Virtual Keyboard for Head Mounted Display-based Wearable Devices

Ming-Wei Chang
Industrial Technology Research Institute
Hsinchu, Taiwan
E-mail: MingWeiChang@itri.org.tw

Tzi-cker Chiueh
Industrial Technology Research Institute
Hsinchu, Taiwan
E-mail: tcc@itri.org.tw

Chia-Ming Chang
Industrial Technology Research Institute
Hsinchu, Taiwan
E-mail: jim.chang@itri.org.tw

***Abstract*—Wearable devices eliminate the need of physically taking out a mobile device before operating on it and are emerging as the next wave of mobile systems. Head-mounted display (HMD) is a key building block of wearable devices, and offers users immediate access to relevant information in a glance. However, most existing user input mechanisms accompanying HMDs are designed for interactive information exploration rather than for extended text entry. This paper describes the design, implementation and evaluation of a text input system for HMDs called Air Typing, which requires only a standard camera and is shown to be comparable in effectiveness to single-hand text input on tablet computers in a lab setting. Air Typing features a novel two-level virtual keyword layout, which substantially improves the typing speed by cutting down unnecessary hand movements during typing and greatly simplifies the associated image processing task by doing away with fine-grained matching between fingertips and keys. The current Air Typing prototype incorporates an OpenCV-based virtual key press detection algorithm that runs on the featured two-level virtual keyboard. In our tests, an experienced user's typing speeds of one-hand text input and of two-hand text input under Air Typing are 13 and 15 words per minute (WPM), respectively.**

Keywords—Head mounted display; typing in air; virtual keyboard; keyboard layout; hand tracking; fingertip detection and tracking

I. INTRODUCTION

Head-mounted display (HMD) was originally designed to provide an immersive experience for users of virtual reality systems, but is now re-purposed as the main visual output of wearable devices such as Google Glass. In addition to enabling immediate access to information directly relevant to the context on the spot, HMD also offers the advantages of a larger effective screen size and lower power consumption because of its shorter distance to the viewer's eyes. However, the input mechanism of most existing HMD-based wearable devices is mainly designed to support interactive exploration and navigation in the virtual information space, and is not suitable for text entry for an extended period of time. While speech input could be a plausible alternative, it still has usability challenges such as privacy concern and error correction. This paper presents the design, implementation and evaluation of a virtual text entry system called Air Typing, which is designed to enable users of HMD-based wearable devices to effortlessly engage in text-based communications, such as instant messaging, email, and posting on social network sites.

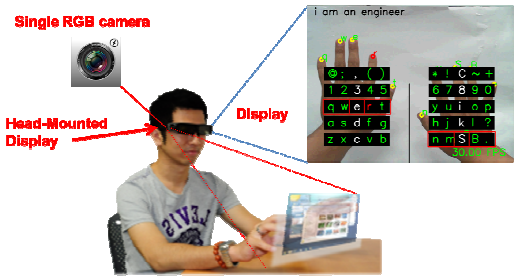

Fig. 1. User view of a HMD-based wearable device supporting the Air Typing technology

As shown in Figure 1, Air Typing does not require any special hardware other than a standard-resolution camera that often comes with a wearable device. It applies computer vision techniques to track in real time the positions and movement of a user's fingers and converts these events to keystrokes. Usability, accuracy and computational overhead are three major considerations in the design of Air Typing. After exploring a variety of keyboard layouts, including the QWERTY layout used in physical computer keyboards, we decided to adopt a two-level keyboard design that requires the user to first pick a region of the virtual keyboard and then select a specific key in the region. More specifically, pressing a key involves moving the hand to the key's region on the keyboard and flexing the corresponding finger. This keyboard layout improves the typing speed in two ways. First, the extent of hand movement during typing is markedly smaller. Second, pressing a key does not require a finger to be physically on top of the key, and this reduces both the user's mental load and the computational overhead associated with each key press. The main disadvantage of this two-level keyboard layout design is it requires additional learning time on the part of new users.

Because Air Typing relies only on the video input to detect key presses, it employs and integrates a set of video object tracking algorithms that collectively are able to track the position and movement of individual hands and fingers, including events that correspond to when the user actually

978-1-4799-7615-7/14/$31.00 ©2014 IEEE

presses a key by bending a finger. In addition to detecting the relevant video events reliably and in real time, the main technical challenge in the development of Air Typing's video processing algorithms is how to minimize both the false positive rate and false negative rate of identifying key press-related finger movements. More specifically, when a user bends a finger to press a key, the finger's tip may become temporarily out of sight and confuse the video object tracking engine. We developed two heuristic techniques to solve this problem. First, we used a thinning algorithm to erode the contours of each finger and make fingertip detection more robust even when fingers are bent. Second, we confirmed detected fingertips during a key press by tracking the relative position of each such detected fingertip with respect to the center of the associated hand and ensuring it follows a reasonable trajectory. With this integrated hand/finger/fingertip tracking mechanism in place, Air Typing is able to support both one-hand and two-hand text input. In our tests, the resulting typing speed for an experienced user is 13 WPM for one-hand input and 15 WPM for two-hand input.

II. Related Work

There are two classes of techniques for body-part tracking. The first class is based on pattern recognition, which are mainly used to detect human face [1]. The other is based on image segmentation, which is mainly for hand tracking and gesture recognition. The research work by Kakumanu et al. [2] showed that there are multiple color spaces that could be used for skin modeling and detection. Skin detection using color cue is a challenging task because the color value of skin is affected by various factors such as illumination, background, camera characteristics, and ethnicity. Thus, many researchers have been working on the problems of skin color segmentation, and came up with static methods [3] and dynamic adaptive methods [4] [5] to improve the accuracy and effectiveness of skin detection in scenarios with dynamically changing illumination and environmental conditions.

Several researchers have proposed the use of hand gestures as a human machine interface [6] [7] [8]. For example, Ince [6] described a motion detection algorithm that is based on a finger blob analysis of skin colors extracted from the captured image, and proposed using two human hand gestures to replace the right and left click action of a mouse.

Electronic glove-based body sensing systems like VType by Evans, Skiena, and Varshney [9], required the user to wear a special glove to enter text in a virtual world. In this glove, each finger is equipped with a sensor that is able to give out its angle position, which is then analyzed by software to detect finger presses and other interesting events. They also developed an algorithm for resolving ambiguities on an overloaded keyboard.

Wu et al. [10] described a research effort that aimed to track a finger's 3D position in real time. With a single camera, they first tracked the 2D position of a finger using skin detection, motion cue, and analysis of the arm's outline. Then, they derive a finger's 3D trajectory by tracking the 2D positions of the user's face, shoulder and elbow. Finally, they use the resulting 3D trajectory of a finger as the user input interface in a teaching environment.

There are several research efforts using computer vision-based methods to build better HCI systems. For example, Zhang et al. [11] described a system that consists of a camera and a sheet of paper. By reference to the paper's position, the user's extended index finger could be reliably located. Then, certain finger patterns could be construed as a user command. For example, resting the fingertip in its current position for three seconds could mean a click. Terajima et al. [12] developed a system that applied a template-matching-based method to track fingers and hands by performing typing motion analysis on outputs from a high-frame-rate camera.

III. Two-Level Keyboard Layout

The baseline design for Air Typing was one that supports one-finger input on the standard QWERTY keyboard. In the end, Air Typing supports a two-level keyboard layout and enables text input with both hands. This section describes the evolution of Air Typing's text input interface design.

A. Traditional Typing Interface

We started with a design that places a virtual QWERTY keyboard on the screen, and requires the user to point one of her fingers to a target key's image on the virtual keyboard when pressing the key. A detailed manual analysis of a recorded video of the finger actions over the virtual keyboard when a user types in an article with a single finger reveals that the typing time is mainly spent on the following two types of actions: (1) moving the finger across the virtual keyboard, and (2) clicking the intended keys with the finger. The QWERTY keyboard layout was designed to space out adjacent letters in individual words. This design improves the typing speed by minimizing the printer jams due to clashes of neighboring typebars, and encourages alternated users of both hands. However, for one-finger and one-hand input, as shown in Figure 2, the QWERTY keyboard layout is less desirable because it entails considerably more hand movement than necessary. For example, Figure 3 shows the typing finger's zigzag-like movement trajectory when typing the word "world" using a single finger. The same hand movement overhead still exists even when the user uses five fingers rather than one finger, although the magnitude of the overhead may be less because different fingers of the same hand may cover consecutive letters of some words.

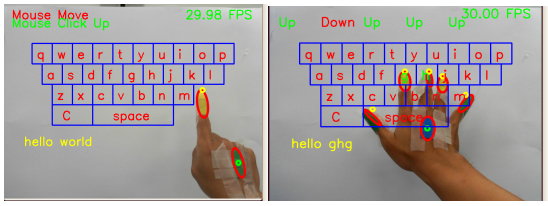

Fig. 2. One-finger and one-hand input against the QWERTY layout

Table I shows the total time taken to enter "hello world" using one finger, five fingers and the mouse, all against the same virtual keyboard similar to the one in Figure 2, and the percentage of this time that is due to hand movement time. The

difference between the total time and the hand movement time is due to key presses. In all three cases, the hand movement time accounts for more than 85% of the total input time. This provides strong motivation for a better keyboard layout that could reduce the hand movement time.

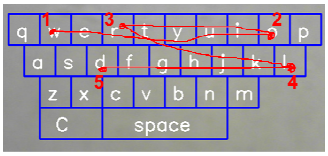

Fig. 3. Movement trajectory of the input finger when typing 'world' using a single finger

TABLE I. THE TOTAL INPUT TIME FOR ENTERING "HELLO WORLD" USING ONE FINGER, FIVE FINGERS AND THE MOUSE AGAINST A VIRTUAL KEYBOARD, AND THE PART THAT IS DUE TO HAND MOVEMENT DURING TYPING

	1 finger	**5 fingers**	**Mouse**
Hand Movement (sec)	8.678 (85.1%)	7.492 (91.9%)	7.02 (90.1%)
Total input time (sec)	10.25	8.16	7.796

B. Two-Level Virtual Keyboard for One-Hand Input

One way to reduce the extent of hand movement during typing is to use a two-level keyboard layout in which a user uses her hand position to select a region of the keyboard as the active region, and one of her fingers to select a key in the active keyboard region. Figure 4 shows a 2-level virtual keyboard for one-hand five-finger input. The first level is the keyboard map in the center, which has 12 entries (labeled 0-11), and the second level is the 12 keyboard regions on top, where each region has 5 keys. Each entry in the keyboard map corresponds to a keyboard region on top. A user selects a keyboard region by moving the center of her hand to the corresponding entry of the keyboard map, and then bends her i-th finger to activate the i-th key in the selected keyboard region. For example, when the hand's center is on the 8-th entry of the keyboard map, the corresponding keyboard region contains the following keys: *zxcvb*. Similarly when the hand's center is on the 5-th entry of the keyboard map, the corresponding keyboard region contains the following keys: *hjkl?*. When a user selects a keyboard map entry, she does not need to place the hand's center exactly on top of the target entry; instead she just needs to move her hand so that the "cursor" moves to the target entry. This is similar to how an optical mouse works.

This two-level virtual keyboard has several advantages when compared with the QWERTY keyboard. First, because the first-level keyboard map is much smaller, the extent of hand movement required is significantly reduced. Second, at any point in time, the typing hand always covers five keys. This is not the case in the QWERTY keyboard. For example, when the thumb is on top of the p key, the other four fingers do not cover any key. Third, selecting a key in the active keyboard region requires the user to bend the corresponding finger, but not place the corresponding finger on top of the chosen key. This appreciably improves the typing efficiency and simplifies the video processing algorithm.

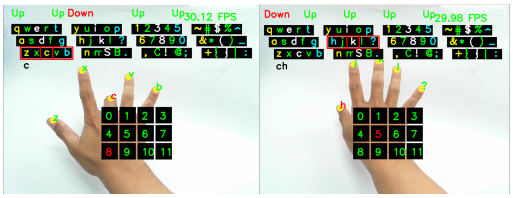

Fig. 4. The proposed two-level virtual keyboard consists of the first-level keyboard map (in the center) and the second-level keyboard regions, where each keyboard region corresponds to an entry in the keyboard map. S and B represent the space and backspace key, respectively.

The proposed two-level virtual keyboard provides various visual feedbacks to help the user to correctly select the intended keys. For example, when a keyboard map entry or a key in the active keyboard region is selected, it is always highlighted in red. In addition, when a user successfully presses a key, an audio cue is also provided. Finally, suppose a user's typing hand has only four fingers, it is relatively straightforward to modify this two-level virtual keyboard so that each keyboard region contains only four rather than five keys.

C. Two-Level Virtual Keyboard for Two-Hand Input

Figure 5 shows Air Typing's two-level virtual keyboard for two-hand ten-finger input. A user moves either of her two hands up and down to select each hand's active keyboard region, and then bends a finger to ``press" a key in the corresponding hand's active keyboard region. In this design, there could be two active keyboard regions at a time, but Air Typing is able to associate a bent finger with the corresponding active keyboard region.

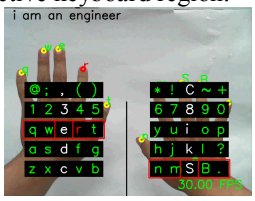

Fig. 5. The proposed two-level virtual keyboard for two-hand input embeds keyboard regions into the keyboard map.

There are two key differences when compared with the one-hand-input keyboard configuration. First, instead of separating the keyboard map and from the keyboard regions, the two-hand-input configuration embeds keyboard regions directly into the keyboard map. Second, a user just needs to move her hands up and down, rather than both up and down and left and right in the case of the one-hand-input keyboard configuration. This further reduces the user's mental load associated with the selection of the active keyboard region.

IV. HAND/FINGER/FINGERTIP MOVEMENT TRACKING

A. Overview

The enabling technology for Air Typing is to apply computer vision techniques to recognize a typing hand's position, which selects a keyboard region, and fingertip movement, which selects a key in the active keyboard region. The video object processing engine in the Air Typing prototype

first detects a typing hand's skin colors, and improves the robustness of hand area detection by applying a median filter to removing noise, then computes the typing hand's contour, locates the hand's center, pinpoints the fingertips, and finally tracks each fingertip's movement that corresponds to key pressing. In addition, the engine is able to perform the above tracking for both hands simultaneously in real time. While designing Air Typing's video object processing engine, we made the following assumptions:

- A user initializes the engine by positioning the center of the back of her typing hand at a fixed point of the display so as to inform the engine of the hand's skin color and area on the display.
- The background cannot contain colors similar to the hand's skin color, and the user needs to re-initialize the engine whenever there is a significant change in the lighting condition.
- When a user inputs text using Air Typing, she must spread out her five fingers so that the engine could reliably detect individual fingers and fingertips. That is, her fingers cannot touch each other.

We will detail each of these steps in the following subsections.

B. *Hand Tracking*

Figure 6 (b) shows the result of applying a back projection operation [13] on Figure 6 (a) using a model histogram constructed at the initialization time. There are obvious black noise pixels on the fingertip and hand portion of the back-projected image. These noise pixels make it difficult to detect the hand's contour and fingertips reliably.

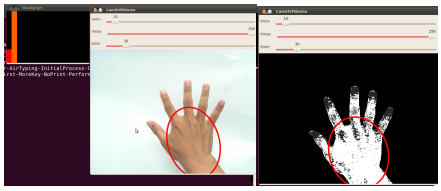

Fig. 6. Detection of a typing hand using CAMSHIFT (a) and the result of back projection (b)

To solve this problem, we first increase the value range of the skin color, and the noise level in the hand area of the resulting back-projected image is significantly reduced, as shown in Figure 7. However, the noise level outside the hand area is increased. To remove these noise pixels, we apply a median filter [14] to smooth out the neighboring pixels in the input picture, and Figure 8 shows the effectiveness of a 7x7 median filter in removing noises introduced by back projection.

After enhancing the input image by increasing the skin color's value range and applying a median filter, we call the OpenCV function FindContours()on the back projection of the transformed input image to extract all contours in it. Then we identify the hand's contour by locating the contour that has the maximum area and has more than a constant number of points on the contour. This heuristic is able to successfully filter most of the non-hand or noise contours.

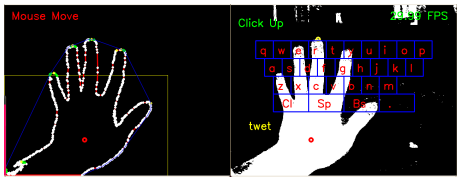

Fig. 7. Increasing the value range of the hand's skin color decreases the noise level in the hand area but increases the noise level outside the hand area.

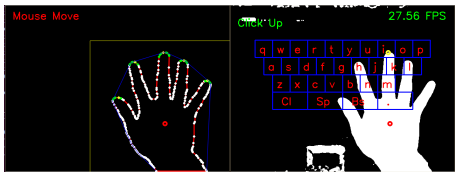

Fig. 8. Removing the noise level outside the hand area by applying a 7x7 median filter to the input image.

We adopt the Continuous Adaptive Mean Shift Algorithm (CAMSHIFT) [15] to track a typing hand's palm area; specifically our implementation is based on the sample code in the OpenCV package [16]. Although the sample CAMSHIFT code is efficient and fast, it cannot effectively handle backgrounds that are similar in color to the hand's skin, and is not robust in the presence of lighting condition changes or hand movements.

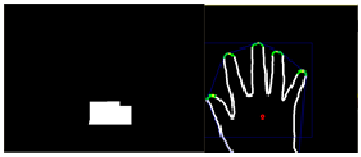

Fig. 9. The resulting palm area after cutting off the wrist and eroding the fingers away.

Air Typing uses the center of the back of the palm to select a keyboard region. To identify this area of a typing hand, we first locate a point on the hand contour that has the smallest Y coordinate, which ideally corresponds to the fingertip of the middle finger, and then carve the wrist part out to form a fixed-sized search window anchored at this fingertip. To further remove fingers, we apply a morphological operation `erode` [17] on the search window to thin the fingers away and arrive at the palm image, as shown in Figure 9. Finally, we apply the CAMSHIFT algorithm to nail down the center of the typing hand's palm area and track its movement subsequently.

C. *Finger/Fingertip Detection*

To detect fingertips, we go through the typing hand's contour, and compute a k-curvature or inner product for such three sample points in which the middle sample point's Y coordinate is smaller than the Y coordinate of the other two sample points [10], and pick out those middle sample points whose corresponding dot product value is positive(< 90 degrees) and larger than a threshold, and whose Y coordinate is the smallest compared with its neighbor sample points. These middle sample points are supposed to be the fingertips.

The above fingertip detection algorithm works reasonably well when the fingers do not have any movement. However, as soon as a user starts to bend her fingers to indicate key pressing, these finger movements seriously degrade this algorithm's effectiveness, as shown in Figure 10, because fingertips become less obvious. We devise two additional techniques to boost the robustness of the above fingertip detection algorithm in the presence of finger movement. The first technique is to tightly integrate fingertip detection with key pressing event detection. That is, when a key pressing event is detected, the fingertip detection algorithm's inner product value threshold is dynamically adjusted to accommodate the fact that a finger's tip tends to become progressively more rounded while the finger is pressing a key, and progressively less rounded while the finger is releasing a key.

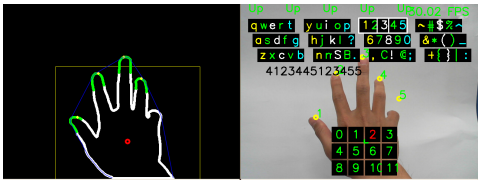

Fig. 10. The tips of both the ring and little finger become less pronounced when the user bends her little finger.

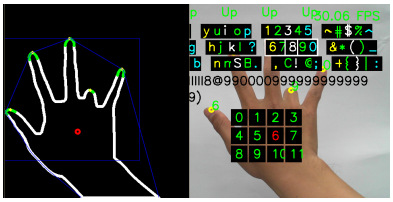

Fig. 11. The tip of the ring finger is still detectable when it is bent after applying the erode operation to the fingers.

The second technique we devised to improve the robustness of fingertip detection by eroding the fingers, which lessens the "rounding" effect when a user bends her fingers, as shown in Figure 11. Note that we have applied the same erode operation for palm area detection to thin the fingers. So this erode operation does not incur any additional performance overhead.

D. Key Press or Click Detection

To reliably detect a key press when a user bends a finger, we track the relative movement of every fingertip with respect to the center of the palm. This approach also allows us to detect finger bends even in the presence of minor concurrent hand movements. We record the position of every fingertip and the palm center in every frame and keep a history of such information for the past five frames. Figure 12 shows the idea of using the relative velocity of a fingertip and the palm center to detect key presses or releases.

In every frame, we first derive the movement velocity of every fingertip and of the palm center using their current coordinates and their coordinates five frames ago, then compute the relative velocity between every fingertip and the palm center, pick the fingertip with the highest relative velocity, and declare a key press event when the magnitude of the relative velocity is larger than a threshold and the relative velocity is heading towards a negative Y direction, and a key release event when the magnitude of the relative velocity is larger than a threshold and the relative velocity is heading towards a positive Y direction.

Fig. 12. Using the relative velocity between a fingertip and the palm center to detect key presses and releases when there is minor concurrent hand movement.

When a key press or release event is detected, the parameters of the fingertip detection algorithm are adjusted accordingly to account for the progressively more or less rounded fingertips during these events. This is an instance of tight integration between components of Air Typing's video object processing engine.

Despite the best attempt to detect fingertips, there are times in which the video object processing engine loses track of some of the fingertips, and needs a recovery mechanism to enable the finger tracking process to proceed normally. In the normal case, the engine records the coordinates of the five fingertips, and sorts them from left to right according to their X coordinate. Whenever the engine detects only fewer than 5 fingertips in a frame, it needs to first determine which fingertip is lost, and then decides how to best estimate the lost finger's current position.

When the number of fingertips detected in the current frame is fewer than five, we compare the coordinates of these detected fingertips and the coordinates of the five fingertips in the previous frame, and find for each detected fingertip in the current frame its corresponding closest fingertip in the previous frame. The fingertips in the previous frame that do not match any fingertip in the current frame are considered lost.

To estimate the current coordinate of each lost fingertip, we first initialize it with its coordinate in the previous frame, and heuristically compensate its X coordinate if it is a thumb and its Y coordinate if it is any of the other fingers. Although not implemented in the current prototype, it is possible to feed the knowledge of lost fingers and their estimated coordinates back to fingertip detection, so that the latter could adjust its threshold to pinpoint and reconstruct the lost fingertips directly.

We have experimented with applying the Kalman filter to predict each fingertip's coordinate at run time and using the predictions as the estimated coordinates for lost fingertips. But the Kalman filter approach incurs nontrivial computational overheads. That's why we eventually choose a heuristic compensation approach in the current prototype instead.

V. PERFORMANCE EVALUATION

A. Evaluation Methodology

We have implemented an Air Typing prototype on a Linux virtual machine that runs on an Intel X86-based Windows desktop machine. The hardware/software specification of the Air Typing prototype is shown in Table II. This Air Typing prototype implements the proposed two-level virtual keyboard and the video object processing engine that features hand, finger and finger movement tracking, and supports both one-hand-five-finger input and two-hand-ten-finger input, with or without markers.

TABLE II. THE HARDWARE/SOFTWARE SPECIFICATION OF THE AIR TYPING PROTOTYPE

Physical machine spec.	Intel i7-2600 @ 3.4 GHz ~ 3.7 GHz , RAM 8GB
Host OS	Windows 7 64 bit SP1
Virtual machine monitor	VMware Player 4.0.3
Virtual machine spec.	4 CPU cores, 2GB RAM
Guest OS	Ubuntu 10.04 LTS
Computer vision library	OpenCV version 2.0.0
Camera	Microsoft LifeCam Studio
Video resolution	VGA (640x480)

When a test user tests the Air Typing interface, he looks at a screen in front of him and types in the air (not against the screen), with a camera facing and capturing his typing hands. The screen shows a composition of the two-level virtual keyboard and the image of his typing hand(s), as shown in Figure 4 and 5. Four input methods are tested and compared, including one-hand five-finger input without marker (OHFF), two-hand ten-finger input without marker (THTF), one-hand five-finger input with marker (OHFFM), all against the two-level virtual keyboard, and one finger on iPad's virtual keyboard (OHOFi). In all cases, the same 107-key test text, which is partial content of English typing test texts [19] and listed in Table III, and every reported number is an average of 5 test runs.

B. Typing Speed

Table IV shows the typing speed and typing error rate results of the four input methods using the 107-key test text in Table III. In terms of typing speed, THTF is the fastest, followed by OHOFi, OHFF and OHFFM. OHFF is faster than OHFFM, but only slightly. This suggests that the Air Typing prototype's video object processing engine is quite effective because it is able to approximate what can be achieved with explicit markers. Somewhat surprisingly, OHFFM is worse than OHOFi by 9.95% in typing speed. Although in theory OHFFM requires less hand movement time than OHOFi, the on-screen virtual keyboard of OHOFi still feels more tangible and requires less hand-eye coordination than the in-the-air virtual keyboard of OHFFM. In the end, OHOFi fares better than OHFFM. Fortunately, THTF beats OHOFi in typing speed by 7.5%. However, in terms of the movement speed, THTF is 28.6% faster than that of OHOFi. This suggests that even though THTF is only slightly faster than OHOFi in the overall speed, the former is significantly faster in movement speed. As for OHFF and OHFFM, we observed that they are very close to OHOFi in movement speed. This result demonstrates that the two-level virtual keyboard indeed provides quantifiable improvement over the traditional keyboard layout in reducing the hand movement time, so much so that the resulting saving in hand movement time more than compensates for the additional typing overhead due to more demanding hand-eye coordination.

TABLE III. A 107-KEY TEST TEXT USED IN THE EVALUATION STUDY OF THE FOUR INPUT METHODS

Sentence 1	It takes twenty four hours for the earth to go around one time. Pepper peel was beside himself with delight.	107 keys

TABLE IV. TYPING SPEED RESULTS OF THE FOUR INPUT METHODS USING THE 107-KEY TEST TEXT IN TABLE III

	THTF	**OHOFi**	**OHFFM**	**OHFF**
Time (sec)	81.38	87.49	97.15	96.56
Movement Time (sec)	41.66 (71.4%)	58.38 (100%)	58.08 (99.5%)	57.60 (98.7%)
WPM(Word Per Minute)	15.78 (107.5%)	14.68 (100%)	13.22 (90.05%)	13.30 (90.6%)
Error Rate	2.49%	0.62%	1.56%	2.18%
FN Rate	0.93%	0%	0.62%	0.62%

The typing speed results reported above already include the time required to correct typing errors during the tests. We recorded the testers' typing actions and analyzed the resulting videos to derive the typing error rate and error pattern. The typing error rates for THTF and OHFF are 2.49% and 2.18%, respectively. That is, there are 2.49 and 2.18 key press errors per 100 key presses. As expected, OHFFM has a 1.56% error rate, which is lower than that of OHFF and THFF. As we can see, OHOFi has the lowest error rate and FN rate at 0.62% and 0% respectively.

A more detailed analysis reveals that there are two classes of typing errors:

(1) Clicking a correct key in an incorrect keyboard region: This results from either the user's mistakes or the system's hand tracking errors.

(2) Clicking an incorrect key in a correct keyboard region: This mainly results from false positives of the system's key press detection, for example, mistaking the little finger is bent when it is the ring finger that the user is bending. As shown in Figure 13, system issues a click event (key "r") for ring finger at frame 13. However, there is a false positive of click detection because the value of y-coordinate of little finger slightly changes at the same time. In this case, we can improve our click detection by picking the finger with the most significant movement.

(3) Clicking a correct key in a correct keyboard region but the system is not responding: This corresponds to false negatives of the system's key press detection. As shown in Table IV, the false negative (FN) rate of the Air Typing prototype's key press detection is 0.93% for THTF and 0.62% for OHFF, respectively.

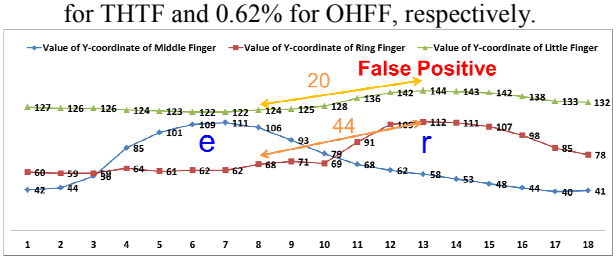

Fig. 13. One case of false positive of click detection occurs when a user bends the ring finger, the little finger is slightly bent as well.

C. Computational Overhead

When the median filter kernel is set to 5x5, and the numbers of contour erode calls for hand tracking and finger detection are 30 and 6 respectively, the average per-video-frame processing times for OHFF and THTF are 11.9 msec and 12.8 msec, respectively. Therefore, the current Air Typing prototype is able to support OHFF and THTF in real time, i.e., 30 frames per second. However, these performance measurements are taken on a desktop machine. So it is still worthwhile to devise additional performance optimizations that could further reduce the per-frame processing overhead to make Air Typing feasible on mobile devices.

TABLE V. BREAKDOWN OF THE PROCESSING TIME OF EACH VIDEO FRAME INTO THE DELAYS OF THE CONSTITUENT STEPS OF OHFF AND THTF SUPPORT IN AIR TYPING AND THEIR ASSOCIATED OPENCV API CALLS IF APPLICABLE

Step	API Call	OHFF	THTF
Format Conversion	cvtColor; cvSplit	41%	35%
Median Filter	cvSmooth	26%	22%
Erode Contour	cvErode	4%	3%
Find Contour	cvFindContours	8%	8%
Fingertip Detect	k-curvature	4%	8%
Hand Track	Camshift	10%	13%
Click Detect		0.01%	0.02%
Input Control		0.87%	0.6%

Table V shows the detailed breakdown of the per-frame video processing overhead into the delays of constituent steps in Air Typing's video object processing pipeline for OHFF and THTF. The most time-consuming step is Format Conversion, which includes conversion from the BGR to the HSV color space and splitting a single multi-channel array into multiple single-channel arrays. The second most time-consuming step is Median Filter, which is designed to remove noise. The Erode Contour step sharpens the fingers to facilitate the detection of fingers. The third most time-consuming step is Hand Track, which includes wrist removal, another round of eroding the finger contours, and invocation of the Camshift algorithm. Although Click Detect, which detects actual key press events, is the most complicated step, it actually ranks the last in terms of processing overhead. The input Control step converts a key press event to an output of the pressed key.

Because Median Filter is a time-consuming step, we varied the filter kernel size to examine the performance impact of this parameter. Figure 14 shows the result of this sensitivity study. The processing overhead of a 15x15 kernel is about 3 times higher than that of a 5x5 kernel. Because a 5x5 kernel provides a decent noise filtering capability, the default median filter of the Air Typing prototype uses a 5x5 kernel.

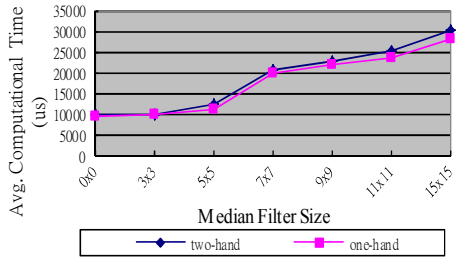

Fig. 14. Computation time comparison for different median filter kernels

D. Training Time

A major design decision in Air Typing is choosing a non-conventional keyboard layout that is different from the QWERTY keyboard layout. An immediate question for this design is the training time required to attain a decent typing speed. Figure 15 shows the improvements in typing speed over a five-day period by one of the authors of this paper for two input methods: THTF and OHFFM. The amount of practice each day is typing the test text in Table VI five times. This result is by no means conclusive but provides an interesting data point to address the learning curve concern.

VI. KNOWN LIMITATIONS

Because Air Typing relies heavily on computer vision techniques, it also suffers from all the usual problems that plague these techniques. The current Air Typing prototype is tuned in a particular lighting condition in our lab. Whenever there is a serious change in the lighting condition, the effectiveness of Air Typing is noticeably affected. In particular, lighting condition change could prevent the typing hand's contour from being reliably extracted, which in turn blocks the follow-on fingertip detection and key press event detection step.

When the background of an Air Typing interface contains pixels that are similar in color to the typing hand's skin, the accuracy of hand tracking, finger and fingertip detection, and click detection could be adversely affected significantly. This problem could be mitigated with more comprehensive and responsive background modeling.

Different users may have different skin colors, palm sizes, stretch distances and angles between the camera and typing hand(s). How to programmatically or automatically adjust the OpenCV parameters in the Air Typing prototype to

accommodate a wide variety of users is an important technical challenge.

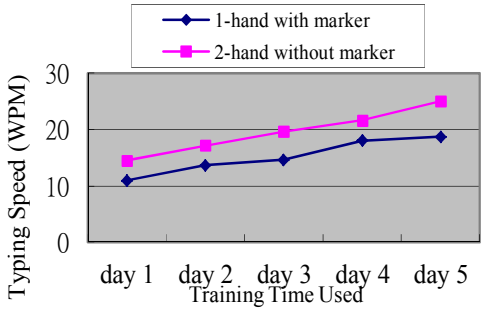

Fig. 15. Improvements in typing speed for THTF and OHFFM over days of practice

TABLE VI. A 100-KEY TEST TEXT USED IN THE EVALUATION STUDY OF TRAINING TIME

Sentence 1	i am an engineer in ccma.	25 keys
Sentence 2	this is our demo for air typing.	32 keys
Sentence 3	thank you for watching and have a nice day.	43 keys
Total		100 keys

VII. CONCLUSION

The first wave of wearable devices that feature a head mounted display, such as Google Glass, are emerging as a new form of mobile devices that potentially could replace existing smart phones. These devices come with primitive input methods that are designed mainly for simple navigation and exploration of the information space displayed on the screen. Although speech input is one possibility for text input, the privacy and social acceptance concern prevents it from being accepted by mainstream users. Therefore, how to support an effective text input method for HMD-based wearable devices becomes an important challenge. This paper describes the design, implementation and evaluation of a text input system called Air Typing, which requires only a standard camera and enables a user to type text in the air as quickly as typing on the virtual keyboard on iPad. This work made the following research contributions to text input method:

- A two-level virtual keyboard layout that reduces the hand movement time during typing,
- A video object processing engine that is specifically designed to detect key presses or clicks by tracking the movement of fingertips, and
- A fully operational Air Typing prototype that demonstrates its effectiveness and serves as the basis of extensive analysis of its typing speed, performance overhead, and usability.

In the immediate future, we will port the Air Typing prototype to Google Glass/iPhone, and convert it into an App on Google Glass/iPhone. Through this App, we will collect more detailed usage data of Air Typing, and enhance Air Typing based on these usage data.

REFERENCES

[1] Viola, Paul, and Michael J. Jones. "Robust real-time face detection." International journal of computer vision 57.2 (2004): 137-154.

[2] Kakumanu, Praveen, Sokratis Makrogiannis, and Nikolaos Bourbakis. "A survey of skin-color modeling and detection methods." Pattern recognition 40.3 (2007): 1106-1122.

[3] Oliveira, V. A., and A. Conci. "Skin Detection using HSV color space." H. Pedrini, & J. Marques de Carvalho, Workshops of Sibgrapi. 2009.

[4] Ibrahim, Nada B., Mazen M. Selim, and Hala H. Zayed. "A dynamic skin detector based on face skin tone color." Informatics and Systems (INFOS), 2012 8th International Conference on. IEEE, 2012.

[5] Dadgostar, Farhad, and Abdolhossein Sarrafzadeh. "An adaptive real-time skin detector based on Hue thresholding: A comparison on two motion tracking methods." Pattern Recognition Letters 27.12 (2006): 1342-1352.

[6] Ince, Ibrahim Furkan, Manuel Socarras-Garzon, and Tae-Cheon Yang. "Hand Mouse: Real Time Hand Motion Detection System Based on Analysis of Finger Blobs." JDCTA 4.2 (2010): 40-56.

[7] Zabulis, Xenophon, Haris Baltzakis, and Antonis Argyros. "Vision-based hand gesture recognition for human-computer interaction." The Universal Access Handbook. LEA (2009).

[8] Vo, N., Tran, Q., Dinh, T. B., Dinh, T. B., & Nguyen, Q. M. "An efficient human-computer interaction framework using skin color tracking and gesture recognition." Computing and Communication Technologies, Research, Innovation, and Vision for the Future (RIVF), 2010 IEEE RIVF International Conference on. IEEE, 2010.

[9] EVANS, Francine; SKIENA, Steven; VARSHNEY, Amitabh. "VType: Entering text in a virtual world." submitted to International Journal of Human-Computer Studies.

[10] Wu, Andrew, Mubarak Shah, and Niels da Vitoria Lobo. "A virtual 3D blackboard: 3D finger tracking using a single camera." Automatic Face and Gesture Recognition, 2000. Proceedings. Fourth IEEE International Conference on. IEEE, 2000.

[11] Zhang, Z., Wu, Y., Shan, Y., & Shafer, S. "Visual panel: virtual mouse, keyboard and 3D controller with an ordinary piece of paper." Proceedings of the 2001 workshop on Perceptive user interfaces. ACM, 2001.

[12] Terajima, Kazuhiro, Takashi Komuro, and Masatoshi Ishikawa. "Fast finger tracking system for in-air typing interface." CHI'09 Extended Abstracts on Human Factors in Computing Systems. ACM, 2009.

[13] http://docs.opencv.org/doc/tutorials/imgproc/histograms/back_projection/back_projection.html

[14] http://en.wikipedia.org/wiki/Median_filter

[15] Allen, John G., Richard YD Xu, and Jesse S. Jin. "Object tracking using camshift algorithm and multiple quantized feature spaces." Proceedings of the Pan-Sydney area workshop on Visual information processing. Australian Computer Society, Inc., 2004.

[16] http://opencv.org/

[17] Sato, Yoichi, Yoshinori Kobayashi, and Hideki Koike. "Fast tracking of hands and fingertips in infrared images for augmented desk interface." Automatic Face and Gesture Recognition, 2000. Proceedings. Fourth IEEE International Conference on. IEEE, 2000.

[18] http://en.wikipedia.org/wiki/Kalman_filter

[19] http://www.cves.org.tw/tcool-file/list.php?id=7