Multimed Tools Appl (2015) 74:2687–2715
DOI 10.1007/s11042-013-1501-1

Hand tracking and gesture recognition system for human-computer interaction using low-cost hardware

Hui-Shyong Yeo · Byung-Gook Lee · Hyotaek Lim

Published online: 31 May 2013

Abstract Human-Computer Interaction (HCI) exists ubiquitously in our daily lives. It is usually achieved by using a physical controller such as a mouse, keyboard or touch screen. It hinders Natural User Interface (NUI) as there is a strong barrier between the user and computer. There are various hand tracking systems available on the market, but they are complex and expensive. In this paper, we present the design and development of a robust marker-less hand/finger tracking and gesture recognition system using low-cost hardware. We propose a simple but efficient method that allows robust and fast hand tracking despite complex background and motion blur. Our system is able to translate the detected hands or gestures into different functional inputs and interfaces with other applications via several methods. It enables intuitive HCI and interactive motion gaming. We also developed sample applications that can utilize the inputs from the hand tracking system. Our results show that an intuitive HCI and motion gaming system can be achieved with minimum hardware requirements.

Keywords Gesture recognition · Hand/Finger tracking · HCI · Kinect · Motion game · NUI

1 Introduction

Computer technologies have grown tremendously over the past decade. As technologies progress even further, existing HCI techniques are becoming a bottleneck. Typical HCI has been the norm all this while and people are unreasonably curious on how things can be done to change the nature of HCI. The most common mode of HCI is relying on simple mechanical devices, i.e. keyboards and mice. These devices have grown to be familiar but are less natural and intuitive in interacting with computers. Besides that, with typical controllers, there is a

H.-S. Yeo
Department of Ubiquitous IT, Dongseo University, 617-716 Busan, Korea
e-mail: hsyeo@dongseo.ac.kr

B.-G. Lee
Department of Visual Contents, Dongseo University, 617-716 Busan, Korea
e-mail: lbg@dongseo.ac.kr

H. Lim (✉)
Division of Computer and Information Engineering, Dongseo University, 617-716 Busan, Korea
e-mail: htlim@dongseo.ac.kr

strong barrier between the user and the game, which causes less immersion in the game world. Author in [2] suggested motion controls as the future of gaming and discussed on how gesture-based controls have revolutionized the way people play video games.

1.1 Related works

Gesture enabled HCI transcends barriers and limitations by bringing the user one step closer to actual one to one interactivity with the computer. There have been much active research towards novel devices and techniques that allow gesture enabled HCI in recent years [23, 27]. There are generally two approaches to interpreting gestures in HCI by computers. First attempt to solve this problem resulted in a hardware-based approach [28]. This approach requires user to wear bulky devices, hindering ease and naturalness of interacting with the computer. Although the hardware-based approach provides high accuracy, it is not practical in users' everyday life. This has led to active research on more natural HCI technique, which is computer vision-based [1, 4–6, 9, 10, 13, 15–17, 20, 23–27, 29, 30, 32, 35, 36, 38]. This approach uses cameras and computer vision techniques to interpret gestures. Research on vision-based HCI has enabled many new possibilities and interesting applications. Some of the most popular examples are tabletop [26], visual touchpad [24], TV remote control [4, 32], augmented reality [5] and mobile augmented reality [17]. Vision-based HCI can be further categorized into marker-based and marker-less approach. Several studies utilize color markers or gloves [5, 17, 20, 36] for real time hand tracking and gesture recognition. This approach is easier to implement and has better accuracy, but it is less natural and not intuitive. Other studies focused on marker-less approach by using different techniques such as Haar-like features [1, 9, 10], Convexity defects [25], K-Curvature [15, 24, 30, 38], Bag-of-features [11, 12], Template Matching [16, 26], Circular Hough Transform [6], Particle Filtering [4], and Hidden-Markov Model [38]. Most studies on marker-less approach focused on recognize static hand poses [1, 4–6, 10, 15–17, 29, 30, 35, 36, 38], or dynamic gestures only [20, 25], but not both. It means the variety of inputs is very limited. Several researchers [1, 9, 10] use Haar-like features [34], which requires high computing power. The classifier preparation stage also consume a lot of time [1]. Some studies [15, 24, 30, 38] use K-curvature to find peaks and valleys along a contour, and then classify these as fingertips. However, it is also CPU intensive because all points along the contour perimeter must be calculated. Besides, they [15, 16, 24–26, 30, 35] did not solve the problem of differentiating between human face and hand regions because they assumed that only hand regions are visible to the camera. Therefore, the problem when the hand is blocking in front of the face is also not discussed. While authors in [12] utilize Haar-like features [34] to remove the face region first, it suffers from background color leakage problems [9, 12] due to its simple background subtraction method. Most of the studies [1, 6, 13, 15, 16] only focused on efficient hand recognition algorithm but did not translate the detected hand into functional inputs. Some authors [9, 11, 25] utilize static hand gesture recognition to simulate gaming inputs by translating different static gestures into keyboard events. However, the limited set of static gestures makes game control difficult and boring. While most of the researchers [4, 9, 26, 35] develop sample application as a proof-of-concept, the hand tracking capability is limited to their application only and is not able to interface with other applications other than passing simple mouse events.

1.2 Motivation

Typical controllers are unsuited to harnessing the full potential of a human being in HCI, whether in navigating applications or playing games. The keyboard, mouse and

game controller are the most prevalent interfaces. However, they can only offer a narrow bridge across the gap in interaction between human and computer. Besides that, controller inputs are significantly different from the outputs. Popular motion controllers such as the Nintendo Wii, PlayStation Move and Xbox Kinect changed the way in which users interact with video games, but they are costly and limited to a specific gaming console only. Therefore, our main motivation is to implement a robust hand/finger tracking and gesture recognition system for HCI using low-cost hardware that is readily available in most consumers' homes.

Our works make the following contributions:

- We focused on effectively tracking hand and fingers in 2D space, including location, direction, orientation, static hand gestures and simple gesture motion.
- We focused on implementing simple but robust tracking methods, while compensating for motion blur, different backgrounds, hand trembling, and confusion between face and hand regions.
- We focused on creating a flexible and intuitive HCI system with variety of inputs usable for controlling games and applications. The system is also able to interface with other applications via different methods.

The remainder of the paper is structured as follows. Section 2 provides the design rationale and architecture of our proposed solution. In Section 3, we present the prototype implementation with walkthrough results. Section 4 discusses the system thoroughly and provides user evaluations alongside the discussions. Finally, in Section 5 we conclude our paper and discuss how this line of research can be further explored.

2 System design and rationale

2.1 Overall system architecture

The suggested environment setup is shown in Fig. 1. The overall system consists of two parts (Fig. 2), back-end and front-end. The back-end system consists of three modules: Camera module, detection module, and interface module. They are summarized as follows:

i. Camera module: This module is responsible for connecting and capturing image output from different types of detectors (regular webcams and depth cameras), and then processing this output with different image processing techniques. The output of this module is a smoothed binary image with clean contours suitable for hand detection.
ii. Detection module: This module is responsible for detection and tracking of hand and fingers using our proposed method. Finite State machine and Kalman filter are applied to improve the accuracy and stability of tracking. The output of this module is hand and finger locations in 2D space.
iii. Interface module: This module is responsible for translating the detected hand and fingers into functional inputs and interfacing with other applications. Several translation modes and interfacing methods are provided.

The front-end system consists of three proof-of-concept sample applications that utilize the inputs from the back-end system. The samples include a fruit slicing motion game, a simple 3D object viewer and a Google Earth [14] plugin navigator. They are summarized as follows:

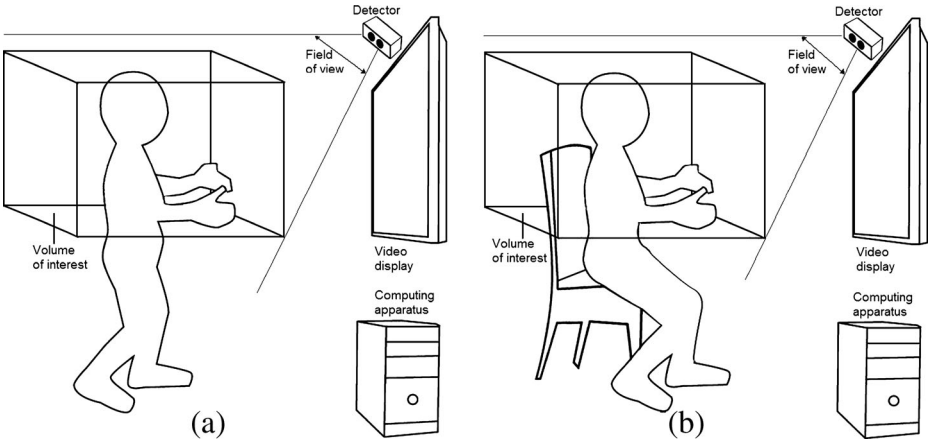


Fig. 1 Suggested environment setup **a** standing, **b** sitting

i. Fruit slicing motion game: A simple multiplayer networked motion game that is inspired by the popular mobile game: Fruit Ninja. Users are able to slice fruits and earn coins by simply moving their hands in mid-air.
ii. Simple 3D object viewer: A simple 3D object viewer which allows users to perform view manipulation actions by using hand motions such as pinch to zoom, swipe, rotate, dragging, etc.
iii. Google Earth plugin navigator: A simple embedded Google Earth plugin that allows users to navigate Google Earth [14] using hand motions. Different gestures such as pinch, swipe and wave are directly translated into zoom, rotate and place landmark respectively in Google Earth.

The hardware requirements are minimal; the system consists of a detector (low-cost USB webcam or depth camera), a computing apparatus (desktop), and a video display (monitor or projector), as shown in Fig. 1.

2.2 Camera module

In this module (Fig. 3), image frames are being retrieved from a USB webcam or Kinect camera, and then processed through several steps using image processing techniques. Then, the pre-processed image frame is passed to the next module for detection. Each processing step in this module is discussed in detail in the next sub sections (sections 2.2.1 to 2.2.7).

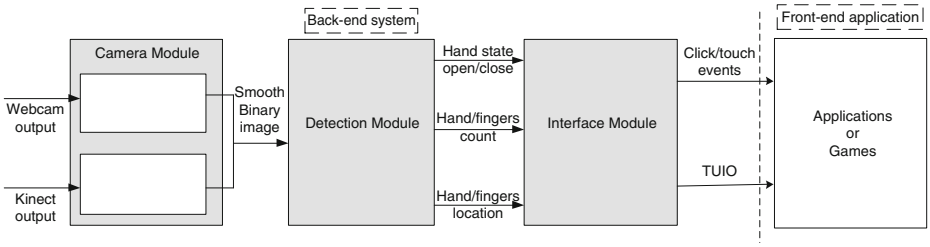


Fig. 2 Overall system architecture (*left*: back-end tracking system. *right*: front-end application)

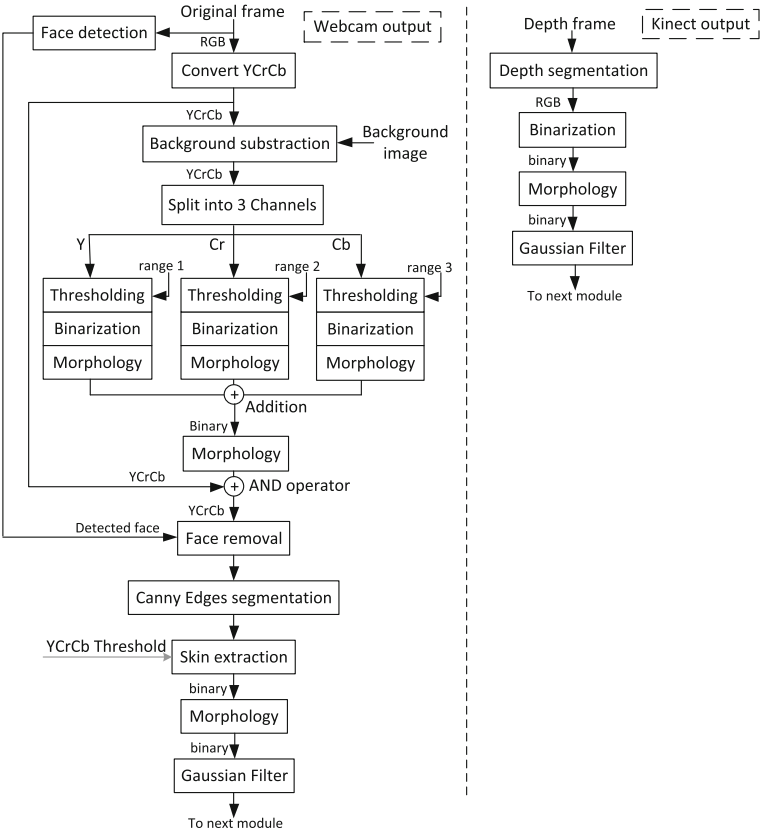


Fig. 3 Camera module architecture (*left*: Webcam output, *right*: Kinect output)

2.2.1 Connecting different cameras

Our main motivation is to design a system that is able to utilize low-cost USB webcams, which is readily available in most users' homes. However, as low-cost depth cameras are becoming more common nowadays, we also included the support for depth cameras in our system. The depth camera is slightly better in terms of hand segmentation from the background, but it suffers from lower resolution and frame rates. It results in less smoother HCI. It is also significantly more expensive than a USB webcam.

At this step, image frames are being retrieved from the camera at 30–60 frames per second (fps), depending on camera type. It is then passed to the next step for background subtraction. For the case of depth camera, only the depth segmentation step is required. It is further discussed in section 2.2.7.

2.2.2 Background subtraction

Background subtraction is performed to effectively segment the user from the background. Typical methods use a static background image and calculate the absolute difference between the current frame and the background image. Usually RGB color space is used while some studies [8, 22, 31] propose HSV or YCrCb color space as a more efficient

alternative. Nonetheless, all these color spaces still possess limitations because "color leakage" will occur if the foreground object contains colors similar to the background. In our approach, we calculate the absolute difference in YCrCb color space and split it into three channels (Y, Cr and Cb); then process each channel independently. We use a different min and max threshold for each channel, and then remove noise in each channel using a morphology operator. Finally, we merge these channels back together by an addition operator and then masking it with the original frame by using an AND operator. The output is then a YCrCb image containing only new objects that enter the scene.

2.2.3 Face removal

Our hand region extraction method is based on skin color extraction; therefore, both hand and face regions will be extracted. In some cases, it is hard to differentiate between a closed fist and the face (Fig. 4(a) and (b)). Therefore, we utilize Haar-like features [1, 10] by Viola Jones [34] to detect the face region and then remove it by simple flood fill. Hence, face region will not be extracted during the next step (skin color extraction in section 2.2.5). Haar-like features is efficient and it can detect human faces with high accuracy and performance [34]. Preparing a good Haar classifier is a time consuming task [1], so we use the well-trained classifier provided in OpenCV [3] library.

2.2.4 Canny edges segmentation

The face removal may not work perfectly under every circumstance, i.e. when the user's face is not facing the camera or when the user's hand is blocking the face. This will result in a single connected contour consisting of both face and hand (Fig. 4(c)) after the skin color extraction step in section 2.2.5. It is not desired and it makes hand shape analysis difficult. To solve this problem, we apply Canny Edges detector to find edges around the contour. Then, we can effectively separate hand contour from face contour by drawing thick lines along the contour perimeter. We also slightly increase the camera contrast and using a low threshold as the Canny Edges parameter. It will allow the Canny Edges detector to detect weak edges even though both face and hand skin color are very similar, ensuring that both contours are well separated and no leakage will occur.

2.2.5 Skin color extraction

Studies [8, 22, 31] show that YCrCb color ranges are best for representing the skin color region. It also provides good coverage for different human races. The basic value chosen in

Fig. 4 **a**, **b** Hand and face contour after skin extraction without face removal **c** Hand and face contour when the hand is blocking in front of face

our implementation is based on the value suggested by D. Chai [8], as shown in Fig. 5(a). The value is used as a threshold to perform skin color extraction in YCrCb color space and it is very efficient in covering a wide range of skin colors. However, it causes any object that contains a color similar to skin such as orange, pink and brown to be falsely extracted. Hence, we modified the value by setting a narrow default range (Fig. 5(b)) so that it will efficiently extract only the skin region but not others objects. Since the modified threshold may not suitable for different skin colors, we provide a simple one-click calibration mechanism to overcome this limitation. It requires the user to place their hand in front of the camera and click on the area of hand, and then a coarse range will be automatically determined by the system. Scrollbars are also provided to allow users to fine-tune the parameters for yielding a better skin extraction result.

2.2.6 Morphology operations and image smoothing

In order to remove noise efficiently, we apply a morphology Opening operator (Erosion followed by Dilation) in several stages; during background subtraction and after skin extraction. After that, we apply a Gaussian filter and threshold binarization with proper value to smooth the contours and remove pixel boundary flickers.

2.2.7 Depth segmentation

With the depth information from the depth camera (Kinect), we can easily remove the background and keep the hand contours only by applying a fixed depth threshold. Anything beyond this depth will be discarded and not taken into consideration when performing hand detection in the next module. Hence, steps from section 2.2.2 to 2.2.5 can be omitted.

2.3 Detection module

The output from the camera module is a binary image with smoothed and polished hand contours. In this module (Fig. 6), more information will be extracted, such as hand locations, hand open/close state, finger counts, fingers locations, and fingers directions.

2.3.1 Contour extraction and polygon approximation

The output from the previous step is simply a binary image consisting of white blobs and black background. The white blobs, also known as contours, must be extracted first. A contour is a list of points that represent a curve in an image. By setting a minimum size threshold, we perform blob pruning to remove contours with very small areas as those are probably unwanted noise. It is also common to approximate a contour representing a polygon using another contour having fewer vertices. Hence, we apply polygon approximation on the extracted contour to make it

Fig. 5 Default threshold value for YCrCb skin extraction **a** D. Chai **b** modified

$$77 \le Cb \le 127 \ \textit{and}\ 133 \le Cr \le 173 \qquad \text{(a)}$$

$$54 \le Y \le 163$$
$$131 \le Cr \le 157$$
$$110 \le Cb \le 135 \qquad \text{(b)}$$
$$\textit{where}\ Y, Cb, Cr = [0, 255]$$

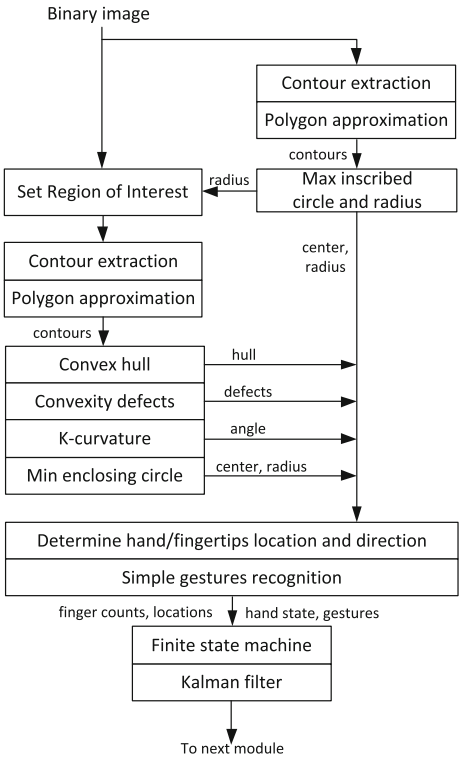


Fig. 6 Detection module architecture

more suitable for shape analysis in the next step. Simple contours do not give much information about a hand, but we can find the interior contours and combine with other information to determine specific hand gestures, which will be discussed in section 2.3.6 (OK and O sign).

2.3.2 Palm center and radius determination

Palm center is an area determined as the maximum inscribed circle inside the contour (Fig. 7(a) and (b)). It calculates the shortest distance of each point in the contour to the contour perimeter, and the point with largest distance is the center of the maximum inscribed circle. This process is quite computationally intensive. In order to speed up this process, first we downscale the image and further limit the region of interest (ROI) to the middle of the contour's bounding rectangle (Fig. 7(c)). Finally, we only search for every N-point instead of all points in the contour. We chose N=4 for balanced performance and accuracy.

2.3.3 Setting Region of Interest (ROI) and finding min enclosing circle

Using the radius (r_a) from the maximum inscribed circle, we limit the effective ROI to only the area surrounding the palm center, which we define as a circle with a radius of 3.5 times (r_a). This will effectively remove unwanted arm area. Then, contour extraction and polygon approximation is performed again on this smaller ROI. Finally, we can find the min enclosing circle and its radius (r_b) on this new contour. The radius (r_b) will be used for checking palm openness in section 2.3.6.

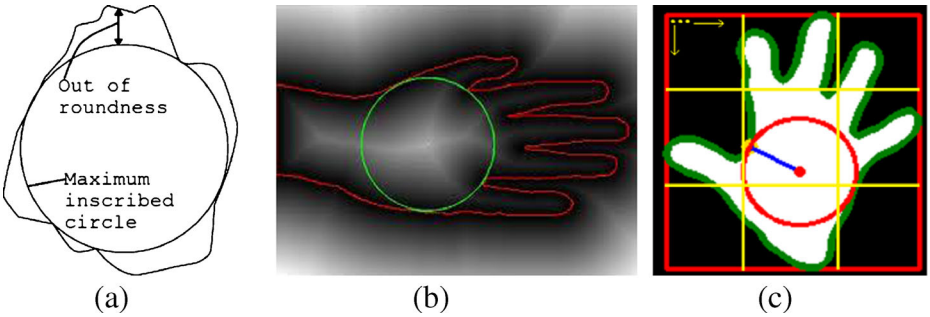


Fig. 7 **a** Maximum inscribed circle **b** Inside hand **c** Limiting region of interest

2.3.4 Convex hull and convexity defects extraction

A useful way of comprehending the shape of hand or palm is to compute a convex hull for the object and then compute its convexity defects [3], as complex hand shapes are very well characterized by such defects. Figure 8 illustrates the concept of convexity defects using an image of the human hand. The dark contour line is the convex hull around the hand. Each of the gridded regions (A, B, C … H) is the convexity defect in the hand contour relative to the convex hull. For a single convexity defect, there is a start point (p_s), depth point (p_d), end point (p_e) and depth length (l_d) as shown in Fig. 8. We find all the points and store them in an array for further analysis.

2.3.5 Determining hand/fingertips location and direction

Our proposed method evolved from previous research on hand tracking using convexity defects [25] and K-curvature [15, 24, 30, 38]. By analyzing the contour's characteristics, we can determine precise fingers locations. It must meet several criteria below before they are considered as fingertips:

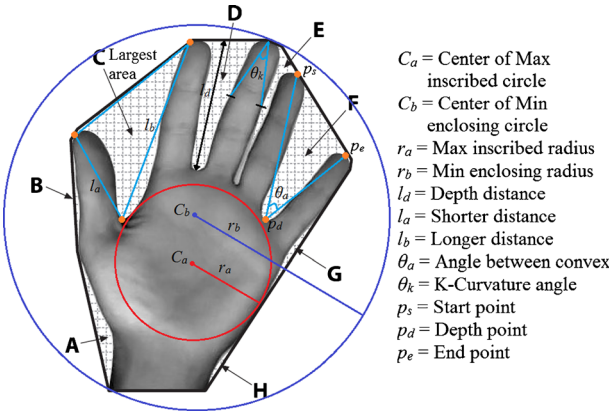


Fig. 8 Convex hull, convexity defects of hand shape, extracted from [3] Learning OpenCV

Fig. 9 Hand gestures supported in our system (from *left* to *right*: Open, Close, Claw, OK, O, Gun, Pinch, Flat, Pointing)

i. Depth of each defects (l_d) must be longer than palm center radius (r_a) but shorter than min enclosing circle radius (r_b), $r_a < l_d < r_b$
ii. Angle (θ_a) between start point (p_s) and end point (p_e) must be less than 90°, θ_a<90
iii. Local minimal K-curvature (θ_k) of point must be lower than 60°, θ_k<90

First, we find convexity defects that have a depth (l_d) longer than the palm radius (r_a) but shorter than (r_b) and the angle (θ_a) between start point (p_s) and end point (p_e) lower than 90°. Then we store its start point (p_s), depth point (p_d) and end point (p_e) into an array (A_p). We remove one of the neighbor points that are very close to each other. From this point, we search bi-direction (forward and backward) along the hand contour and compute the K-curvature (θ_k) of each point, which is the angle between the two vectors $[C_i(j), C_i(j - k)]$ and $[C_i(j), C_i(j + k)]$. (20 points each in direction and k is a constant of 30). The idea is that contour point with a small K-curvature (θ_k) will represent a potential peak or valley. Since our method only computes K-curvature for points near to convex hull points, we can reduce the computational cost and effectively remove the valley points. Then we find the point with local maxima peak angle from these 20 points to ensure that a more accurate fingertip location can be determined. Finally, we can determine the finger's direction by finding the line between the peak point and the middle point of $C_i(j - k)$ and $C_i(j + k)$.

To find the thumb finger, we find the convexity defects region with the largest area (area C) between all the convexity defects that have depth (l_d) longer than palm center radius (r_a) and an angle (θ_a) lower than 90°. Using the hand image in Fig. 8 as an example, it will be the shaded area C. The point with the shorter distance (l_a) to its depth point will be the thumb while the point with the longer distance (l_b) will be the index finger.

The three assumptions above work well in determining fingers, except in one case, which is when only one finger is available. If there are no convexity defects with an appropriate depth distance (l_d) in assumption (i), and no angle (θ_a) between start point (p_s) and end point (p_e) less than 90° in assumption (ii), then we proceed to assumption (iii) to find the K-curvature (θ_k) of the points of convexity defects that has (θ_a) larger than 90°.

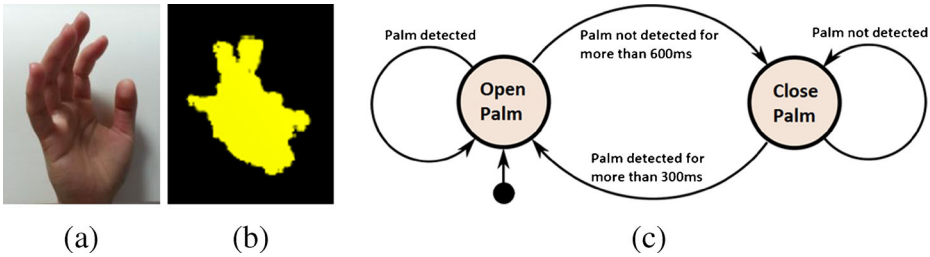


Fig. 10 **a** Self-occlusion **b** Motion blur **c** State Machine to tolerate fast changes in state

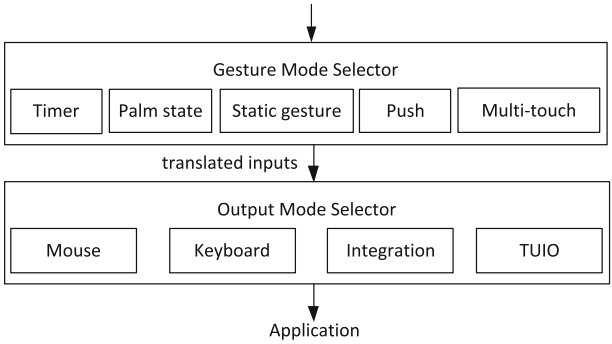


Fig. 11 Interface module architecture

2.3.6 Simple hand gesture recognition

With all the information from the previous steps, we can perform gesture recognition based on simple and heuristic assumptions. Currently, we are able to recognize several static gestures as shown in Fig. 9. They are summarized as follows:

i. Open Palm—4–5 fingers detected, the center point (C_b) of min enclosing circle is far from center point (C_a) of maximum inscribed circle.
ii. Close Palm—0–1 fingers detected, the center point (C_b) of min enclosing circle is near to the center point (C_a) of maximum inscribed circle.
iii. Claw—4–5 fingers detected, center point (C_b) of min enclosing circle is at medium distance from center point (C_a) of maximum inscribed circle.
iv. Ok Sign—3 fingers detected, an interior contour of appropriate size detected.
v. O Sign—No fingers detected, an interior contour of appropriate size detected.
vi. Gun Sign—2 fingers detected, angle (θ_a) between fingers is about 80 to 100°.
vii. Pinch—2 fingers detected, angle (θ_a) between fingers is less than 80°.
viii. Pointing—1 finger detected, no thumb detected.
ix. Finger Tap—Fingertip's z-coordinate exceeds 3 cm from z-coordinate of palm center.

2.3.7 Simple finite state machine

We apply a simple finite state machine (Fig. 10(c)) in order to tolerate minor false positives and motion blur. This is because not all five fingers may be detected at all times. It may be due to several reasons, i.e. user's hand orientation causing self-occlusion from the line of

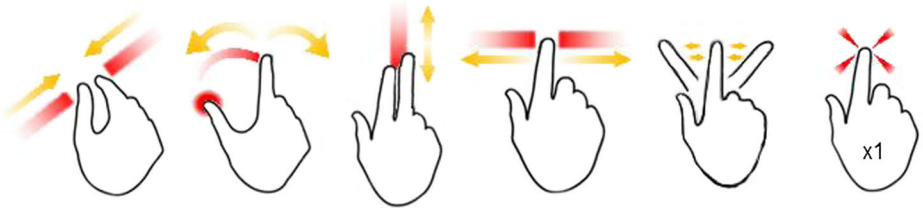


Fig. 12 Multi-touch gestures supported in our system (from *left* to *right*: Pinch to zoom, Rotate, Two points scroll, Swipe, Wave, Finger push to click)

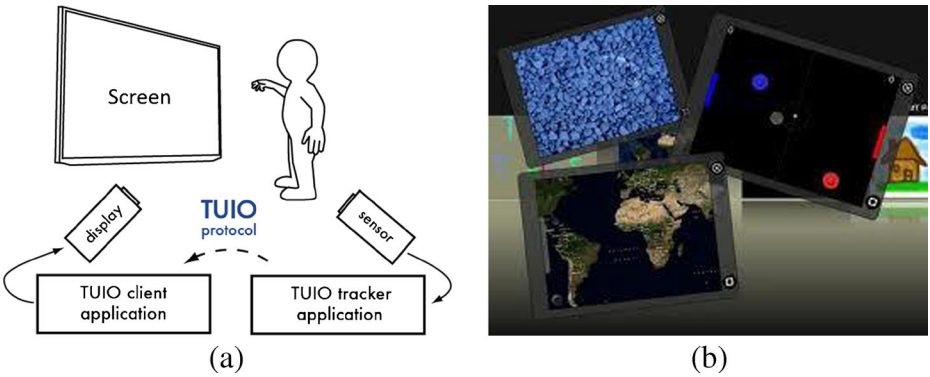


Fig. 13 **a** Modified TUIO architecture **b** Multi-touch for Java (MT4j) example

sight of camera (Fig. 10(a)), fast hand movement speed causing motion blur results in blurry hand contour to be analyzed (Fig. 10(b)). It is based on the assumption that the motion blur will only occur for a very short period and not continuously. A short delay time is added to tolerate fast changes in hand state before altering the real state. This mechanism is also applied on the face removal step.

2.3.8 Kalman filter

The fact that human hand tend to tremble when wandering in the air results in jumpy and unstable hand location. This will causes difficulties when precise control is required such as when controlling cursors or clicking small buttons in a Windows desktop environment. In order to overcome this problem, we implement a stabilizing mechanism by using Kalman filter. Kalman filter [18, 37] is a computational algorithm that implement a predictor-corrector type estimator to deduce optimum estimation of past, present and future state of a linear system in the sense that it minimizes the estimated error covariance. Therefore, by applying Kalman filter on the extracted hand locations, we can estimate and refine the optimal locations by removing unstable noises. It will result in smoother and less jumpy hand locations or trajectory paths that are more suitable for controlling applications.

2.4 Interfacing module

This module (Fig. 11) is responsible for translating the detected hand/finger points, hand gestures and hand depth into meaningful actions such as mouse clicks, keyboard key input, or multi-touch events including swipe, rotate and pinch. These actions are then passed to the appropriate application based on the chosen output method such as passing mouse and keyboard events or TUIO messages.

Table 1 Camera specifications and estimated price

Hardware Requirements	Price	Specifications
PS3 Eye Camera	USD 25	240p (60fps), 480p (60fps)
Logitech HD Pro C910	USD 80	240p (60fps), 480p (30fps), 720p (10fps)
Microsoft Kinect Camera	USD 150	480p RGB (30fps), 480p Depth (30fps)

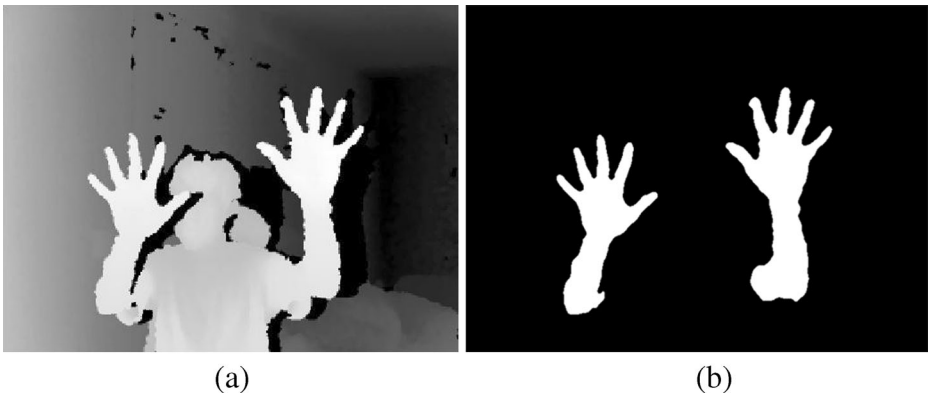


Fig. 14 **a** Original depth image from Kinect **b** Binary image after depth segmentation

2.4.1 Translating hand and fingers into functional inputs

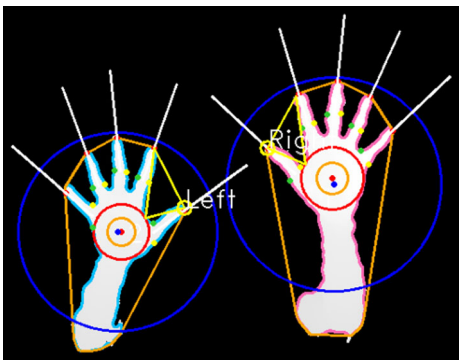
The detected hand, fingers and gestures can be translated into functional inputs based on several modes. The most common way is using a timer, e.g. when user's hand location is inside a certain predefined region for a certain period; it is considered as a click. Another common way is to detect changes in palm openness or gesture, e.g. when the palm state changes from open to close, it is interpreted as a click. Different gestures from section 2.3.6 (Fig. 9) can be mapped into different actions based on user preferences. For depth cameras, we can utilize the z-location changes in hand or fingers to simulate a click (Fig. 12(right)), which is not possible with normal webcams. Our system provides several modes and allows users to choose based on their preferences.

Furthermore, the basic hand and finger locations can be translated into advance multi-touch inputs such as in Fig. 12. Our prototype only implemented several common multi-touch actions such as pinch to zoom, swipe, rotate, scroll, drag and wave. Nevertheless, it is up to the front-end application to decide the modes by only retrieving the basic hand locations, and then translating these into advance gestures by analyzing point movement.

2.4.2 Interfacing with different applications

Our back-end system is able to interface with different applications by passing the inputs via different methods. It can be directly integrated into the front-end application as we did with

Fig. 15 Final output with all detected features



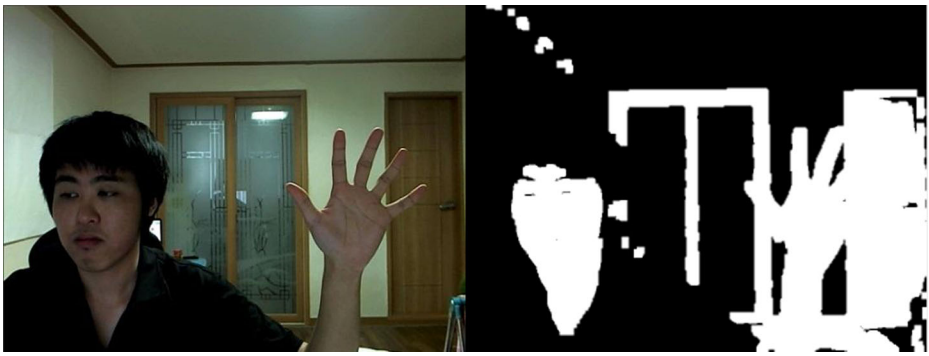


Fig. 16 Using YCrCb threshold by D. Chai [8] (*left*: original image, *right*: after skin extraction)

our sample motion game (Fig. 32). It is also able simulate mouse and keyboard events in Windows environment. By utilizing TUIO as the underlying communication protocol, it can easily pass inputs to any TUIO enabled client application such as Multi-touch for Java (MT4j) (Fig. 34). Therefore, in order for general-purpose applications to make use of the inputs, they can listen for either mouse and keyboard events or TUIO messages, and then process it as input commands.

TUIO [19] is an open framework which defines a common protocol and API for tangible multi-touch surfaces. It allows the transmission of an abstract description of interactive surfaces, including touch events and tangible object states. It encodes control data from a tracker application and sends it to any client application that is capable of decoding the protocol (Fig. 13(a)).

Multi-touch for Java (MT4j) [21] is an open source Java framework which supports different input devices with a special focus on multi-touch support including the TUIO protocol. It comes with prebuilt sample applications (Fig. 13(b)) that allow users to test the multi-touch functionality.

2.5 Front-end sample applications

In order to evaluate the usefulness of our hand tracking system, we developed several sample applications including a fruit slicing motion game, a simple 3D object viewer and a Google Earth plugin navigator. The game and object viewer were developed using Microsoft XNA

Fig. 17 Using a narrower YCrCb threshold (*left*: original image, *right*: after skin extraction)

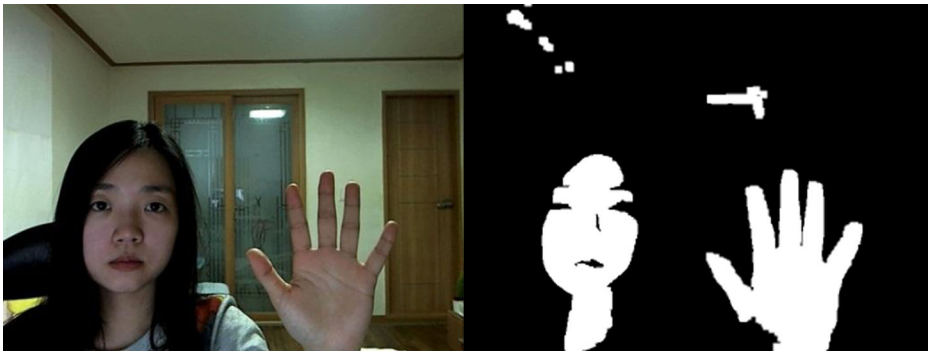


Fig. 18 Using a narrower YCrCb threshold (*left*: original image, *right*: after skin color extraction)

Game Studio while the Google Earth plugin navigator was developed using Google Earth API.

Microsoft XNA Game Studio [7] is an integrated development environment (IDE) for development of games in Microsoft Visual Studio using C# language. It simplifies game development by providing a set of tools which are easy to use and learn.

The Google Earth API [14] allows developers to embed Google Earth, a true 3D digital glove into web pages. The API allows users to draw markers and lines, drape images over the terrain, add 3D models, or load KML files while allowing developers to build sophisticated 3D map applications.

2.5.1 Fruit slicing motion game

A simple fruit slicing motion game inspired by Fruit Ninja was developed. It was designed in a way that can utilize the simple hand inputs from the detection system, such as hand location and hand state (open or close). It allows users to slice fruit virtually by moving their hands in the air (Fig. 32). In order to simplify the development process, the detection system is directly integrated into the game system. The game can support two players on the same

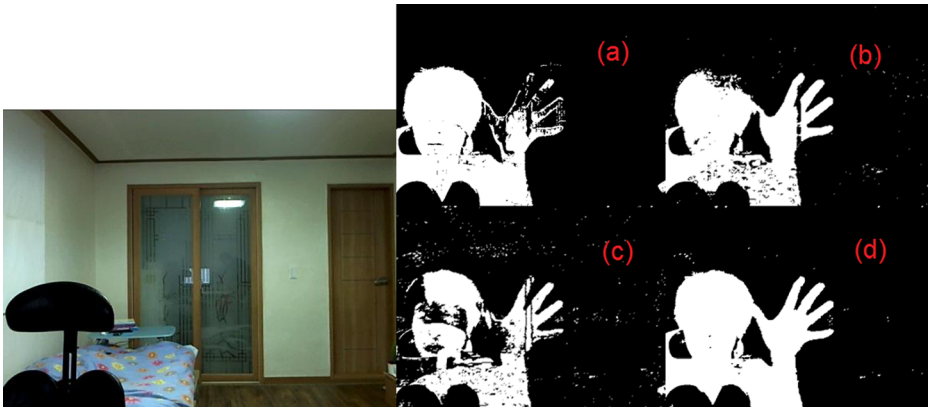


Fig. 19 (*left*) Static background image (*right*) After background subtraction with current image **a** Y channel **b** Cr channel **c** Cb channel **d** Combine 3 channels by addition operation

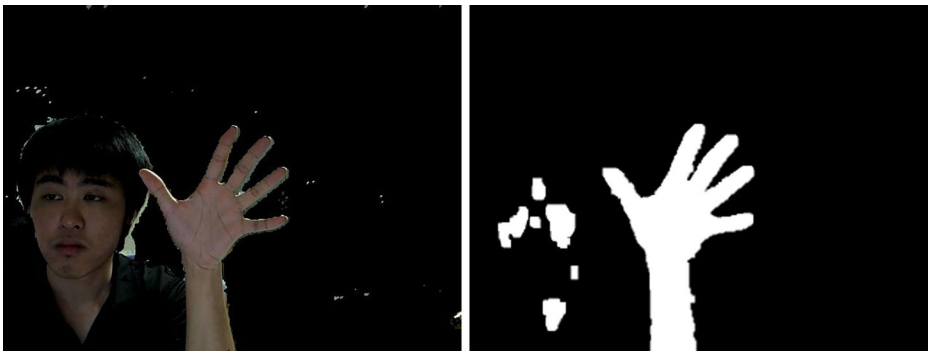


Fig. 20 (*left*) using combined mask from Fig. 18(right(d)) (right) Skin extraction using image from Fig. 19(a)

PC, or two different PCs in the same network. Each player's view is also relayed to the other player's screen in real time via split screen view.

2.5.2 Simple NUI 3D object viewer

We developed a simple 3D object viewer prototype. It allows users to view 3D models with view manipulations such as rotation, zooming, scaling, translation, etc. using hand motions (Fig. 33(a)). It is a good example of utilizing more advance inputs from the detection system such as multiple finger locations, pinch to zoom, swipe, rotate and drag actions.

2.5.3 Google Earth plugin navigator

We embedded a simple Google Earth plugin navigator in our system. It allows users to navigate the Google Earth system using hand motions (Fig. 33(b)). As it is only a proof-of-concept implementation, only common actions such as zoom, rotate and add placemark were implemented.

3 System implementation and results

3.1 Hardware and software requirements

We implemented the system in a PC with an Intel i3-2100 processor and 4GB ram. The system ran on the Windows 7 operating system. We used different cameras as shown in

Fig. 21 (*left*) Original frame (*center*) Skin color extraction using narrow threshold (*right*) Morphology operation and Gaussian smoothing

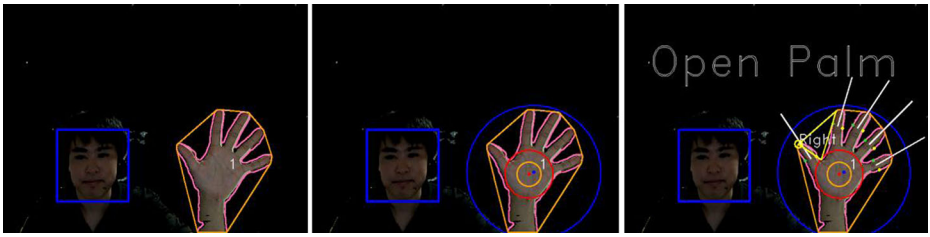


Fig. 22 (*left*) Find convex hull and convexity defects (*center*) Find max inscribed circle and min enclosing circle (*right*) Find fingertips location, finger direction, thumb location, differentiate right hand and recognize Open Palm gesture

Table 1. We developed our system using Visual Studio 2008 with C# as the programming language. We used EmguCV library (OpenCV wrappers for C#) for the image processing portion. CL-Eye Platform Driver and OpenNI framework were used for connecting the PS3 Eye Camera and Kinect camera. TUIO framework was used for sending TUIO messages between applications. The sample fruit motion game and 3D object viewer were developed with Microsoft XNA Game Studio while the Google Earth plugin navigator was developed using the Google Earth API.

3.2 Back-end system

3.2.1 Camera output

Figure 14(a) shows the original output from the Kinect camera, which is a depth image. It is then passed through the camera module for depth segmentation, binarization, morphology and Gaussian smoothing. The result is Fig. 14(b). Then, it is passed to the detection module for hand tracking and gesture recognition as discussed in section 2.3. The result is shown in Fig. 15.

Figures 16 and 17 show the difference between using a wide and a narrow YCrCb threshold values. We can see that a background with similar colors as skin is not extracted when using a narrow threshold. In Fig. 18, we can see that both the face region and hand region are equally extracted on user with different skin color, but the background with similar color is not detected.

Figure 19 (left) shows the static background image that is used for background subtraction from the current frame. Figure 19 (right) shows that after background subtraction, it is split into three channels and each channel is processed independently

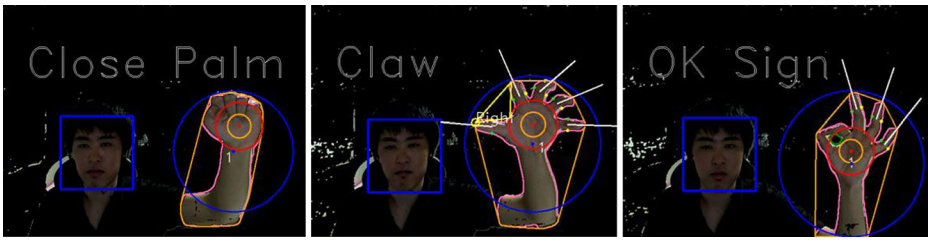


Fig. 23 (*left*) Close Palm gesture (*center*) Claw gesture (*right*) OK Sign gesture

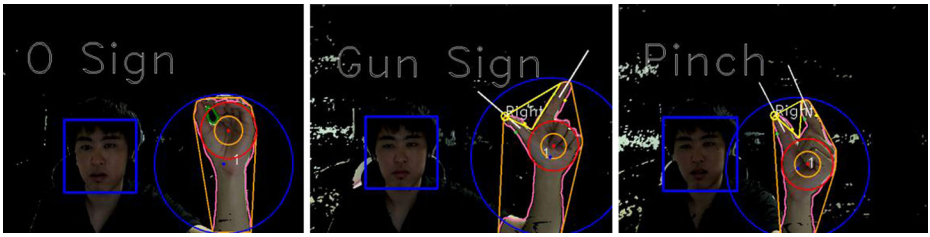


Fig. 24 (*left*) O Sign gesture (*center*) Gun Sign gesture (*right*) Pinch gesture

before they are recombined. We can see how each channel compensates for each other in Fig. 19(a, b, c) and results in Fig. 19(d). It solved the color leakage problem existing in typical background subtraction methods. Finally, using the combined binary image as a mask, efficient background subtraction can be performed (Fig. 20 (left)). Then, skin color extraction on this image yields the result in Fig. 20 (right).

Figures 21 and 22 show our hand detection method in a step-by-step manner (from left to right) based on our proposed method in section 2.3.

Figures 23, 24 and 25 show that our system is able to recognize different static gestures based on our proposed method in section 2.3.6. In Fig. 25(right), fingers that are tapped forward are highlighted in cyan color while normal fingers are highlighted in red color.

Figure 26 (right) shows the YCrCb image with Canny Edges detected. After the skin extraction method, it will yield the result shown in Fig. 27 (left). After the contour extraction step, the face region is effectively removed (Fig. 27 (center)), even though Haar-like features face removal is not working due to occlusion. Figure 28 shows that even though both hands are overlapping each other, with the help of Canny Edges segmentation, we can separate the two hand contours effectively.

Figure 29 shows that our system is able to locate the thumb finger and differentiate between the left and right hands even though the hands are in a weird position.

By utilizing the Kalman filter, hand locations can be stabilized as shown in Fig. 30. This makes it more usable in terms of applications and game control. Our future works plan to recognize these motion gestures using the Hidden-Markov model (HMM).

Figure 31 shows our simple one-click calibration method. Users only need to click once on the palm area to retrieve the suitable YCrCb threshold. Users can further fine-tune the value for better results using the scrollbars provided.

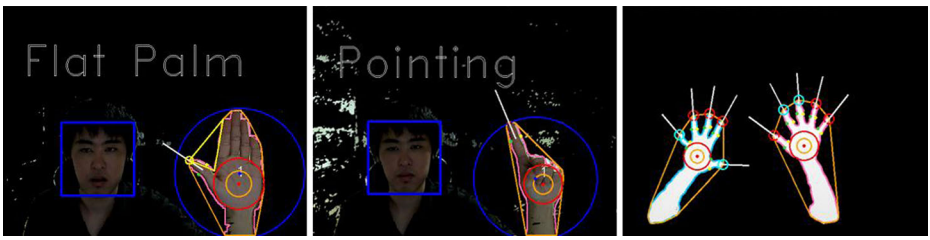


Fig. 25 (*left*) Flat Palm gesture (*center*) Pointing gesture (*right*) Tap fingertip forward to click (clicked: *cyan* fingertips, not clicked: *red* fingertips)

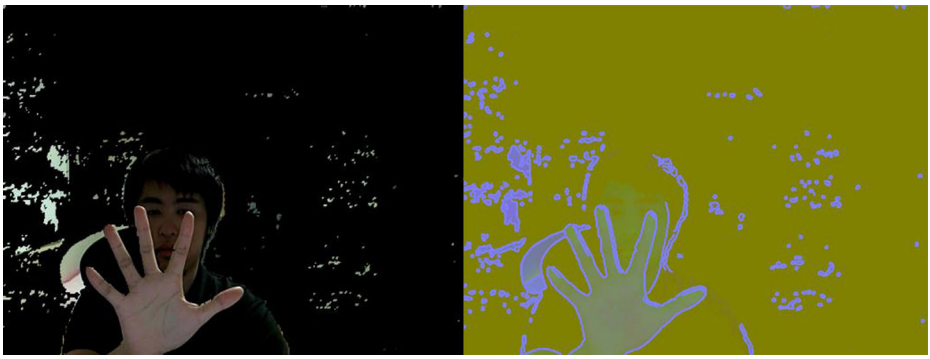


Fig. 26 (*left*) Original frame (*right*) Convert to YCrCb image and addition with Canny Edges

3.3 Front-end system

Figures 32 and 33 show screenshots of our sample games (Fruit slicing) and applications (Simple 3D object viewer and Google Earth navigator), respectively. Figure 34 shows that our system is able to interface with third party applications using the TUIO protocol. All ten finger points can be used to perform multi-touch operations.

4 Discussions

In this section, we will discuss the accuracy and stability of our system. We also discuss limitations of our implementation. Finally, we evaluate the system based on user accuracy and user feedback.

4.1 Architecture advantages and benefits

The common method for background subtraction suffers from color leakage even when the background color is only slightly similar to human skin color. Our method does not suffer from this limitation because we split and process each channel independently, as shown in Fig. 19(right). Additionally, we also use a narrow YCrCb threshold to extract the skin regions only, as shown in Figs. 17 and 18. We provide a simple one-click calibration method (Fig. 31) that allows users to recalibrate the value

Fig. 27 (*left*) Skin extraction after Canny Edges segmentation (*center*) Contour extraction, keeping only the largest contour, effectively removing face region (*right*) Hull extraction without face

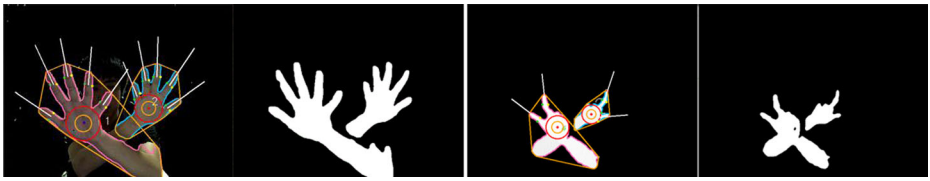


Fig. 28 Segmenting hand crossing each other with Canny Edges (*left*) webcam (*right*) Kinect

to suit different skin colors. By combining this narrow threshold and our background subtraction method, we can effectively remove background areas that contain colors similar to skin color, while avoiding color leakage. However, this approach is more sensitive to lighting changes and will introduce some noise. Therefore, we solve this problem by using morphology Opening operator to remove the noise.

Our skin color extraction method is similar to previous approaches [8], except for the usage of a narrower range. Despite using a narrow range, the face region is still extracted. Hence, we use Haar-like features [34] face detection to effectively remove the face region (Fig. 22(left)). Since it is based on feature detection, face removal will not work well if the hand is blocking the face region. This would cause a connected big contour (Fig. 4(c)) to be extracted by our skin extraction method. Hence, we apply Canny Edges to effectively separate the hand contour from the face contour, as shown in Figs. 27 and 28.

Our hybrid hand and fingers tracking method is based on hand shape analysis that evolved from previous studies on Convexity Defects, K-Curvature and maximum inscribed circles. With this information, we can find hand and fingertip locations, direction and thumb location. Then, using the thumb location, we can differentiate between the left and right hands by calculating the cross product between vectors (Fig. 29). One of the main advantages of our proposed solution is that users are not required to wear long sleeve shirts. As the vision-based approach will inevitably suffer from motion blur and self-occlusion, we apply a simple Finite State Machine to tolerate fast changes in hand state. We also apply Kalman filter (Fig. 30) to stabilize the detected hand and fingers location. This makes the system more usable in applications that require precise control. With simple assumptions discussed in section 2.3.6, our system is able to recognize different static hand gestures, as shown in Figs. 22, 23, 24 and 25.

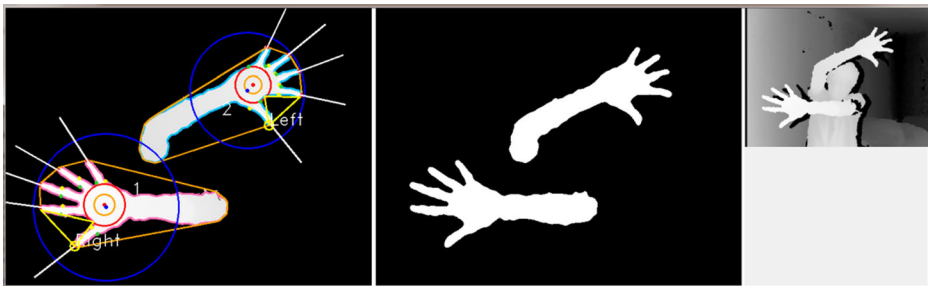


Fig. 29 Differentiating left and right hand by identifying the thumb in counter-clockwise manner

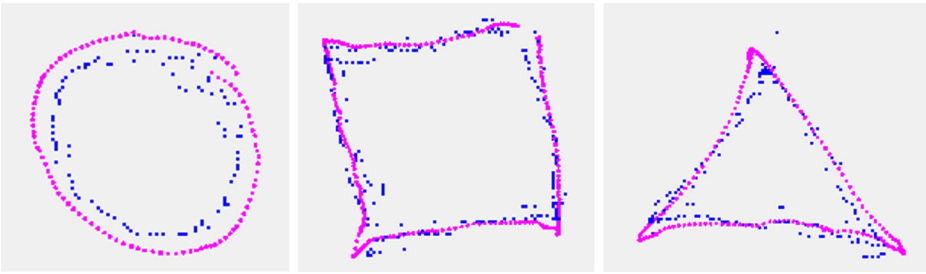


Fig. 30 Kalman filter for stabilizing points (original: *blue*, stabilized: *purple*)

In this paper, we do not limit the scope of our research to hand tracking and recognition methods only. Instead, we effectively translate the detected results into functional inputs for system and game control. We also provide samples that demonstrate how we can utilize the detected results for interactive applications. Our system is able to interface with other third party applications easily via different methods, such as passing mouse and keyboard events or TUIO messages.

Our system is also able to support both regular USB webcams and Microsoft Kinect depth cameras. The main differences between the two cameras (Table 1) can be summarized as follows:

i. Kinect is more expensive than regular USB webcams (150usd vs. 25usd)
ii. Kinect provides valuable depth information. It allows accurate push to click or finger tap to click as shown in Fig. 25 (right). It also allows efficient hand segmentation from noisy backgrounds (Figs. 14 and 15). Background subtraction, skin extraction, or face removal can be omitted.
iii. Kinect depth sensor is limited to 30 fps while regular webcam can achieve 60fps for smoother interactions.

4.2 Limitations

4.2.1 Hybrid algorithm

Our hybrid hand and finger tracking method evolved from previous research, mainly on convexity defects, K-curvature and maximum inscribed circle. While it is good in characterizing hand shape, it is still not perfect because some non-hand objects may possess similar characteristic to the hand.

Fig. 31 One click calibration method and scroll bar to fine tune threshold value

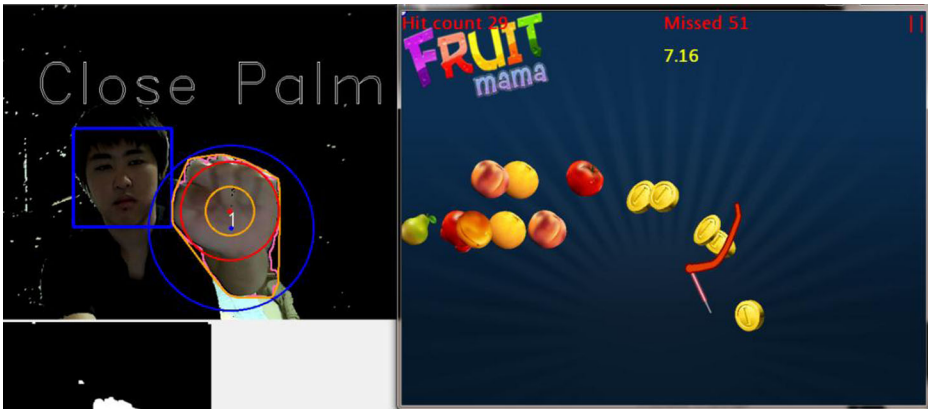


Fig. 32 User playing sample fruit game with hand motion

4.2.2 Self-occlusion

As our system only uses a single camera placed in front of the user, occlusions are inevitable. This can occur when one hand is blocking in front of the other hand, or when the hand is rotated along the y-axis. In a simple experiment, our system can tolerate up to 30° of rotation in both directions. In future work, we aim to overcome this limitation by incorporating additional cameras. This will also provide valuable depth information by calculating the disparity between cameras.

4.2.3 Environment lighting and screen reflection

As the hand tracking method on regular webcam is mainly based on skin color extraction, environmental lighting conditions are important factors in affecting the system accuracy. The system works best in indoor environment with moderate lighting condition but it will perform poorly under extremely dark or bright condition.

Our system setup requires users to stay about 40 cm to 1 m in front of the camera. The most common setup is to place the camera on top of a monitor screen (Fig. 1). However, this setup will cause reflection of screen light on the user's hand. The reflection intensity greatly depends on environmental lighting and current screen brightness. Under low light condition

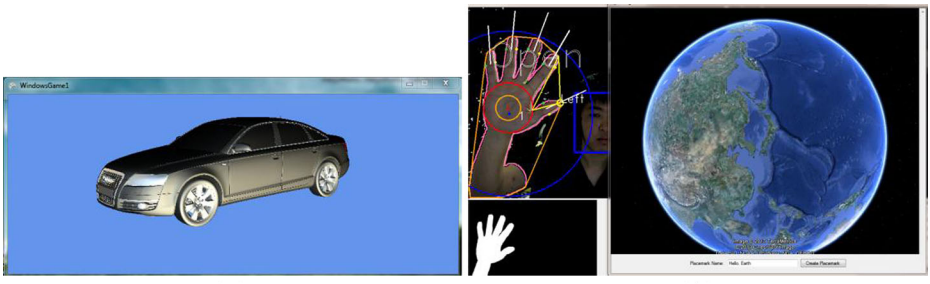


Fig. 33 **a** User testing 3D object viewer **b** Google Earth plugin navigator

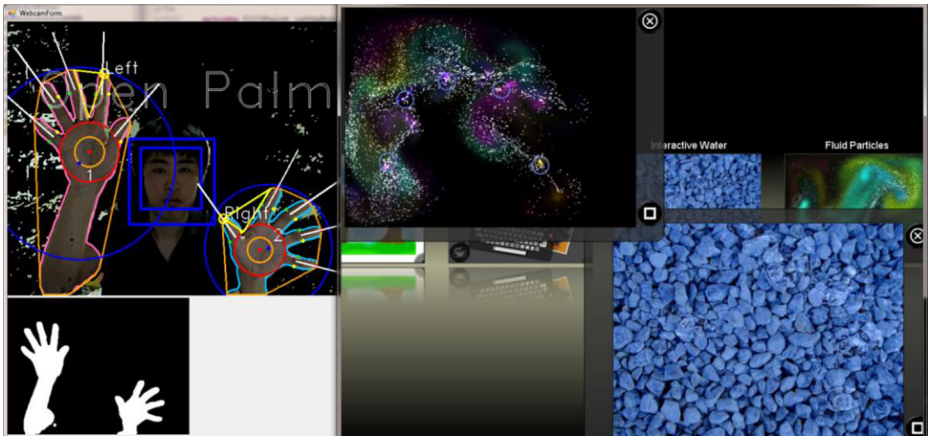


Fig. 34 User testing MT4j application using ten fingertips for multi-touch via TUIO protocol

and whenever the screen color changes greatly, it will cause our vision-based skin color extraction method to perform poorly.

One way to overcome this issue is by setting a large range on the YCrCb threshold value. However, this will cause any object with color similar to skin color to be falsely detected. Both limitations discussed above do not occur when using Kinect depth camera.

4.3 Evaluation

We invited ten members of our department to evaluate our prototype system. We asked test subjects to undergo an accuracy test twice (first trial and after training). Then, we asked test subjects to test our sample applications and provide feedback based on different criteria.

4.3.1 Accuracy test

We designed a simple accuracy test application that represents bulls-eye shooting, as shown in Fig. 35(left). It generates a bulls-eye at random locations for 5 s at a time.

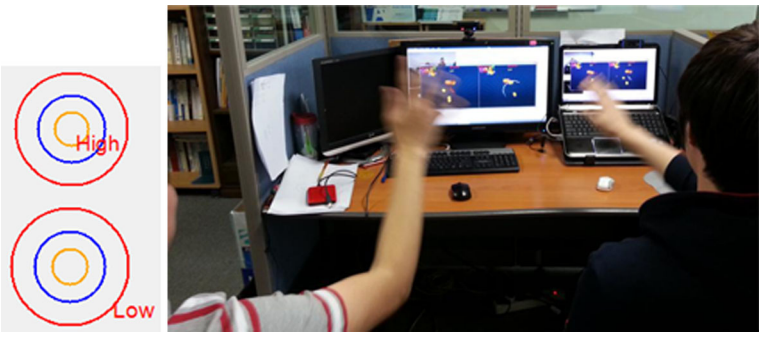


Fig. 35 (*left*) Bulls-eye for accuracy test (*right*) Two players playing sample fruit motion game on two PCs connected over a local area network

Table 2 Average test score from users (first trial)

Accuracy	Missed	High	Mid	Low	Accuracy	Missed	High	Mid	Low
User 1	14/30	2/30	11/30	3/30	User 6	15/30	2/30	8/30	5/30
User 2	8/30	3/30	13/30	6/30	User 7	8/30	0/30	12/30	10/30
User 3	14/30	3/30	6/30	7/30	User 8	15/30	1/30	8/30	6/30
User 4	10/30	5/30	9/30	6/30	User 9	9/30	3/30	9/30	9/30
User 5	9/30	3/30	9/30	9/30	User 10	14/30	2/30	3/30	11/30
Average	Missed=38.67 % High=8 % Mid=29.33 % Low=24 %								

Test subjects are required to click on the bulls-eye using their hand (close palm to click) before it disappears (timeout). Each test consisted of 30 clicks. We asked test subjects to perform the accuracy test for the first time, without any previous training. Then, they are given 5 min to become familiar with the hand tracking system. After that, they are asked to perform the accuracy test again for the second time. The results are then collected for accuracy analysis (Tables 2 and 3).

4.3.2 Users evaluation

We then asked the test subjects to evaluate our system by testing our sample applications including fruit game, 3D object viewer, Google Earth navigation, MT4J applications and controlling a cursor in the Windows desktop environment. The evaluation criteria are based on level of fun, control accuracy, intuitiveness, usability, and comfort, rated in the scale of one to five. Users' feedback are collected for analysis (Table 4).

Table 3 Average test score from users (after training)

Accuracy	Missed	High	Mid	Low	Accuracy	Missed	High	Mid	Low
User 1	5/30	7/30	15/30	3/30	User 6	3/30	8/30	8/30	11/30
User 2	3/30	8/30	12/30	7/30	User 7	5/30	7/30	11/30	7/30
User 3	6/30	8/30	10/30	6/30	User 8	8/30	4/30	6/30	12/30
User 4	1/30	7/30	13/30	9/30	User 9	2/30	6/30	9/30	13/30
User 5	3/30	6/30	9/30	12/30	User 10	4/30	3/30	11/30	12/30
Average	Missed=13.33 % High=21.33 % Mid=34.67 % Low=30.67 %								

Table 4 Average feedback score from users

	Fun	Accuracy	Intuitiveness	Usability	Comfort
(A) Fruit Slicing Motion Game	4.7/5	4.7/5	3.9/5	3.8/5	3.5/5
(B) 3D object viewer control	3.8/5	3.4/5	3.5/5	3.0/5	3.4/5
(C) Google Earth navigation	3.5/5	3.1/5	3.4/5	2.4/5	2.8/5
(D) MT4J applications	4.2/5	3.5/5	4.1/5	3.8/5	3.3/5
(E) Controlling Windows cursor	2.5/5	2.5/5	3.1/5	2.9/5	2.0/5

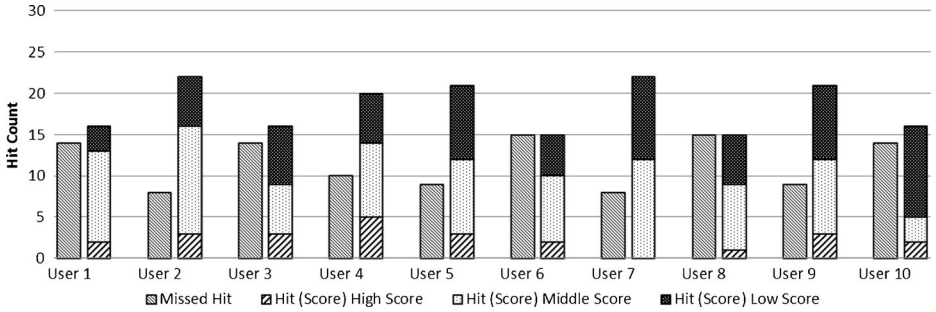


Fig. 36 Average test score from users (first trial)

4.3.3 Discussions

From the results of the accuracy test (Figs. 36 and 37), we observe that users' accuracy score greatly improved on the second trial. The mean correctness went from 61.33 % (first trial) to 86.66 % (second trial) with precision of 21.33 % being high, 34.67 % being middle and 30.67 % being low. It proved that the system is easy to learn and can be usable in HCI after simple training.

From the user evaluation results (Fig. 38), we can observe that users are very satisfied with the fruit slicing motion game, but not at controlling a Windows cursor. This is because precise and stable mouse control is required in a Windows OS environment and can hardly be achieved via a vision-based approach because of human nature, where the user's hands tend to tremble when held/moved in the air. Although we solved the unstable cursor problem by applying Finite State Machine and Kalman filter, users tend to compare the usability with typical mouse

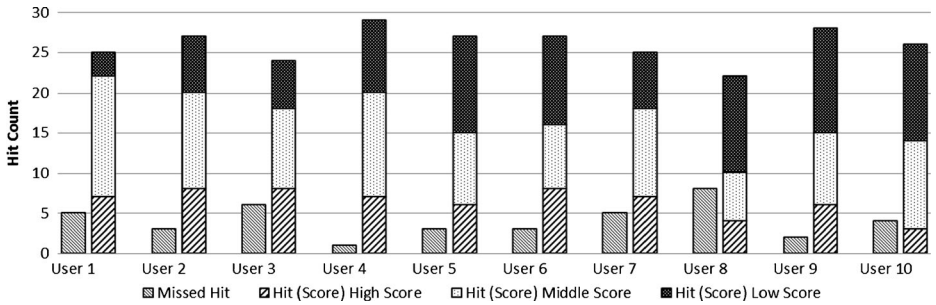


Fig. 37 Average test score from users (after training)

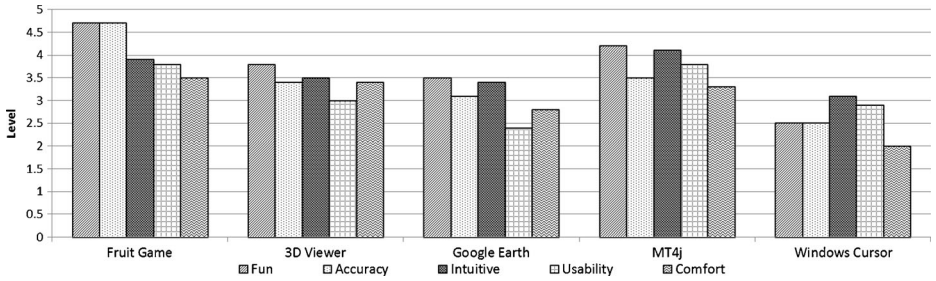


Fig. 38 User feedback score for different applications

control, which is much more stable and precise. As both the simple 3D object viewer and Google Earth plugin navigator are only proof-of-concept implementations, they support only a limited set of hand gestures and features. Hence, we can expect moderately low usability scores.

5 Conclusions and future work

In this paper, we developed a robust marker-less hand/finger tracking and gesture recognition system for human-computer interaction using low-cost hardware. Users can interact with PC applications or games by performing hand gestures instead of relying on physical controllers. It is therefore more natural and intuitive. This kind of interactive gaming is also much cheaper and flexible compared to console motion gaming.

Our solution is intuitive and low-cost. It can be easily calibrated to suit any skin color and work against almost any background. It can track hand and finger locations in real time and is able to recognize simple hand gestures. Our method is also able to differentiate the left hand from the right hand, even when they are crossing each other. Our solution has overcome several limitations existed in previous studies such as the requirement for long sleeves, backgrounds without skin color objects and non-overlapped hand and face regions in front of the camera's point of view. We aim to provide a thorough benchmark comparison between previous methods and our method in the future works.

Nonetheless, our system still suffers from several limitations as discussed in section 4.2. Our future work aims to overcome those limitations and improve the hand/finger tracking algorithm. We also aim to improve the stability and accuracy of our tracking system where high precision fingertip locations can be obtained. It will allow advanced NUI applications such as floating virtual keyboard to be more practical, instead of just being a gimmick. We also plan to incorporate additional cameras into our system. It can easily solve the problem of self-occlusion and would also provide valuable depth information that allows us to represent detected hands into a 3D model. We also plan to recognize more advanced gestures using Neural-Network (NN) or Hidden-Markov Model (HMM). Finally, we target to extend our domain scenarios and apply our tracking mechanism into digital TV and mobile devices. A real demo of the system can be located in [33].

Acknowledgments This work was supported in part by the National Research Foundation of Korea under Grant 2011-0009349. The authors wish to thank Mr. Dylan Zhu for the language editing. Thanks are also due to all reviewers for their comments and recommendations, which have greatly improved the manuscript.

References

1. Barczak ALC, Dadgostar F (2005) Real-time hand tracking using a set of cooperative classifiers based on Haar-like features. Res Lett Inf Math Sci 7:29–42
2. Benedetti W (2009) Motion controls move games into the future. [Online] http://www.msnbc.msn.com/id/31200220/ns/technology_and_science-games/
3. Bradski G, Kaehler A (2008) Learning OpenCV: computer vision with the OpenCV library. O'Reilly Media, Incorporated
4. Bretzner, L, Laptev I, Lindeberg T (2002) Hand gesture recognition using multi-scale colour features, hierarchical models and particle filtering. Automatic face and gesture recognition, 2002. Proceedings. Fifth IEEE International Conference on (pp. 423–428). IEEE
5. Buchmann V et al (2004) FingARtips: gesture based direct manipulation in Augmented reality. Proceedings of the 2nd international conference on Computer graphics and interactive techniques in Australasia and South East Asia (pp. 212–221). ACM

6. Burns A-M, Mazzarino B (2006) Finger tracking methods using eyesweb. Gesture in human-computer interaction and simulation 156–167
7. Carter C (2007) Microsoft® xna™ unleashed: graphics and game programming for xbox 360 and windows. Sams
8. Chai D, Ngan KN (1999) Face segmentation using skin-color map in videophone applications. IEEE Trans Circ Syst Video Technol 9(4):551–564
9. Chen Q (2008) Real-time vision-based hand tracking and gesture recognition. University of Ottawa
10. Chen, Q, Georganas ND, Petriu EM (2007) Real-time vision-based hand gesture recognition using haar-like features. Instrumentation and Measurement Technology Conference Proceedings, 2007. IMTC 2007. IEEE (pp. 1–6). IEEE
11. Dardas NH, Alhaj M (2011) Hand gesture interaction with a 3D virtual environment. Int J ACM Jordan 2(3):186–194
12. Dardas NH, Georganas ND (2011) Real-time hand gesture detection and recognition using bag-of-features and support vector machine techniques. IEEE Trans Instrum Meas 60(11):3592–3607
13. Dias JMS, Nande P, Barata N, Correia A (2004) OGRE-open gestures recognition engine. In: Computer graphics and image processing, 2004. Proceedings. 17th Brazilian Symposium on (pp. 33–40). IEEE
14. Google Earth API, https://developers.google.com/earth/
15. Han SI, Mi JY, Kwon JH, Yang HK, Lee BG (2008) Vision based hand tracking for interaction
16. Hasan MM, Mishra PK (2012) Real time fingers and palm locating using dynamic circle templates. Int J Comput Appl 41(6):33–43
17. Hürst W, van Wezel C (2012) Gesture-based interaction via finger tracking for mobile augmented reality. Multimed Tools Appl 62(1):233–258
18. Kalman RE (1960) A new approach to linear filtering and prediction problems. J Basic Eng 82(1):35–45
19. Kaltenbrunner M et al (2005) TUIO: a protocol for table-top tangible user interfaces. Proc. of the The 6th Int'l Workshop on Gesture in Human-Computer Interaction and Simulation
20. Keskin C, Erkan A, Akarun L (2003) Real time hand tracking and 3d gesture recognition for interactive interfaces using hmm. ICANN/ICONIPP 2003:26–29
21. Laufs U, Ruff C, Zibuschka J (2010) Mt4j-a cross-platform multi-touch development framework. arXiv preprint arXiv:1012.0467
22. Mahmoud TM (2008) A new fast skin color detection technique. World Acad Sci 501–505
23. Mahmoudi F, Parviz M (1993) Visual hand tracking algorithms. In: Geometric modeling and imaging–new trends, 2006 (pp. 228–232). IEEE
24. Malik S, Laszlo J (2004) Visual touchpad: a two-handed gestural input device. Proceedings of the 6th international conference on Multimodal interfaces (pp. 289–296). ACM
25. Manresa C, Varona J, Mas R, Perales F (2005) Hand tracking and gesture recognition for human-computer interaction. Electron Letters Comput Vis Image Anal 5(3):96–104
26. Oka K, Sato Y, Koike H (2002) Real-time fingertip tracking and gesture recognition. IEEE Comput Graph Appl 22(6):64–71
27. Pavlovic VI, Sharma R, Huang TS (1997) Visual interpretation of hand gestures for human-computer interaction: a review. IEEE Trans Pattern Anal Mach Intell 19(7):677–695
28. Quam DL (1990) Gesture recognition with a DataGlove. Aerospace and Electronics Conference, 1990. Proceedings of the IEEE 1990 National (pp. 755–760). IEEE
29. Rehg JM, Kanade T (1994) Digiteyes: vision-based hand tracking for human-computer interaction. Motion of non-rigid and articulated objects, 1994. Proceedings of the 1994 IEEE Workshop on (pp. 16–22). IEEE
30. Segen J, Kumar S (1998) Human-computer interaction using gesture recognition and 3D hand tracking. Image Processing, 1998. ICIP 98. Proceedings. 1998 International Conference on (pp. 188–192). IEEE
31. Singh SK, Chauhan DS, Vatsa M, Singh R (2003) A robust skin color based face detection algorithm. Tamkang J Sci Eng 6(4):227–234
32. Takahashi M et al (2011) Human gesture recognition system for TV viewing using time-of-flight camera. Multimed Tools Appl 1–23
33. Video demo of hand tracking system, http://www.youtube.com/user/tcboy88/videos?
34. Viola P, Jones MJ (2004) Robust real-time face detection. Int J Comput Vis 57(2):137–154
35. Von Hardenberg C, Bérard F (2001) Bare-hand human-computer interaction. Proceedings of the 2001 workshop on Perceptive user interfaces (pp. 1–8). ACM
36. Wang RY, Popović J (2009) Real-time hand-tracking with a color glove. ACM Transactions on Graphics (TOG). Vol. 28. No. 3. ACM
37. Welch G, Bishop G (1995) An introduction to the kalman filter. Technical report, UNC-CH Computer Science Technical Report 95041
38. Zabulis X, Baltzakis H, Argyros A (2009) Vision-based hand gesture recognition for human-computer interaction. The Universal Access Handbook. LEA

Hui-Shyong Yeo received his Bachelor degree of Electronics majoring Computer from Multimedia University(MMU), Malaysia in 2011. Presently, he is pursuing the Master of Science in Ubiquitous IT at Dongseo University, South Korea. His research interest includes Human Computer Interaction (HCI), Natural User Interface (NUI), mobile applications and cloud storage.

Byung-Gook Lee received his BS degree in Mathematics from Yonsei University in 1987, the MS and PhD degrees in Applied Mathematics from Korea Advanced Institute of Science and Technology Korea in 1989 and 1993, respectively. From 1993 to 1995, he had worked for DACOM Corp. R&D Center as a senior engineer. Since 1995, he has been with Dongseo University, Korea, where he is currently a professor in the Division of Computer and Information Engineering. His research interests include Computer Aided Geometric Modeling, Computer Graphics, and Computer Vision.

Hyotaek Lim received his BS degree in Computer Science from Hongik University in 1988, the MS degree in computer science from POSTECH and the PhD degree in computer science from Yonsei University in 1992 and 1997, respectively. From 1988 to 1994, he had worked for Electronics and Telecommunications Research Institute as a research staff. Since 1994, he has been with Dongseo University, Korea, where he is currently a professor in the Division of Computer and Information Engineering. His research interests include ubiquitous and mobile networking, storage area networks, and cloud computing.