SIViP
DOI 10.1007/s11760-013-0462-1

ORIGINAL PAPER

A HCI interface based on hand gestures

Chuan-Kai Yang · Yu-Chun Chen

Received: 23 August 2012 / Revised: 11 December 2012 / Accepted: 13 March 2013

Abstract *Human–computer interaction*, or *HCI* for short, concerning how people interact with computers, has long been an important and popular research field. Though not completely realistic, fancy HCI applications such as those shown in the science fiction movies *Minority Report* and *Iron Man* have impressively demonstrated the potential and trend of HCI technologies that will be very soon made available. As one can very often observe, compared with traditional keyboard/mouse interfaces, the exclusive use of hands has distinguished itself by enjoying a more intuitive and natural way for communication. Furthermore, the increasingly popular concept of *ubiquitous computing* has called for convenient and portable input devices, thus making hand gesture inputs even more attractive. For example, a smart phone equipped with the capability of hand gesture recognition could be a good input substitute for its intrinsically small touch screen or keypad. Rather than *data gloves*, which transfer hand gestures through relatively expensive electronic devices, we are more interested in recognizing the gestures of a bare hand. In this regard, there exist works that can track a 2D articulated hand model. In this paper, we make further improvement in computation efficiency and propose novel interfaces to be coupled with the hand tracking system for more user-friendliness.

Keywords Hand tracking · Hand gesture recognition · Human–computer interaction · Skin detection

C.-K. Yang (✉) · Y.-C. Chen
Department of Information Management,
National Taiwan University of Science and Technology,
43, Section 4, Keelung Road, Taipei 10607, Taiwan
e-mail: ckyang@cs.ntust.edu.tw; M9609114@mail.ntust.edu.tw

1 Introduction

Since the invention of computers, the related technologies have been rapidly developed and advanced. In particular, the way we interact with computers has become more and more diversified, and thus, the emergence of the field called *human–computer interaction*, or *HCI* for short. In fact, in a broad sense, if we view many *embedded systems* as different forms of computers, the scope of HCI can include a wide range of scenarios from the music playback on a hand-held device, to the dashboard display on an airplane and so on. Keyboards and mice have long been the main devices we used to interact with computers. However, there also exist other possible interfaces/devices that can be used to communicate with computers. For example, *data gloves* is one of them, and by wearing a data glove, a user's hand gesture can be sensed and used as an input to interact with computers. Such a device can offer desired accuracy, but its relatively higher expense limits its popularity. In recent years, Nintendo's Wii Remote has become one of the most popular input devices as it can provide a "wireless" control interface through which some hand motions can be detected. As a result, it becomes possible to *remotely* control or interact with computers, and especially for gaming. Microsoft's Kinect sensor is another emerging technique that quickly gains significant popularity. By detecting depths through infrared cameras, it opens the door for capturing more motion information and thus has also been widely used for computer or game interactions. However, the need to carry an extra Wii Remote device and the current inaccuracy of Kinect sensors still leave room for further improvement on HCI, especially when more control accuracy is desired. Nevertheless, it is easy to see the exclusive use of hands in the new trends of interaction as it is quite natural for a human to communicate through hand gestures. Another observation is the popularity of portable or hand-held devices, which can be laptops, tablet PCs, or even smart phones. There are at least two things in common on these devices: a relatively smaller screen space and the equipment of a camera. Such a configuration makes a hand tracking system more attractive than before, and this is why this work is motivated. In short, the contribution of this work is to propose a hand tracking system that, based on previous work but further improves the efficiency. In addition, we developed two novel interfaces that could facilitate the mouse and keyboard operations and at the same time make the proposed system much easier to be combined with other window applications.

The rest of the paper is organized as follows. Section 2 reviews works related to this paper. Section 3 provides a background to make it easier for understanding our proposed methods. Section 4 presents our approaches for designing a hand tracking system. Section 5 describes the novel window interfaces we propose. Section 6 concludes this work and gives suggestions for possible future directions.

2 Related work

Vision-based methods for detecting or tracking objects have been important research issues. And it has become popular to make use of this technology for HCI. For example, based on the detected image or video frames, we could perform *face detection* or *human body detection* or even *activity detection*. According to the detected results, computers can then react in a proper way and thus achieve the effect of HCI. Among the vision-based technologies, hand tracking is what we concerned in this work. Generally speaking, there are at least four types of approaches, where they can be based on 3D model information, skin color detection, image difference, or machine learning algorithms.

The first type of approaches is based on a hand's 3D model information, such as the work proposed by Rehg [10]. In general, multiple cameras are needed, and thus, there is not much limitation on how a hand should behave. However, the requirement of multiple cameras and the more involved computation make such an approach relatively inappropriate for common HCI.

The second type of approaches depends on skin detection, as the appearance of skin colors could be an important clue of the existence of a hand. Based on this idea, many systems incorporate such a functionality for hand detection. For example, Manresa et al. [9] extracted the skin colors of a user during the initialization stage and then use the extracted information to further identify the locations of hands. Zaletelj et al. [17] first performed face detection, which is usually considered relatively easier, and then made use of the detected facial skin colors to help discover the hand colors. In general, in spite of its simplicity, skin color detection still suffers from the following limitations. For example, the exposure of arms or the confusion of backgrounds with skin-like colors, and so on, could make it more difficult to judge the exact locations of target hand (palm) area.

The third type of approaches is based on image difference, as proposed by [4,7,16] for hand detection. The main idea is to first maintain (derive) a background image, and then, we can calculate its difference with other images. The advantage of such an approach is its ability to locate an object in motion. In particular, in [4,7] a *color difference* color system is used as it is known to be less affected by the lighting variation. The most serious limitation of an image difference approach is on the stability of the background image, which cannot work properly if the camera is a hand-held one or if the background is not static.

The last type of approaches makes use of machine learning. Rowley et al. [11] were among the earliest ones to apply the machine learning technique on face detection. Viola et al. [13] proposed a very classical approach, called *AdaBoost*, to detect human face, and then applied a *cascading classifier* to significantly speed up the detection. Due to such a successful application of machine learning algorithm, Wu et al. [15] applied a *neural network* approach for hand detection. Fahn et al. [3] made use of *AdaBoost* to detect finger tips. Though being able to locate the finger tips of a hand, it is not always possible to derive the entire hand gesture or shape, thus limiting the freedom of hand motion. As a result, they proposed many combinations of hand gestures to overcome the limitation, at the expense of requiring a user to memorize numerous hand gestures. Therefore, its usefulness may be affected.

To sum up, there are many possible approaches that can be used for hand tracking, but each of them has its pros and cons. In this work, we adopt the idea of *contour tracking*, a simplified version of the first type of approaches, and we only need to measure the probability of each possible (hypothetical) contour through the measurement lines on the contour. Compared with many other approaches where the entire image (frame) is involved in the computation, a contour tracking framework is much faster. In addition, a successful contour tracking can also help us to understand the current hand shape and thus the desired motion as a whole.

3 Background

In the ensuing sections, we will describe the involved techniques in this hand tracking system, including *deformable templates*, *particle filtering*, and *sweep tracker*. Particle filters and deformable templates have been combined and applied on hand contour tracking [1,6,12]. Isard et al. [5] built a contour tracking framework, which combines particle filtering and deformable template into a *Condensation* algorithm, as it is abbreviated from **Con***ditional* **Dens***ity* *Propag***ation**, and

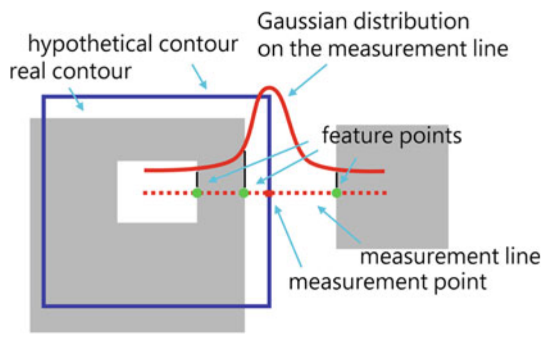

Fig. 1 The measurement method

its strength lies on its ability to track object's contour over a possibly complex background with rather limited resource.

3.1 Deformable templates

Based on the approaches proposed by Blake et al. [1], we represent the hand for tracking as a *deformable template* consisting of a set of B-splines, where each of them is controlled by a vector called *state*. Through such a mechanism, each B-spline can be made to scale, translate, rotate, etc. In the ensuing discussion, each deformable template or *contour*, represents the (hand) shape we plan to track.

3.2 Measurement method

Figure 1 briefly shows how we perform our measurement. For each *measurement point*, there will be a *measurement line*, along which we detect the features in a frame to determine the score of this measurement point. Here, the features are referring to *skin color* or *edge*, to be described detailedly in Sects. 4.3.2 and 4.3.3, respectively. The multiplication of all the scores of the measurement points on all measurement lines lead to a *fitness* value that tells how well a hypothetical contour fits the real contour. In this figure, the blue rectangle is a *hypothetical contour*, the dark gray region the *real contour*, the red dotted line a measurement line, the red line the Gaussian distribution that is used to calculate the score, the red dot the measurement point, and the green dots the detected features, respectively. As a result, each red dot will lead to a score. In practice, as the calculation of a *fitness* involves the multiplication of scores measured on many measurement lines, we apply the acceleration method proposed by Tosas [12] to pre-calculate the scores on the Gaussian distribution so that at the run time, the score of each measurement point can be obtained through a simple *table-lookup*.

3.3 Condensation algorithm

By combining the concepts of particle filtering and deformable template, the Condensation algorithm [5] treats each contour as a particle, which represents the state of a hypothetical contour x_i, and its corresponding weight π_i is the fitness of the contour. The main idea of Condensation algorithm is to repeatedly perform one of the three operations of *resampling*, *prediction* and *measurement* to "evolve particles". We now briefly describe these three operations.

Resampling At the beginning of this operation, we will perform sampling on the particle set by n times, where the probability for a particle to be chosen is equal to its weight π_i. As a result, those particles with higher weights may get selected many times, thus causing the replication of these particles in the updated particle set for the next round. On the other hand, some lower-weighted particles may not be included in the updated particle set as they are not sampled. After the resampling process, the new particles do not have their associated weights. How the new weights are obtained is described later in the *measurement* stage.

Prediction As the particles in the new particle set come from the previous phase, the *dynamics* should be put into consideration. In fact, to incorporate the dynamics means to *predict*, the future behavior of the particles involved. In this work, we adopt the *second-order auto-regressive processes (ARPS)*, proposed by Blake et al. [1], to represent the state x_t at time t by a linear combination of its previous two states, together with a Gaussian noise:

$$x_t = A_2 x_{t-2} + A_1 x_{t-1} + Bw \tag{1}$$

where A_1 and A_2 are the *deterministic* components, while B a randomized component and w an independent random variable $N(0, 1)$, respectively. The values of A_1, A_2 and B are determined according to the experiments in this paper.

Measurement In this stage, all particles are new particles, with their states being predicted according to their corresponding dynamics. However, their new weights are not yet assigned. For this, we calculate the *fitness* of each particle, i.e., each hypothetical contour, with respect to the target contour, by analyzing corresponding image features. This calculation determines the weight of a particle. After all, the weights of all particles are determined, these weights have to be *normalized* to be within 1 to facilitate ensuing computation. The resulting particle set is again sent back to perform the first operation, that is, resampling, repeatedly, until the tracking process being stopped.

3.4 Sweep tracker

In our implementation, we adopt a *hierarchical deformable template* where the first level is the palm, the second level

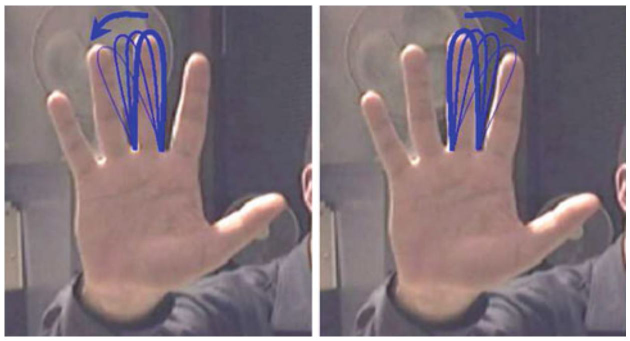

Fig. 2 The angular search, cited from [12]

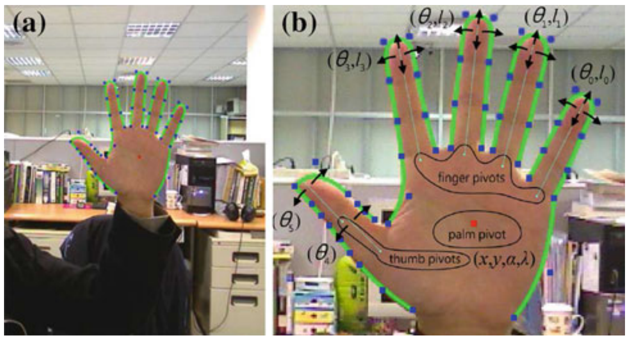

Fig. 3 The hand contour template

includes four fingers and the proximal phalanx of the thumb (the part closer to the palm), and the third level is the distal phalanx of the thumb (the part closer to the thumb tip). The Condensation algorithm is used to track the palm, while we use the sweep tracker algorithm [12] to track the angle and length of fingers. The latter algorithm is a *deterministic* algorithm and can offer much faster and better response when compared with existing *partitioned sampling* approach [8] that is normally used to track hierarchical or *tree-structured* objects. After the weights of all particles being calculated, we pick the particle with the highest weight and then search the angle for each finger in the corresponding hypothetical contour to obtain the *fitness*. The angle leading to the highest fitness is chosen as the detected angle for that finger. To accelerate the search process, the initial angle and length for a finger are set to be their corresponding values in the previous frame. The search for a finger's angle starts from the initial angle, rotating counter-clockwisely, rotating back to the initial angle, and then rotating clockwisely, as shown in Fig. 2. Once the angle with the best fitness score for a finger is found, we can then determine the finger's length, and the involved algorithm will be described in detail in the next section.

4 Implementation and improvements

In this section, we describe how the hand tracking is fulfilled and how the improvements are made. In particular, we apply an *articulated hand template* to track a hand, and based on the basic framework presented in the previous section, we will describe a measurement model based on *skin color*, and how skin colors are detected. In addition, we discuss how to detect a *closing-hand* gesture where there are no gaps between fingers. Finally, we propose methods to further improve upon existing approaches, including the Condensation algorithm, resampling mechanism, finger length detection, and the sweep tracker algorithm. To sum up, these improvement altogether can make the proposed hand tracking system more computation efficient and at the same time much easier to be combined with other potential applications.

4.1 Articulated hand template

By assuming that the hand to track moves perpendicularly to where the camera is pointing at, we adopt the articulated hand template similar to [12]. There are totally 50 control points for the B-splines that constitute the hand contour, as shown in Fig. 3a. In order to let each finger to move freely, each finger can rotate about its *pivot*, which is located at the *finger pivots* region on the palm. Moreover, to be able to express more gestures, each finger is allowed to bend, thus affecting its length. To make detection easier, we only allow the thumb to move parallelly to the palm plane, and therefore, it can bend toward other fingers. As a result, the thumb is composed of two segments, where the first segment (proximal phalanx) is connected to the palm, and the remaining part is the second segment (distal phalanx). As shown in Fig. 3b, there are 14 degrees of freedom, $(x, y, \alpha, \lambda, \theta_0, l_0, \theta_1, l_1, \theta_2, l_2, \theta_3, l_3, \theta_4, \theta_5)$, where (x, y) is the central location, and (α, λ) are the rotation angles of the tracked hand, respectively, while (θ_i, l_i) are the angle and length of a particular finger. In general, (x, y) is set to be the center position of the frame, α is set to be 0 degree, while λ is set be 1, initially and respectively. Moreover, the angle for each finger is also set to be 0. Note that the thumb pivot is not centered around its shape, so that the associated contour can more easily model its movement. Finally, except the thumb, the length of each finger is initially set to be 1. As for the thumb, since we assume it can only move parallelly to the palm, the lengths of its two segments are fixed.

4.2 Dynamic model

To predict an articulated contour model, we assume that each knuckle behaves like an independent oscillator where the values of damping constant β, natural frequency f, and

Table 1 The parameters in the dynamic model

	β (s^{-1})	f (Hz)	ρ
x	6	0	50 pixels
y	6	0	45 pixels
α	6	0	0.3 rad
λ	6	0	0.1

root-mean-squared average displacement ρ are set according to [1]. As in this work, we apply the Condensation algorithm only on representing the dynamics of the palm, and we show the palm-related parameters in Table 1. Similar to Tosas's work [12], we assume the tracked hand is not always vibrating, and therefore, f is always set to be 0. In our implementation, as discussed in Sect. 3.3, the dynamic model involves three terms, A_1, A_2 and B. Here, we empirically set the values of A_1 and A_2 to be -0.6 and 1.6, respectively, to obtain good prediction effect, while B is still generated using the original dynamic model.

4.3 Contour measurement model

The hand contour fitness is calculated based on what is described in Sect. 3.2, and the measurement is also done through *measurement lines*; however, we also take *colors* into consideration.

4.3.1 Measurement line

Figure 4 depicts the configurations of measurement lines. There are totally 70 measurement lines, out of which 19 are on the palm, 10 on each finger but the thumb, 5 on the proximal phalanx of the thumb, and 6 on the distal phalanx of the thumb, respectively. The measurement lines on the palm or the thumb are set perpendicular to their corresponding contours, while their settings on the remaining fingers are a bit different. For each finger (except the thumb), 8 of the measurement lines that are closer to the palm are deployed to be perpendicular to the finger orientation (angle), while the rest two parallel with the finger orientation. The length of each measurement line is about 20 pixels. Regarding how to obtain measurement lines perpendicular to associated contours, it can be referred to Blake et al.'s work [1] for more details. In fact, not only the orthogonality is concerned, but also the interior and exterior of the involved contour should be distinguished for ensuing processing. For illustration purpose, here we draw a measurement line with different colors where the further into the exterior region a more saturated red color is used, while the interior region, a more saturated blue color. Note that as we need to examine each and every pixel on these measurement lines, we implement the *Bresenham* algorithm [2] for the line-drawing process, and due to the potential rotations of lines, the lengths of lines may not be exactly 20 pixels. However, we will make sure that the average length of measurement lines is still around 20 pixels.

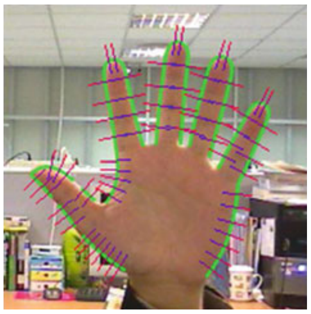
Fig. 4 The measurement lines are perpendicular to the hand contour

4.3.2 Skin-based measurement for hand contours

For skin color detection, we adopt the *linear container classifier* proposed by Tosas [12]. To detect the skin colors of a user, such a classifier can be obtained very efficiently in terms of computation and memory consumption. As a result, by adopting such a classifier alone, we already can track a target skin-colored object with a certain degree of successfulness.

The way to compute the fitness of a hypothetical contour is as follows. Along each measurement line on the hypothetical contour, from the outermost pixel to the innermost pixel, if two consecutive pixels are detected to be of skin colors, then we judge that a contour edge point is encountered. The distance between this point and the center of the measurement line, i.e., the measurement point, is calculated and table-lookuped to obtain a score. The scores from all measurement lines are multiplied to derive the fitness of this hypothetical contour. Each aforementioned score is calculated based the integral samplings from a Gaussian distribution, and the table to lookup is shown in Table 2. Note that the score ranges from 0.5–2, which means each measurement can double or halve the fitness calculated so far. In the case where no skin pixels are detected, the largest distance, i.e., 10 is resulted, and thus, the score of 0.5 is returned. As there are 70 measurement lines, the value of fitness ranges from $2^{70} \approx 1.180 \times 10^{21}$ to $0.5^{70} \approx 8.47 \times 10^{-22}$.

A naive way to obtained the aforementioned distance may involve floating point computation such as *squared root* that is normally considered computation expensive. To simplify and thus accelerate this calculation, we could map the pixels on the measurement line to the range from -10 to 10 by the following steps. We maintain a "distance" variable that is initially 0 and increases by one when we advance from the

Table 2 The measurement scores

Dist.	Score	Dist.	Score	Dist.	Score
0	2.0000	4	1.8801	8	0.8737
1	1.9995	5	1.7232	9	0.6408
2	1.9922	6	1.4805	10	0.5000
3	1.9610	7	1.1767		

outermost pixel to the innermost pixel along a measurement line. When a skin-colored pixel is encountered, we subtract half the length of the measurement line from the distance variable, divide it by the length of the measurement line to normalize the value to be within the range from 0 to 1, and finally multiply it by 21 to finish the mapping. By taking the absolute value of this mapped value and a table-lookup, the corresponding score can be retrieved.

A table-lookup and a simplified distance calculation make the computation very efficient. Furthermore, the way of using two skin pixel occurrence helps to distinguish the correct contour. More specifically, on one hand, if all pixels on a measurement line are skin-colored or non-skin-colored, the corresponding fitness value will be halved. On the other hand, only when a more correct contour is encountered, it is then possible to have a score higher than 0.5. As a result, such a mechanism helps to enlarge the difference between a low-fitness hypothetical contour and a high-fitness one.

4.3.3 Edge detection

Adopting the skin color detection alone is still not enough, as a user may close his/her hand as shown in Fig. 5 so that the previously mentioned measurement process may get erroneous results. This is because a half portion of many measurement lines will be enclosed by the skin-colored area, and thus, their associated scores will be halved to affect the fitness value. To solve this, we adopt the edge detection techniques as in Tosas's work [12]. The idea is to apply an *edge detector*, which outputs a larger value when a larger change of image brightness is encountered. A threshold is used to guard against a valid edge, and when there are more than one edge points on a measurement line, only the one that is closest to the measurement point is chosen. When both skin colors and edge features are considered, a 2D Gaussian function is used to calculate the score. And again to speed up the computation, the involved function evaluation computation can be pre-calculated so that at the run time only table-lookups are needed. Figure 5 shows the contour of fingers for a closing-hand gesture that can be correctly detected after applying the edge detection mechanism.

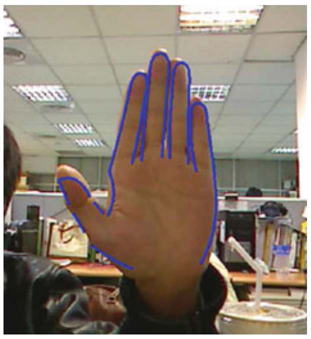

Fig. 5 The hand contour after applying the edge detection

4.4 Improved resampling mechanism

As mentioned in Sect. 3.3, the Condensation algorithm consists of three steps, resampling, prediction, and measurement. To further improve the performance, we choose to only apply particle filtering on the palm, which is associated with 250 particles, while dealing with the rest via the sweep tracker algorithm. In Blake et al.'s work [1], the resampling process is random, whereas in Tosas's work [12], the resampling process was only performed on the top 10 % of the old particles. We make an even drastic change by resampling only on the best one. In practice, such a choice not only speeds up the computation, but also maintains a reasonable quality.

4.5 Improved sweep tracker algorithm

In Sect. 3.4, we described the sweep tracker algorithm proposed by Tosas [12]. The basic idea is to search for the positions of fingers and the algorithm is more efficient than the original *particle sampling* algorithm in that it improves in terms of both speed and accuracy. We implement the sweep tracker algorithm and found that it can be further improved due to the fact that human gesture may change rapidly and human fingers can only rotate up to a certain range of angles, which can be obtained experimentally.

More specifically, as the original sweep tracker algorithm starts its search from the best previously estimated angle and length, during our implementation, we observed the following problem. Assuming in the previous state, a finger is with its shorted length due to a *fist* gesture, and at the next moment, the gesture is rapidly changed so that the corresponding finger increases its length. As a result, the search for the current angle of the finger may be incorrect and thus also an incorrect estimation of the finger's length. It may take a hand tracking systems more rounds of search to finally "catch up" with the change of gestures, so a delay of hand tracking can be perceived. To resolve this issue, when a finger's length is smaller than a threshold and we are to perform the sweep tracker algorithm to search for all angles, an increase in finger's length,

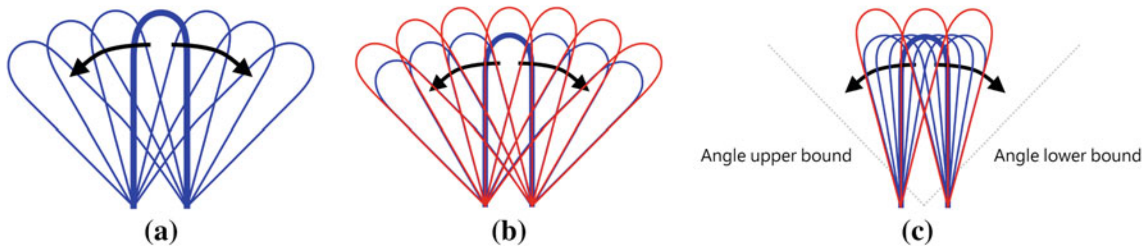

Fig. 6 Sweep tracker algorithm and its improved version

called *SearchLength*, is considered. Figure 6a shows the original sweep tracker's angle search for a finger's movement, and note that it is a bit simplified for illustration purpose. Figure 6b shows how we increase the search range, where the increased portion is marked in red. Through numerous experiments, we found that setting the value of *SearchLength* to be 0.2 leads to the best result. Finally, as the original sweep tracker algorithm always searches angles up to a fixed range, in our implementation, we adjust the angular search range to be the previous angle minus or plus *SearchAngle*, so that the actual search range is reduced, and at the same time, we can refine the step size of the angular search. Therefore, with the same number of searches, we can capture more precise finger change. Figure 6c shows the reduced but denser sweep tracker search, where the two gray lines are the original angular search bounds. In our implementation, with *SearchAngle* being 0.3, we get the most satisfactory result.

4.6 Improved finger length detection algorithm

One of the most important applications of finger length estimation is to detect the events like *click* and *dragging* events, as will be described in the next Section. Originally, we adopt the approach proposed by Tosas [12]; however, we found there is room for further improvement.

According to Tosas [12], as shown in Fig. 7, the way to find a finger's length is to make use of two longer *measurement lines*, which are parallel to the finger's angle, and span from the finger's *pivot* region to the position that is a bit farther than the finger tip. And to find the length of a finger, we start from the end in the pivot region to the other end, until a "non-skin-colored" pixel is found and thus the end of skin area, or equivalently the end of the finger is determined, so the length of the finger can be obtained.

More detailedly, to make the whole process more accurate, the interference of noise must be considered. For this, and according to Tosas [12], the updated finger's length is detected by maintaining an array of size 25 (pixels). By treating the skin area as the foreground, the following morphological operations are performed: an erosion of radius 1 (pixel), a dilation of radius 25, and an erosion of radius 24. As a result, the array is examined to determine the changed

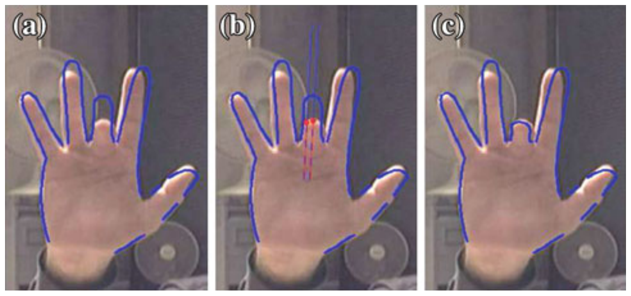

Fig. 7 Finger length detection, cited from [12]. **a** The blue contour shows the original (longer) finger contour. **b** The red contour is used to measure the changed finger's length. **c** The updated finger contour

finger's length. However, in order to perform these morphological operations, one extra array is needed.

We propose an improved version for finding the updated finger's length without the need of an extra array, and most importantly, we only need to scan each pixel just once to derive the answer. One important observation is that, the purpose of the first morphological operation is to get ride of continuous skin pixels whose length is less than 3 pixels, as these pixels are most likely to be noise. Therefore, they should be just ignored. At the same time, we also need to patch the hole of size 25 due to the dilation operation. To do that, we use one variable, called *count25*, to record the length of consecutive non-skin pixels examined so far. As shown in Fig. 8a, where T means a skin pixel and F means a non-skin pixel. As this one skin pixel region will be gone after the morphological erosion operation, the *count25* value should be 3 when it comes to the last pixel. Similarly in Fig. 8b, *count25* is 4. However, in Fig. 8c, the value for *count25* should be 1 as there are already three consecutive skin pixels. Figure 9 shows a more complex example. Let us examine some representative cases. At pixel (A), after seeing three consecutive skin pixels, *count25* is reset to 1, as we see the first non-skin pixel. At pixel (B), after seeing only two skin pixels, *count25*, which has remained as 6 for the past two skin pixels, should be added up to 9 to reflect the equivalent result after applying the morphological erosion operation. At pixel (C), it is similar to the case at pixel (A). At pixel (D), it is similar to the case in pixel (B). Finally, at pixel (E), since *count25* has been accumulated to 25, we then can retrieve the last loca-

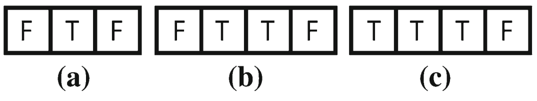

Fig. 8 Three scenarios, where the rightmost pixel represents the current pixel

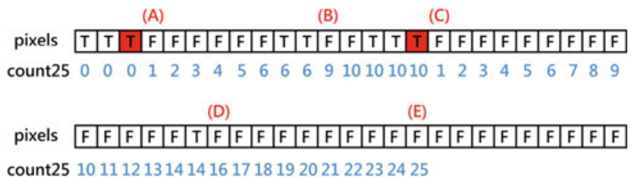

Fig. 9 A more complex scenario

Table 3 The noise variation bounds

Variation	Lower bound	Upper bound
x	2 pixels	4 pixels
y	2 pixels	4 pixels
α	0.03 rad	0.05 rad
λ	0.02	0.03

tion where the (long enough consecutive) skin pixels were encountered, that is, the location prior to pixel (C) will be returned. Note that this is also the finger's length that we want to detect, and we have achieved the same result without using an extra array. As a consequence, it is also much faster.

4.7 Improved dynamic noise mechanism

In order to make hand tracking system to function correctly even in face of rapid gesture changes, Tosas [12] propose a dynamic noise variation mechanism called *Variable Process Noise*, which offers stable tracking result as a hand stays relatively stationary and still tracks accurately in the case where more dynamic hand gestures are experienced. Here, the dynamic noise is the term Bw described in the Condensation algorithm, where w is a normally distributed noise $\mathcal{N}(0, 1)$, and B is a constant governing the size of the associated noise.

However, as we have made improvement over the resampling process proposed by Tosas [12], the original dynamic noise approach is no longer applicable. To address this, we propose to modify the noise variation according to the previous two best particles. Table 3 shows the noise variation upper bounds and lower bounds that we reference to adjust the value of B. Recall that a (hand) particle's state vector contains four parameters (x, y, α, λ), and if both the variations of the previous two particles are larger than their corresponding upper bounds, then B is set to be 1.75 times of its original value; if both of the variations of the previous two particles are smaller than their corresponding lower bounds, then B is set to be half of its original value; otherwise, B stays the same. Through such a modification, even a more rapid hand motion can be tracked.

5 Novel window interface

In this section, we present the novel window interface we designed to facilitate friendly HCI. The proposed window interfaces allow a user to perform window operations without mouse or keyboard while only a common webcam is needed. Moreover, with the help of hand gestures, oftentimes it becomes much easier to achieve many window operations. For example, opening a webpage, enlarge or shrink current window, and so on, can be accomplished with a short-distance hand movement and a (detected) click. When a user wants to input texts, a simple hand gesture can bring up a virtual keyboard interface to suit the need of entering numbers, alphabets, and so on.

5.1 System flow

Figure 10a shows the system flow in this work. During the *standby mode*, our system continuously monitoring the *fitness* of each frame, and when it is higher than an *initial threshold*, our system enters the *initialization mode*. When the fitness is higher than the initial threshold for 10 consecutive frames, then we enter into the *operation mode*. When the fitness is lower than a *stop threshold* for 3 consecutive frames, we view that the hand tracking is effectively lost, and therefore, we return to the *standby mode*. In the operation mode, as shown in Fig. 10b, there are three states to denote three different interface situations, which can be switched on alternatingly by bending the distal phalanx of the thumb with the ring finger being bent in advance. In the first state, there is no interface displayed and our system will just emulate the mouse movement by following the hand. In the second state, a mouse interface is displayed, and at the same time, the mouse movement is still emulated. In the third state, a keyboard interface is displayed for entering textual input, and to avoid potential interference, the mouse simulation is turned off.

5.2 Initialization

During the *standby mode*, our system will initialize many parameters, and as a user presents his/her hand in front of the webcam, the standby mode begins. In this mode, the hand contour is drawn in red, as shown in Fig. 11a, and this indicates the case that the hand's fitness is still below a desired *initial threshold*, or equivalently, the user's hand contour has not been placed in a proper position. Figure 11b represents the case where the fitness value has reached the initial thresh-

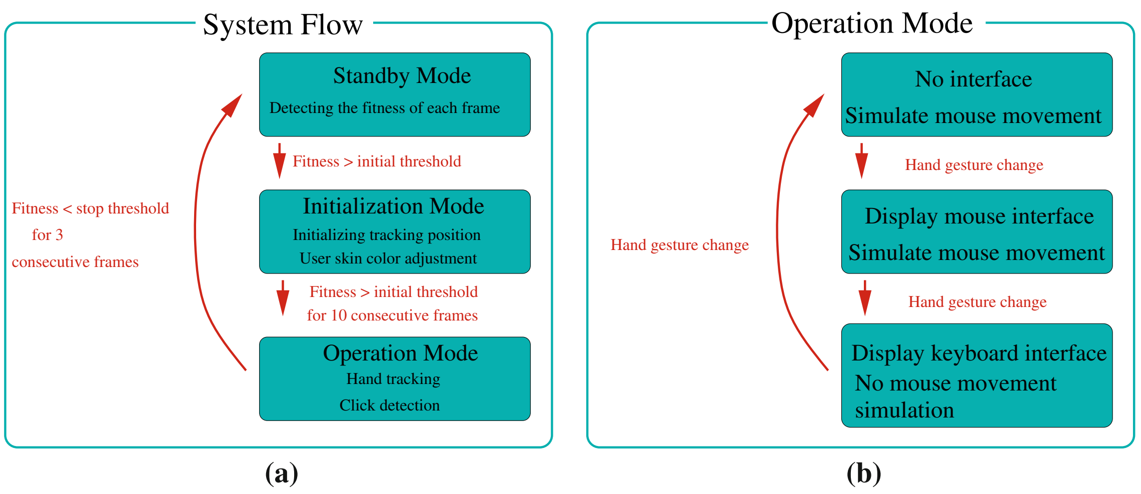
System Flow
Standby Mode
Detecting the fitness of each frame
Fitness > initial threshold
Initialization Mode
Initializing tracking position
User skin color adjustment
Fitness > initial threshold for 10 consecutive frames
Operation Mode
Hand tracking
Click detection
Fitness < stop threshold for 3 consecutive frames
(a)
Operation Mode
No interface
Simulate mouse movement
Hand gesture change
Display mouse interface
Simulate mouse movement
Hand gesture change
Display keyboard interface
No mouse movement simulation
Hand gesture change
(b)

Fig. 10 **a** The system flow of this system. **b** The operation mode in the system flow

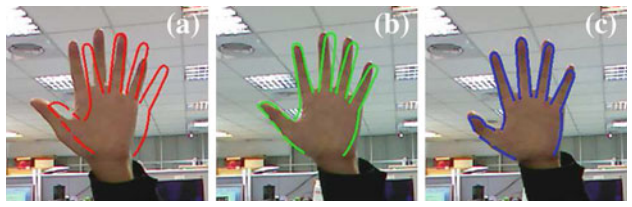

Fig. 11 Hand contour initialization

old, so the hand contour is drawn in green to show that our system enters the initialization mode and starts the tuning of some parameters, which includes the skin color classifier and hand position initialization. Once this matching status has lasted for 10 consecutive frames, the hand contour turns to blue, as shown in Fig. 11c, it means the hand tracking process starts to function, that is, our system enters into the *operation mode*. On the other hand, if the fitness is smaller than a *stop threshold* for 3 consecutive frames, our system deems this as *lost of hand contour*, and thus goes back to the *standby mode*, and waiting to enter the *initialization mode* next time.

As mentioned, the parameter tuning in the initialization mode includes two parts, hand position initialization and skin color classifier. Hand position initialization means setting the hand contour with the highest fitness to be the current tracked position. As for skin color classifier, our system makes use of the 10 consecutive frames to find the best *linear container classifier*, and in particular, the user's skin colors and the current lighting will both be taken into consideration.

5.3 Click/dragging detection

A hand tracking system needs to track not only where a hand is, but also what its gesture is. Regarding the gesture, the most important task is to detect the click and dragging events.

Note that the clicking position can also be determined as the whole hand is under the tracking process. To emulate a dragging, we set a finger length range; that is, once the tracked hand is moving with the finger's length falling within the specified range, a dragging is triggered.

Click threshold For detecting a click, we need to detect the change of a finger's length. To do so, we applied the technique EWMA (Exponentially Weighted Moving Average) [12] to see if the length of a tracked finger falls below a threshold, called *lower control limit (LCL)*, and if so, a click is triggered.

Dealing with hand vibration Even when we are able to detect a click, oftentimes some hand vibrations may cause undesired click events. To avoid this, we modify the click triggering mechanism so that only when a finger's length is shorter than *LCL* for 2 consecutive frames, a click is then triggered.

Click position In this work, there are mainly two interfaces involved, namely the mouse interface and keyboard interface. When the mouse interface is enabled, we assume the hand center is where we want to set the mouse cursor. In addition, when a click happens, if the clicking finger is the index finger, we assume the left mouse button is pressed; similarly, the middle finger corresponds to the right mouse button. On the other hand, when the keyboard interface is enabled, we assume that each finger can press any of the buttons on the virtual keyboard, and therefore, the click position, in this case, is where a finger shrinks its length.

Dragging event Dragging is an indispensable operation commonly adopted in a window environment. The way we detect a dragging event is as follows. Whenever we detect a click, we store the current finger's length as *Click-Length*. By means of this length, we can derive a dragging

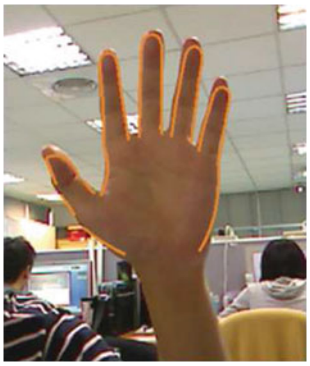

Fig. 12 A color change to indicate the occurrence of a depth change event

threshold by the following:

$$DragTreshold = ClickLength + Margin \quad (2)$$

where *Margin* can be different for each finger and is an adjustable parameter. A larger value indicates that it takes longer time to end a dragging. When a finger's length is smaller than *Drag Treshold*, a dragging event starts; otherwise, a dragging event ends.

5.4 Depth change event

A hand-based tracking system can offer more freedom when compared with traditional systems using mice and keyboards. One interesting and novel event that our window interface can detect is a depth change. Recall that during the initialization mode mentioned in Sect. 5.1, some hand configuration can be recorded. Such information can be used to determine if the depth of the tracked hand changes significantly by comparing the hand size before and after the movement. Once such an event is recognized, the hand contour changes its color to orange, as shown in Fig. 12, and it can be linked to an interesting window operation: push the current focused window to the background. Such a motion is not trivial for traditional mouse and keyboard system, whereas it is a very natural movement for a hand.

5.5 Hand devibration

When simulating mouse movement with a hand, the hand position, shown in the video frame, will be mapped to the screen. In general, as a screen normally has a higher resolution, the intrinsic minor hand vibration will be inevitably enlarged and thus affect the click position. To solve this, we perform a weighted average of the previous detected hand position, *PrevPos*, and the currently detected one, *CurrPos* by the following:

$$CurrPos = 0.2 \times CurrPos + 0.8 \times PrevPos \quad (3)$$

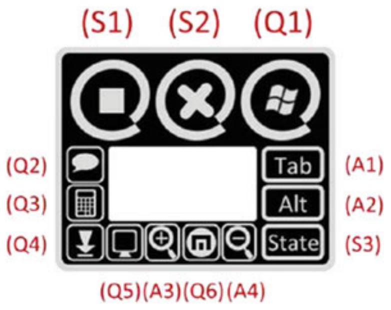

Fig. 13 The mouse interface

Through such a scheme, the original vibration phenomenon is effectively reduced by such a smoothing mechanism.

5.6 Virtual touch panels

To facilitate desired window HCI, we have also implemented two virtual touch panels, one for the mouse, and one for the keyboard, as shown in Figs. 13 and 14, respectively. These interfaces can be displayed alternatingly via a specific hand gesture or some hot keys. When a user first enters the *operation mode*, he/she can use the gesture mentioned in Sect. 5.1 to switch on the mouse interface. To make it easier to use, when the mouse interface is shown, its displayed position will be adjusted so that the current mouse cursor (the center of the tracked hand contour) will become the center of this mouse interface. This adjustment allows a user to quickly locate the desired keys without moving for a long distance. There are three groups of (function) keys, as shown in Fig. 13, where some red labels are marked to ease the description. The labels with *S* initials are related to the hand tracking system, where (*S1*) is *start or stop*, which can be used to combine with a *media player* to control a media's playback, (*S2*) is to close a system or window, while (*S3*) is to display some system state information. The labels with *Q* initials are related to Windows applications shortcuts, where (*Q1*) is equivalent to the same key on a keyboard to start some basic Windows functions, while (*Q2*) to (*Q6*) execute the applications of *MSN*, *Calculator*, *Showing Desktop*, *File Manager*, and *Web Browser*, respectively. Finally, the labels with *A* initials are for more complex combination of keys, where (*A1*) and (*A2*) for switching the current active window, while (*A3*) and (*A4*) serve the same functionalities as combining the *Ctrl* key and the mouse wheel to produce the *zoom in* and *zoom out* operations.

By the same switching gesture, the keyboard interface will pop up to replace the mouse interface. To avoid confusion, the keyboard interface will not block the mouse cursor when it appears, and as mentioned, the mouse movement simulation also stops. Similarly, the functions on the keyboard interface can be explained according to the red labels. The

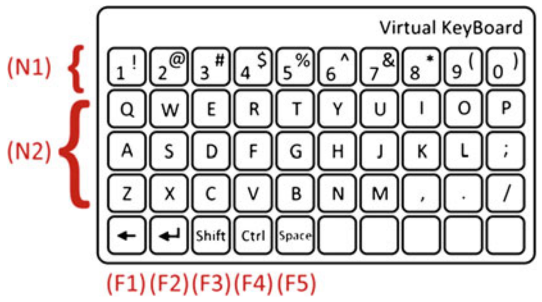

Fig. 14 The keyboard interface

region marked as (*N1*) contains the keys for entering numbers, while the region marked as (*N2*) the keys for entering English alphabets. The keys with the *F* initial labels are for executing functions, which correspond to the same functionalities as the keys on a normal keyboard. Note that when a key on the keyboard interface is pressed, the key will change its color to provide a user the feeling of pressing a key. When a key combination such as *Shift* and *Ctrl* are pressed, both of them will remain red, until the user presses another key so that their original colors are restored.

5.7 Results and system environment

This system was implemented on a machine with Intel Core 2 Duo 1.86 GHz, 3 GB memory, running on Windows XP. The camera in use is Logitech's QuickCam which can process frames of size 640 × 480 at the speed of 30 frames per second. The programming language for development is C# and the DirectShow library is used for capturing video frames. For the result demonstration, as it is difficult to present HCI through static images, we opt to produce some video demos of the proposed system. These files are packed together and zipped and can be downloaded from http://star7.cs.ntust.edu.tw/ckyang/hand_sivp_video.zip. There are several video files included. The file named *initialization.avi* shows how the proposed system performs the initialization, as described in Sect. 5.2. The file named *switch_interface.avi* demonstrates how we can switch among the different interface configurations through a specific hand gesture, as mentioned in Sect. 5.1. The file named *dragging.avi* shows how the technologies discussed in Sects. 5.3 and 5.6 are combined to perform a window dragging through the proposed mouse interface. The file named *push.avi* illustrates a scenario where we make use of the detection of a depth change event, as explained in Sect. 5.4, to "push" the current active window to the background while exposing the window that is the next to the top to become the current active one. The file named *keyboard.avi* demonstrates how we can enter textual inputs through a virtual keyboard interface, whose functionalities are detailed in Sect. 5.6. Finally, the files named *sweeptracker_original.avi* and *sweeptracker_improved.avi* compare the results from the original sweep tracker algorithm proposed by Tosas [12] and from our improved sweep tracker algorithm. As shown in these two videos, a rapid gesture change such as altering from a fist gesture to an opening hand gesture may result a delay in the detection, as mentioned in Sect. 4.5. On the other hand, by adopting our improved algorithms both on sweep tracker and finger length detection, as described in Sects. 4.5 and 4.6, the sudden change can be more promptly captured and reflected, thus offering a more visually pleasing result.

6 Conclusion and future work

This paper presents a hand tracking system that improves both the Condensation algorithm proposed by Blake and Isard [1], and the hand tracking system proposed by Tosas [12]. The proposed system can offer *real-time performance* and thus is more suitable for being integrated with other applications. We have also developed two novel window interfaces with virtual touch panels to be coupled with the hand tracking system to further assist desired and friendly human–computer interactions in a Windows system. Compared with other systems, the proposed work can detect events such as click, dragging and also a novel type of a depth change, and with the help of many hand gestures, it is a more powerful tool for HCI.

However, for the concerns of simplicity and therefore efficiency, currently, our hand gestures is still confined to be "2D", that is, a tracked hand cannot rotate toward the screen, as potential occlusions could happen to confuse the recognition result. In the future, how to efficiently generalize the hand tracking system to be able to deal with full 3D motions is worth further studying. In fact, Wang et al. [14] did some work along this line; however, it is at the expense of constructing of a *hand pose database* in advance. Furthermore, the functionality of *multi-touch*, currently not implemented for the purpose of efficiency, is also another possible future direction.

Acknowledgments This work was supported in part by the National Science Council under the Grant NSC 97-2221-E-011-109.

References

1. Blake, A., Isard, M.: Active Contours. Springer, Berlin (1998)
2. Bresenham, J.E.: Algorithm for computer control of a digital plotter. IBM Syst. J. **4**(1), 25–30 (1965)
3. Fahn, C.S., Yang, C.N.: Two-hand fingertip identification and gesture recognition techniques applied for human-computer interaction systems in real time. Master's thesis, National Taiwan University of Science and Technology (2009)
4. Hardenberg, C.V., Berard, F.: Bare-hand human-computer interacion. In: Proceedings of the ACM Workshop on Perceptive User Interfaces (2001)
5. Isard, M., Blake, A.: Condensation—conditional density propagation for visual tracking. Int. J. Comput. Vis. **28**(1), 5–28 (1998)

6. Isard, M., MacCormick, J.: Hand tracking for vision-based drawing. Technical report, Visual Dynamics Group, Department of Engineering Science, University of Oxford (2000)
7. Letessier, J., Berard, F.: Visual tracking of bare fingers for interactive surfaces. In: UIST '04: 17th Annual ACM symposium on User Interface Software and Technology, pp. 119–122 (2004)
8. MacCormick, J., Isard, M.: Partitioned sampling, articulated objects, and interface-quality hand tracking. In: European Conference on Computer Vision (2000)
9. Manresa, C., Varona, J., Mas, R., Perales, F.J.: Hand tracking and gesture recognition for human-computer interaction. Electron. Lett. Comput. Vis. Image Anal. **5**(3), 96–104 (2005)
10. Rehg, J.: Visual analysis of high dof articulated objects with application to hand tracking. Ph.D. thesis, Carnegie Mellon University (1995)
11. Rowley, H., Baluja, S., Kanade, T.: Neural network-based face detection. IEEE Pattern Anal. Mach. Intell. **20**, 22–38 (1998)
12. Tosas, M.: Visual articulated hand tracking for interactive surfaces. Ph.D. thesis, Nottingham University (2006)
13. Viola, P., Jones, M.: Robust real-time face detection. Int. J. Comput. Vis. **57**(2), 137–154 (2004)
14. Wang, R.Y., Popović, J.: Real-time hand-tracking with a color glove. ACM Trans. Graph. **28**(3) (2009). http://dl.acm.org/citation.cfm?id=1531369
15. Wu, X., Xu, L., Zhung, B., Ge, Q.: Hand detection based on self-organizing map and motion information. In: IEEE International Conference on Neural Networks and Signal Processing (2003)
16. Yuan, X., Lu, J.: Virtual programming with bare-hand-based interaction. In: Proceedings of the IEEE International Conference on Mechatronics and Automation (2005)
17. Zaletelj, J., Perhavc, J., Tasic, J.F.: Vision-based human-computer interface using hand gestures. In: Eight International Workshop on Image Analysis for Multimedia Interactive Services (WIAMIS' 07) (2007)