國立交通大學

電子工程學系 電子研究所

碩 士 論 文

# 適用於人機介面之單一攝影機手勢辨識技術

# Single-camera Hand Gesture Recognition for Human-Computer Interface

研 究 生：姜政銘

指導教授：王聖智 教授

中 華 民 國 一 Ｏ 二 年 八 月

# 適用於人機介面之單一攝影機手勢辨識技術

# Single-camera Hand Gesture Recognition for Human-Computer Interface

研 究 生：姜政銘　　　　Student：Cheng-Ming Chiang

指導教授：王聖智 教授　　Advisor：Prof. Sheng-Jyh Wang

國 立 交 通 大 學

電子工程學系 電子研究所

碩 士 論 文

A Thesis
Submitted to Department of Electronics Engineering and
Institute of Electronics
College of Electrical and Computer Engineering
National Chiao Tung University
in partial Fulfillment of the Requirements
for the Degree of
Master of Science
in
Electronics Engineering

August 2013

Hsinchu, Taiwan, Republic of China

中 華 民 國 一 〇 二 年 八 月

# 適用於人機介面之單一攝影機手勢辨識技術

研究生：姜政銘　　　指導教授：王聖智 教授

國立交通大學

電子工程學系　電子研究所碩士班

## 摘要

在本篇論文中，我們提出了一個僅利用一台可見光攝影機便可遙控操作人機介面之手勢辨識技術。此系統主要是由一台投影機及一台安裝在大型面板左側的可見光攝影機組成，我們希望此系統能讓操作區域不再侷限於面板之前，以達到可以遙控操作的目的。在此條件下，背景可能是非常雜亂的，我們主要探討的議題包括如何在此情況下快速地找到手的位置，以及辨識使用者的手勢，以取代滑鼠的使用。在我們的演算法中，我們會先利用簡單的校正過程來取得手的初始位置，以及影像座標系和投影螢幕坐標的相對位移關係，接著，我們利用追縱演算法來取得手的位置，並使用手部偵測來輔助追縱演算法的判斷，使這兩種演算法能夠發揮相輔相成的作用。此外，我們並使用手勢辨識來判斷目前的手勢為何。最後，我們將影像中手的位置投影回投影幕上，這樣即可利用手的移動來控制游標的移動並讓系統做出對應的手勢反應。

# Single-camera Hand Gesture Recognition for Human-Computer Interface

Student：Cheng-Ming Chiang      Advisor：Prof. Sheng-Jyh Wang

Department of Electronics Engineering, Institute of Electronics
National Chiao Tung University

## Abstract

In this thesis, we propose a novel hand gesture recognition technique for a remote-control human computer interface (HCI) using a single visible-light camera. The system is mainly composed of an image projector and a camera installed on the left side of the panel. We wish to develop a human computer interface that is not limited to finger touching on the board, but allows remotely controlling the system. In this system, we develop our human computer interface in order to find the hand location and to recognize human hand gesture in cluttered backgrounds in real time. In our approach, we first use a simple calibration process to get the initial position of the hand and the relation between image coordinates and the projected board coordinates. After that, we develop a tracking algorithm to get the position of hand, with the help of a hand detection algorithm. Next, we use a gesture recognition technique to recognize the current gesture. We also integrate the detection algorithm with the tracking algorithm to boost the performance. Finally, by projecting the detected hand position onto the projected screen, we can replace the use of mouse and use hand gesture to control the system.

# 誌謝

能夠完成此篇論文，首先要感謝指導教授 王聖智老師的教導。在研究上，老師總是會在我們遇到困難時適時地提點我們。另一方面，又保有讓我們獨立思考、培養創造力的空間。除了研究之外，老師也會教導我們讀書、報告、寫作上的一些方法，乃至於生活及待人處事上，老師也都會與我們分享他的經驗及心得。這兩年來，真的是受益良多，衷心地感謝老師的指導，也很開心能夠當王老師的學生。

也感謝這兩年來所有的實驗室成員，謝謝博士班學長：慈澄、禎宇、家豪；碩士班學長姐：柏翔、耀笙、彥廷、心憫、秉修、儲培、利容；同屆同學們：介暐、姿婷、秉宸、佳峻；以及學弟妹：冠廷、汝欣、非凡、振源、奕中、子銘、浩瑋，因為有你們，讓我的碩士生活更多采多姿且不孤單。不論是一同討論課業、研究，一同運動、出遊，或是吃飯聚餐，都是美好的回憶。希望畢業後大家還能繼續保持聯絡，也祝福大家都有一個美好的未來。

除此之外，也很感謝從小到大求學過程中的同學們及好朋友們，也謝謝所有教導過我的老師們。你們豐富了我的求學過程，讓我能夠很順利地一直到完成碩士學位。

最重要的，感謝的我父母及家人，有你們的栽培，我才能完成碩士學位。也謝謝家瑜在一直陪伴在我身邊鼓勵我。妳們是我的最大動力來源。

# Content

# List of Figures

# List of Tables

# Chapter 1 Introduction

Human-computer interface (HCI) plays an important role in our daily life. Computers and machines can do lots of things for us. How to interact with them is what we are interested in. Keyboard and mouse are the commonest HCIs. As Computer Vision and Machine Learning algorithms become more and more powerful, computers can communicate with human more directly. Today, we can use HCIs more conveniently and intuitively, such as the use of a multi-touch system or a remote-control system.

Multi-touch technologies has been widely used in smart phones and tablets. However, for a large-scale human-computer interface, the cost is still a main issue. Furthermore, this multi-touch system can only detect the position of objects or fingers when the objects and fingers touch the screen [2]. This system restricts the operation zone to the front area of the touch screen. It is sometimes inconvenient to touch all the locations of a large-scale device. Hence, a human-computer interface that allows remote-control has become more and more useful. For this type of HCI systems, we can use intuitive gestures to interact with the computer and the operation zone is no longer restricted to a small region.

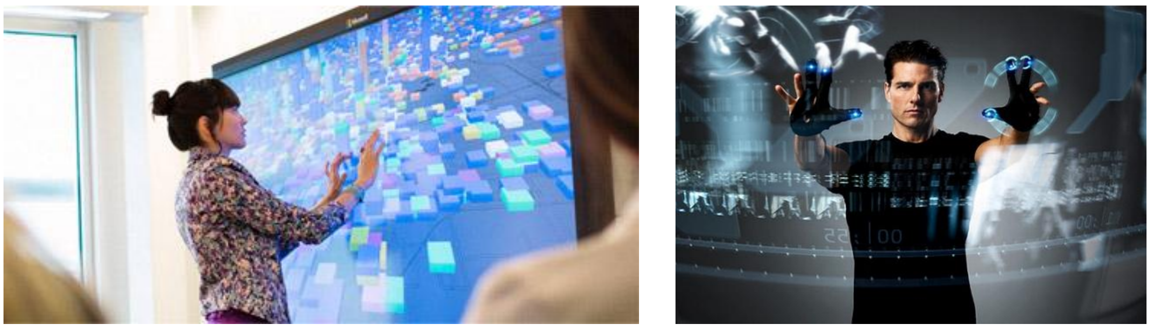


Figure 1-1 (a) Large-scale touch device by Microsoft [2]. (b) The scene in the movie Minority Report [3].

In this thesis, we propose a hand gesture recognition algorithm and apply this algorithm to remote-control human computer interface. Hence, we develop two applications to demonstrate the feasibility of this system. One application is a jigsaw puzzle game system and the other is a slide presentation system. Since the variation of the hand is quite dramatic and the operation zone is not restricted to the board, how to find the hand in an image and how to recognize its gesture precisely are the main issues of this thesis. For a convenient HCI, computer should understand what the user conveys as soon as the user uses a gesture. Hence, the other issue we concern is the processing time. Moreover, we aim to solve these problems with the use of one visible-light camera.

In our algorithm, to interact with the computer, we first use a simple calibration process to get the relation between the image and the projected screen. We find the location of the hand using a tracking algorithm initially. We propose an integrator that can automatically revise the location of hand using a detection technique if the tracking result is not precise enough. This mechanism can also detect the presence of the hand so it needs not track the hand in every frame. With the precise location of the hand, we recognize the current gesture. Finally, the computer acts based on the detected gesture.

The thesis is organized as follows. In Chapter 2, we first introduce some recent famous remote-control HCI systems. We also introduce the tracking and detection algorithms that relate to our work. Next, we describe the proposed algorithm in detail for a remote-control human-computer interface in Chapter 3. Experimental Results are shown in Chapter 4. In Chapter 5, we describe two applications of our hand gesture recognition algorithm. Finally, we give conclusions in Chapter 6.

# Chapter 2  Backgrounds and Related Works

There are many HCIs nowadays which allow users to control the computer with hands and the users can keep a distance from the computer. In Section 2.1, we will discuss some famous HCIs. For our work, we need hand detection and tracking algorithms. Hence, we will describe the related works of these algorithms in Section 2.2 and in Section 2.3.

## 2.1 Remote-Control Human-Computer Interface

In this section, we introduce three famous HCIs in time order which are all remote-control systems. We describe the hardware of these systems and then describe the adopted algorithms. We also discuss their advantages and disadvantages. Finally, we compare our proposed human-computer interface with theses HCIs.

### 2.1.1 Color Glove by MIT Media Lab

In 2009, Wang and Popović proposed an easy-to-use and inexpensive interface that can estimate hand pose at interactive rates [4]. The user needs to wear a color glove for interaction. The glove is designed with a specific color pattern, as shown in Figure 2-1. There are twenty colored patches with ten distinct colors on the glove.

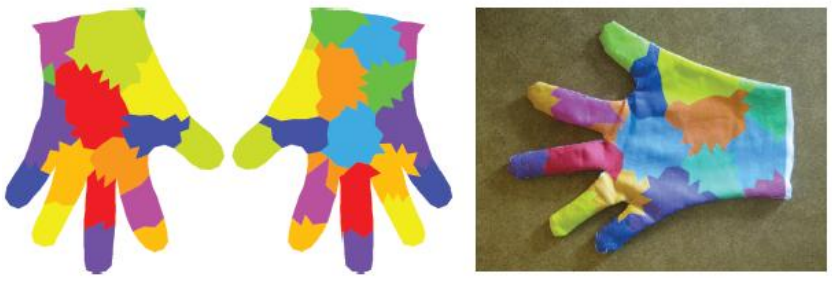


Figure 2-1 The glove designed by [4].

The authors construct a database of hand poses in advance which contains the sign language alphabets, common hand gesture, and random jiggling of fingers. Each image in the database is resized to a tiny image ($40 \times 40$). As shown in Figure 2-2, once a color pattern in single image is detected, they normalize the image into a tiny image. After that, in order to find the most likely hand pose, they use this tiny image as the query to search the pose database with the nearest neighbor method. Finally, the authors propose a few algorithms to improve the speed and accuracy in searching the database. They also improve the temporal smoothness to reduce jitters.

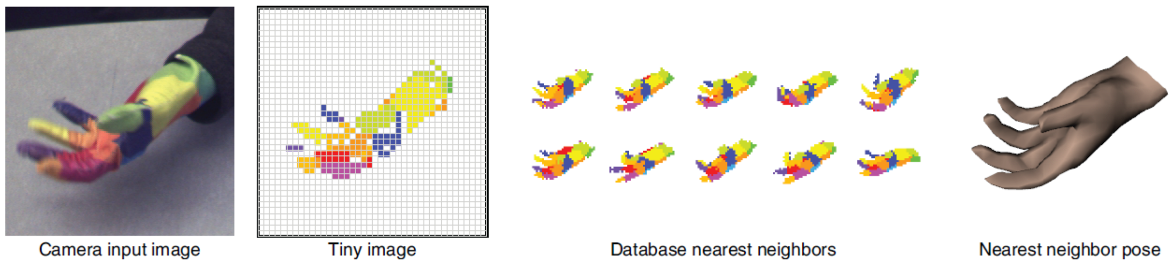


Figure 2-2 Pose estimation process in [4].

The advantages of their system are that it needs only one camera and thus it is cheap and easy to use. However, wearing a specific glove is somewhat inconvenient for general interaction. As a result, we want to design a bare-hand gesture recognition algorithm that has the advantages of low cost and easy to use.

### 2.1.2 Microsoft Kinect

Microsoft developed a remarkable device, Kinect, in 2010. The device can recognize the gestures of human body and the voice of the user. Users do not have to hold a controller like Nintendo Wii [5]. The use of Kinect is not only for the video game console Xbox 360, but it also allows the developers to write their own applications. Kinect is a powerful device to develop user interface of gesture recognition or speech recognition, as shown in Figure 2-3.

The hardware architecture is shown in Figure 2-4. There are three camera lenses on Kinect, an RGB camera and a pair of an infrared emitter and sensor. The infrared pair can generate a depth image using the “Light Coding” technique [6] [7]. The infrared emitter projects pre-defined infrared patterns into the surroundings. The infrared sensor captures the image of the distorted patterns reflected from the objects. The device estimates the depth map of the scene by measuring the distortion rate of those specific patterns. With the depth map, it is easier to recognize gestures by using 3-D information. Moreover, based on the depth image, Kinect can rapidly and accurately predict the 3-D positions of body joints and then detect the skeleton of the user. The detailed algorithm is described in [9].

Figure 2-3 The applications of Microsoft Kinect [8].

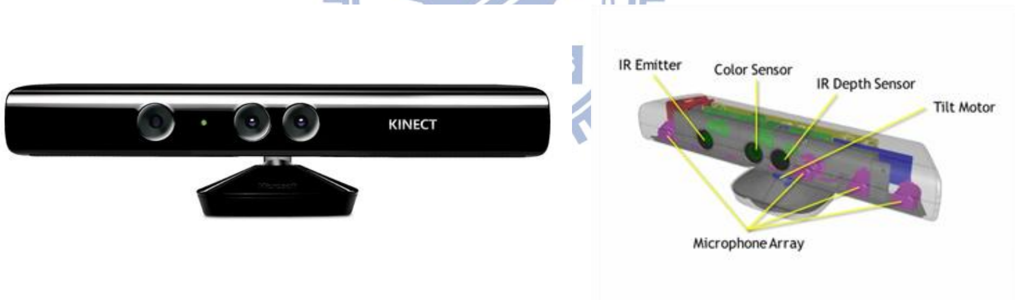


Figure 2-4 The hardware architecture of Kinect [8].

Without doubt, Kinect is an excellent device. Developers can create many creative and convenient user interface with the help of Kinect. However, the price of Kinect is not affordable for every user. Consequently, the goal of our thesis is to develop a cheap user interface that can recognize hand gestures.

### 2.1.3 Leap Motion

The Leap Motion, released in 2012, is able to detect hands, fingers and pen-like objects very precisely. The company [10] claims that the Leap Motion has the precision up to 0.01mm which excels Microsoft Kinect. On the other hand, unlike Kinect, the operation zone for the Leap Motion is limited. It is suitable for close interaction, like above the device for about 25 mm to 600 mm. The appearance of the Leap Motion is shown in Figure 2-5.

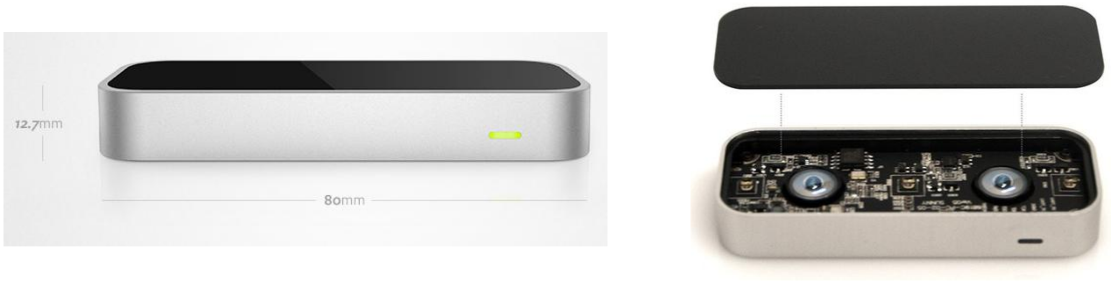


Figure 2-5 Appearance of the Leap Motion.
(a) Outward appearance [10]. (b) Inward appearance [11].

The Leap Motion is composed of two CCD cameras and three infrared LEDs. It can reconstruct the 3-D coordinates above the device, as shown in figure 2-6 [11]. The x-axis is parallel to the device, pointing toward the right side of the screen. The y-axis points upward and the z-axis, which represents the depth information, points away from the screen. Using a powerful mathematical algorithm, the Leap Motion can track hands, fingers and pointable objects and record the motion information.

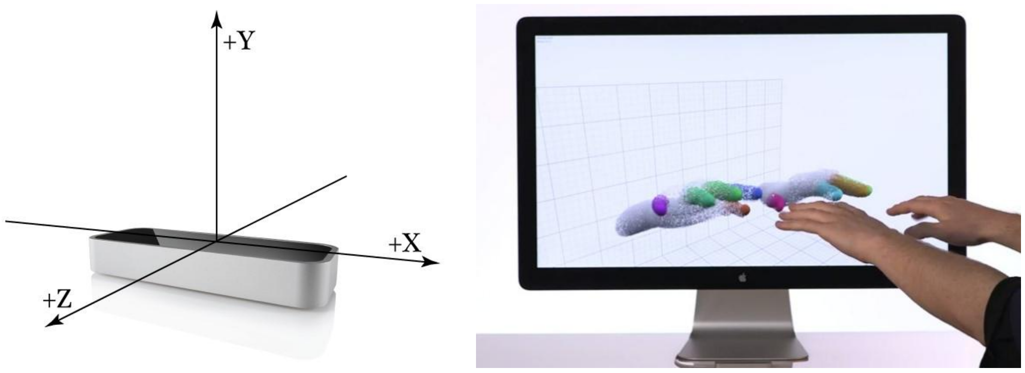


Figure 2-6 (a) The coordinates constructed by the Leap Motion. (b) Reconstructed 3-D hands

Despite the high precision of hand tracking, the Leap Motion is not suitable to be applied to a large-scale HCI due to its restriction of the operation zone. Hence, we want to develop a hand gesture recognition HCI that is not limited to laptops or PCs.

## 2.2 Hand Posture Recognition Technique for Large-scale Touch Panel

The work of our thesis follows the work in [1]. In this section, we introduce what have been done and what we want to improve in the previous work. At first, we introduce the whole system which is a vision-based large-scale touch panel. After that, we describe the main hand posture recognition algorithm on which our thesis is based.

### 2.2.1 System Overview

In [1], they propose a hand posture recognition algorithm for large-scale touch panel. They use a pair of visible-light cameras, installed on the top corners of the board, to detect the touch location [12], as shown in Figure 2-7. Before detecting the touch location, the most important thing is to find the hand in the image, as shown in the flowchart in Figure 2-8. The algorithm is to be discussed later.

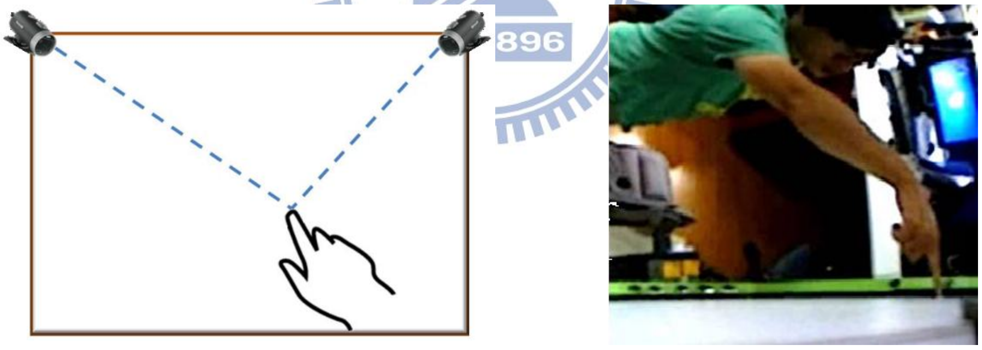


Figure 2-7 (a) Proposed system in [1]. (b) Horizontal camera view [1].

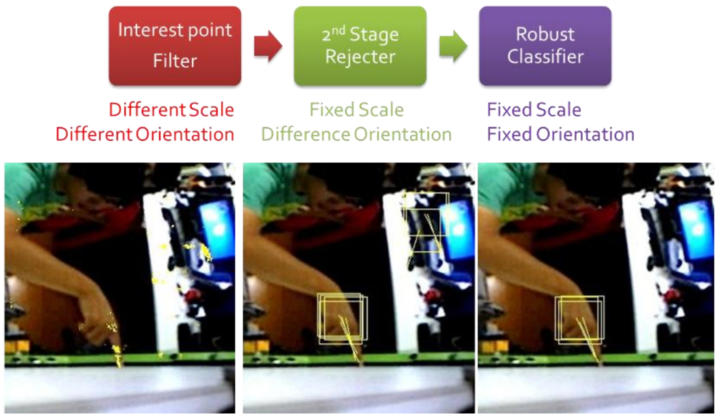


Figure 2-8 Flowchart of the hand posture recognition [1].

### 2.2.2 Algorithm for Hand Detection

In this section, we focus on the hand detection algorithm that are closely related to our work. They use a machine learning technique called "Random Forest [13]", with the idea of randomized pooling area [14], to train classifiers. They use these trained classifiers to recognize hand postures.

Generating a lot of training data is a prerequisite thing for training. However, it is not easy to obtain a large data set. Hence, they use synthesized data for training. Using background removal method, they get the foreground hand images and then merge these hand images with possible background images. However, the merged images may look abnormal due to the sharp edges on the boundaries of the hands. They apply a Gaussian smoothing filter over those merged images to make them look more realistic. In order to detect hands with different orientations, they also rotate the hands in various pre-defined orientations. The above process, as shown in Figure 2-9, can generate thousands of training data, which makes the training of classifiers more reliable.

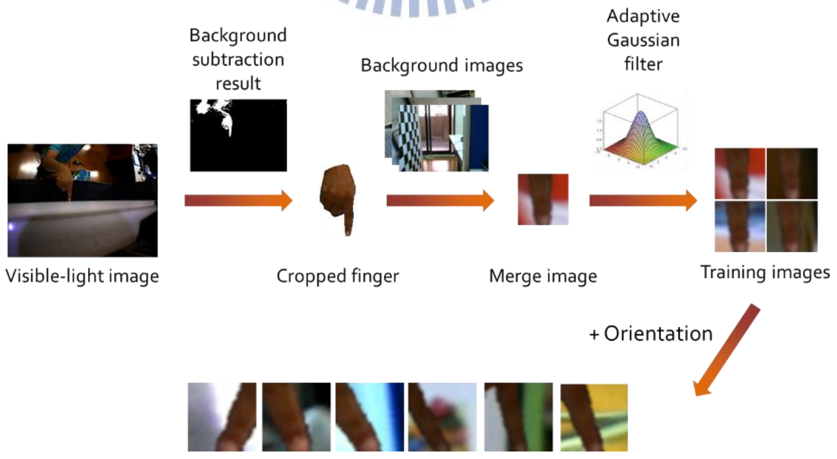
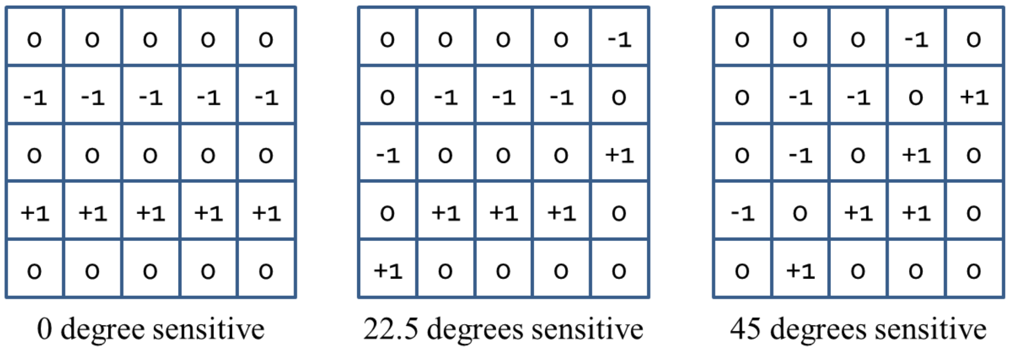


Figure 2-9 System flow of generating training data [1].

How to describe the features of the images is an important issue in computer vision. In [1], they design edge-sensitive filters to generate edge maps, as shown in Figure 2-10. For the purpose of fast computational speed, they use the pre-defined filters to extract edges, rather than the commonly used Histogram of Orientated Gradient (HOG) [15] descriptor.



| | | | | |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| -1 | -1 | -1 | -1 | -1 |
| 0 | 0 | 0 | 0 | 0 |
| +1 | +1 | +1 | +1 | +1 |
| 0 | 0 | 0 | 0 | 0 |

0 degree sensitive

| | | | | |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | -1 |
| 0 | -1 | -1 | -1 | 0 |
| -1 | 0 | 0 | 0 | +1 |
| 0 | +1 | +1 | +1 | 0 |
| +1 | 0 | 0 | 0 | 0 |

22.5 degrees sensitive

| | | | | |
|---|---|---|---|---|
| 0 | 0 | 0 | -1 | 0 |
| 0 | -1 | -1 | 0 | +1 |
| 0 | -1 | 0 | +1 | 0 |
| -1 | 0 | +1 | +1 | 0 |
| 0 | +1 | 0 | 0 | 0 |

45 degrees sensitive

Figure 2-10 The designed edge-sensitive filters [1].

After extracting edges from the images, we need to determine the pooling areas to compute the final descriptors of the images. In previous works, SIFT [16] and HOG [15] use rectangular pooling areas. The work in [17] uses rectangular pooling areas of pyramidal structure so as to gain information of different scales. However, the pre-defined pooling areas may not describe the object precisely. Hence, [18] proposes randomized pooling areas, which may have better performance than pre-defined ones. In the previous work in [1], they propose an algorithm to score pooling areas and pick the most suitable ones for the classifiers. The flowchart is shown in Figure 2-11(a).

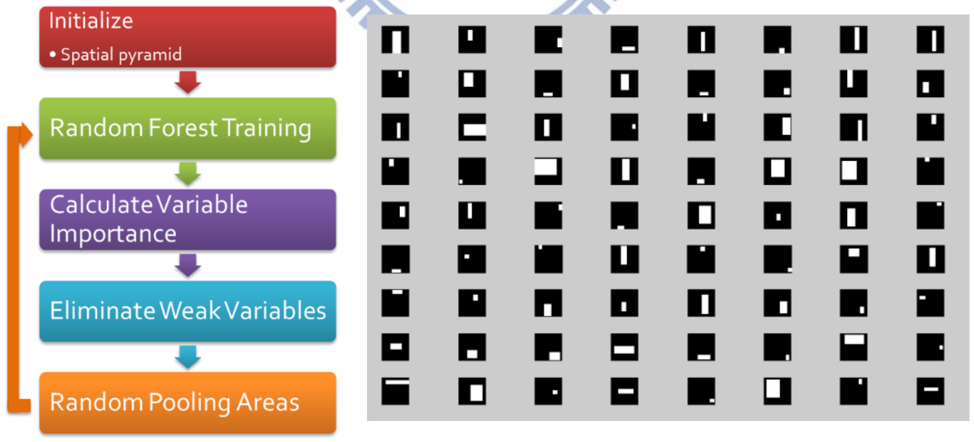


Figure 2-11 (a) Flowchart of generating pooling areas [1]. (b) The generated pooling cells

First, they use two-level spatial pyramid rectangular grids as the pooling areas, together with some randomized rectangular grids. Using these pooling areas to calculate feature vectors, they use Random Forest to train the classifier. In the training process, Random Forest structure can tell us the score of relative importance among variables called “variable importance”. Hence, they use this information to discard the weakest 20-percent pooling areas and regenerate the rectangular girds randomly. After that, they use these new pooling areas to train the Random Forest structure iteratively. After 200 iterations, the remaining pooling areas represent the more important features of the object, as shown in Figure 2-11(b).

As mentioned above, they generate training data with different orientations. They regard these orientations as different classes. Hence, it is actually a multi-class training problem, as shown in Figure 2-12.

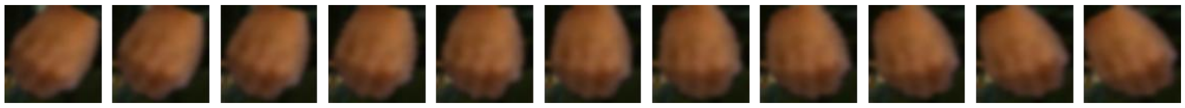


Figure 2-12 Multi-class representation of different orientations [1].

## 2.3 Related Works in Tracking

In this section, we introduce two categories of tracking algorithms: Bayesian tracking approach and tracking-by-detection approach. The former one uses probabilistic model to predict the location of the object and then uses the latest measurement to refine the location. The latter one treats tracking as a binary classification problem and trains an online classifier to find the object.

### 2.3.1 Bayesian Tracking

For the problem of tracking, we need to define two models [19]: system model (dynamic model) and measurement model (observation model), as expressed in Equation 2-1 and 2-2, respectively:

$$\mathbf{x}_k = \boldsymbol{f}_k(\boldsymbol{x}_{k-1}, \boldsymbol{v}_{k-1}) \text{ and} \tag{2-1}$$

$$\mathbf{z}_k = \boldsymbol{h}_k(\boldsymbol{x}_k, n_k), \tag{2-2}$$

where $\{\mathbf{x}_k, k \in N\}$ represents the state sequence, and $k$ is time index. $\{\mathbf{v}_{k-1}, k \in N\}$ is an i.i.d. process noise sequence and $\{\mathbf{n}_k, k \in N\}$ is an i.i.d. measurement noise sequence. $\boldsymbol{f}_k$ and $\boldsymbol{h}_k$

are possibly nonlinear functions. Figure 2-13 illustrates the idea of Bayesian tracking.

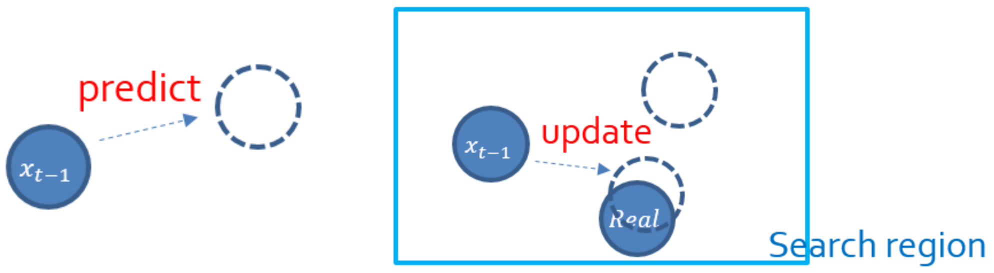


Figure 2-13 Illustration of Bayesian tracking.
(a) Prediction step using the system model. (b) Update step using the measurement model.

From a Bayesian perspective, given the observed data $\mathbf{z}_{1:k}$, we want to calculate the state $\mathbf{x}_k$ at time $k$. It means that we have to construct the pdf $p(\mathbf{x}_k|\mathbf{z}_{1:k})$. The pdf is calculated via two steps: prediction step and update step [19].

$$p(\boldsymbol{x}_k|\boldsymbol{z}_{1:k-1}) = \int p(\boldsymbol{x}_k|\boldsymbol{x}_{k-1})p(\boldsymbol{x}_{k-1}|\boldsymbol{z}_{1:k-1})d\boldsymbol{x}_{k-1} \tag{2-3}$$

Equation 2-3 shows the prior pdf of the state at time $k$, where $p(\boldsymbol{x}_k|\boldsymbol{x}_{k-1})$ is defined by the system model (Equation 2-1). Once the measurement $\mathbf{z}_k$ at time $k$ is available, it is used to update the prior pdf by Equation 2-3.

$$p(\boldsymbol{x}_k|\boldsymbol{z}_{1:k}) = \frac{p(\boldsymbol{z}_k|\boldsymbol{x}_k)p(\boldsymbol{x}_k|\boldsymbol{z}_{1:k-1})}{p(\boldsymbol{z}_k|\boldsymbol{z}_{1:k-1})} \tag{2-4}$$

Equation 2-4 shows the equation of the update step, where

$$p(\boldsymbol{z}_k|\boldsymbol{z}_{1:k-1}) = \int p(\boldsymbol{z}_k|\boldsymbol{x}_k)p(\boldsymbol{x}_k|\boldsymbol{z}_{1:k-1})d\boldsymbol{x}_k \tag{2-5}$$

is the normalizing constant. Here, $p(\boldsymbol{z}_k|\boldsymbol{x}_k)$ is defined by the measurement model (Equation 2-2).

Using Equation 2-3 and Equation 2-4 iteratively, we can form the optimal solution for Bayesian tracking. However, it is a conceptual solution only. In general, it cannot be solved analytically unless for some restrictive cases, such as Kalman filter [20]. We can also use some approximation algorithms to approximate the optimal solution, such as extended Kalman filters [21] and particle filters [22].

### 2.3.2 Tracking by Detection

Recently, many people consider tracking not just as posterior density estimation. Instead, they use a detection algorithm to help the tracking algorithm to discriminate the object from the background. There are two remarkable papers which inspire our work: "Ensemble Tracking [23]" and "Tracking-Learning-Detection [24]". Typically, a traditional particle filter algorithm may be restricted to working with histograms. However, for high-dimensional feature space, it requires large memory storage. The tracking-learning-detection method is no longer restricted to these limitations. Moreover, the histogram can be updated over time, rather than using the pre-defined one all the time. In the next paragraph, we will briefly introduce the idea of "Ensemble Tracking".

In [23], Avidan treats tracking as a binary classification problem. They train an online Adaboost [25] classifier composed of weak classifiers and a strong classifier to distinguish the object from the background. Each pixel in the image is regarded as a training example. Weak classifiers represent the hyperplanes in the feature space. The strong classifier generates a confidence map for the image, as shown in Figure 2-14.

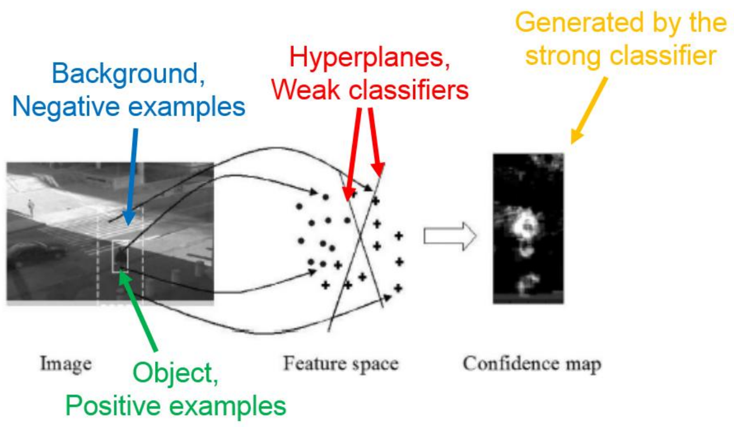

Generated by the strong classifier
Background, Negative examples
Hyperplanes, Weak classifiers
Image
Object, Positive examples
Feature space
Confidence map

Figure 2-14 Idea of ensemble tracking

For each video frame, they use the strong classifier to create a confidence map and then use the mean-shift algorithm [26] to find the peak value of the confidence map. Based on the peak, the new bounding box of the object is obtained. Next, they label the pixels within the bounding box as positive examples while the others as negative examples. They test all the weak classifiers and keep the *K* best ones. Finally, they train new *T* - *K* classifiers and update the strong classifier, where *T* is the total number of weak classifiers. The above iterative process gives a robust and low-computational cost tracker.

# Chapter 3  Proposed Algorithm

In this chapter, we explain our algorithms in detail. First, we show our system architecture and explain the design consideration of our system in Section 3.1. In Section 3.2, we provide a novel and simple calibration method to get the initial position of the hand and the information between image coordinates and projected-board coordinates. After that, in Section 3.3, we present the tracking algorithm which tracks the hand in an image and predict the hand location. Next, Section 3.4 explains the hand detection algorithm which we use to determine the hand gesture in our system. After that, we provide a combination of tracking and detection algorithm to complement each other to get a better performance in Section 3.5. In Section 3.6, we define some intuitive and useful continuous gestures and present how to recognize these gestures in order to replace the mouse. Figure 3-1 shows the flowchart of our algorithm.

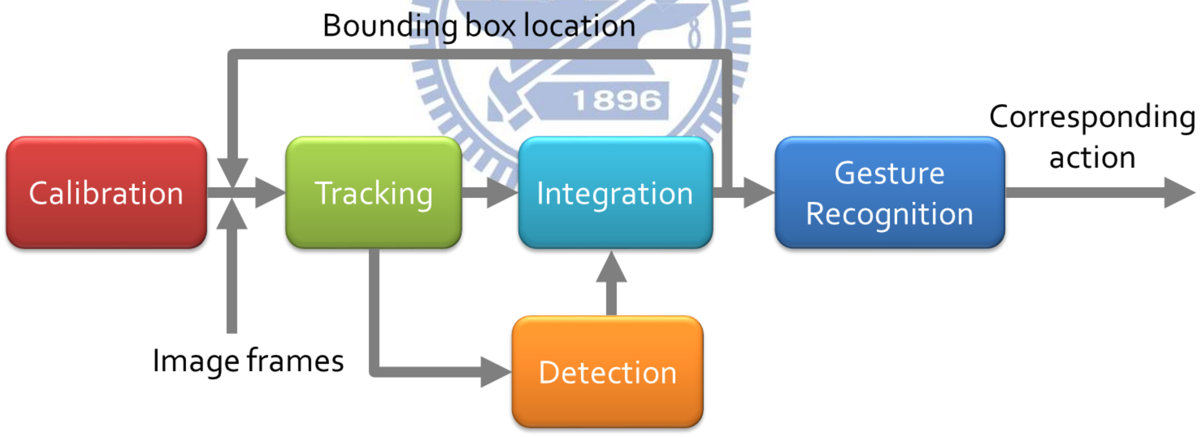


Figure 3-1 Flowchart of our algorithm.

## 3.1  System Architecture

Derived from [1], our system is based on a 70-inch projection-based surface (shown in Figure 3-2(a)) and a projector. The main purpose of our system is that we want to change from the touch panel system into a remote-control system. In the proposed system, the user can interact with this remote-control human-computer interface (HCI) about 1.5 meters to 2.0

meters in front of the panel. Our HCI is designed mainly for the presentation, such as giving a lecture or meeting. Hence, the users should not occlude the view of the board, but have to stand sideways instead, as shown in Figure 3-2(c).

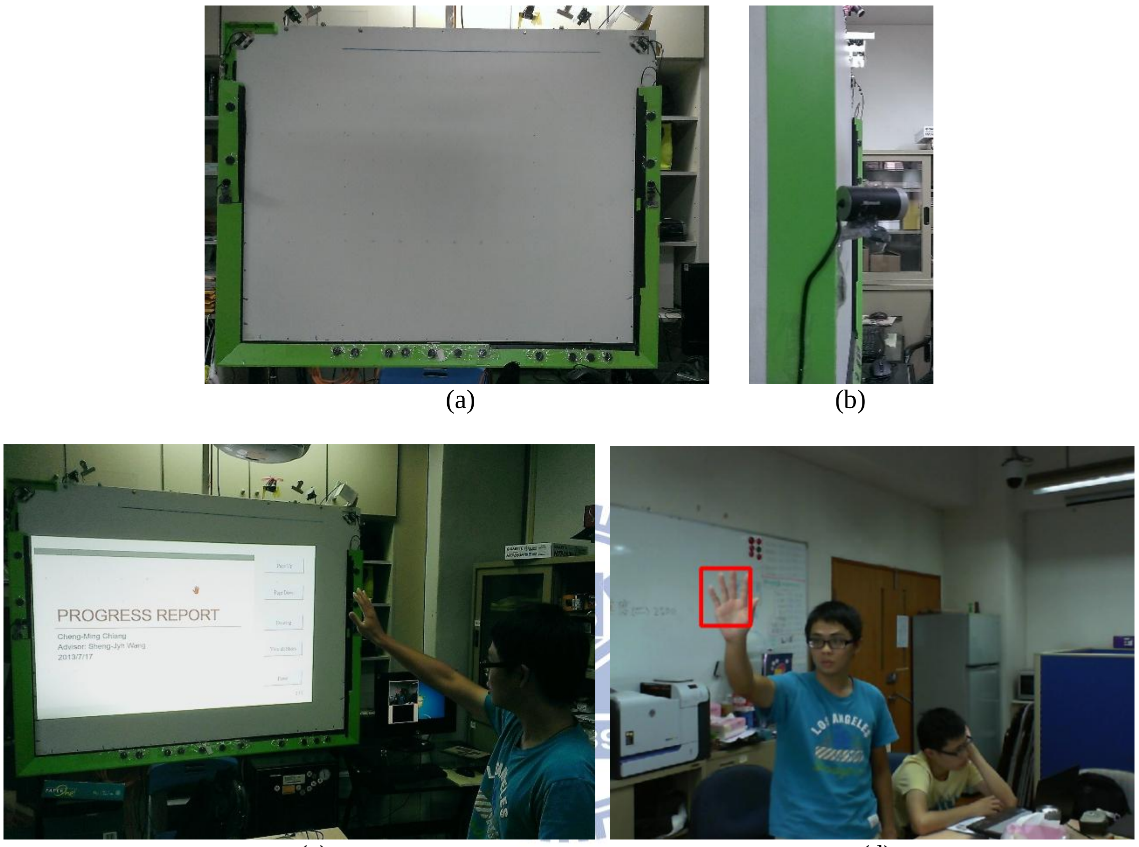


Figure 3-2 System architecture.

(a) Our large-scale projection-based panel. (b) Camera installed on the left side (left side view).

(c) The look of the system. (d) The image captured by the camera.

To keep our system as simple as possible, unlike the special device needed as mentioned in Section 2.1, we use only one visible-light camera. As shown in Figure 3-2(b), the camera is mounted on the left side of the board, with the principal axis pointing toward the front of the board.

With the use of a single visible-light camera and the requirement that the operation is no longer limited to touching on the board, we have several challenges. One is that we need to find the hand location in the cluttered background in image coordinates and then project the location onto the projected board. The other is that there are not only in-plane rotation but also out-ofplane rotation and scale changes. Besides, we have to recognize different gestures in this situation. In the subsequent sections, we will present a process to solve these problems.

## 3.2 Calibration

The ultimate goal of our system is to replace mouse by hand. Hence, the user can use his/her hand to control the cursor. For the detected hand in the image (Figure 3-3(a)), we must find its corresponding position on the projected board, as shown in Figure 3-2(c). Because of using only one visible-light camera, we do not have the depth information and cannot build the 3-D world coordinates. Hence, we regard this as a 2-D to 2-D mapping problem and we design a simple calibration process to solve this problem.

First, after the user standing in the operation zone, the user points his/her hand toward the specified rectangle, as show in Figure 3-3. Once the gesture "Five" is detected by the system, the calibration will start automatically. The hand detection algorithm will be described in detail in Section 3.4. Next, the system will use a tracking algorithm described in Section 3.3 to track the hand. After that, the user moves the hand to point toward the four corners of the projected screen, upper left, lower left, lower right and upper right, as shown in Figure 3-4. As moving the hand to point toward the four corners, we will get a trajectory map of the hand (Figure 3-5(b)). The trajectory map has a shape like a rectangle which indicates the operation zone of the hand seen from the camera. Finally, the user uses the gesture "Zero" to finish the calibration process, as shown in Figure 3-3(b).

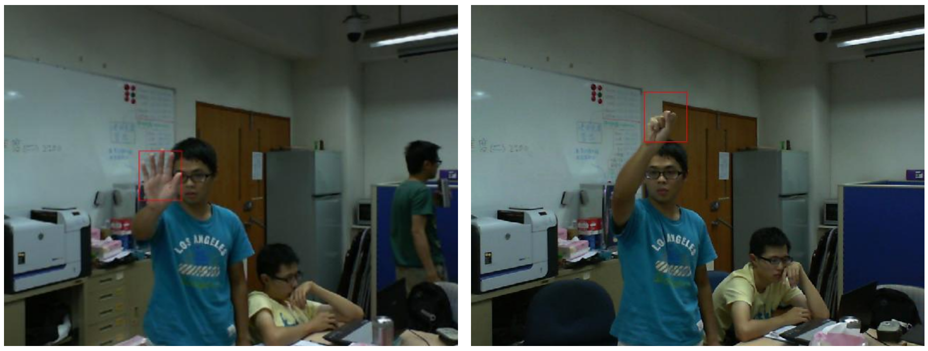


Figure 3-3 Calibration process for the start and the end. (a) Start. (b) End.

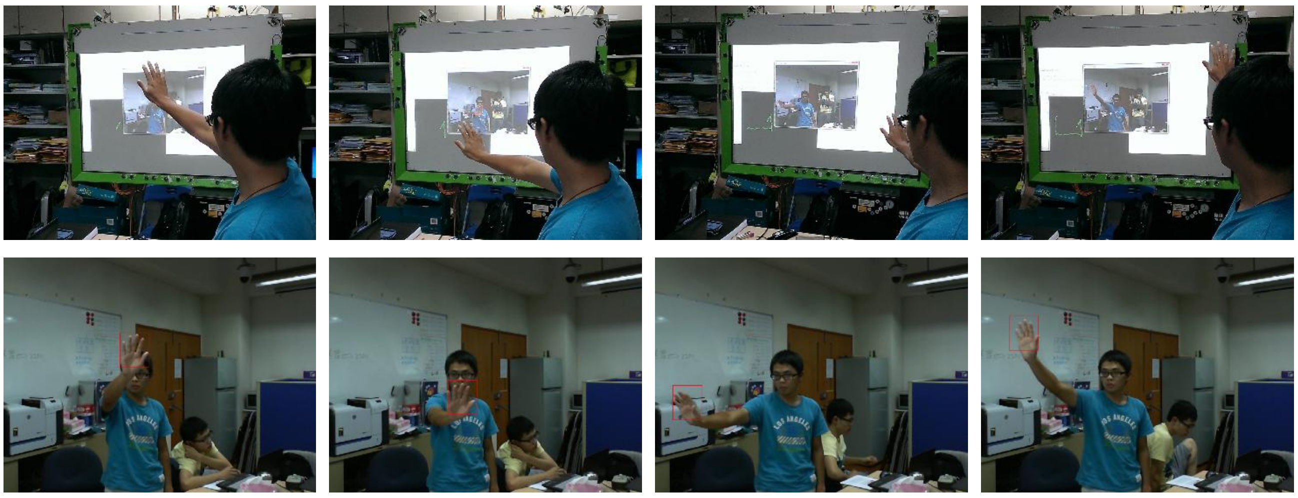


Figure 3-4 Calibration process for the four corners

(a)(b)(c)(d) The hand positions as pointing toward the four corners.

(e)(f)(g)(h) The corresponding image views.

The main purpose of our calibration process is that we want to find the relationship between this operation zone in the image and the projected screen; that is, the mapping from the rectangle of the operation zone (Figure 3-5(b)) to the rectangle of the projected screen (Figure 3-2(c)). We define the ratio of the projected screen to the operation zone as

$$ratio_x = width_{projected\ screen}/width_{operating\ region}, \text{ and} \tag{3-1}$$

$$ratio_y = height_{projected\ screen}/height_{operating\ region}. \tag{3-2}$$

We also define $x_{min}, y_{min}, x_{max}$ and $y_{max}$ as the position of the left, top, right and bottom boundaries of the operation zone respectively. Hence, for each detected hand location $(x_{image}, y_{image})$ in the image coordinates, we can find the corresponding location $(x_{screen}, y_{screen})$ in the projected screen coordinates using the following equations:

$$x_{screen} = \left(x_{image} - x_{min}\right) \cdot ratio_x, \text{ and} \tag{3-3}$$

$$y_{screen} = \left(y_{image} - y_{min}\right) \cdot ratio_y. \tag{3-4}$$

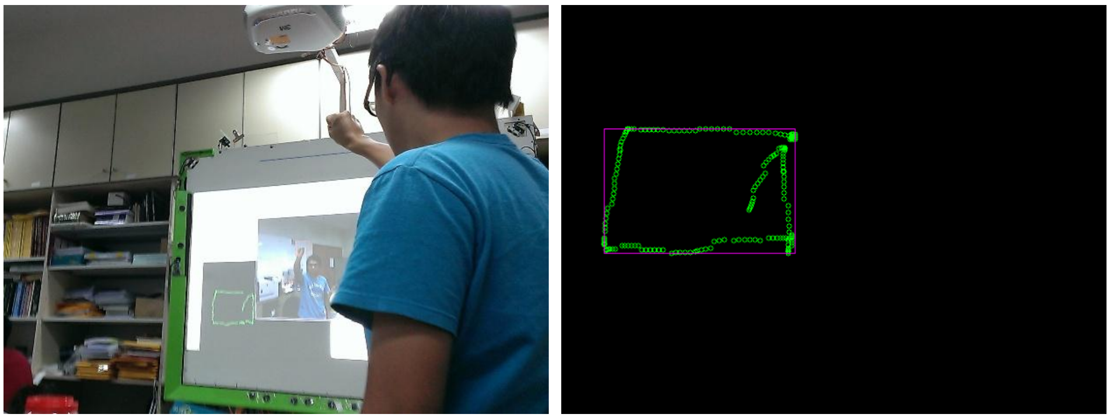


Figure 3-5 (a) Use “Zero” to end calibration process. (b) The trajectory map of the calibration process.

## 3.3 Hand Tracking

Our goal is to develop a hand gesture recognition algorithm for human-computer interface. We want to modify the previous work [1] for large-scale touch panel into a remote-control system. One main challenge for the human-computer interface is the processing time. Roughly speaking, the frame rate of the algorithm has to be at least 10 to 15 frames per second to reach real-time performance. However, the frame rate in the previous work [1] is only about 2~4 frames/sec. The most time consuming part is the sliding window process for hand detection. If considering temporal information, we can reduce a lot of time on searching the best window. Hence, we introduce a tracking algorithm into our hand gesture recognition system. The related works for tracking have been described previously in Section 2.3. Here, we adopt the tracking-by-detection method. The idea of our tracking algorithm is originated from [24] proposed by Z. Kalal. However, in [24], it has several limitations. One is that it cannot deal with non-convex objects. The other one is that the bounding box of the object must not contain the background parts. Otherwise, the learning component will learn the background. Hence, we modify this tracking algorithm in order to deal with non-convex objects, such as the different views of the hand images.

We will introduce the proposed tracking method in the succeeding sections. Section 3.3.1 introduces how we choose the features for tracking. With these features, we describe the method to predict the bounding box of the object in Section 3.3.2.

### 3.3.1 Features for Tracking

In [24] and [27], Z. Kalal et al use sparse grid points as the feature points to represent the object. They use the sparse feature points at frame $I_t$ to predict the feature points at frame $I_{t+1}$. These feature points at the consecutive frames are used to predict the object location at frame $I_{t+1}$, as shown in Figure 3-6. The method to predict the bounding box is described in Section 3.3.2. There are equally spacing sampled grid points within the bounding box at frame $I_t$, as shown in Figure 3-7(a). Obviously, these grid points are not reliable enough, so they propose an algorithm called "Median Flow" to get reliable points. First, they use Lucas-Kanade tracker [28] to track these grid points in the subsequent frame $I_{t+1}$. After that, they estimate the errors of the motion flow between $I_t$ and $I_{t+1}$ and filter out unreliable points. There are two measurements they use: Forward-Backward (FB) error and Normalized Correlation Coefficient (NCC). For the FB error, "forward" means that they track the points from $I_t$ to $I_{t+1}$. On the other hand, tracking those points from $I_{t+1}$ to $I_t$ means "backward". The distances between the original points and the tracked points in frame $I_t$ represent the FB error. Roughly speaking, the FB error of a reliable point must be small. Hence, they discard the point with FB error larger than the median of all FB errors. For the other measurement, they extract a patch near the corresponding pair of points in frame $I_t$ and frame $I_{t+1}$. The patches near both points need to be similar. NCC measures the similarity between patches:

$$NCC = \frac{1}{n}\sum_{x,y}\frac{(I(x,y)-\bar{I})(T(x,y)-\bar{T})}{\sigma_I\sigma_T}, \tag{3-5}$$

where $\bar{I}, \bar{T}, \sigma_I$ and $\sigma_T$ are the average and standard deviation of the patches $I(x,y)$ and $T(x,y)$, respectively. If the point has similarity measurement smaller than the median of all similarity measurements, this point is regarded as an unreliable point and is discarded. After the two filtering processes, the remaining points are used to estimate the new position of the bounding box, as shown in Figure 3-7(a)(b).

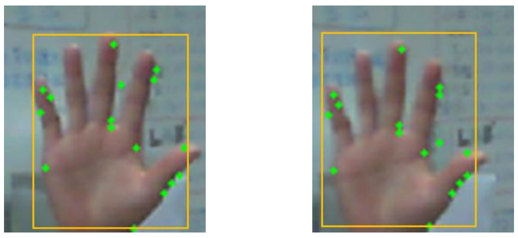


Figure 3-6 The feature points and the bounding box at (a) frame *t*. (b) frame *t+1*.

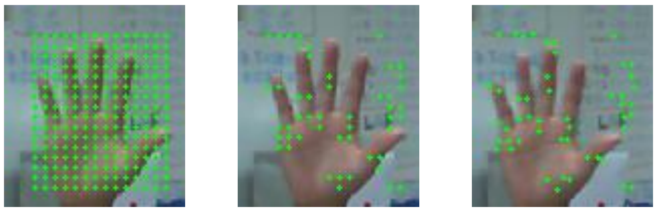
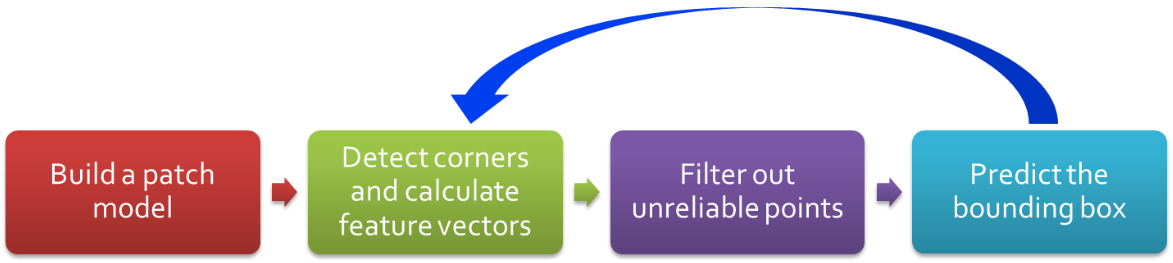


Figure 3-7 The feature points obtained using Kalal's algorithm.

(a) Grid points. (b) Remaining points at frame *t*. (c) Remaining points at frame *t+1*.

In their work, the major problem is that the some reliable points may belong to the background unless the bounding box contains only the object without background. However, in our case, the hand is a non-convex and non-rigid object. The rectangular bounding box may not contain the object only. Hence, we propose a method which finds the reliable feature points and avoids the interference of the background. Figure 3-8 shows the flowchart of our tracking algorithm.


Build a patch model
Detect corners and calculate feature vectors
Filter out unreliable points
Predict the bounding box
Figure 3-8 Flowchart of our tracking algorithm.

In the calibration process, we get the hand image and we build a patch model for this hand image. Figure 3-9 shows the process of building the patch model. We use a famous corner detector [29] to find the feature points instead of grid points. For each detected corner, we extract a $9 \times 9$ patch centered on the detected point. We describe the patches using the edge information which will be discussed in detail in Section 3.4. Hence, each detected corner is represented by a feature vector. The patch model stores those feature vectors. Although there are some background points in the patch model, we use a motion estimation method to filter out these background points.

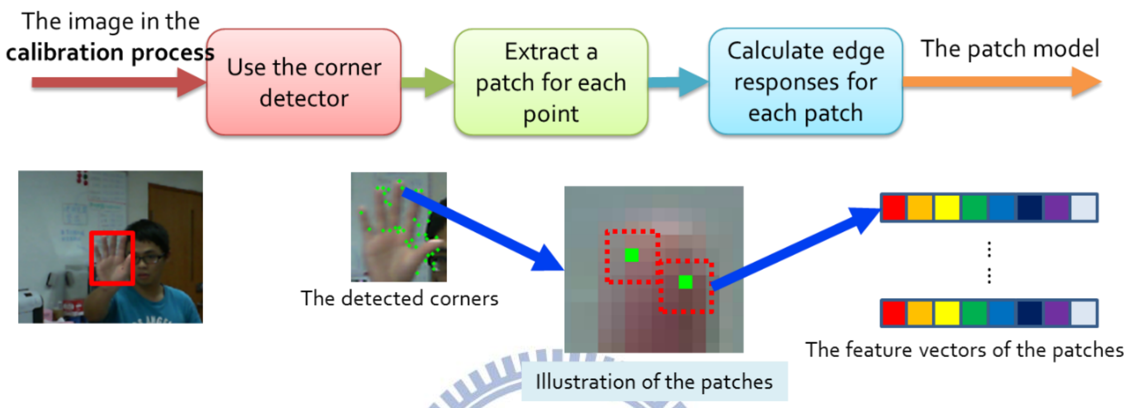

The image in the **calibration process**
Use the corner detector
Extract a patch for each point
Calculate edge responses for each patch
The patch model
The detected corners
Illustration of the patches
The feature vectors of the patches

Figure 3-9 Illustration of building the patch model.

Assume that we have the bounding box at frame *t*. We want to find the meaningful feature points at frame *t* and frame *t+1* and then predict the bounding box at frame *t+1*. For the frame *t*, we first find the corners using corner detector (Figure 3-10(a)). With the detected corners, we use a 3-stage filtering process to filter out unreliable points and background points. The first stage filter out non-hand points. We extract a patch for each detected corners and then compute the edge responses of the extracted patches. We calculate the correlation coefficient $corr$ between the feature vector of each detected patch and the feature vectors of all the patches in the model. The correlation coefficient is defined as

$$corr = \frac{\sum_{i=1}^{n}(x_i-\bar{x})(y_i-\bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i-\bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i-\bar{y})^2}}. \quad (3\text{-}6)$$

We find the maximum correlation between the detected patch and all the patches in the model. If the maximum correlation is smaller than a threshold, the corresponding detected corner is discarded (Figure 3-10(b)). The second stage filter out the deformed points. We filter out the points with lower similarity (NCC) as mentioned previously (Figure 3-10(c)). Note that we set the threshold of similarity to 0.85, instead of the median in [27]. The final stage filters out the background points. We use an optical flow method proposed by Horn and Schunck [30] to find the moving areas, as shown in Figure 3-11. We also use a morphological opening to reduce noise. Finally, we retain the feature points which lie in the moving region (Figure 3-10(d)). After these process, the remaining points are used to estimate the new position of the bounding box. Figure 3-6 shows the remaining points at frame *t* and frame *t+1*.

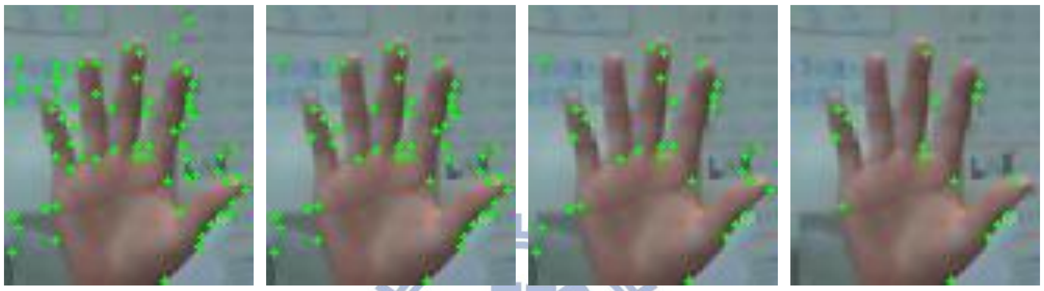


Figure 3-10 Filtering process of the feature points.

(a) Detected corners. (b) Filtered by correlation. (c) Filtered by similarity. (d) Filtered by optical flow.

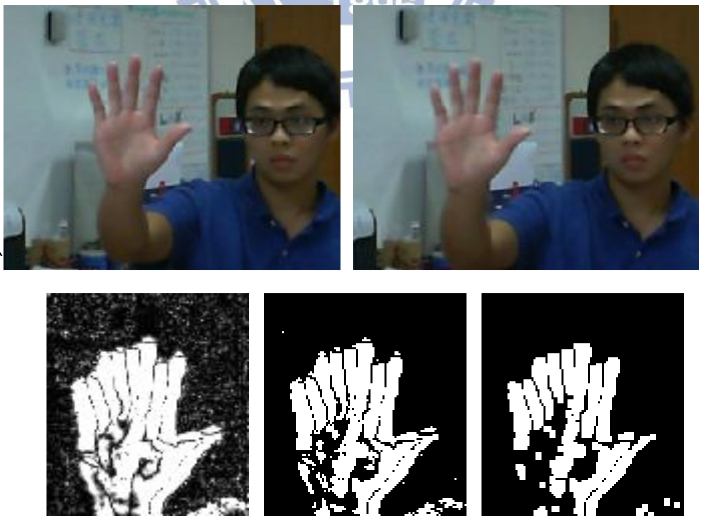


Figure 3-11 Optical flow process.

(a) The image at frame *t*. (b) The image at frame *t+1*.

(c) Optical flow. (d) After thresholding. (e) After morphological operation.

### 3.3.2 Prediction of the Bounding Box

With the reliable feature points (Figure 3-6), we use the Median Flow method [27] to estimate the bounding box. Using the Lucas-Kanade tracker, we can get the motion vector between two consecutive frames. For each feature point $(x_i, y_i)$ in the previous frame, we have the predicted points $(\hat{x}_i, \hat{y}_i)$ in the next frame. $(d_{ix}, d_{iy}) = (\hat{x}_i - x_i, \hat{y}_i - y_i)$ denotes the motion vector, as show in Figure 3-12. For the center of the previous bounding box, $(x_{old}, y_{old})$, the center of the new bounding box, $(x_{new}, y_{new})$, is estimated by

$$x_{new} = x_{old} + median(d_{1x}, d_{2x}, \dots, d_{nx}) \text{ and} \tag{3-7}$$

$$y_{new} = y_{old} + median(d_{1y}, d_{2y}, \dots, d_{ny}), \tag{3-8}$$

where $n$ is the number of feature points.

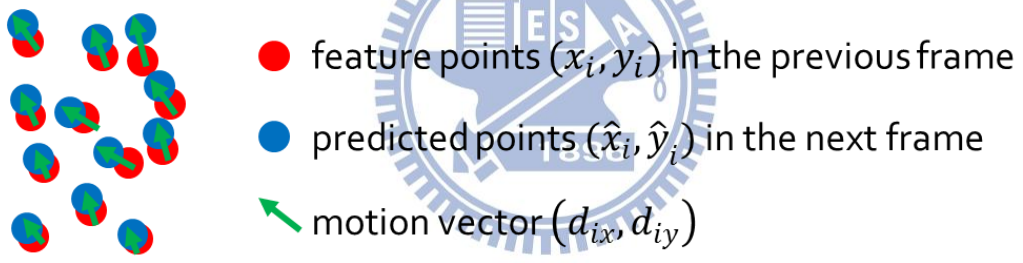

feature points $(x_i, y_i)$ in the previous frame
predicted points $(\hat{x}_i, \hat{y}_i)$ in the next frame
motion vector $(d_{ix}, d_{iy})$

Figure 3-12 Illustration of feature points.

In [27], they also provide a method to resize the bounding box. However in our case, we track the hand with different gestures and the shape may be slightly changed. This resizing strategy is not robust to different and non-rigid objects. To deal with this problem, we use a hand detection algorithm to find the proper size of the hand which will be discussed in Section 3.4.

## 3.4 Hand Detection and Classification

In this section, we propose an algorithm to detect and classify the hands based on the work introduced in Section 2.2. In our algorithm, we train a "Five-Zero" classifier which can classify two types of the hand gestures, “Five” and “Zero”, as shown in Figure 3-13. We also train two detectors, ”Five-Background” and “Zero-Background”, which are able to determine the presence of the two gestures in the image.

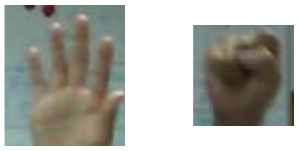


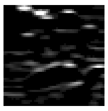
Figure 3-13 Hand gestures (a) Five. (b) Zero.

The detection algorithm is composed of two parts. Section 3.4.1 describes the feature extraction and Random Forest training process. After that, with the pre-trained model, we introduce real-time hand gesture detection and classification in Section 3.4.2.

### 3.4.1 Features Extraction and Random Forest Training

As mentioned in Section 2.2, the previous work convolve the image with pre-defined edge filters to generate the edge maps. They quantize 180 degrees into 8 bins. The step size is 22.5° and there are 8 edge filters. However, this information may not be sufficient to describe an object. Taking Figure 3-13(b) as an example, both the right side and the left side of the fist have vertical edges. Using 180°-orientation edge filters, we cannot discriminate the difference between these two edges even though the two edges represent different orientations. To solve this problem, we use 360°-orientation edge filters rather than 180° to generate the edge maps. We quantize 360 degrees into 8 bins. The step size is 45°. Ideally, using 16 bins with 22.5° step size is better than using 8 bins with 45° step size. However, with the double edge filters and edge maps, the processing time is also doubled. To keep the processing time as short as possible, we use 45° step size filters instead of 22.5°. The final edge filters used in our algorithm are $(45 \times k)°, k = 0,1, \ldots ,7$ sensitive. The edge filters and the result of the edges maps are shown in Figure 3-14. Certainly, 22.5°-sensitive filters are better than 45°-sensitive filters, but the difference in the detection accuracy is not too large. Later in Chapter 4, we will make a detailed comparison.

| | | | | |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| -1 | -1 | -1 | -1 | -1 |
| 0 | 0 | 0 | 0 | 0 |
| +1 | +1 | +1 | +1 | +1 |
| 0 | 0 | 0 | 0 | 0 |



(a) 0°-sensitive.

| | | | | |
|---|---|---|---|---|
| 0 | 0 | 0 | -1 | 0 |
| 0 | -1 | -1 | 0 | +1 |
| 0 | -1 | 0 | +1 | 0 |
| -1 | 0 | +1 | +1 | 0 |
| 0 | +1 | 0 | 0 | 0 |

(b) 45°-sensitive.

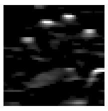
| | | | | |
|---|---|---|---|---|
| 0 | -1 | 0 | +1 | 0 |
| 0 | -1 | 0 | +1 | 0 |
| 0 | -1 | 0 | +1 | 0 |
| 0 | -1 | 0 | +1 | 0 |
| 0 | -1 | 0 | +1 | 0 |

(c) 90°-sensitive.

| | | | | |
|---|---|---|---|---|
| 0 | +1 | 0 | 0 | 0 |
| -1 | 0 | +1 | +1 | 0 |
| 0 | -1 | 0 | +1 | 0 |
| 0 | -1 | -1 | 0 | +1 |
| 0 | 0 | 0 | -1 | 0 |

(d) 135°-sensitive.

| | | | | |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| +1 | +1 | +1 | +1 | +1 |
| 0 | 0 | 0 | 0 | 0 |
| -1 | -1 | -1 | -1 | -1 |
| 0 | 0 | 0 | 0 | 0 |

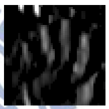


(e) 180°-sensitive.

| | | | | |
|---|---|---|---|---|
| 0 | 0 | 0 | +1 | 0 |
| 0 | +1 | +1 | 0 | -1 |
| 0 | +1 | 0 | -1 | 0 |
| +1 | 0 | -1 | -1 | 0 |
| 0 | -1 | 0 | 0 | 0 |

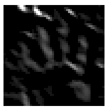
(f) 225°-sensitive.

| | | | | |
|---|---|---|---|---|
| 0 | +1 | 0 | -1 | 0 |
| 0 | +1 | 0 | -1 | 0 |
| 0 | +1 | 0 | -1 | 0 |
| 0 | +1 | 0 | -1 | 0 |
| 0 | +1 | 0 | -1 | 0 |



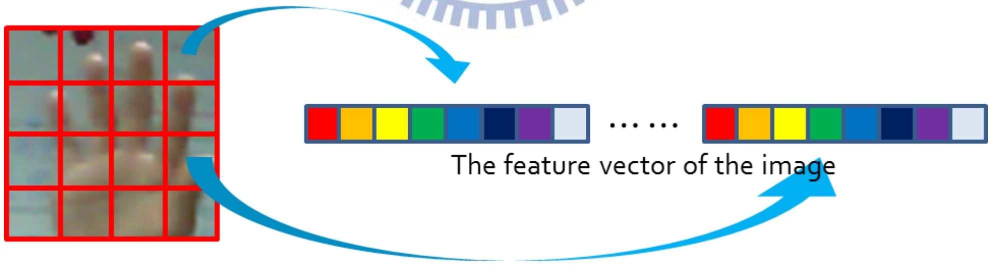
(g) 270°-sensitive.

| | | | | |
|---|---|---|---|---|
| 0 | -1 | 0 | 0 | 0 |
| +1 | 0 | -1 | -1 | 0 |
| 0 | +1 | 0 | -1 | 0 |
| 0 | +1 | +1 | 0 | -1 |
| 0 | 0 | 0 | +1 | 0 |

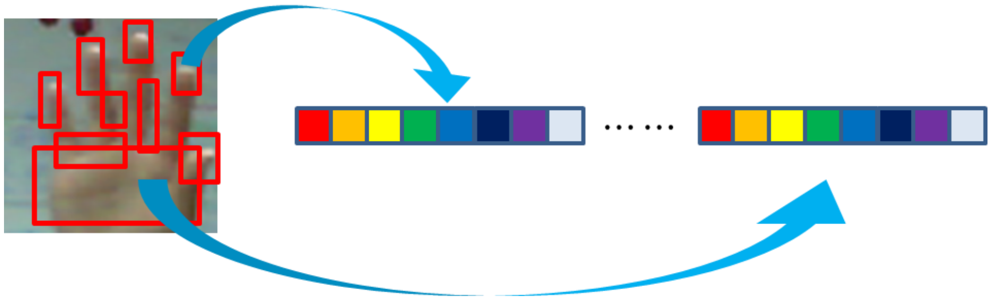


(h) 315°-sensitive.

Figure 3-14 The edge filters and the corresponding edge maps for the image "Five".



(a) Illustration of the grid pooling cells.



(b) Illustration of the randomized pooling cells.

Figure 3-15 Comparison between traditional grid cells and randomized pooling cells.

After generating the edge maps, we use the structure of randomized pooling cells to calculate feature vectors. Figure 3-15(a) illustrates the traditional grid cells. The image is partitioned into equally spacing grid cells. After that, we extract the edge information in each cell. The final feature vector of the image is concatenated by the feature vectors in all the grid cells. Figure 3-15(b) shows the idea of randomized pooling cells. The feature vector of the image is obtained by extracting the edge information in the randomized pooling cells. Ideally, the randomized pooling cells convey more meaningful information than the traditional grid cells. We show the flowchart of the overall training algorithm for the randomized pooling cells in Figure 3-16.

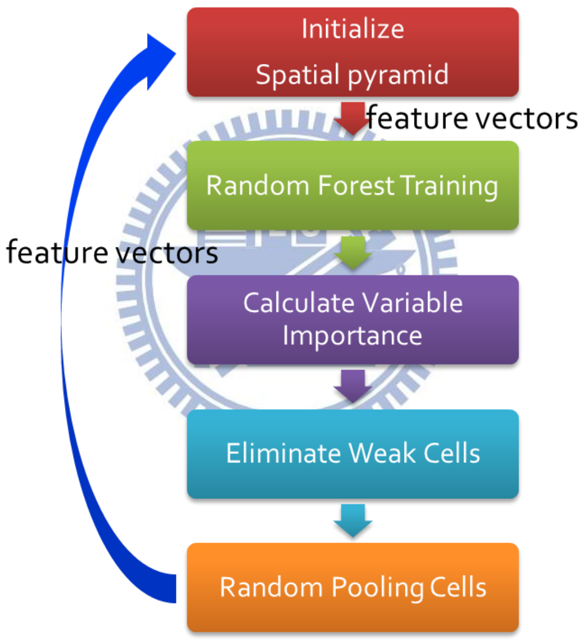


Figure 3-16 Flowchart of generating pooling areas.

The randomized pooling cells are trained in an iterative process. Initially, we use a three-level spatial pyramid to initialize the pooling cells. In addition to the spatial pyramid cells, we also generate pooling cells randomly. We use 120 pooling cells in our algorithm. We add some restrictions on the randomized pooling cells. One is the size of the cells. In [1], they make a restriction that

$$width_{cell} > 0.1 \times width_{image} \text{ and} \tag{3-9}$$

$$height_{cell} > 0.1 \times height_{image}. \tag{3-9}$$

We find that this threshold may be too small. For example, the hand image in our system is about $50 \times 50$, which means the threshold is 5 pixels. However, the object may be a little shifted in the location in different images. The 5-pixel wide cell may contain the edge information in one image, but may contain no information in another image. Hence, we loosen the restriction to 9 pixels so that we can solve the shifting problem. Figure 3-17 illustrates the edge information in a pixel-wise view. The other restriction is on the similarity. For the randomly generated pooling cells, as seen in Figure 2-11(b), some cells are very similar. In this instance, it seems useless to generate so many cells. It may also overemphasize some specific features. Hence, in the process of randomly generating pooling cells, if the generated cell is similar to the existing cells, we will re-generate the cell until no similarity.

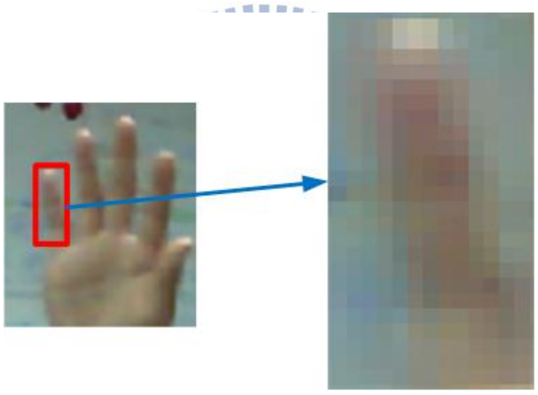


Figure 3-17 Pixel-wise view of the hand.

With these pooling cells, we can calculate feature vectors as illustrated in Figure 3-15(b). We use the integral image [31] to speed up. With integral images, the computations to extract the edge responses for the cells is just simple additions and subtractions. Here, we extract 8 edge responses for each cell. Hence, the dimension of the feature vector for each image is $8 \times 120 = 960$, as shown in Figure 3-18. Note that we randomly pick 3 cells as a group in order to deal with the normalization problem as described in [1]. We normalize the edge responses in each group. Next, the feature vectors of all the training images are fed into Random Forest [13] to train a forest model. In the training process, Random Forest is able to calculate the relative importance of each feature variable. We sum up the importance scores in each group (24 feature variables). By comparing the scores of the groups, we can determine which groups of the pooling cells are important. The important cells of the groups are retained, while the other cells of the groups are eliminated. The eliminated cells are re-generated randomly with the restriction mentioned above. With these new cells, we can calculate the feature vectors using the new cells. After that, the feature vectors are fed into Random Forest again. After many iterations, we can get the final forest model and the randomized pooling cells. Figure 3-19 shows the half pooling cells for the "Five-Background" detector. As expected, there are many long pooling cells which look like fingers.

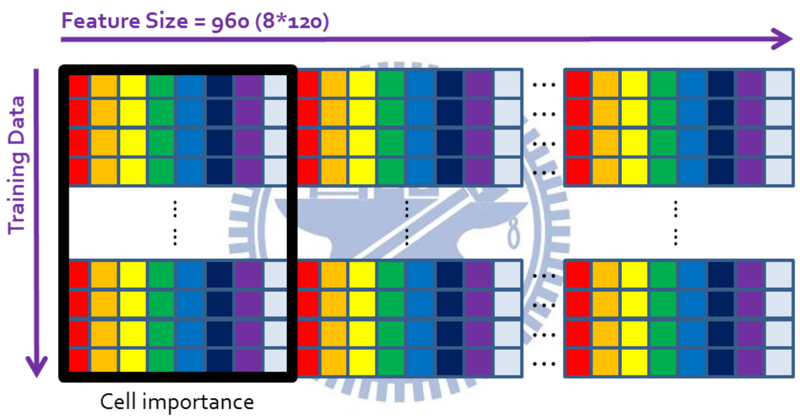


Figure 3-18 Illustration of the feature vectors and variable importance.

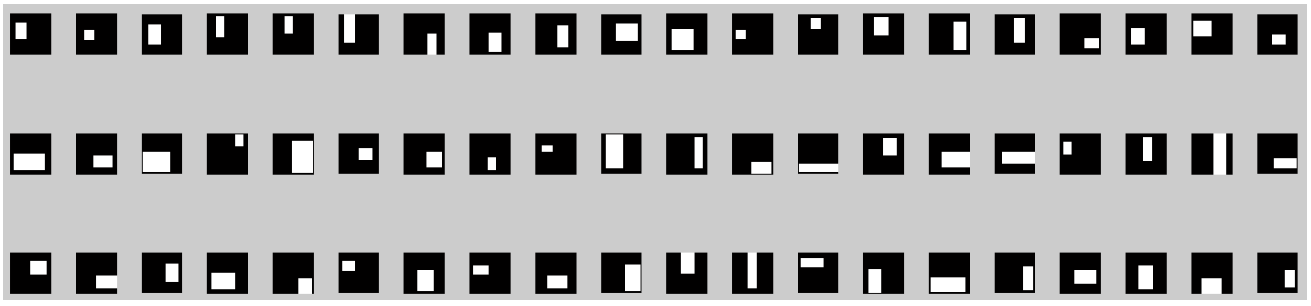


Figure 3-19 Pooling cells of the "Five-Background" detector.

In our system, we train two detectors "Five-Background" and "Zero-Background". They can discriminate between the specified gestures and backgrounds. We also train a "Five-Zero" classifier, which can recognize the gestures. Training data is very important in many machine learning techniques. We do not use the synthesized training images for the gestures “Five” and “Zero” in our algorithm. The appearance may be somewhat different between the synthesized images and the realistic images, as shown in Figure 3-20. The training images that we use are all realistic images. The realistic gesture images are obtained using the tracking algorithm. We track the hand and crop the image of the bounding box. The training images of the “Background” are randomly sampled from the possible background images. The size of the “Zero” images is $50 \times 50$. The size of the “Five” images is $70 \times 60$. Before calculating the feature vectors, we resize all the images to the same size $50 \times 50$.

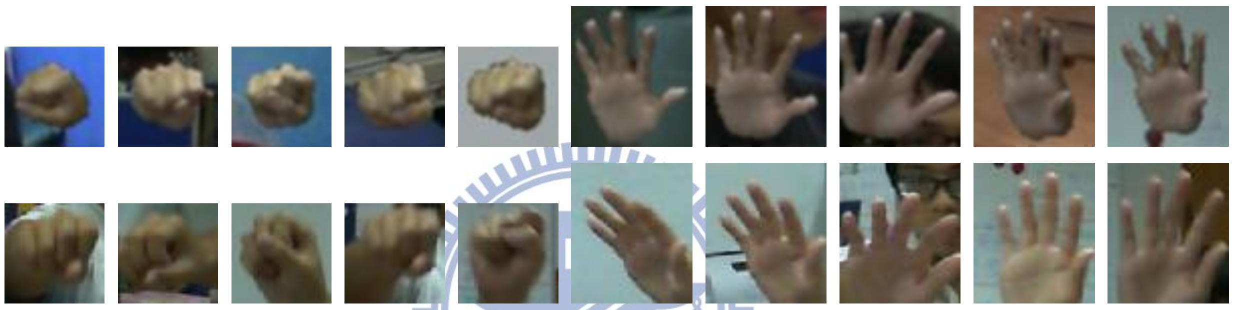


Figure 3-20 Comparison between the synthesized images and the realistic images.

The three cases are all considered as binary classification problem. The ratio of the positive training data to the negative training data is 1:1. The numbers of training images of the gesture “Five”, “Zero” and backgrounds are all 38669. As shown in Figure 3-18, although the viewpoint change of the hand images is wide, our algorithm can still well classify the gestures.

The important parameters in Random Forest training are the number of trees in the forest, the depth of a tree, and the minimum sample counts in a tree. In our algorithm, we use 70 trees in the forest. The depth and minimum sample counts in a tree are 30 and 30, respectively. We will make a comparison and discuss the influence of the parameters in Chapter 4.

### 3.4.2 Real-Time Hand Detection and Classification

Once the classifiers are trained, we use these models to do the task of detection and classification. For the testing stage of Random Forest, the feature vectors are passed to every trees. In the nodes of the tree, it is just a simple comparison with the threshold. Each tree gives a posterior of the classification result. The final result is obtained by averaging all the posteriors. Hence, the classification speed is fast.

Scaling is an important issue in detection. In conventional methods, they resize the candidate testing image at every candidate position and use different sizes of the image to find the best one. However, it is a time consuming task and may be impractical for real-time applications. Since we have trained the pooling cells in the training stage, we resize these pooling cells in advance instead of resizing the testing images, as shown in Figure 3-21. As a result, we test the pooling cells with different sizes to find the most appropriate one. This is a more efficient way to deal with the scaling issue.

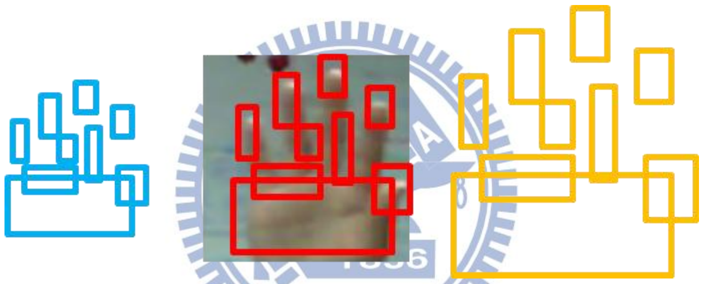


Figure 3-21 Illustration of resizing the pooling cells.

## 3.5 Integration of Tracking and Detection

In this section, we propose an integration method that combines the tracking and detection algorithm to form a robust system. There are two major problems that we are interested in. One is the presence of the hand. The other is the precise location of the hand. The method of the integrator is summarized in Algorithm 1.

For the first issue, we can determine the presence of the hand in every frame. For every image frame, we use the detectors to detect the presence of the hand for the region near the previous bounding box. We test both detectors of “Five” and “Zero”. If both detectors does not find the positive patch, it means that there is no hand in the operation zone. Figure 3-22 illustrates the existence of the hand.

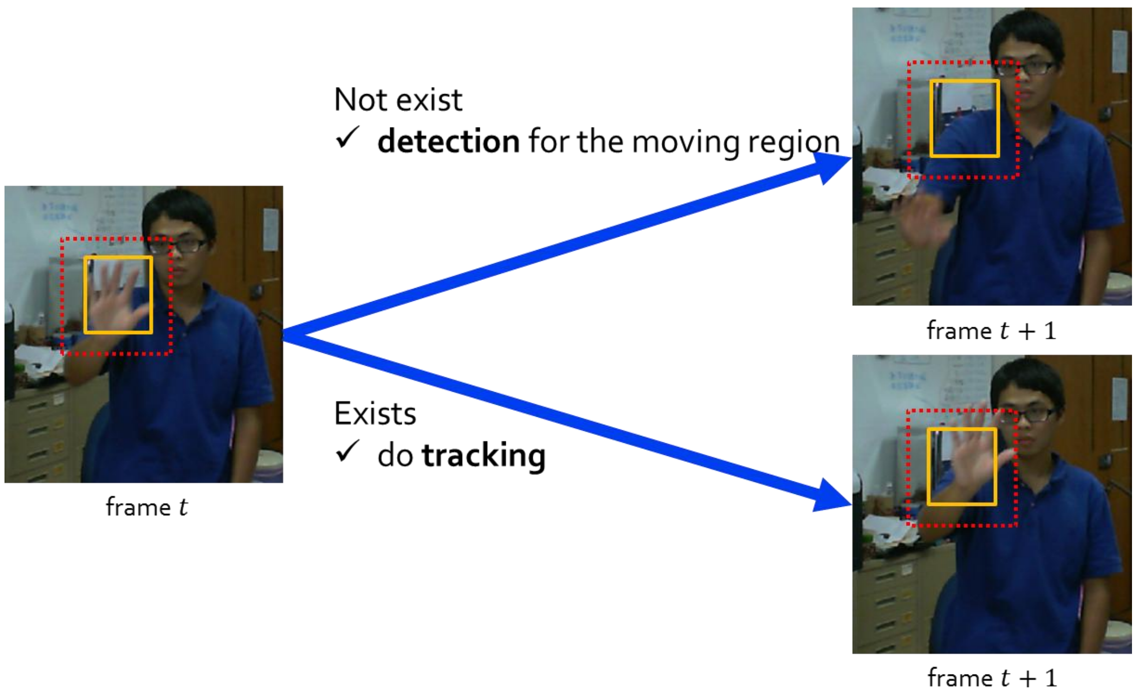


Figure 3-22 Example of existing hands.

For the second issue, we want to find the precise location of the hand with the help of the detectors. In our tracking algorithm, we can track the hand in the next frame. However, there may be some problems in the tracking algorithm. One is that the tracking result is not guaranteed to be perfect. The other problem is for the filtering process in the tracking algorithm. In fact, there may be too few points to predict the location of the hand. Hence, we use the detectors to solve both problems. First, if there are enough points to predict the bounding box, the hand is tracked using the tracking algorithm initially. After that, we use the detector to detect the hand location for refinement. We do not do the refinement in every frame since it is a little time consuming. Moreover, using detectors to predict the bounding box, the trajectory may oscillate as the time advances. It is inconvenient for the users. We want a smoother trajectory. As a result, we refine the prediction for every 5 frames. Second, if there are too few points to predict the hand location, we use the detector to find the hand location directly. Figure 3-23 illustrates the refinement for the tracking result.

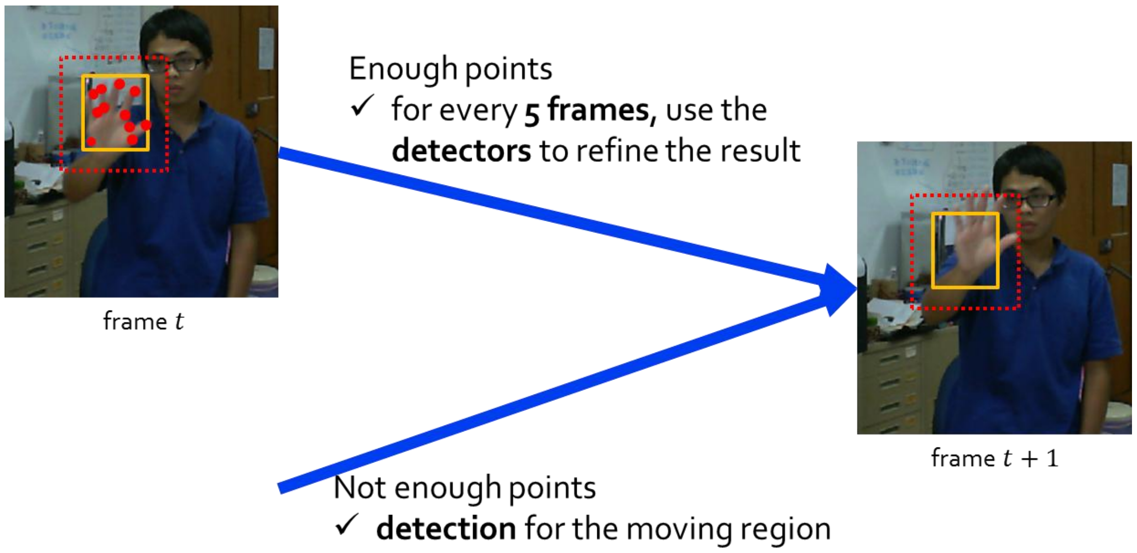

frame $t$
Enough points
✓ for every **5 frames,** use the **detectors** to refine the result
frame $t+1$
Not enough points
✓ **detection** for the moving region

Figure 3-23 Illustration of the refinement of the hand.

**Algorithm 1**. *Integrator*

```
If hand does not exist in the operation zone:
    If there is moving object:
        1.)  Use the detectors to find the hand.
             Output location.
    Else:
        2.)  No action.
Else:
    If there is enough feature points:
        3.)  Tracking.
        For every 5 frames, if $posterior_{tracked\ patch} < threshold$:
            (a)  Use the detectors to refine prediction.
                 Output location.
        Else:
            (b)  Output location using the tracking result.
    Else:
        If there is moving object:
            (a)  Use the detectors to find the hand.
                 Output location.
        Else:
            (b)  No action.
Check existence.
```

## 3.6 Hand Gesture Recognition

The purpose of our thesis is to develop a useful and intuitive gesture recognition algorithm, which is able to replace the use of mouse. The system needs not to contact with the device directly. It is a remote-control human-computer interface. With the help of tracking and detection method, we can define some simple and intuitive gestures and recognize them to perform some tasks. Hence, in Section 3.5.1, we define the gestures and the corresponding tasks. In Section 3.5.2, we introduce the gesture recognition method.

### 3.6.1 Gesture Definition

As mentioned in Section 3.4.1, we train the classifiers which can detect “Five” and “Zero” and discriminate these two gestures. We use them to design some simple gestures that can control the cursor of the computer and perform basic functionality. First, we use the gesture “Five” to represents the cursor. Moving the hand is like moving the cursor, as shown in Figure 3-24(a). The position of the cursor on the screen is calculated by Equation (3-3) and (3-4). As shown in Figure 3-24(b), the eyes, hand and the cursor are on a straight line. It is intuitively to control the cursor.

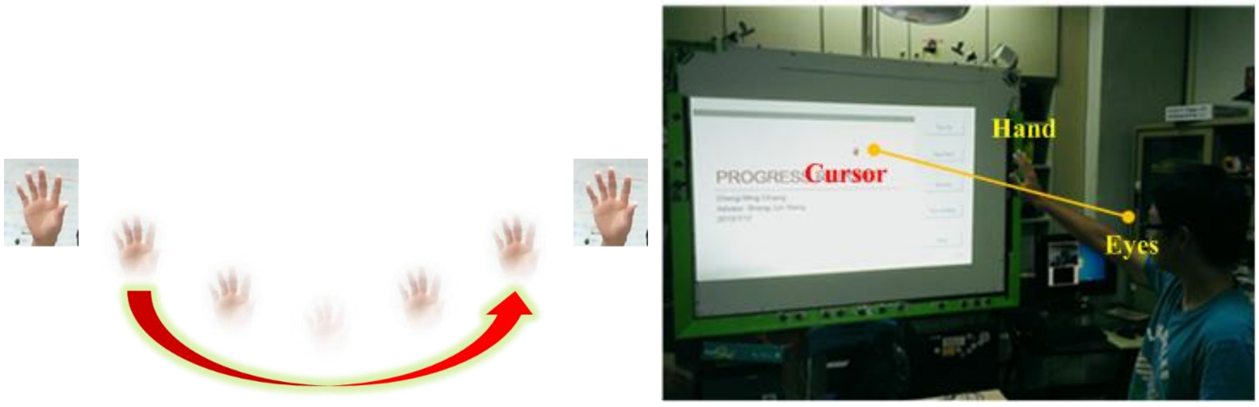


Figure 3-24 Illustration of moving the cursor.
(a) Moving the cursor with gesture “Five”. (b) Relative position of eyes, hand and the cursor.

The second functionality is the "Click" function. When the gesture changes from "Five" into "Zero", it represents the pressing on the left button of the mouse. In this state, we can move the hand with gesture "Zero" to drag some object. As we change the gesture from "Zero" back to "Five", it means releasing the left button of the mouse. Figure 3-25 illustrates the "Click" and "Drag and Drop" processes.

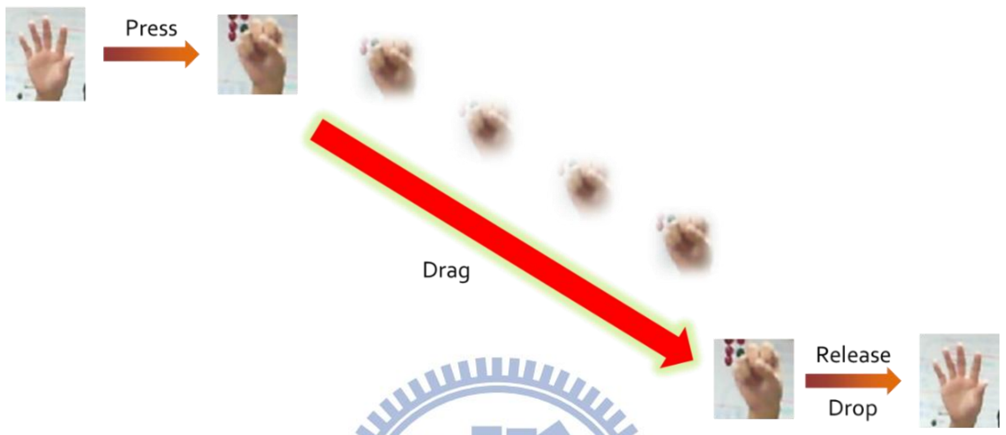


Figure 3-25 Illustration of the "Click" and "Drag and Drop" processes.

### 3.6.2 Gesture Recognition

With the precise hand location, we only need to recognize the gestures "Five" and "Zero". We use the pre-trained classifier "Five-Zero" to discriminate these two gestures. As a matter of fact, the detection rate of the classifiers cannot be perfect. For example, when moving the hand with the gesture "Five", there may be one or two frames classified as "Zero". Under this situation, there is a false alarm for the "Press" functionality. The false alarm is very inconvenient for the users. Hence, we propose a voting mechanism to stabilize the recognition algorithm. We use a sliding window in the temporal domain. The sliding window records the classification result for each frame in the window. The current action is not altered until the majority of the results changes from one gesture into the other gesture. Figure 3-26 illustrates the voting mechanism. Using the voting mechanism increases not only the stability but also the recognition time. In our system, the size of the temporal sliding window is 5 frames. It means that the recognition process is a delayed action for 5 frames. Since our overall algorithm is performed fast enough, this delayed action does not influence the use of the system.

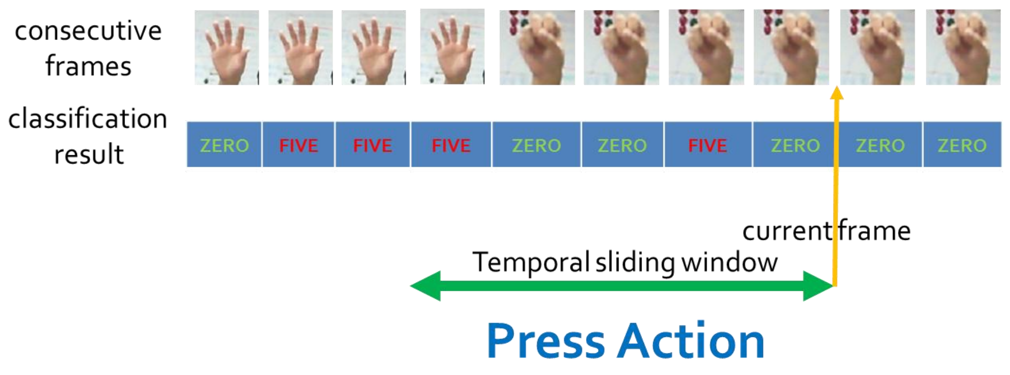


Figure 3-26 Illustration of the voting mechanism.

# Chapter 4 Experimental Results

In this chapter, we show some results of our hand detection algorithm. In Section 4.1, we make a comparison between 22.5°-sensitive and 45°-sensitive filters. We also test different parameters in Random Forest training process. Moreover, we test our algorithm on images with different illumination situations. Section 4.2 shows some image frames of our detection, tracking and gesture recognition algorithms.

## 4.1 Hand Detection Results

In this section, we show the detection results of different edge filters, different training parameters and different illumination images. We show the error rate and the average testing time of three classifiers: “Five-Zero”, “Five-Background” and “Zero-Background”. We use 6196 testing images for the gesture “Five” and 5014 testing images for the gesture “Zero”. We randomly sample 10000 background testing images from the indoor scenes.

Table 4-1 shows the final classifiers in our algorithm. The feature vectors are calculated by eight 45°-sensitive edge filters. The number of trees in the forest is 70.

Table 4-1 The detection results of the classifiers in our algorithm.

| | 70 trees / 8 edge filters (45°) | | | |
|---|---|---|---|---|
| | Number of images | Miss-classified | Error rate | Average testing time |
| Five | 6196 | 10 | 0. 1427 % | 0.000686 seconds |
| Zero | 5014 | 6 | | |
| Five | 6196 | 18 | 0.7162 % | 0.000714 seconds |
| Background | 10000 | 98 | | |
| Zero | 5014 | 74 | 1.3987 % | 0.000680 seconds |
| Background | 10000 | 136 | | |

As we can see in the above table, the classifier “Five-Zero” has the best performance. Since the appearance of the gestures “Five” and “Zero” diverges greatly, this classifier outperforms the others. The “Zero-Background” detector has the largest error rate since the feature of the gesture “Zero” is not more distinct than the gesture “Five”.

In table 4-2, we show the detection results of using sixteen 22.5°-sensitive edge filters to calculate feature vectors. We also show the detection results of using 50 trees in Random Forest.

Table 4-2 The detection results for different edge filters and training parameter.

| | 70 trees / 16 edge filters (22.5°) | | | 50 trees / 8 edge filters (45°) | | |
|---|---|---|---|---|---|---|
| | Miss-classified | Error rate | Average testing time | Miss-classified | Error rate | Average testing time |
| Five | 12 | 0. 1517 % | 0.001022 seconds | 10 | 0. 1427 % | 0.000646 seconds |
| Zero | 5 | | | 6 | | |
| Five | 18 | 0.7039 % | 0.001203 seconds | 16 | 0.7162 % | 0.000735 seconds |
| Background | 96 | | | 100 | | |
| Zero | 62 | 1.2322 % | 0.001014 seconds | 76 | 1.3454 % | 0.000760 seconds |
| Background | 123 | | | 126 | | |

The results of using 16 edge filters is as what we expected. If we use 16 edge filters, the error rate decreases slightly. However, the processing time is about 1.5 times of the 8 edge filters. Hence, we use 8 edge filters in our algorithm so as to reach real-time performance. In our training process, increasing the number of trees reduces the training error rate. However, it does not help a lot in reducing the testing error rate.

In [1], they quantize 180 degrees to 8 bins. The edge filters are 22.5°-sensitive. Instead, we quantize 360 degrees to 8 bins. The edge filters are 45°-sensitive. We make a comparison of these two filters in Table 4-3.

Table 4-3 The comparison results for different sensitivity edge filters.

| | 70 trees / 8 edge filters (45°) | | 70 trees / 8 edge filters (22.5°) | |
|---|---|---|---|---|
| | Miss-classified | Error rate | Miss-classified | Error rate |
| Five | 10 | 0. 1427 % | 22 | 0. 4550 % |
| Zero | 6 | | 29 | |
| Five | 18 | 0.7162 % | 12 | 0.6113 % |
| Background | 98 | | 87 | |
| Zero | 74 | 1.3987 % | 103 | 1.9848 % |
| Background | 136 | | 195 | |

The left side of table 4-3 shows the results using 45°-sensitive filters. The right side shows the results of using 22.5°-sensitive filters. Obviously, the error rate of using 360° orientations is lower than that by using 180° orientations. Hence, we use 360°-orientation filters to calculate the feature vectors.

We change our testing images to different illumination conditions to see whether our algorithm is invariant to the illumination. Some testing images are shown in Figure 4-1. For each type of the images, the left one is the brighter image. The middle one is the original image used in Table 4-1. The right one is the darker image. Table 4-4 shows the testing results for the brighter and darker images.

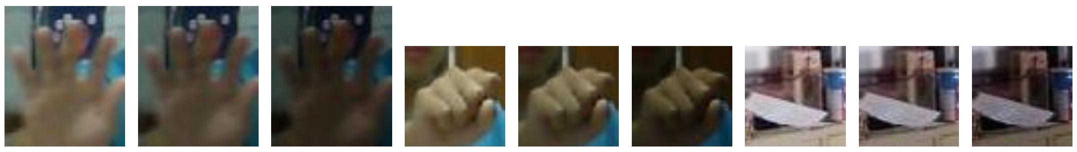


Figure 4-1 Testing images for different illumination.

Table 4-4 The detection results for different illumination images.

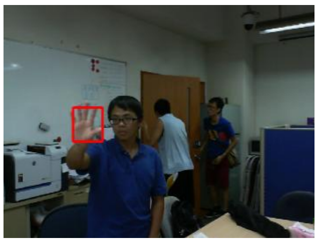
| | Bright images | | Dark images | |
|---|---|---|---|---|
| | Miss-classified | Error rate | Miss-classified | Error rate |
| Five | 6 | 0. 1427 % | 14 | 0. 1338 % |
| Zero | 10 | | 1 | |
| Five | 11 | 0.8891 % | 48 | 0.7533 % |
| Background | 133 | | 74 | |
| Zero | 66 | 1.4786 % | 183 | 1.9915 % |
| Background | 156 | | 116 | |

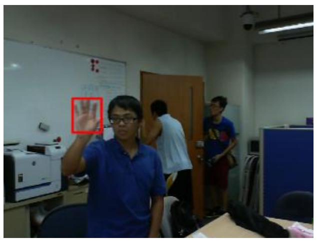
Table 4-4 shows that the illumination change has a little impact on the error rate. Averagely, it does not influence our performance too much except for the dark images in the "Zero-Background" case. For darker images, some edge information may get lost. Hence, the images are difficult to be classified correctly.

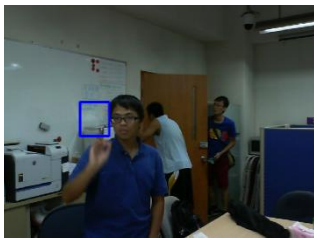
## 4.2 Results for the Overall Algorithm

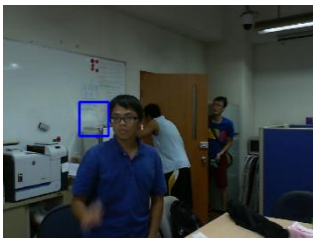
In this section, we show the results the overall algorithm. There are four types of results: the presence of the hand, cursor tracking, clicking and drag-and-drop action. We show some image frames for the four types of actions. The frames are extracted in a sequence with 1177 frames. Our algorithm is implemented using C++ code in Microsoft Visual Studio 2010. We use OpenCV [32] library for some image processing and computer vision algorithms. The hardware we use is Intel Core i5-3470 3.2 GHz CPU with 8GB memory.

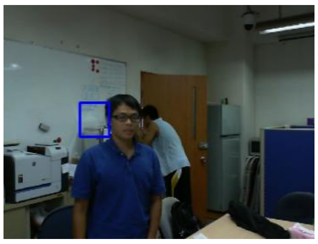
Figure 4-2 shows the image frames for determining the presence of the hand. The red rectangle represents the current gesture is "Five". The blue rectangle means that there is no hand near the previous bounding box.

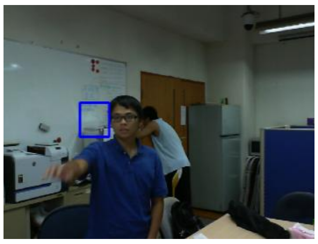

(a) Frame 744.

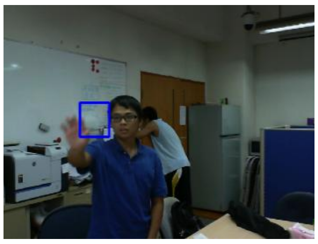

(b) Frame 747.


(c) Frame 751.


(d) Frame 753.


(e) Frame 788.


(f) Frame 820.


(g) Frame 823.

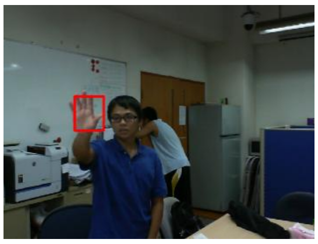
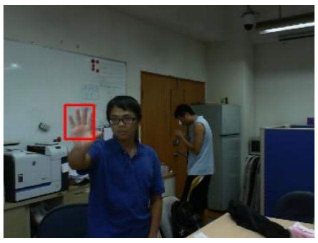
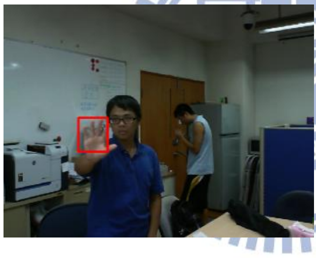
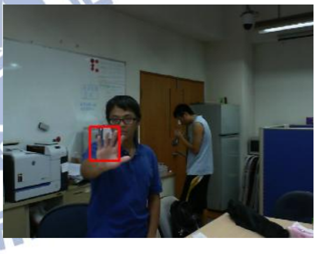

(h) Frame 826.

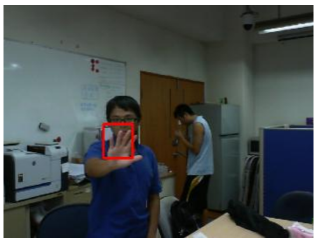
Figure 4-2 Image frames for determining the presence of the hand.

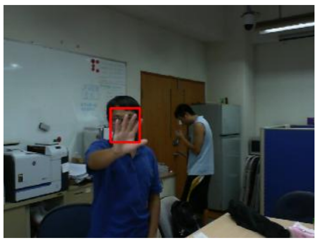
Figure 4-3 shows the results for tracking the gesture "Five". As described in Section 3.6.1, we use the gesture "Five" to represent the cursor.

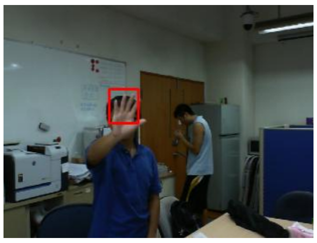

(a) Frame 190.

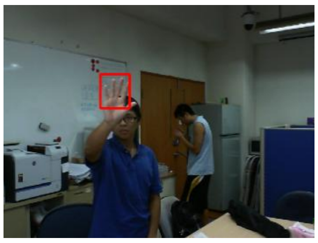

(b) Frame 195.

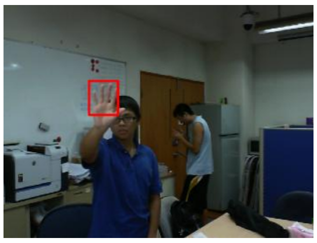

(c) Frame 200.

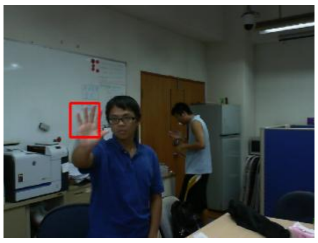

(d) Frame 205.

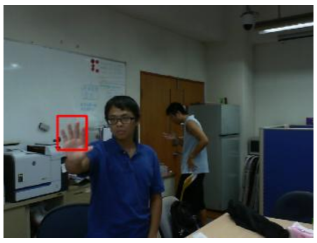

(e) Frame 210.

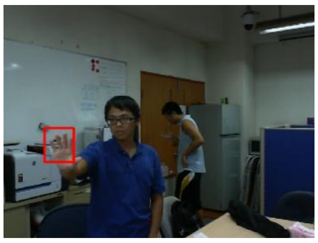

(f) Frame 215.


(g) Frame 220.


(h) Frame 225.


(i) Frame 230.


(j) Frame 235.


(k) Frame 240.

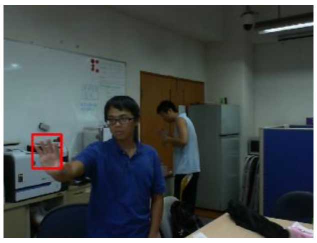
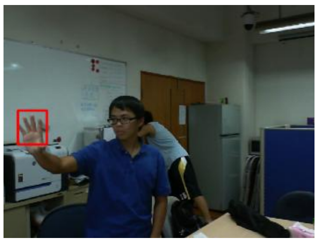
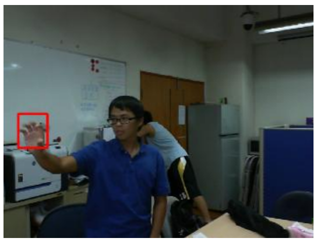
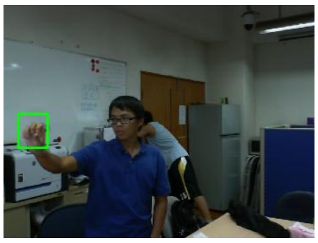

(l) Frame 245.

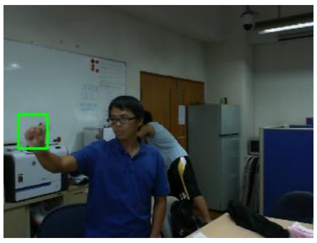
Figure 4-3 Image frames for tracking the gesture "Five".

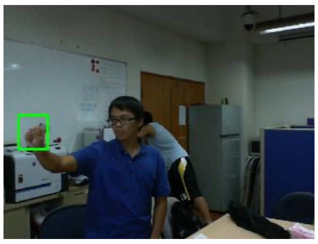
Figure 4-4 shows the image frames for the “click” action. The action is performed when changing the gestures. The green rectangle represents the gesture “Zero”

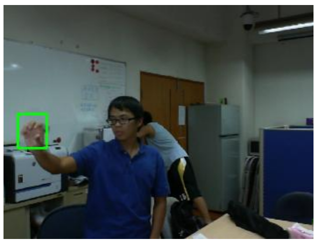

(a) Frame 506.

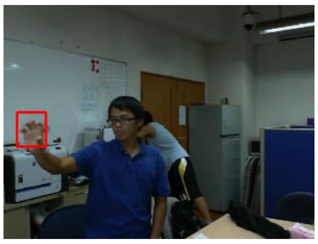

(b) Frame 510.


(c) Frame 511.


(d) Frame 513.


(e) Frame 518.


(f) Frame 523.


(g) Frame 524.

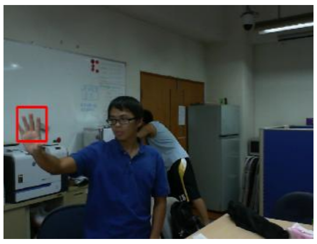
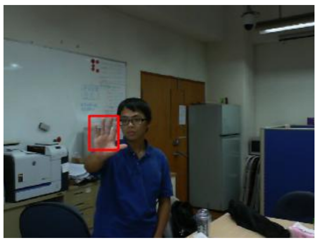
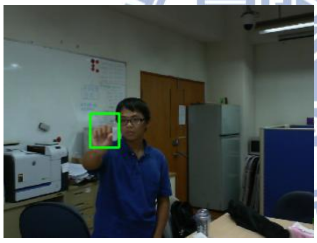
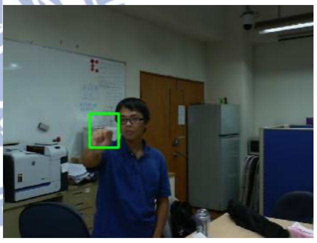

(h) Frame 527.

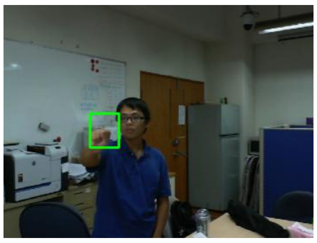
Figure 4-4 Image frames for the “click” action.

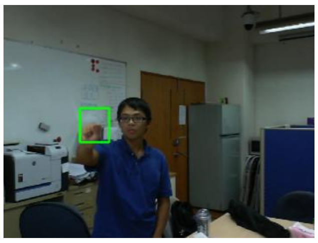
Figure 4-5 shows the “drag and drop” action which tracks the gesture “Zero”

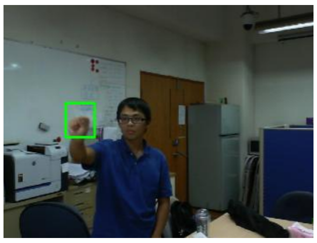

(a) Frame 47.

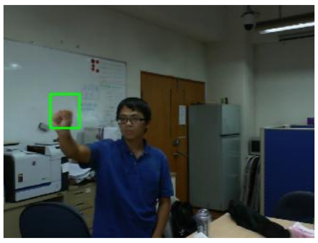

(b) Frame 53.

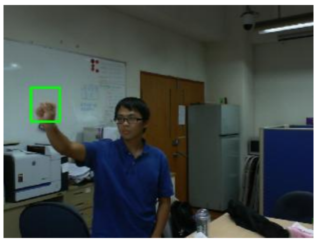

(c) Frame 59.

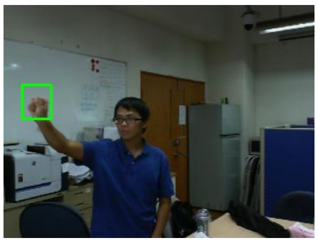

(d) Frame 65.

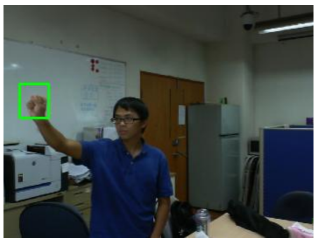

(e) Frame 71.

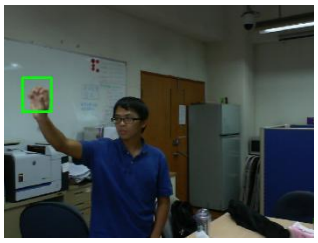

(f) Frame 77.


(g) Frame 83.


(h) Frame 89.


(i) Frame 95.


(j) Frame 101.


(k) Frame 109.

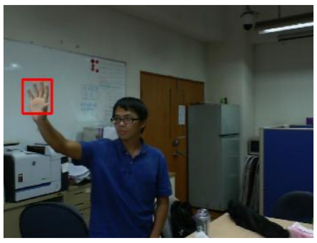

(l) Frame 112.

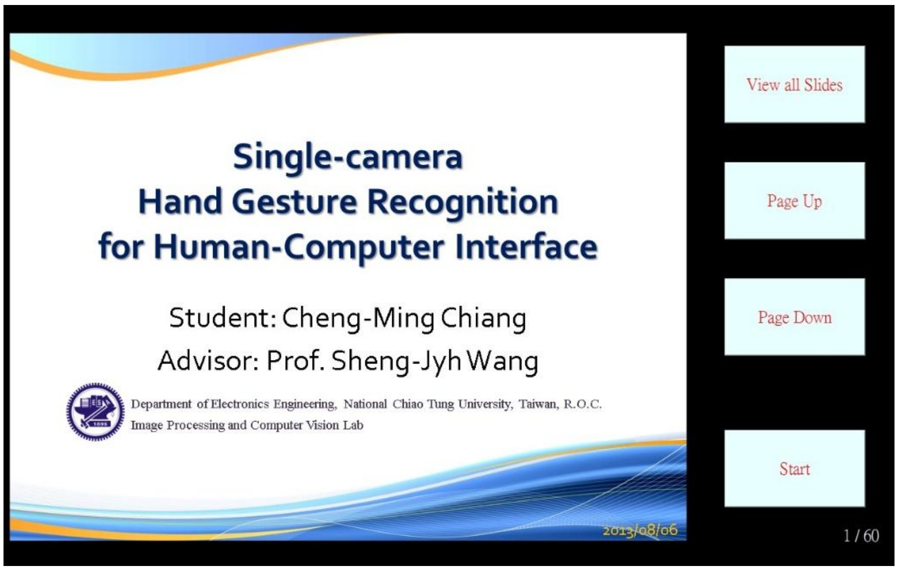
Figure 4-5 Image frames for the “drag and drop” action.

# Chapter 5  System Applications

To confirm the feasibility of our algorithm, we design two human-computer interface applications. One is the slide presentation system introduced in Section 5.1. We can use the designed gestures to control the presentation slides intuitively. The other is a puzzle game system, which is described in Section 5.2. These systems prove that our algorithm can reach real-time performance and is easy to use. We also show some demo image frames in Section 5.1 and Section 5.2.

## 5.1  Presentation System

Nowadays, when people give a speech or make a presentation, they usually use slides. Most people use Microsoft PowerPoint as the presentation tool, with the aid of keyboard, mouse the laser pointer. However, in a large meeting room, we may not have the keyboard or the mouse at hand. Hence, the laser pointer is widely used. Even though the laser pointer is convenient to use, it is not intuitive and its functionality is restricted. Consequently, we design a presentation system that can be controlled by using some intuitive gestures.



Single-camera
Hand Gesture Recognition
for Human-Computer Interface

Student: Cheng-Ming Chiang
Advisor: Prof. Sheng-Jyh Wang

Department of Electronics Engineering, National Chiao Tung University, Taiwan, R.O.C.
Image Processing and Computer Vision Lab

2013/08/06

Figure 5-1 Interface of the presentation system.

Figure 5-1 shows the interface of our presentation system. We design some useful buttons on the right side, such as "Page Up" and "Page Down". Sometimes the number of the slides is very large. If we want to show a specific page, it is very inconvenient for the original presentation system during the presentation time. Hence, as shown in Figure 5-2, we add a button "View all Slides", which shows all the slides and we can easily choose the slide we want.

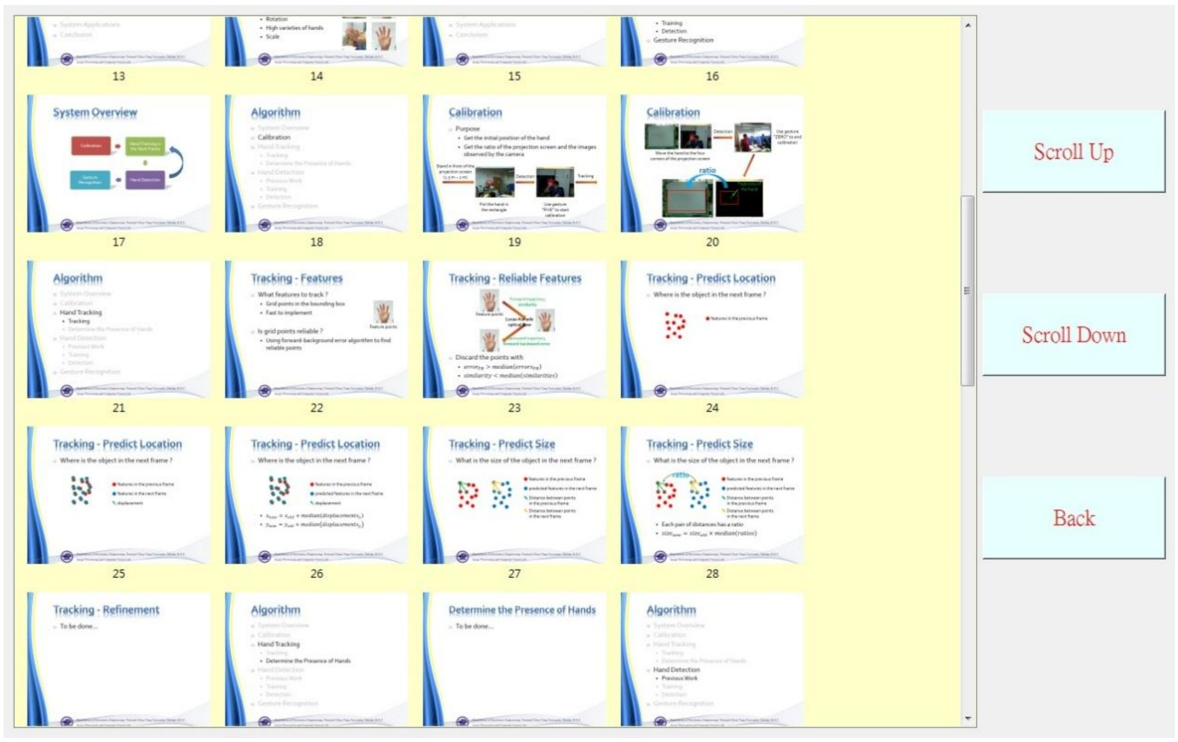
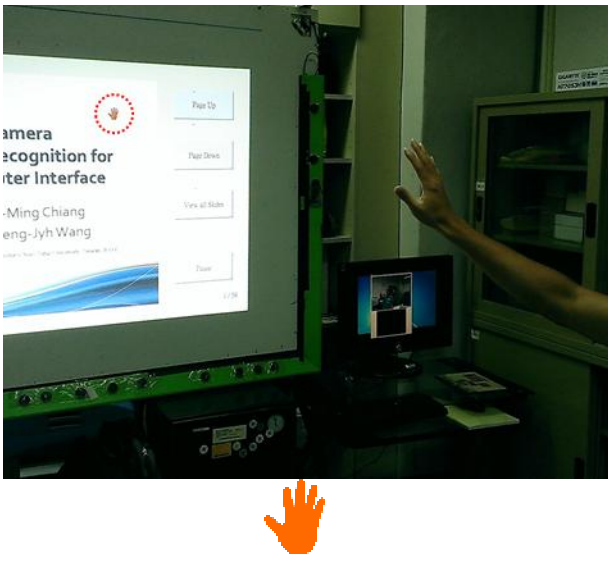
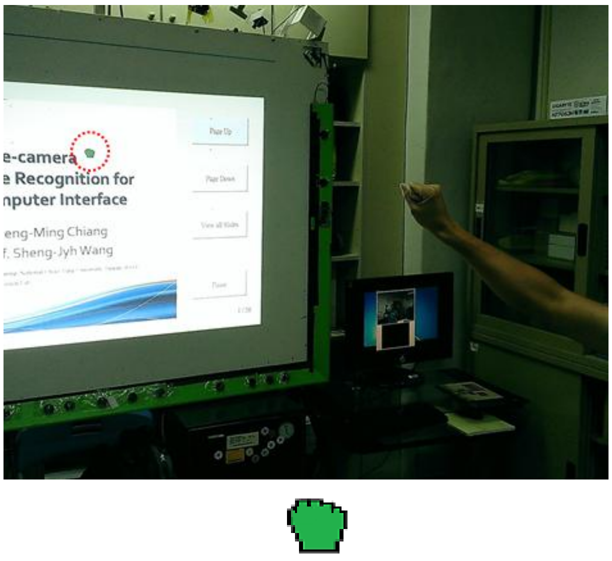
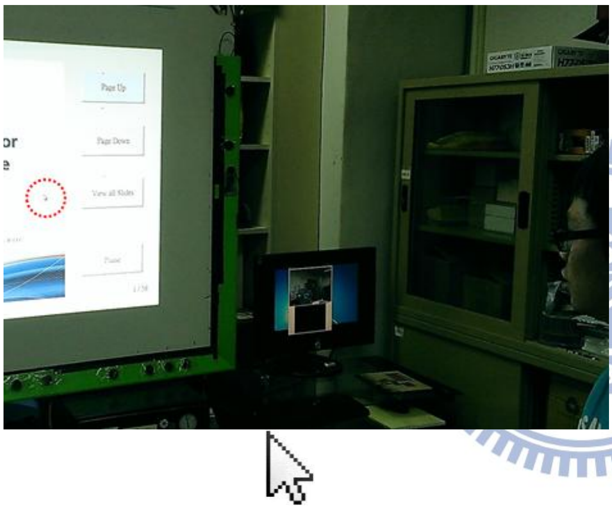


Figure 5-2 "View all Slides" interface.

We also introduce some meaningful cursor icons in the system. The icons reflect the current state of the hand gestures, just like interacting with the computer. There are four types of icons. As shown in Figure 5-3(a) and Figure 5-3(b), the cursor icon changes into "Five" or "Zero" as the gesture change. If the hand does not exist in the operation zone, the cursor changes to the normal icon "Arrow", as shown in Figure 5-3(c). When the user wants to control the cursor again, he/she just points his/her hand toward the cursor. That is to say, he/she makes the cursor, his/her hand and his/her eyes on a straight line. During the tracking process, if the hand moves outside the operation zone, the system will remind the user by the icon "Forbidden", as shown in figure 5-3(d).


(a) The icon “Five”


(b) The icon “Zero”


(c) The icon “Arrow”

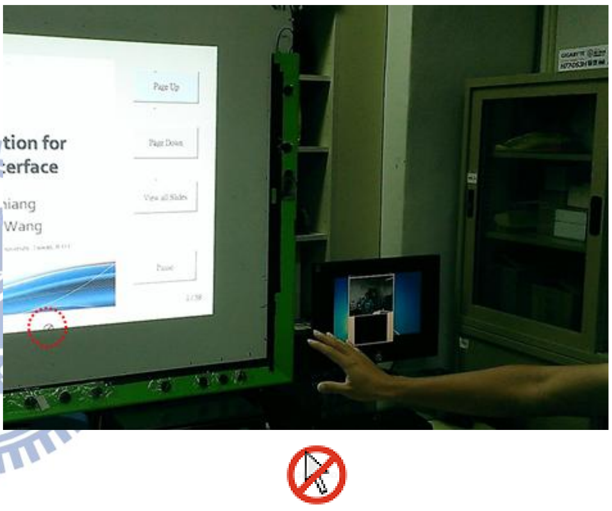
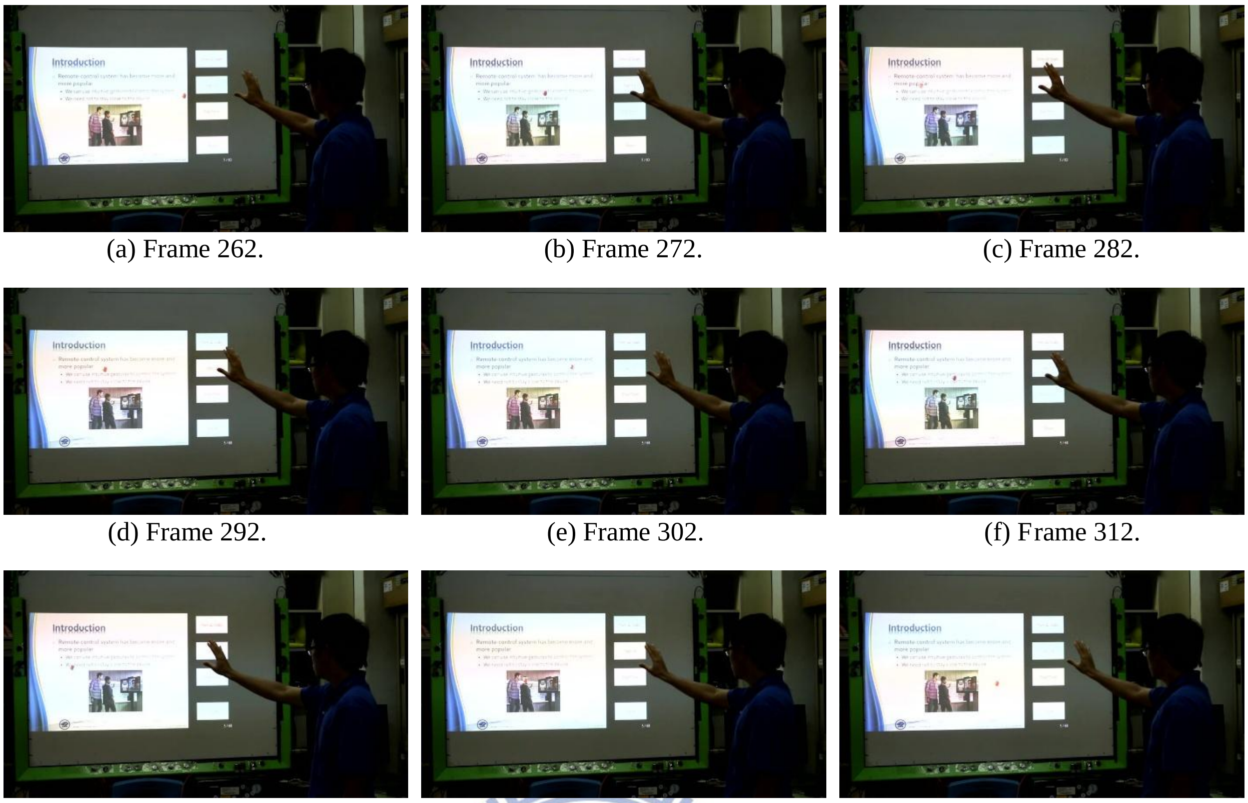

(d) The icon “Forbidden”

Figure 5-3 The interactive cursor icons.

The subsequent figures show some image frames for the specific actions in our demo video which contains 1599 frames. Figure 5-4 shows how we control the cursor as a pointer to present the slides. As described previously, we use the icon “Five” to represent the cursor when the gesture is “Five”.



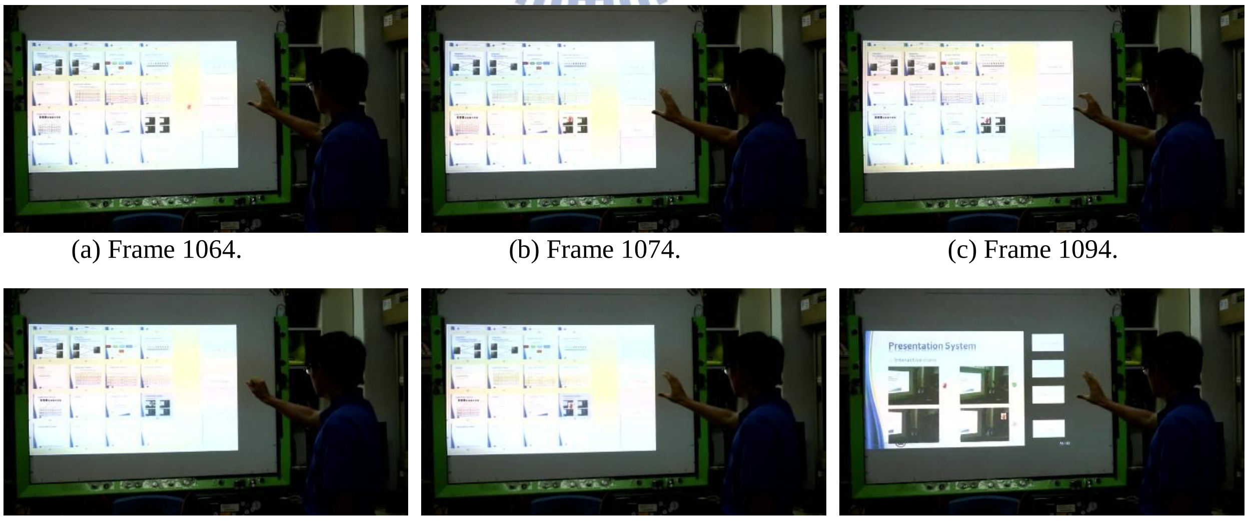
(a) Frame 262. (b) Frame 272. (c) Frame 282.

(d) Frame 292. (e) Frame 302. (f) Frame 312.

(g) Frame 322. (h) Frame 332. (i) Frame 342.

Figure 5-4 The image frames for controlling the cursor in our demo video.

Figure 5-5 shows some image frames in which we use our system to perform the “view all slides” action.



(a) Frame 1064. (b) Frame 1074. (c) Frame 1094.

(d) Frame 1104. (e) Frame 1109. (f) Frame 1114.

Figure 5-5 The image frames for the “view all slides” action in our demo video.

## 5.2 Puzzle Game System

In addition to the presentation system, we also develop a puzzle game system. The user can use some intuitive gestures to play the game. The key idea of controlling the system is the “drag and drop” action. This action is accomplished by the gestures defined in Section 3.6.1. Changing the gesture from “Five” into “Zero” means choosing the item. The user can move his/her hand with the gesture “Zero” to drag the item. Finally, he/she changes his/her gesture from “Zero” into “Five” to drop the item. Figure 5-6 shows the interface of our puzzle game system. The left region of the interface is the puzzle pieces. The user drags the pieces to the right-top region to accomplish the puzzle game. The original image of the puzzle is shown on the right-bottom side.

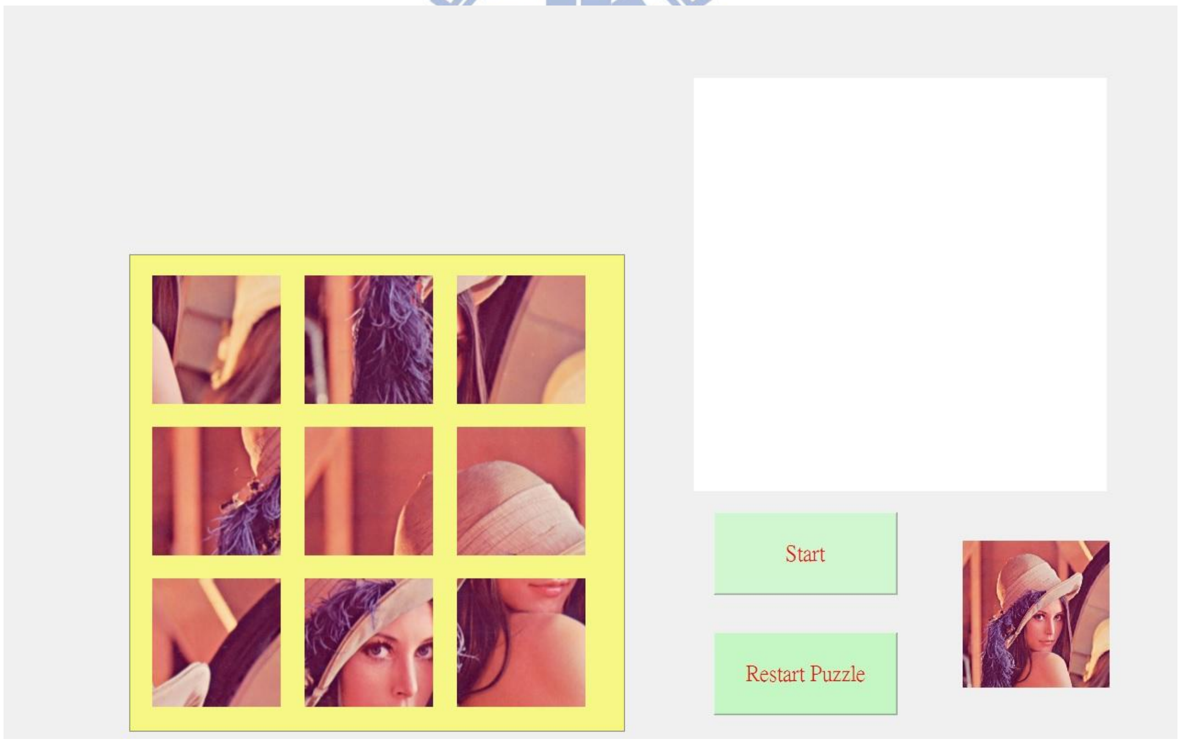
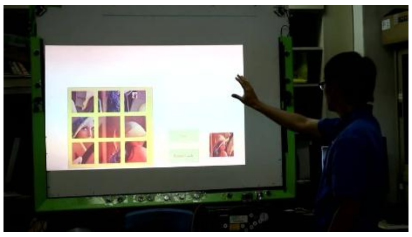
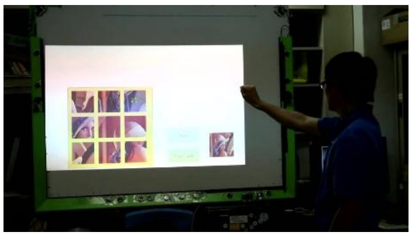
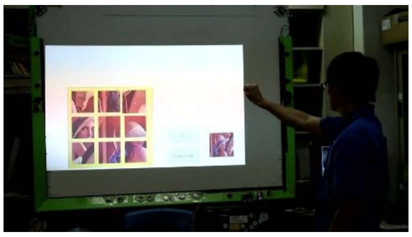


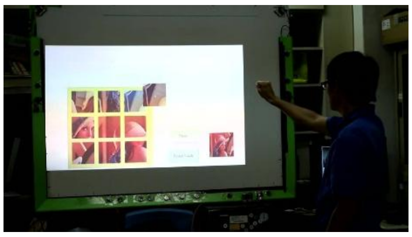
Figure 5-6 The interface of the puzzle game system.

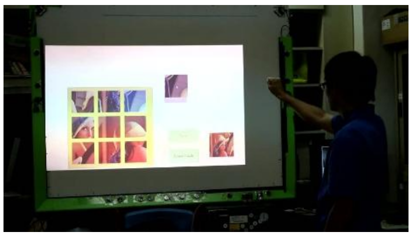
Figure 5-7 shows some image frames in our demo video for the puzzle game system. The figure shows the “drag and drop” action.

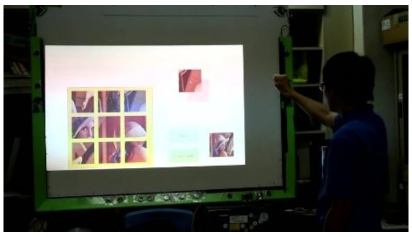

(a) Frame 37.

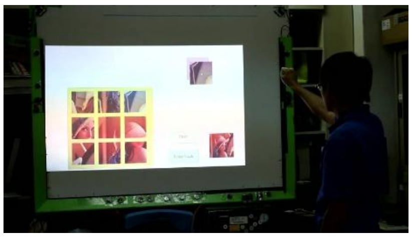

(b) Frame 47.

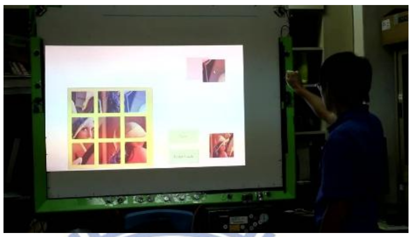

(c) Frame 57.

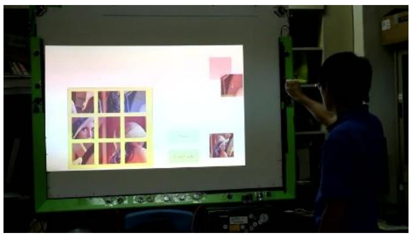

(d) Frame 67.

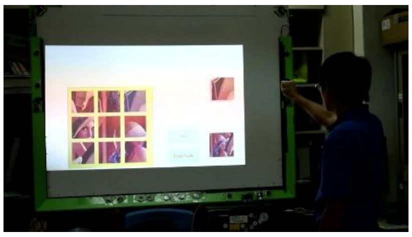

(e) Frame 77.

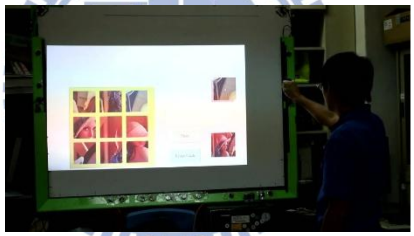

(f) Frame 87.

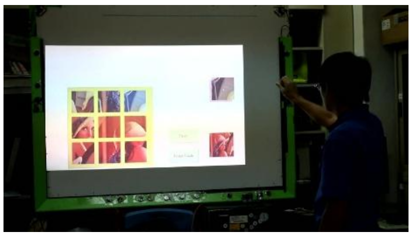

(g) Frame 107.

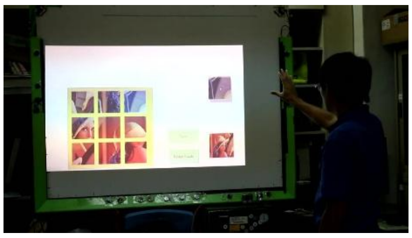

(h) Frame 117.

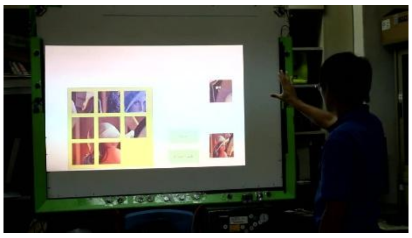

(i) Frame 127.


(j) Frame 137.


(k) Frame 147.


(k) Frame 152.


(m) Frame 157.


(n) Frame 162.

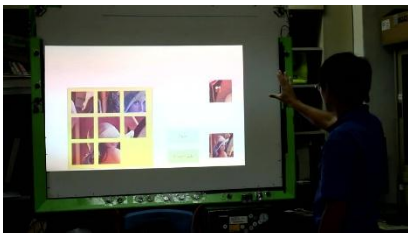

(o) Frame 167.

Figure 5-7 The image frames for the "drag and drop" action in our demo video.

# Chapter 6 Conclusion

We propose a hand gesture recognition algorithm for a human-computer interface. We define some useful gestures to control the system in an intuitive way. In our algorithm, we use the tracking algorithm to track the position of the hand initially. We also use the Random Forest method to train the detectors and the randomized pooling cells. With the help of the detectors, we can determine the presence of the hand and refine the position predicted by the tracker. After the position of the hand is determined, we use the classifier to recognize the gestures.

We use our proposed algorithm to develop two system applications. One is the slide presentation system. The users can use a few intuitive gesture to control the slides during presentation. They do not need a mouse to control the computer. The other system is a puzzle game system, in which our system is used to play the puzzle game.

In our system, we need only one RGB camera, which is low cost and can be installed easily. Moreover, the system can reach real-time performance. The flexibility of the system is higher than multi-touch systems. We can easily add some useful gestures to control the human-computer interface.

# References

[1] P. H. Chen., “Hand Posture Recognition Technique for Large-scale Touch Panel,” *M.S. thesis, Dept. Electronic Engineering, National Chiao Tung University, Hsinchu, Taiwan,* 2012.

[2] http://www.microsoft.com/office/perceptivepixel/products/default.aspx

[3] http://movie.douban.com/photos/photo/2011595591

[4] R. Y. Wang, J. Popovi, “Real-time hand-tracking with a color glove,” *ACM SIGGRAPH 2009 papers, pp. 1-8*, 2009.

[5] http://www.nintendo.com/wii

[6] C. Albitar, P. Graebling, and C. Doignon, "Robust Structured Light Coding for 3D Reconstruction," *IEEE 11th International Conference on Computer Vision*, pp. 1-6, 2007.

[7] http://www.primesense.com/solutions/technology/

[8] http://www.microsoft.com/en-us/kinectforwindows/

[9] J. Shotton, A. Fitzgibbon, M. Cook, T. Sharp, M. Finocchio, R. Moore, A. Kipman, and A. Blake, "Real-time human pose recognition in parts from single depth images," *IEEE Conference on Computer Vision and Pattern Recognition, pp. 1297-1304*, 2011.

[10] https://www.leapmotion.com/product

[11] http://www.zhihu.com/question/20252985

[12] R. Hartley, and A. Zisserman, “Multiple view geometry in computer vision”, *2nd ed., Cambridge, UK ; New York: Cambridge University Press*, 2003.

[13] L. Breiman, "Random Forests," *Machine Learning, vol. 45, pp. 5-32*, 2001.

[14] J. Yangqing, H. Chang, and T. Darrell, "Beyond spatial pyramids: Receptive field learning for pooled image features," *IEEE Conference on Computer Vision and Pattern Recognition, pp. 3370-3377*, 2012.

[15] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," *IEEE Conference on Computer Vision and Pattern Recognition, pp. 886-893 vol. 1*, 2005.

[16] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision, vol. 60, pp. 91-110*, 2004.

[17] S. Lazebnik, C. Schmid, and J. Ponce, "Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories," *IEEE Conference on Computer Vision and Pattern Recognition, vol. 2, pp. 2169-2178*, 2006

[18] J. Yangqing, H. Chang, and T. Darrell, "Beyond spatial pyramids: Receptive field learning for pooled image features," *IEEE Conference on Computer Vision and Pattern Recognition, pp. 3370-3377*, 2012.

[19] M. Sanjeev Arulampalam, S. Maskell, N. Gordon, and T. Clapp, "A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking," *IEEE Transactions on Signal Processing, vol. 50, pp. 174-188*, 2002.

[20] M. S. M. Asaari and S. A. Suandi, "Hand gesture tracking system using Adaptive Kalman Filter," *International Conference on Intelligent Systems Design and Applications, pp. 166-171*, 2010.

[21] S. J. Julier and J. K. Uhlmann, “New extension of the Kalman filter to nonlinear systems”, *SPIE Proceedings of Signal Processing, Sensor Fusion, and Target Recognition VI, pp. 182-193*, 1997.

[22] C. Shan, T. Tan, and Y. Wei, "Real-time hand tracking using a mean shift embedded particle filter," *Pattern Recognition, vol. 40, pp. 1958-1970*, 2007.

[23] S. Avidan, "Ensemble Tracking," *IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 29, pp. 261-271*, 2007.

[24] Z. Kalal, K. Mikolajczyk, and J. Matas, "Tracking-Learning-Detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 34, pp. 1409-1422*, 2012.

[25] Y. Freund and R. E. Schapire, "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting," *Journal of Computer and System Sciences, vol. 55, pp. 119-139*, 1997.

[26] Y. Cheng, "Mean shift, mode seeking, and clustering," *IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 17, pp. 790-799*, 1995.

[27] Z. Kalal, K. Mikolajczyk, and J. Matas, "Forward-Backward Error: Automatic Detection of Tracking Failures," *International Conference on Pattern Recognition, pp. 2756-2759*, 2010.

[28] J.-Y. Bouguet, "Pyramidal implementation of the Lucas Kanade feature tracker," *Intel Corporation, Microprocessor Research Labs*, 2000.

[29] S. Jianbo and C. Tomasi, "Good features to track," *IEEE Conference on Computer Vision and Pattern Recognition, pp. 593-600*, 1994,.

[30] B. K. P. Horn and B. G. Schunck, "Determining Optical Flow," *Massachusetts Institute of Technology*, 1980.

[31] P. Viola and M. Jones, "Robust real-time object detection," *International journal of computer vision, vol. 4*, 2001.

[32] http://opencv.org/about.html