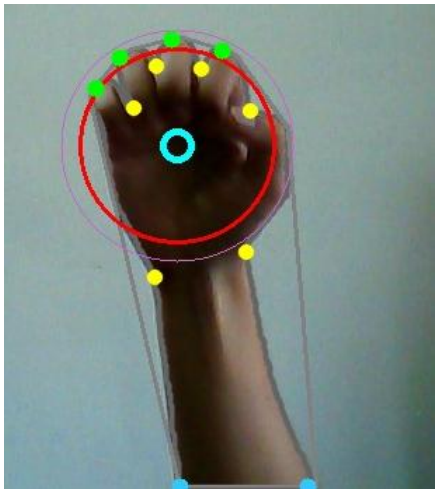
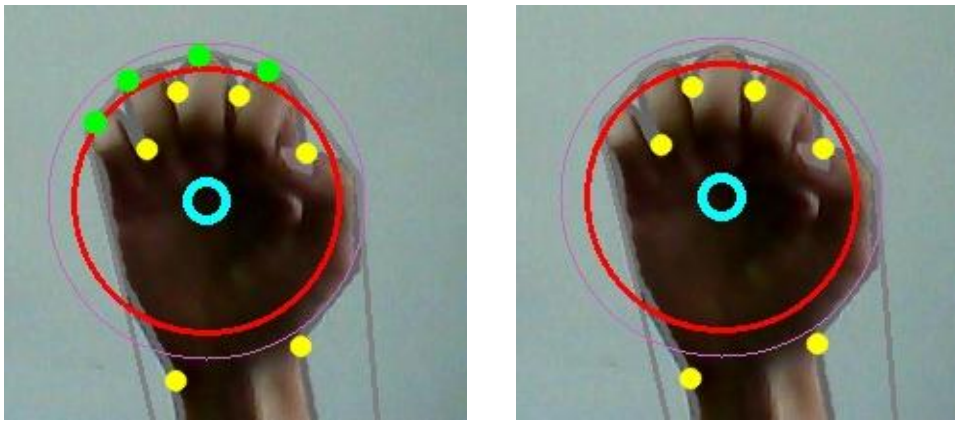
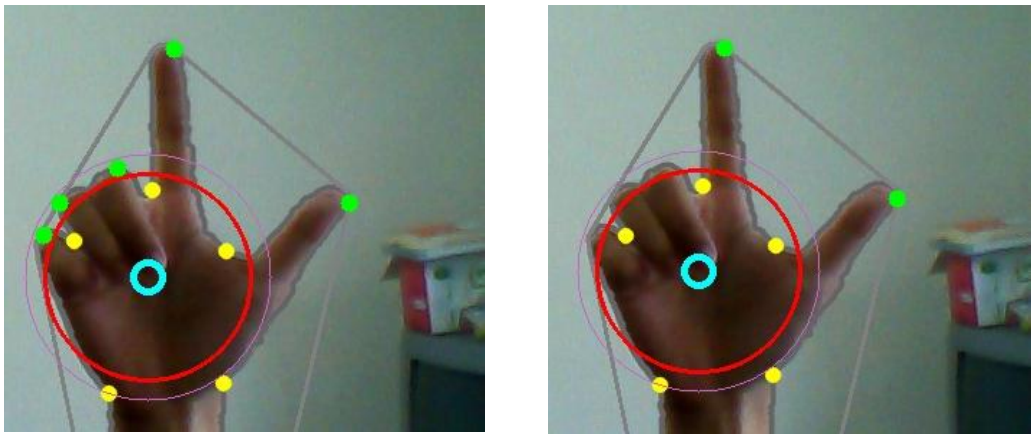


Figure 4.24 Unwanted fingertip points

We set a threshold for fingertips. The threshold can be obtained by 1.2*(radius of palm circle). As figure 4.25 shows, the inner circle is the estimated palm circle and the outer circle will be the threshold distance for fingertips. The distance between fingertips and the palm center must be longer than the threshold; otherwise the fingertips won't show up. The outer circle also covers up all the unexpected fingertips when the gesture is a close hand.



(a)



(b)

Figure 4.25 If fingertip point is inside the circle of 1.2*radius, the point will be removed

Chapter 5 Experiment Results

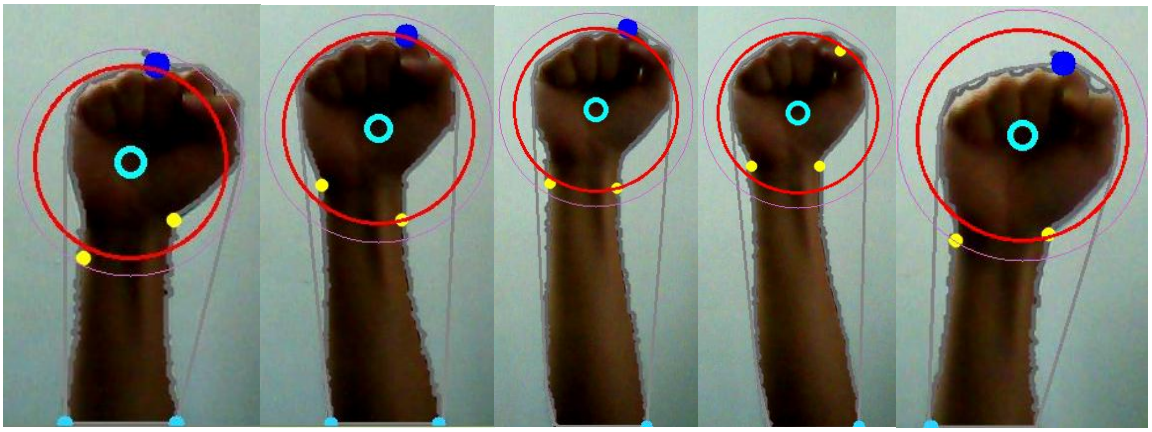
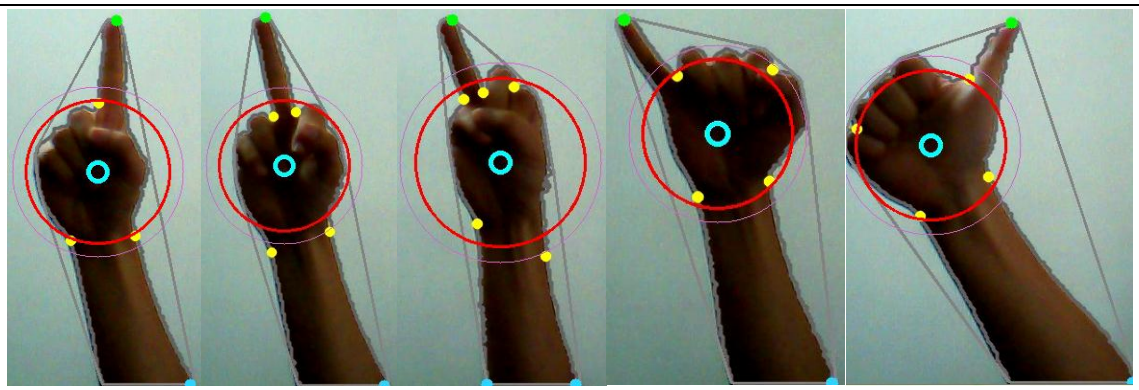
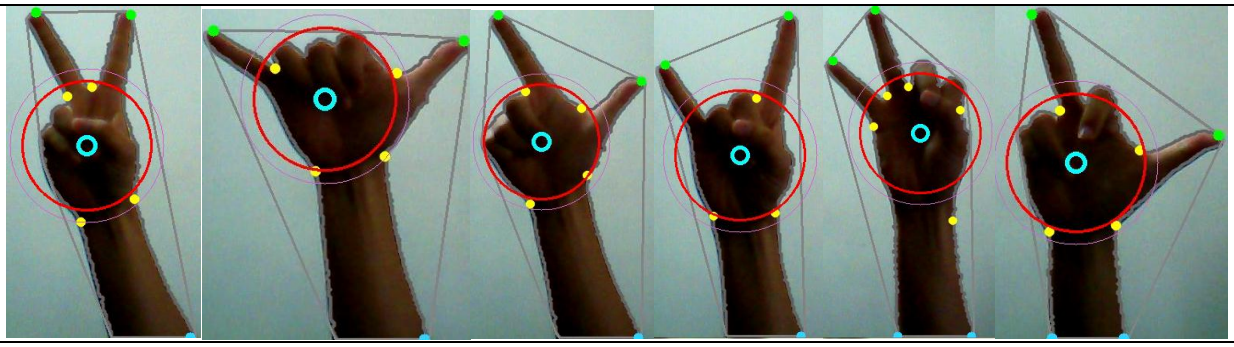
The system is implemented using C programming language under Microsoft Visual Studio 2010. Our experiment is running on a personal computer with Intel® Dual-Core E4500 2.7GHz CPU and 2GB RAM. The video image is captured by the E-books W3 web camera with resolution of 640x480.

Since the hand gesture can be in various different types, we define some certain type of gestures to show our experiment result. In our experiment, arm can be straight or tilt; hand can be straight or tilt also. And we show the results with different number of fingers (0 to 5) under the mentioned condition. Our experimental video has 21846 frames in total, and all the gesture conditions mentioned above are covered.

Figure 5.1 Web camera used in our system

5.1 Experiment 1 – Straight Arm and Hand

In section 5.1, we will represent our result of basic gesture. Basic gesture means that the gesture is not tilt, rotate, and the arm is straight, finger numbers can be from 0 to 5. Those basic gesture are the most frequency natural gesture for human, hence we should have a high accuracy on it.

Finger Numbers	Results
0	
1	
2	

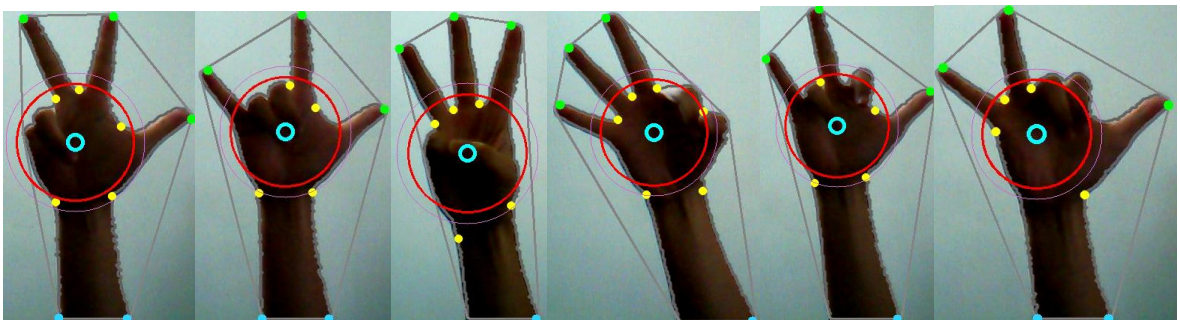
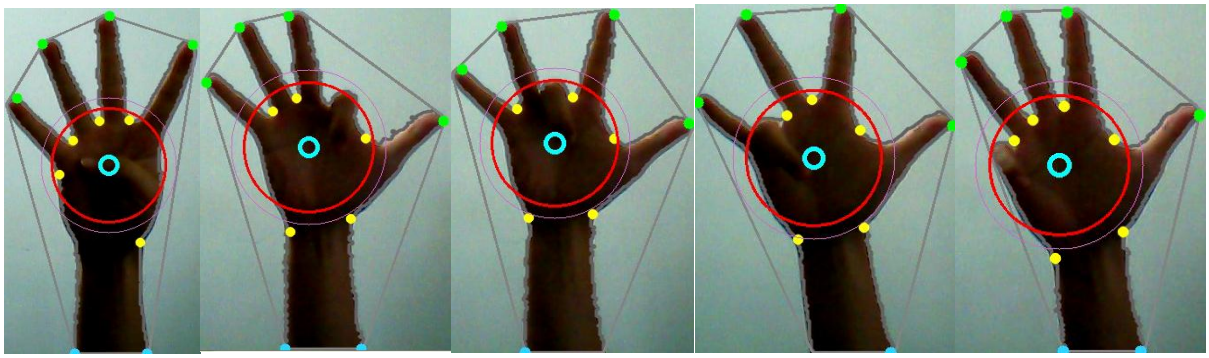
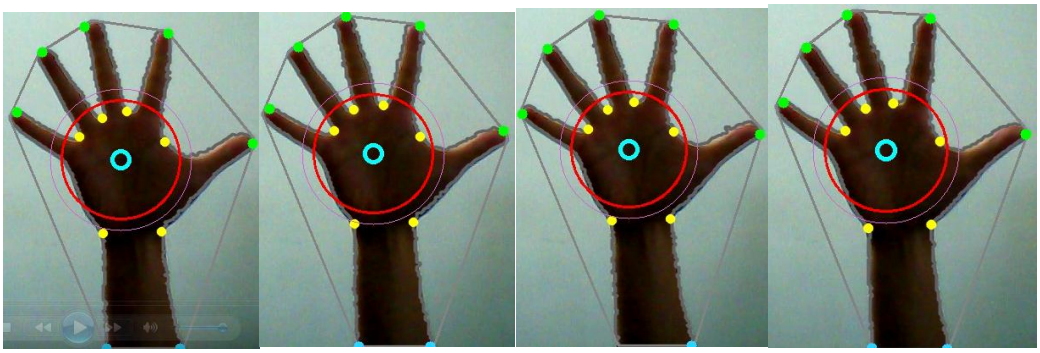
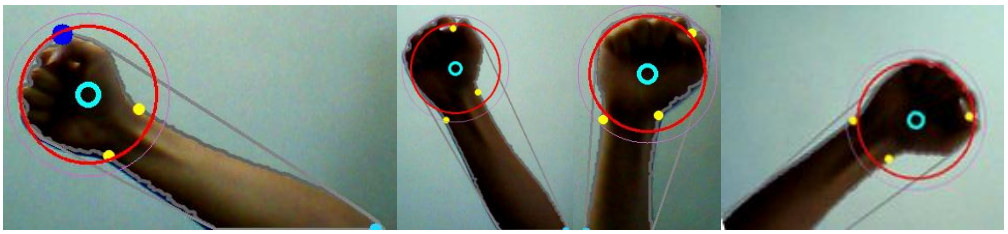
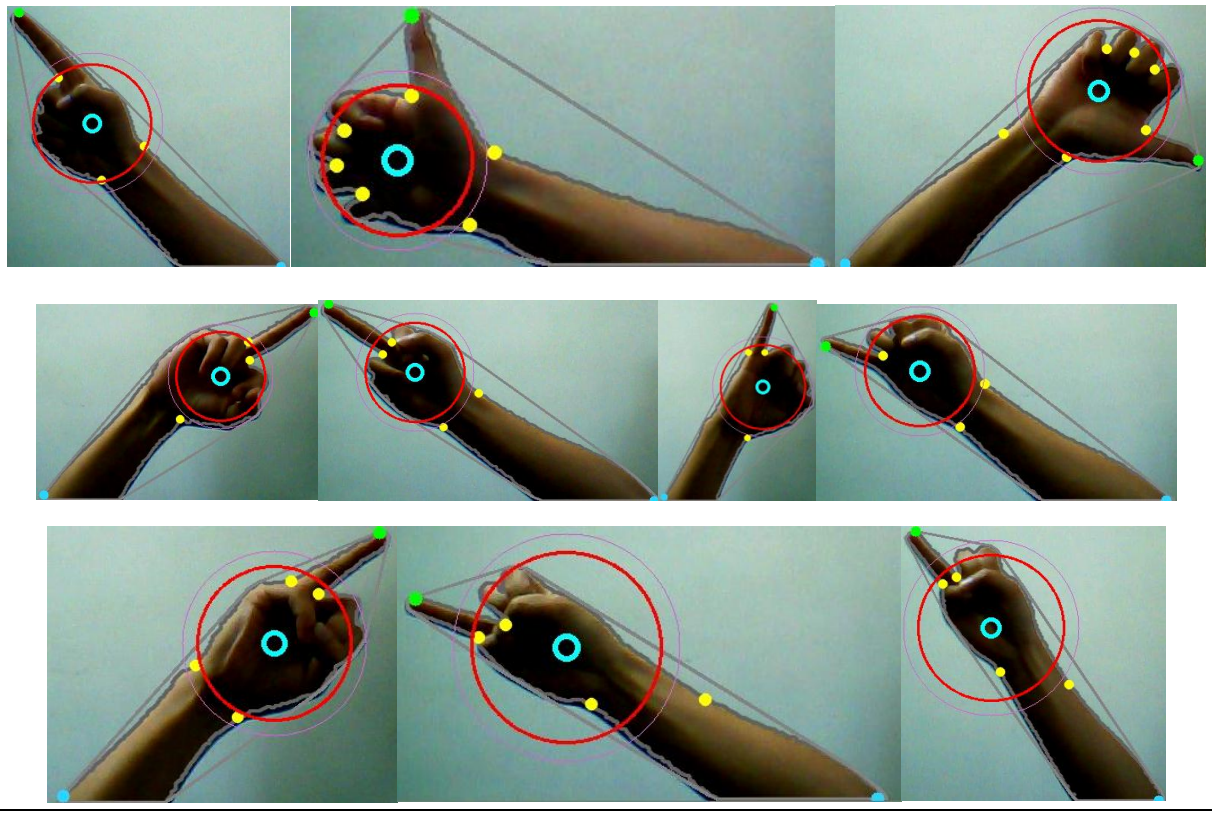
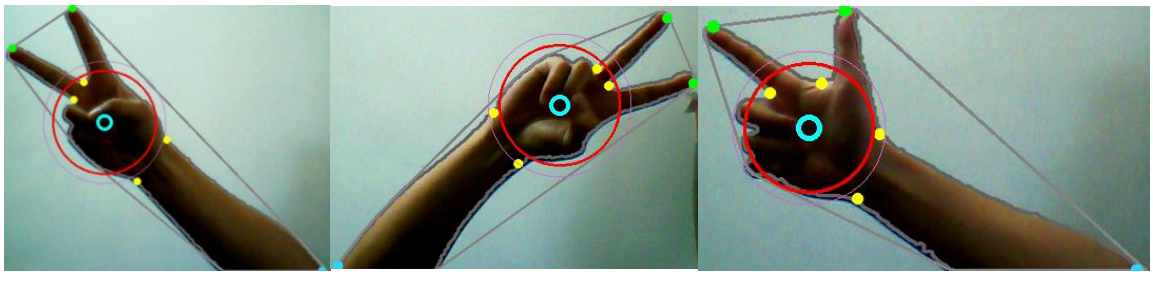
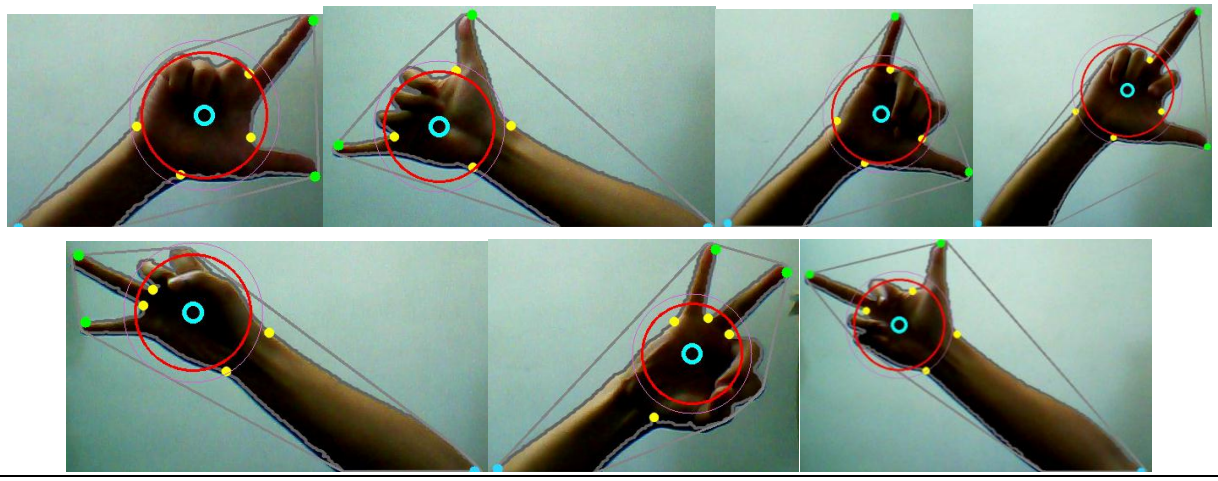
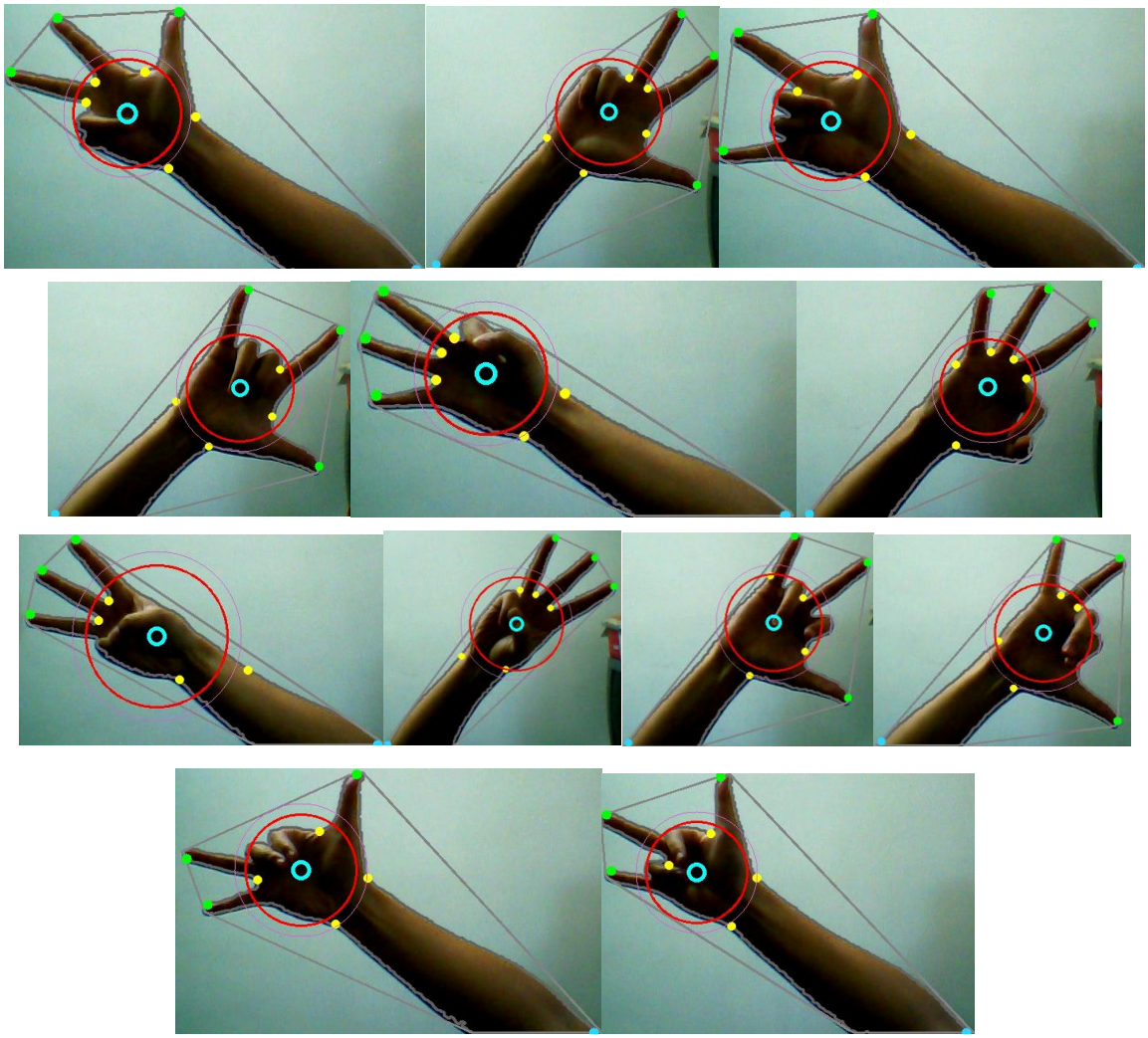
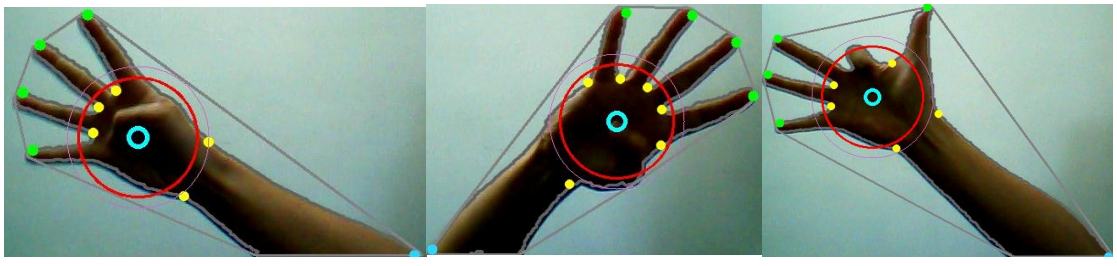
3	
4	
5	

Table 5.1 Result of experiment 1

5.2 Experiment 2 – Tilt Arm

This section shows the results when the arm has different levels of tilt; hand palm remains straight. The finger number can be from 0 to 5.

Finger Numbers	Results
0	
1	
2	

	
3	
4	

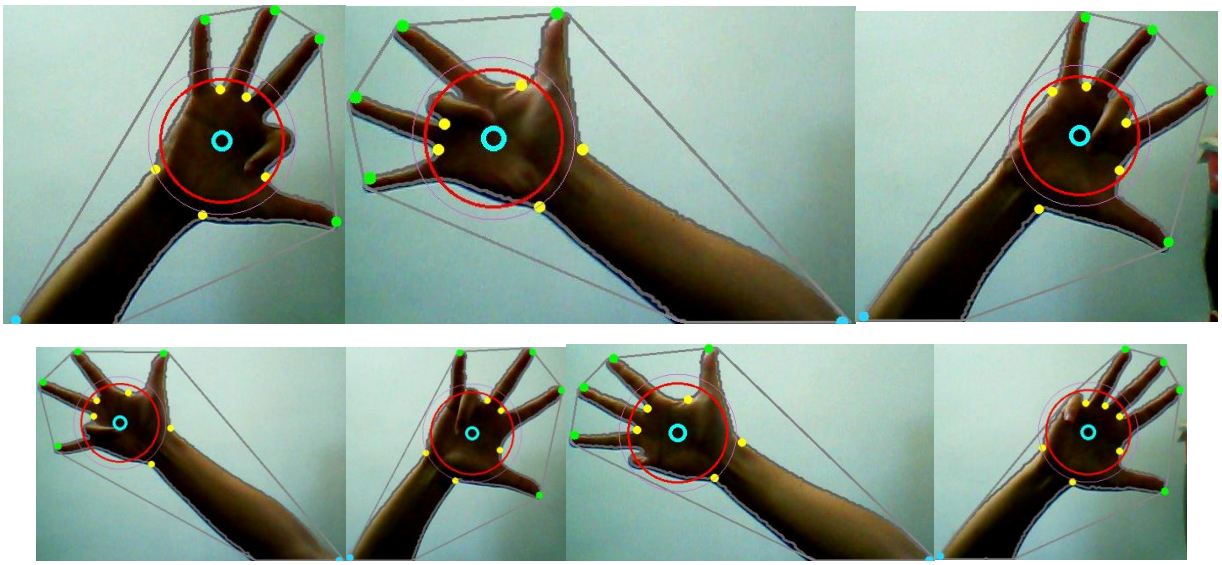
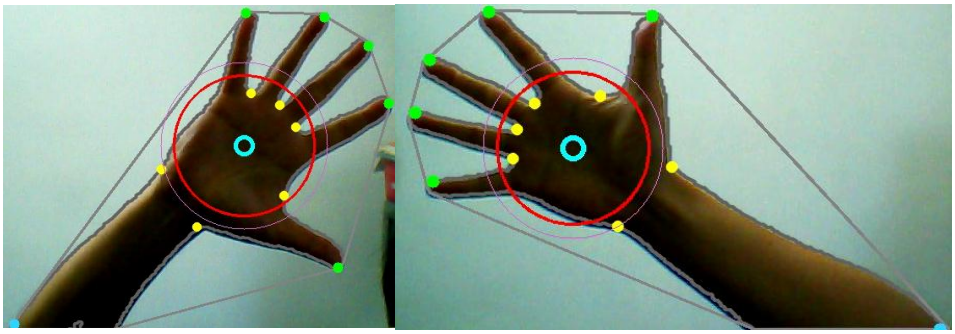
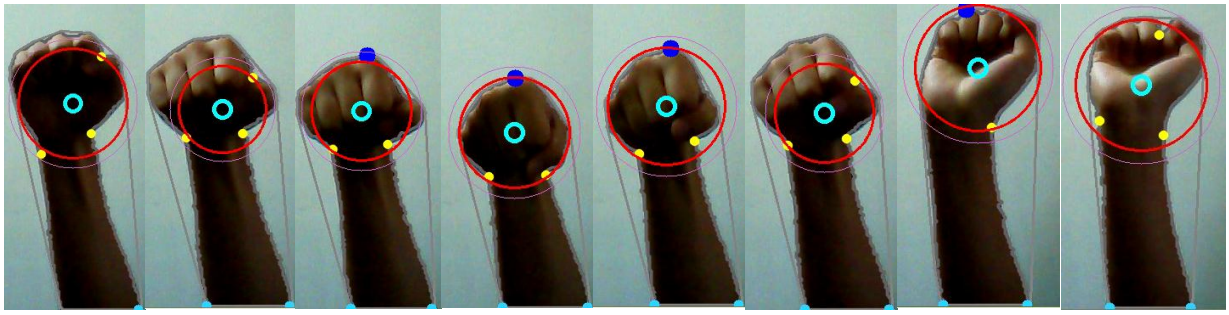
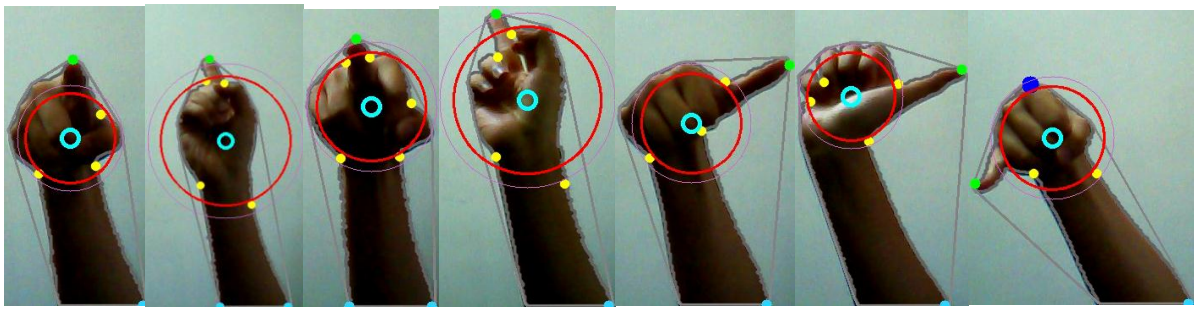
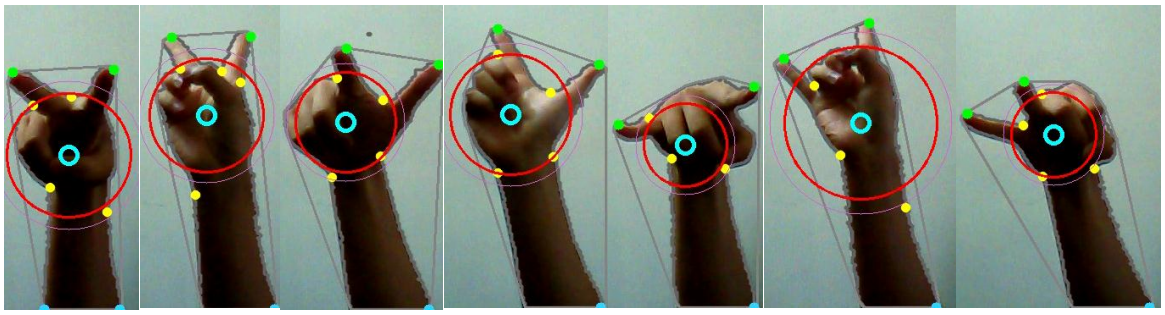
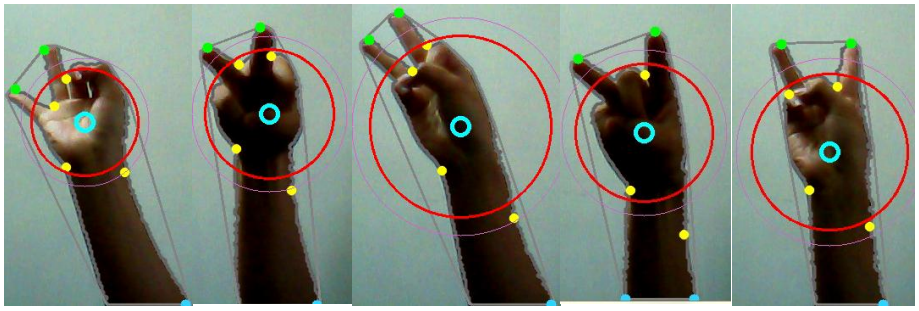
	
5	

Table 5.2 Result of experiment 2

5.3 Experiment 3 – Tilt Hand Palm

This section shows the results when the hand palm has different levels of tilt; arm remains straight. The finger number can be from 0 to 5.

Finger Numbers	Results
0	
1	
2	 

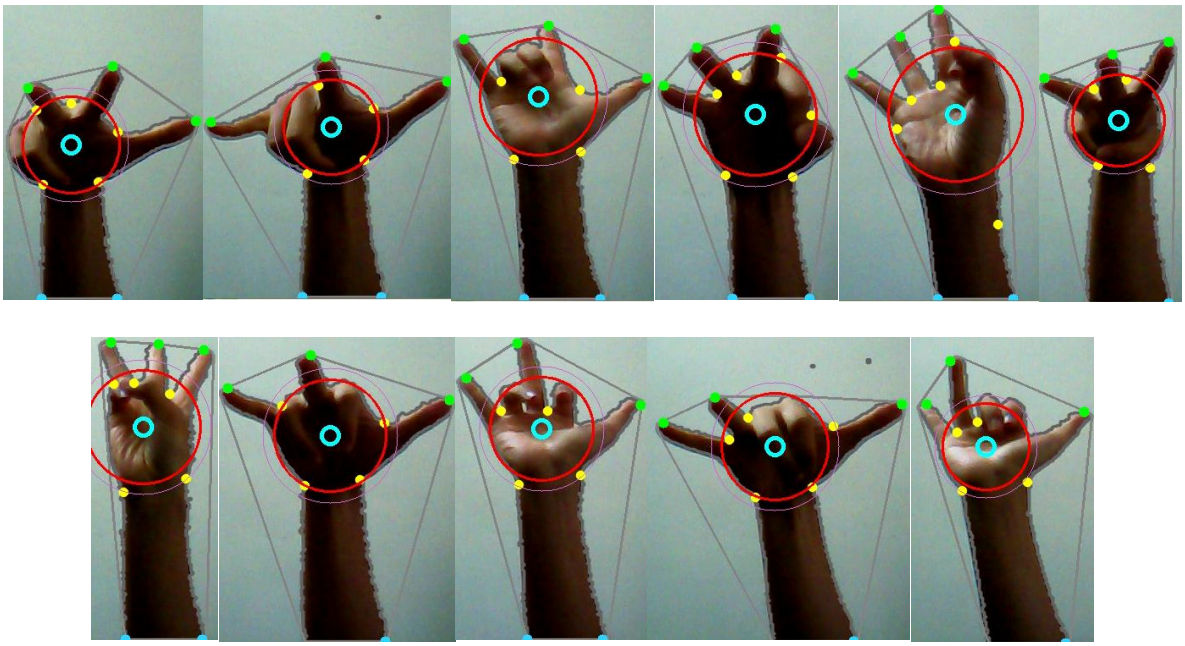
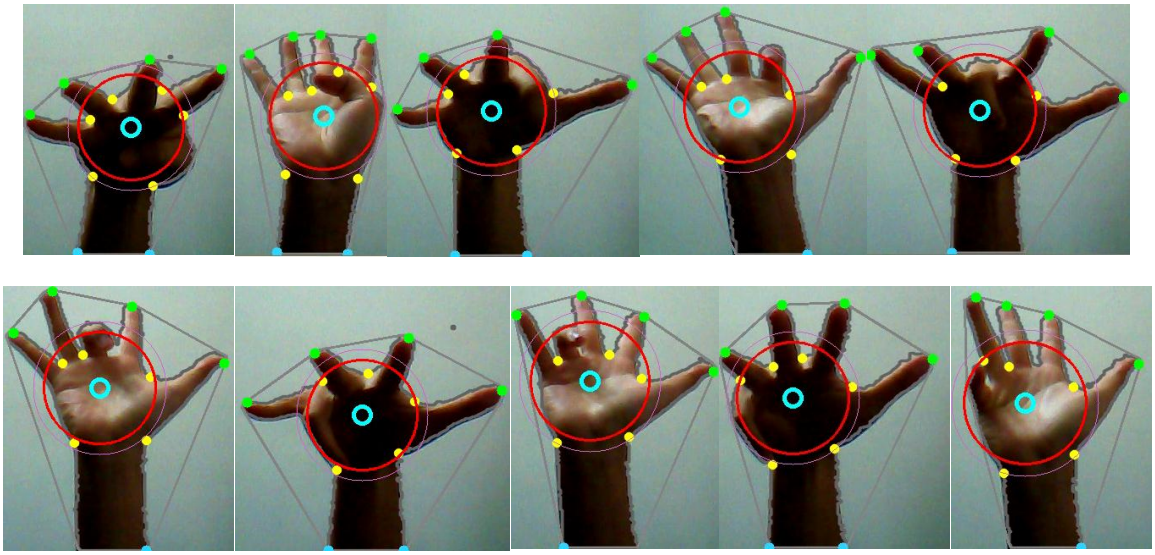
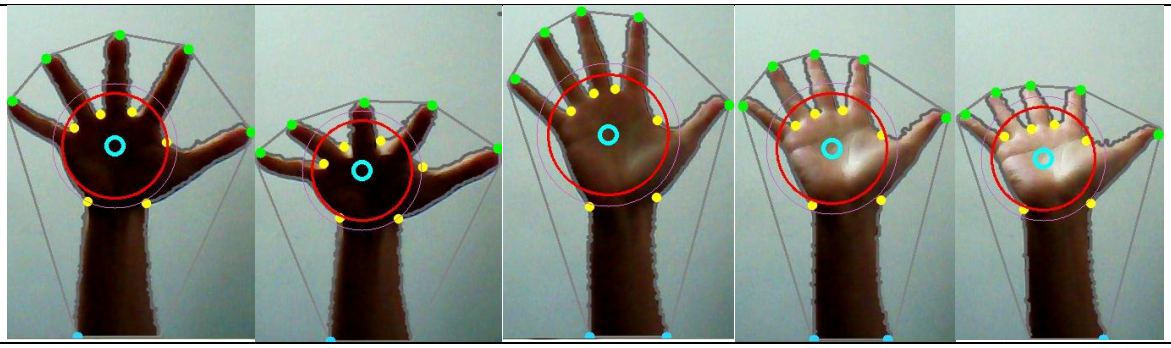
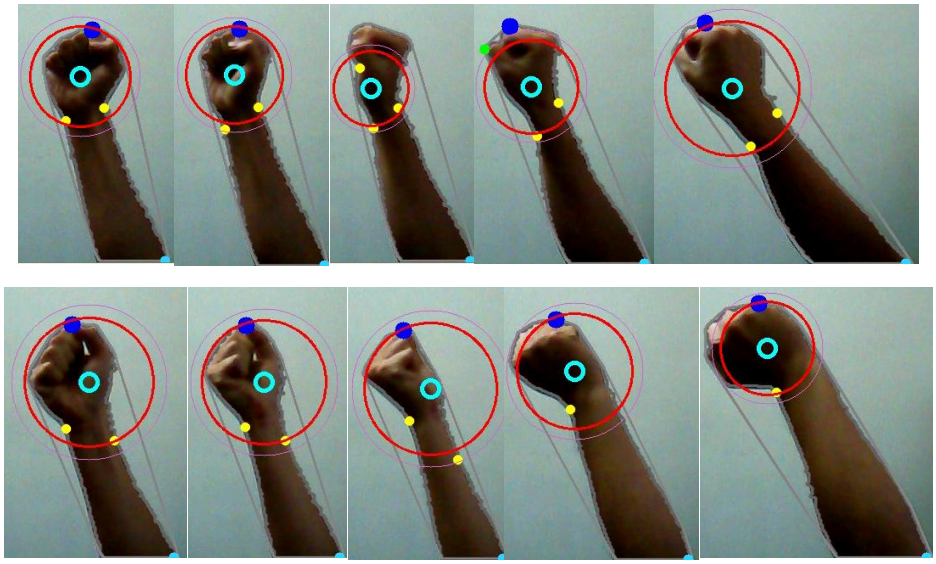
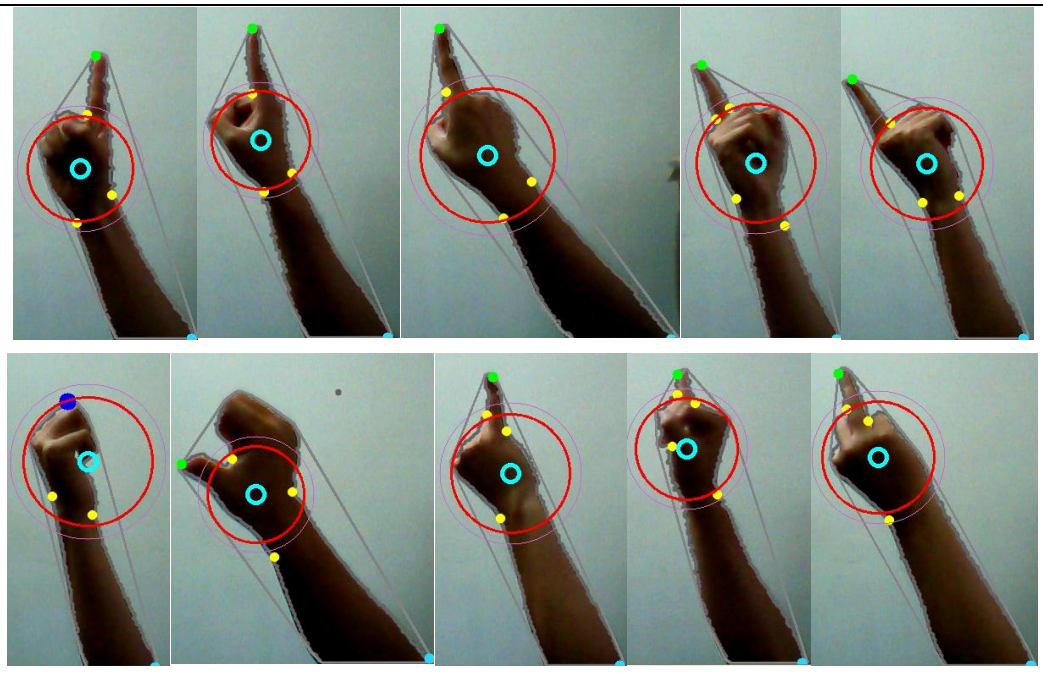
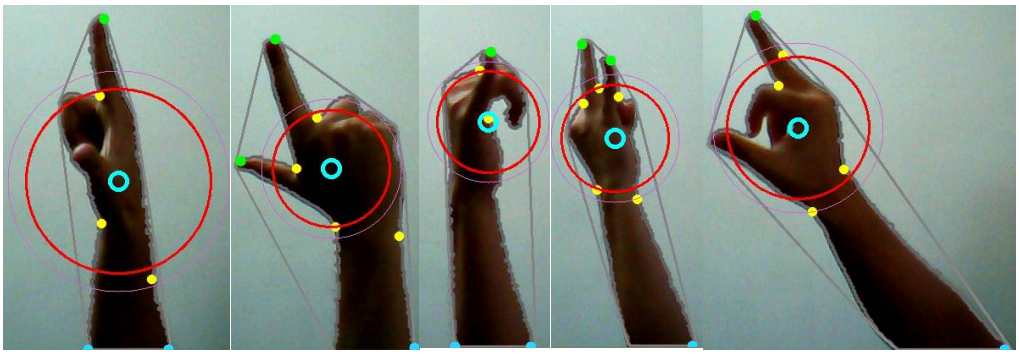
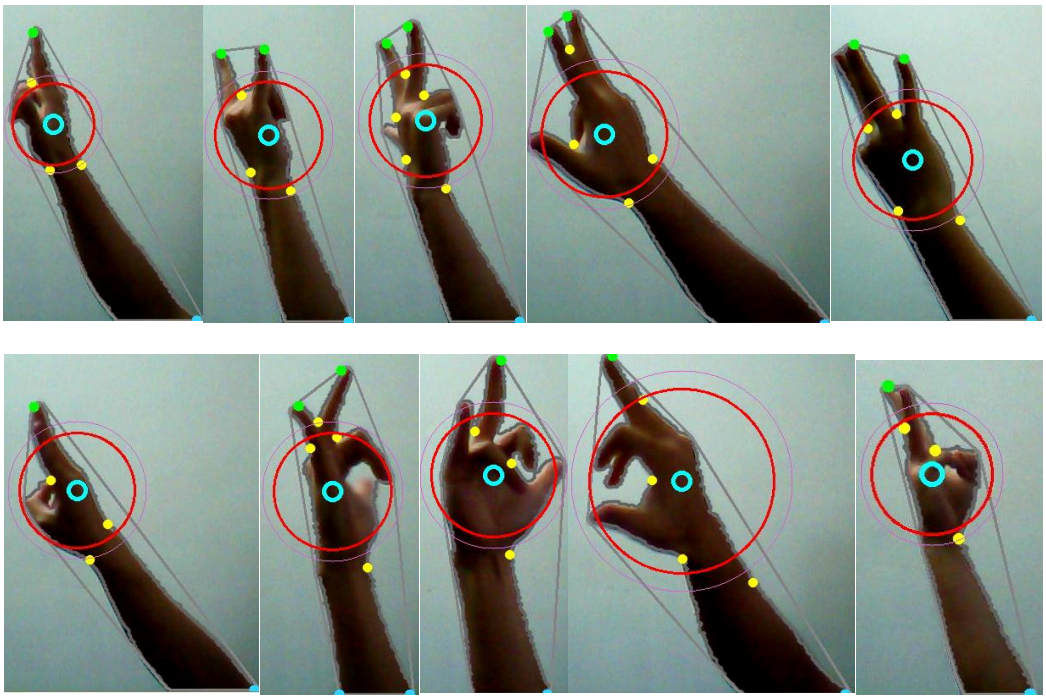
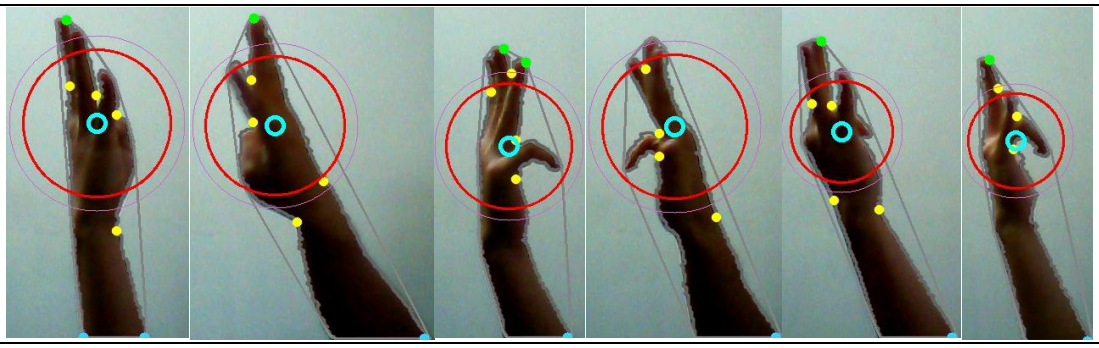
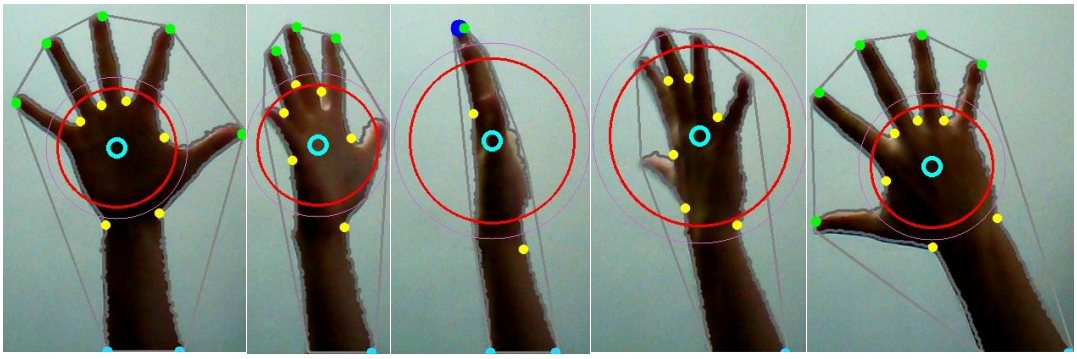
3	
4	
5	

Table 5.3 Result of experiment 3

5.4 Experiment 4 – Rotated Arm

This section shows the results when the arm has different levels of rotation as well as the hand palm. The finger number can be from 0 to 5.

Finger Numbers	Results
0	
1	

2	
3	
4	
5	

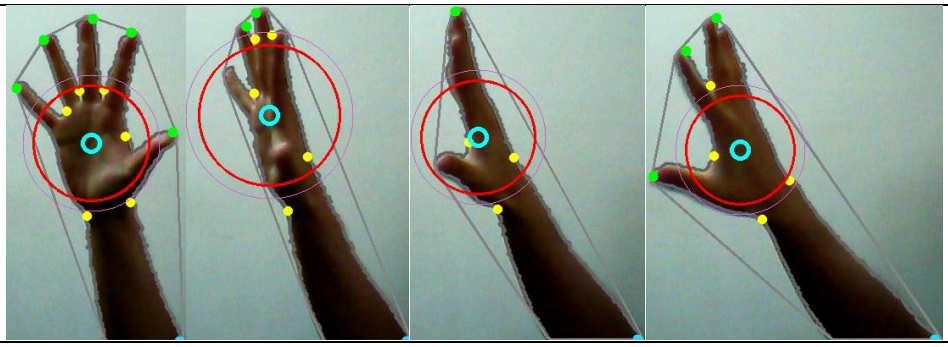
	

Table 5.4 Result of experiment 4

5.5 Experiment Data

This section shows the experiment data of each experiment above; we calculate the accuracy of fingertips and palm. Palm position accuracy can be measured in two ways; one is see if the estimated palm position is inside the hand palm. This one is objective; we can clearly know that if it's inside the palm area. But sometimes the estimation is not good enough even it is inside the hand area; the estimation may be close to the boundary of palm area. So we also measure the palm position accuracy in a subjective way. The estimation will be failed if it's too close to the boundary or even exceed it. But the palm position estimation needn't to be exactly in the palm center, we just eliminate those which are obviously wrong.

Finger Number	Frames	Total Fingertips	Failed Fingertips	Fingertip Accuracy	Objectively Failed Palm	Subjectively Failed Palm	Objective Accuracy	Subjective Accuracy
0	100	0	0	100%	0	0	100%	100%
1	714	714	9	98.45%	0	3	100%	99.57%
2	982	1964	16	99.18%	0	0	100%	100%
3	920	2760	34	98.76%	0	3	100%	99.89%
4	709	2836	21	99.25%	0	0	100%	100%
5	154	770	1	99.87%	0	0	100%	100%
Total Frames: 3579 Total Fingertip Accuracy: 99.10% Total Objective Palm Accuracy: 100% Total Subjective Palm Accuracy: 99.91%								

Table 5.5 Data of Experiment 1

The results of experiment 1 are shown in table 5.5, it shows that our method has high accuracy on the basic gestures. Palm position and fingertips can be detected accurately in almost every frame.

Finger Number	Frames	Total Fingertips	Failed Fingertips	Fingertip Accuracy	Objectively Failed Palm	Subjectively Failed Palm	Objective Accuracy	Subjective Accuracy
0	324	0	0	100%	0	0	100%	100%
1	1693	1693	7	99.58%	5	13	99.70%	99.23%
2	1833	3666	19	99.48%	0	20	100%	98.90%
3	2074	6222	42	99.32%	1	5	99.95%	99.75%
4	1871	7484	36	99.51%	0	0	100%	100%
5	383	1915	8	99.58%	0	0	100%	100%
Total Frames: 8178 Total Fingertip Accuracy: 99.46% Total Objective Palm Accuracy: 99.92% Total Subjective Palm Accuracy: 99.53%								

Table 5.6 Data of Experiment 2

The results of experiment 2 are shown in table 5.6, palm position and fingertips can be detected in almost every frame. Tilt arm somehow effected the palm position estimation very slightly since we have more failed palm estimation in experiment 2; but the accuracy is still higher than 99%.

Finger Number	Frames	Objectively Failed Palm	Subjectively Failed Palm	Objective Accuracy	Subjective Accuracy
0	428	0	23	100%	94.62%
1	1508	0	230	100%	84.74%
2	1449	0	88	100%	93.92%
3	1327	0	30	100%	97.73%
4	874	0	3	100%	99.65%
5	331	0	6	100%	98.18%
Total Frames: 5917 Total Objective Palm Accuracy: 100% Total Subjective Palm Accuracy: 93.57%					

Table 5.7 Data of Experiment 3

In table 5.7 we can see that the estimated palm position can 100% be located inside the palm region when the hand palm is tilt. But when there's only one finger, the estimated palm position might be too close to the boundary, causes the accuracy of 1 finger drop down to 84.74%. But the accuracy in general is 93.57%. The accuracy for fingertips is not estimated in experiment 3 and experiment 4 since the hand will be moving a lot, many fingertips will be covered up by the palm circle. Those videos are not ideal for fingertip accuracy estimation.

Finger Number	Frames	Objectively Failed Palm	Subjectively Failed Palm	Objective Accuracy	Subjective Accuracy
0	164	0	40	100%	75.60%
1	761	2	118	99.73%	84.49%
2	997	2	50	99.79%	94.98%
3	1128	5	69	99.55%	93.88%
4	929	6	98	99.35%	89.45%
5	193	0	13	98.36%	93.26%
Total Frames: 4172 Total Objective Palm Accuracy: 99.64% Total Subjective Palm Accuracy: 90.69%					

Table 5.8 Data of Experiment 4

The palm position is hard to estimate when the arm is rotated. Our experiment shows that the accuracy drop down to 90.69%, but estimated palm position is still inside the hand palm region in almost every frame.

Experiment 3 and experiment 4 show that our method has high toleration to tilt and rotate hand. The most common way to find the palm is to calculate the point inside the hand which has the largest distance. But the estimation can be wrong when the palm region is not big enough, the point which has the largest distance could be in the arm as figure 5.2 shown.

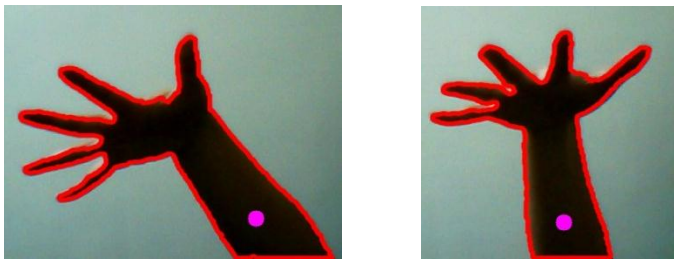


Figure 5.2 The point with the largest distance

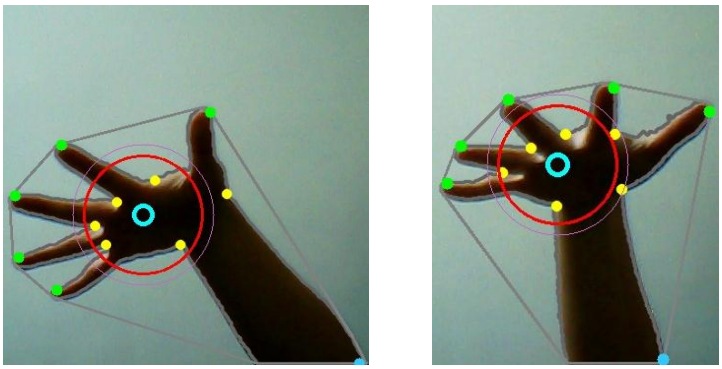


Figure 5.3 Result of Our method

5.6 Extend to two hands

We extend our method to two hands by applying the detections to the longest and the second longest contour. The restriction of it will be no overlapped hands. In order to eliminate the affection of body and face, we use a depth camera to capture the hand contour.

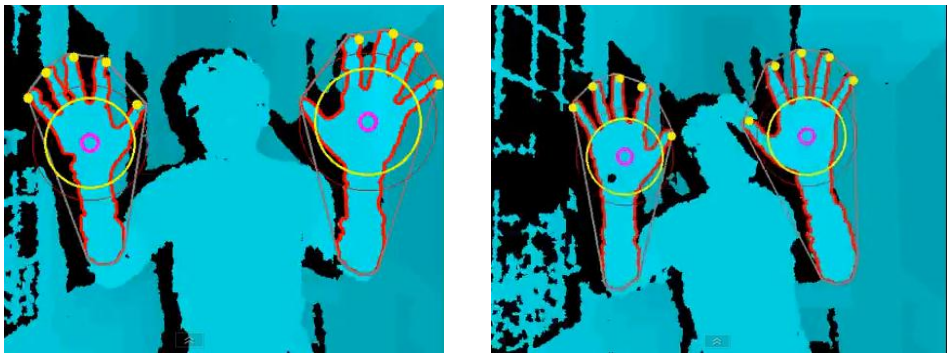


Figure 5.4 Extend to two hands

Chapter 6 Conclusions and Future Work

6.1 Conclusions

In this system, we can have an accurate palm and fingertip positions estimation based on a hand contour. Fore-arm can be included to the contour and the system has a good toleration to rotate and tilt hand. We obtained the color image from single web camera, and transform the color space into HSV color space. Our skin color region is defined under HSV color space. A binary image of hand can be obtained according to the skin color definition. Two morphological operations include erosion and dilation are performed. Erosion eliminates the noises while dilation smooths the boundary. When the usable image is generated, we apply Pavilidis' algorithm to search for the contours in the image. One or more contours can be found, we choose the largest contour as our hand contour. When the hand contour has been chosen, we need to calculate the convex hull of it. Three-coin algorithm is applied to obtain the convex hull. We can compare the contour and its convex hull to find all the convexity defects. The contour point inside one convexity defect which has the longest distance to the contour will be the depth point of the convexity defect. By observing the depth points in many different gestures, we notice that the depth points are tending to be around the hand palm. Minimum enclosing circle of all the depth points will be calculated to estimate the palm and the palm center. Also the fingertips can be detected by Wen and Niu's method. There are some conditions which we might need to modify our estimation. When the amount of depth points are two or less, the estimation of palm could be wrong. We add the highest point in the contour as an extra depth point. Sometimes the depth points beside the wrist will slipped away and enlarge the minimum enclosing circle; which will cause the wrong estimation of palm. We

calculate the average position of depth points and mix it with the minimum enclosing circle. This will reduce the effect of the slipped point. Another situation is that when a finger is bending, an unwanted fingertip will be found. We set a threshold of 1.2 * radius of minimum enclosing circle to remove those unwanted fingertips. Experiment shows that our method has high accuracy and low computation cost. And it increases the toleration to rotation and tilt, which increases the freedom of usability.

6.2 Future Work

1) Hand and arm recognition – Our system so far assume the input data to be the arm; we can apply hand or arm recognition so that our method can be used in a more flexible condition.

2) Using stereo camera or depth information – We can use two camera from different viewpoint to locate those hand features more accurately. Since the depth camera is a common product nowadays, depth image can be obtained easily. Depth information can be included to know about exact condition of rotation or tilt; and even solve the problem of overlapped hands.

3) Combine with Augmented Reality – AR is getting more and more popular recently. Instead of using paper mark, we can use the hand features to locate the virtual objects; and those objects can be interactive to the hand. So the virtual objects won't be just shown in the video, we can interact with it.

References

[1] H. C. Xu, “Principal Component Analysis on Fingertips for Gesture Recognition” Master Thesis, Department of Applied Marine physics & Undersea Technology, National Sun Yat-sen University, Kaohsiung, Taiwan, 2006.

[2] A. Anagnostopoulos, A. Pnevmatikakis, “A real-time mixed reality system for seamless interaction between real and virtual objects”, Proceedings of the 3rd international conference on Digital Interactive Media in Entertainment and Arts, September 10-12, 2008, Athens, Greece

[3] G. Amayeh, G. Bebis, A. Erol, and M. Nicolescu, “Hand-Based Verification and Identification Using Palm-Finger Segmentation and Fusion,” Computer Vision and Image Understanding, vol. 113, pp.477-501, 2009.

[4] Weiying Chen, Fujiki, R.; Arita, D.; Taniguchi, R.-I, “Real-Time 3D hand Shape Estimation based on Image Feature Analysis and Inverse Kinematics”, Image Analysis and Processing, 2007. ICIAP 2007. 14th International Conference, pp.247-252, 2007

[5] Y.H. Chang and C.M. Chang, “Automatic Hand-Pose Trajectory Tracking System Using Video Sequences”, Master Thesis, Department of Information and Computer Engineering, Chung-Yuan Christian University, Chung-Li, Taiwan, 2010.

[6] M. Soriano, S. Huovinen, B. Martinkauppi, and M. Laaksonen, “Using the skin locus to cope with changing illumination conditions in color-based face tracking,” in *Proceedings of the IEEE Nordic Signal Processing Symposium*, pp. 383-386, 2000.

[7] T. Pavlidis, “Algorithms for Graphics and Image Processing”, Computer Science Press, Rockville, Maryland, 1982.

[8] R. L. Graham. “An efficient algorithm for determining the convex hull of a finite planar set”, Information Processing Letters, 7:175–180, 1972.

[9] Sklansky J., "Measuring Concavity on a Rectangular Mosaic", IEEE Transactions on Computing, 21, p1355, 1972.

[10] M. Soriano, S. Huovinen, B. Martinkauppi, and M. Laaksonen, “Using the skin locus to cope with changing illumination conditions in color-based face tracking,” in Proceedings of the IEEE Nordic Signal Processing Symposium, pp. 383-386, 2000.

[11] R.C. Gonzalez and R.E. Woods, Digital Image Processing, 2nd Ed., Addison Wesley, Reading, Massachusetts, 1992.

[12] S. L. Phung, A. Bouzerdoum, and D. Chai, “A novel skin color model in YCbCr color space and its application to human face detection,” in Proceedings of IEEE International Conference on Image Processing, vol. 1, pp. 289-291, 2002.

[13] X. Wen, Y. Niu, “A Method for Hand Gesture Recognition Based on Morphology and Fingertip- Angle”, The 2nd International Conference on Computer and Automation Engineering (ICCAE), vol. 1, pp. 688-691, 2010

[14] N. Megiddo, "Linear-time algorithms for linear programming in *R*^3 and related problems," *SIAM Journal on Computing* **12** (1983), No. 4, 759–776

[15] S. Skyum, **“**A simple algorithm for computing the smallest enclosing circle”, Information Processing Letters, vol. 37, Issue 3, 1991

[16] P. Bourke, “Equation of a Circle from 3 Points (2 dimensions)”, http://paulbourke.net/geometry/circlefrom3/ , Jan. 1991.

[17] http://en.wikipedia.org/wiki/Dot_product

[18] C. Manresa, J. Varona, R. Mas, F.J. Perales, “Real –Time Hand Tracking and Gesture Recognition for Human-Computer Interaction”, Electronic Letters on Computer Vision and Image Analysis 0(0):1-7, 2000

[19] C.W. Ng, S. Ranganath, “Real-time gesture recognition system and application”, Image and Vision Computing 20, pp. 993–1007, 2002

[20] Xiaoming Yin, Ming Xie, “Finger identification and hand posture recognition for human–robot interaction” *ScienceDirect Image and Vision Computing*, vol.25 , PP. 1291-1300, 2007.