# CamK: Camera-Based Keystroke Detection and Localization for Small Mobile Devices

Yafeng Yin, *Member, IEEE*, Qun Li, *Fellow, IEEE*, Lei Xie, *Member, IEEE*, Shanhe Yi, Ed Novak, and Sanglu Lu, *Member, IEEE*

**Abstract**—Because of the smaller size of mobile devices, text entry with on-screen keyboards becomes inefficient. Therefore, we present CamK, a camera-based text-entry method, which can use a panel (e.g., a piece of paper) with a keyboard layout to input text into small devices. With the built-in camera of the mobile device, CamK captures images during the typing process and utilizes image processing techniques to recognize the typing behavior, i.e., extract the keys, track the user's fingertips, detect, and locate keystrokes. To achieve high accuracy of keystroke localization and low false positive rate of keystroke detection, CamK introduces the initial training and online calibration. To reduce the time latency, CamK optimizes computation-intensive modules by changing image sizes, focusing on target areas, introducing multiple threads, removing the operations of writing or reading images. Finally, we implement CamK on mobile devices running Android. Our experimental results show that CamK can achieve above 95 percent accuracy in keystroke localization, with only a 4.8 percent false positive rate. When compared with on-screen keyboards, CamK can achieve a 1.25X typing speedup for regular text input and 2.5X for random character input. In addition, we introduce word prediction to further improve the input speed for regular text by 13.4 percent.

**Index Terms**—Mobile text-entry, camera, keystroke detection and localization, small mobile devices

## 1 Introduction

In recent years, we have witnessed a rapid development of electronic devices and mobile technology. Mobile devices (e.g., smartphones, Apple Watch) have become smaller and smaller, in order to be carried everywhere easily, while avoiding carrying bulky laptops all the time. However, the small size of the mobile device brings many new challenges, a typical example is inputting text into the small mobile device without a physical keyboard.

In order to get rid of the constraint of bulky physical keyboards, many virtual keyboards have been proposed, e.g., wearable keyboards, on-screen keyboards, projection keyboards, etc. However, *wearable keyboards* introduce additional equipments like rings [1] and gloves [2]. *On-screen keyboards* [3], [4] usually take up a large area on the screen and only support single finger for text entry. Typing with a small screen becomes inefficient. *Projection keyboards* [5], [6] often need a visible light projector or lasers to display the keyboard. To remove the additional hardwares, audio signal [7] and camera based virtual keyboards [8], [9] are proposed. However, UbiK [7] requires the user to click keys with their fingertips and nails, while the existing camera based keyboards either slow the typing speed [8], or should be used in controlled environments [9]. The existing schemes are difficult to provide a similar user experience to using physical keyboards.

To provide a PC-like text-entry experience, we propose a camera-based keyboard CamK, a more natural and intuitive text-entry method. As shown in Fig. 1, CamK works with the front-facing camera of the mobile device and a paper keyboard. CamK takes pictures as the user types on the paper keyboard, and uses image processing techniques to detect and locate keystrokes. Then, CamK outputs the corresponding character of the pressed key. CamK can be used in a wide variety of scenarios, e.g., the office, coffee shops, outdoors, etc. However, to make CamK work well, we need to solve the following key technical challenges.

*(1) Location Deviation:* On a paper keyboard, the inter-key distance is only about two centimeters [7]. With image processing techniques, there may exist a position deviation between the real fingertip and the detected fingertip. This deviation may lead to localization errors of keystrokes. To address this challenge, CamK introduces the initial training to get the optimal parameters for image processing. Then, CamK uses an extended region to represent the detected fingertip, to tolerate the position deviation. Besides, CamK utilizes the features of a keystroke (e.g., the fingertip is located in the key for a certain duration, the pressed key is partially obstructed by the fingertip, etc.) to verify the validity of a keystroke.

*(2) False Positives:* A false positive occurs when a non-keystroke (i.e., a period in which no fingertip is pressing any key) is recognized as a keystroke. Without the assistance of

- *Y. Yin, L. Xie, and S. Lu are with the State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210023, China. E-mail: {yafeng, lxie, sanglu}@nju.edu.cn.*
- *Q. Li and S. Yi are with the Department of Computer Science, College of William and Mary, Williamsburg, VA 23187. E-mail: {liqun, syi}@cs.wm.edu.*
- *E. Novak is with the Computer Science Department, Franklin and Marshall College, Lancaster, PA 17604. E-mail: enovak@fandm.edu.*

*Manuscript received 3 Feb. 2017; revised 24 Dec. 2017; accepted 15 Jan. 2018. Date of publication 25 Jan. 2018; date of current version 29 Aug. 2018. (Corresponding author: Lei Xie.)*
*For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.*
*Digital Object Identifier no. 10.1109/TMC.2018.2798635*

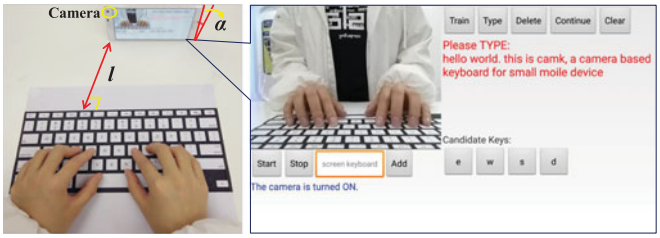



Fig. 1. A typical use case of CamK.

other resources like audio signals, CamK should detect keystrokes only with images. To address this challenge, CamK combines keystroke detection with keystroke localization. For a potential keystroke, if there is no valid key pressed by the fingertip, CamK will remove the keystroke and recognize it as a non-keystroke. Additionally, CamK introduces online calibration, i.e., using the movement features of the fingertip after a keystroke, to further decrease the false positive rate.

*(3) Processing Latency:* To serve as a text-entry method, when the user presses a key on the paper keyboard, CamK should output the character of the key without any noticeable latency. However, due to the limited computing resources of small mobile devices, the heavy computation overhead of image processing will lead to a large latency. To address this challenge, CamK optimizes the computation-intensive modules by adaptively changing image sizes, focusing on the target area in the large-size image, adopting multiple threads and removing the operations of writing/reading images.

We make the following contributions in this paper (a preliminary version of this work appeared in [10]).

- We design a practical framework for CamK, which operates using a smart mobile device camera and a portable paper keyboard. Based on image processing, CamK can detect and locate the keystroke with high accuracy and low false positive rate.
- We realize real time text-entry for small mobile devices with limited resources, by optimizing the computation-intensive modules. Additionally, we introduce word prediction to further improve the input speed and reduce the error rate.
- We implement CamK on smartphones running Android. We first evaluate each module in CamK. Then, we conduct extensive experiments to test the performance of CamK. After that, we compare CamK with other methods in input speed and error rate.

## 2 Related Work

Considering the small sizes of mobile devices, a lot of virtual keyboards are proposed for text entry, e.g., wearable keyboards, on-screen keyboards, projection keyboards, camera based keyboards, etc.

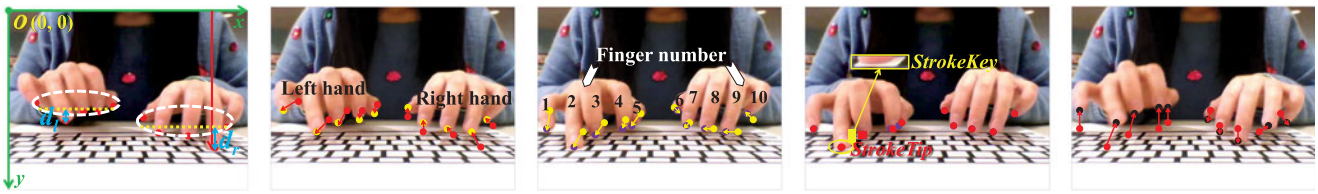
*Wearable Keyboards.* Wearable keyboards sense and recognize the typing behavior based on the sensors built into rings [1], [11], gloves [12], and so on. TypingRing [13] utilizes the embedded sensors of the ring to input text. Finger-Joint keypad [14] works with a glove equipped with the pressure sensors. The Senseboard [2] consists of two rubber pads and senses the movements in the palm to get keystrokes. Funk et al. [15] utilize a touch sensitive wristband to enter text based on the location of the touch. These wearable keyboards often need the user to wear devices around the hands or fingers, thus leading to the decrease of user experience.

*On-Screen Keyboards.* On-screen keyboards allow the user to enter characters on a touch screen. Considering the limited area of the keyboard on the screen, BigKey [3] and ZoomBoard [4] adaptively change the size of keys. ContextType [16] leverages hand postures to improve mobile touch screen text entry. Kwon et al. [17] introduce the regional error correction method to reduce the number of necessary touches. ShapeWriter [18] recognizes a word based on the trace over successive letters in the word. Sandwich keyboard [19] affords ten-finger touch typing by utilizing a touch sensor on the back side of a device. Usually, on-screen keyboards occupy the screen area and support only one finger for typing. Besides, it often needs to switch between different screens to type letters, digits, punctuations, etc.

*Projection Keyboards.* Projection keyboards usually need a visible light projector or lasers to cast a keyboard, and then utilize image processing methods [5] or infrared light [6] to detect the typing events. Hu et al. use a pico-projector to project the keyboard on the table, and then detect the touch interaction by the distortion of the keyboard projection [20]. Roeber et al. utilize a pattern projector to display the keyboard layout on the flat surface, and then detect the keyboard events based on the intersection of fingers and infrared light [21]. The projection keyboard often requires the extra equipments, e.g., a visible light projector, infrared light modules, etc. The extra equipments increase the cost and introduce the inconvenience of text entry.

*Camera Based Keyboards.* Camera based virtual keyboards use the captured image [22] or video [23] to infer the keystroke. Gesture keyboard [22] gets the input by recognizing the finger's gesture. It works without a keyboard layout, thus the user needs to remember the mapping between the keys and the finger's gestures. Visual Panel [8] works with a printed keyboard on a piece of paper. It requires the user to use only one finger and wait for one second before each keystroke. Malik et al. present the Visual Touchpad [24] to track the 3D positions of the fingertips based on two downward-pointing cameras and a stereo. Adajania et al. [9] detect the keystroke based on shadow analysis with a standard web camera. Hagara et al. estimate the finger positions and detect clicking events based on edge detection, fingertip localization, etc [25]. In regard to the iPhone app paper keyboard [26], which only allows the user to use one finger to input letters. The above research work usually focuses on detecting and tracking the fingertips, instead of locating the fingertip in a key's area of the keyboard, which is researched in our paper.

In addition to the above text-entry solutions, MacKenzie et al. [27] describe the text entry for mobile computing. Zhang et al. [28] propose Okuli to locate user's finger based on visible light communication modules, LED, and light sensors. Wang et al. [7] propose UbiK to locate the keystroke based on audio signals. The existing work usually needs extra equipments, or only allows one finger to type, or needs to change the user's typing behavior, while difficult to provide a PC-like text-entry experience. In this paper, we propose a text-entry method based on the built-in camera of the mobile device and a paper keyboard, to provide a similar user experience to using physical keyboards.




(a) Frame 1 (b) Frame 2 (c) Frame 3 (d) Frame 4 (e) Frame 5

Fig. 2. Frames during two consecutive keystrokes.

## 3 Feasibility Study and Overview of CamK

In order to show the feasibility of locating keystrokes based on image processing techniques, we first show the observations of a keystroke from the camera's view. After that, we will describe the system overview of CamK.

### 3.1 Observations of a Keystroke

In Fig. 2, we show the frames/images captured by the camera during two consecutive keystrokes. The origin of axes is located in the top left corner of the image, as shown in Fig. 2a. The hand located in the left area of the image is called *left hand*, while the other is called the *right hand*, as shown in Fig. 2b. From left to right, the fingers are called finger $i$ in sequence, $i \in [1, 10]$, as shown in Fig. 2c. The fingertip pressing the key is called *StrokeTip*, while that pressed key is called *StrokeKey*, as shown in Fig. 2d.

When the user presses a key, i.e., a keystroke occurs, the *StrokeTip* and *StrokeKey* often have the following features, which can be used to track, detect and locate the keystroke.

(1) *Coordinate position:* The *StrokeTip usually has the largest vertical coordinate* among the fingers on the same hand, because the user tends to stretch out one finger when typing a key. An example is finger 9 in Fig. 2a. While considering the particularity of thumbs, this feature may not be suitable for thumbs. Therefore, we separately detect the *StrokeTip* in thumbs and other fingertips.

(2) *Moving state:* The *StrokeTip stays on the StrokeKey for a certain duration* in a typing operation, as finger 2 shown in Figs. 2c and 2d. If the positions of the fingertip keep unchanged, a keystroke may happen.

(3) *Correlated location:* The *StrokeTip is located in the StrokeKey*, in order to press that key, such as finger 9 shown in Fig. 2a and finger 2 shown in Fig. 2d.

(4) *Obstructed view:* The *StrokeTip obstructs the StrokeKey* from the view of the camera, as shown in Fig. 2d. The ratio of the visually obstructed area to the whole area of the key can be used to verify whether the key is really pressed.

(5) *Relative distance:* The *StrokeTip usually achieves the largest vertical distance between the fingertip and remaining fingertips of the same hand.* This is because the user usually stretches out the finger to press a key. Thus the feature can be used to infer which hand generates the keystroke. In Fig. 2a, the vertical distance $d_r$ between the *StrokeTip* (i.e., Finger 9) and remaining fingertips in right hand is larger than that ($d_l$) in left hand. Thus we choose finger 9 as the *StrokeTip* from two hands, instead of finger 2.

### 3.2 System Overview

As shown in Fig. 1, CamK works with a mobile device and a paper keyboard. The device uses the front-facing camera to capture the typing process, while the paper keyboard is placed on a flat surface and located in the camera's view. We take Fig. 1 as an example to describe the deployment. In Fig. 1, the mobile device is a Samsung N9109W smartphone, while $l$ means the distance between the device and the printed keyboard, $\alpha$ means the angle between the plane of the device's screen and that of the keyboard. In Fig. 1, we set $l = 13.5$ cm, $\alpha = 90°$, to make the letter keys large enough in the camera's view. In fact, there is no strict requirements of the above parameters' value, especially when the position of the camera varies in different devices. In Fig. 1, when we fix the A4 sized paper keyboard, $l$ can range in $[13.5 \text{ cm}, 18.0 \text{ cm}]$, while $\alpha$ can range in $[78.8°, 90.0°]$. In CamK, even if some part of the keyboard is out of the camera's view, CamK still works.

The architecture of CamK is shown in Fig. 3. The input is the image taken by the camera and the output is the character of the pressed key. Before a user begins typing, CamK uses *Key Extraction* to detect the keyboard and extract each key from the image. When the user types, CamK uses *Fingertip Detection* to extract the user's hands and detect their fingertips. Based on the movements of fingertips, CamK uses *Keystroke Detection and Localization* to detect a possible keystroke and locate the keystroke. Finally, CamK uses *Text Output* to output the character of the pressed key.

## 4 System Design

According to Fig. 3, CamK consists of four components: key extraction, fingertip detection, keystroke detection and localization, and text output. Obviously, text output is easy to be implemented. Therefore, we mainly describe the first three components.

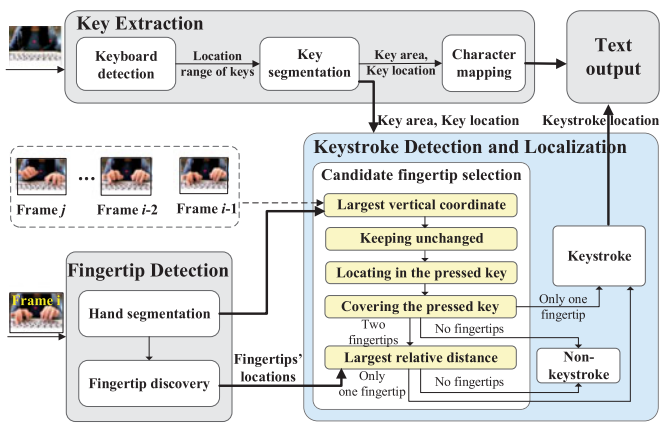
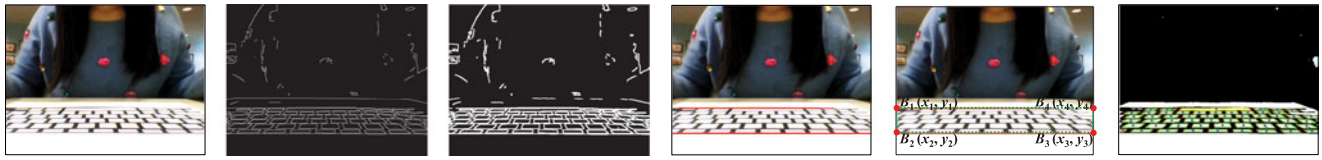



Fig. 3. Architecture of CamK.




(a) An input image (b) Edge detection (c) Edge optimization (d) Keyboard range (e) Keyboard boundary (f) Key segmentation

Fig. 4. Keyboard detection and key extraction.

### 4.1 Key Extraction

Without loss of generality, CamK adopts the common QWERTY keyboard layout, which is printed in black and white on a piece of paper, as shown in Fig. 1. In order to eliminate the effects of background, we first detect the boundary of the keyboard. Then, we extract each key from the keyboard. Therefore, key extraction contains three parts: keyboard detection, key segmentation, and mapping the characters to the keys, as shown in Fig. 3.

#### 4.1.1 Keyboard Detection

We use the Canny edge detection algorithm [29] to obtain the edges of the keyboard. Fig. 4b shows the edge detection result of Fig. 4a. However, the interference edges (e.g., the paper's edge/longest edge in Fig. 4b) should be removed. Based on Fig. 4b, the edges of the keyboard should be close to the edges of keys. We use this feature to remove pitfall edges, the result is shown in Fig. 4c. Additionally, we adopt the dilation operation [30] to join the dispersed edge points which are close to each other, to get better edges/boundaries of the keyboard. After that, we use the Hough transform [8] to detect the lines in Fig. 4c. Then, we use the uppermost line and the bottom line to describe the position range of the keyboard, as shown in Fig. 4d. Similarly, we can use the Hough transform [8] to detect the left/right edge of the keyboard. If there are no suitable edges detected by the Hough transform, it is usually because the keyboard is not perfectly located in the camera's view. In this case, we simply use the left/right boundary of the image to represent the left/right edge of the keyboard. As shown in Fig. 4e, we extend the four edges (lines) to get four intersections $B_1(x_1, y_1), B_2(x_2, y_2), B_3(x_3, y_3), B_4(x_4, y_4)$, which are used to describe the boundary of the keyboard.

#### 4.1.2 Key Segmentation

Considering the short interference edges generated by the edge detection algorithm, it is difficult to accurately segment each key from the keyboard with detected edges. Consequently, we utilize the color difference between the white keys and the black background and the area of a key for key segmentation, to reduce the effect of pitfall areas.

First, we introduce color segmentation to distinguish the white keys and black background. Considering the convenience of image processing, we represent the color in YCrCb space. In YCrCb space, the color coordinate (Y, Cr, Cb) of a white pixel is (255, 128, 128), while that of a black pixel is (0, 128, 128). Thus, we only compute the difference in the $Y$ value between the pixels to distinguish the white keys from the black background. If a pixel is located in the keyboard, while satisfying $255 - \varepsilon_y \leq Y \leq 255$, the pixel belongs to a key. The offsets $\varepsilon_y \in \mathbb{N}$ of $Y$ is mainly caused by light conditions. $\varepsilon_y$ can be estimated in the initial training (see Section 5.1). The initial/default value of $\varepsilon_y$ is 50.

When we obtain the white pixels, we need to get the contours of keys and separate the keys from one another. To avoid pitfall areas such as small white areas which do not belong to any key, we introduce the area of a key. Based on Fig. 4e, we first use $B_1, B_2, B_3, B_4$ to calculate the area $S_b$ of the keyboard as $S_b = \frac{1}{2} \cdot (|\overrightarrow{B_1B_2} \times \overrightarrow{B_1B_4}| + |\overrightarrow{B_3B_4} \times \overrightarrow{B_3B_2}|)$. Then, we calculate the area of each key. We use $N$ to represent the number of keys in the keyboard. Considering the size difference between keys, we treat larger keys (e.g., the space key) as multiple regular keys (e.g., A-Z, 0-9). For example, the space key is treated as five regular keys. In this way, we will change $N$ to $N_{avg}$. Then, we can estimate the average area of a regular key as $S_b/N_{avg}$. In addition to size difference between keys, the camera's view can also affect the area of a key in the image. Therefore, we introduce $\alpha_l$, $\alpha_h$ to describe the range of a valid area $S_k$ of a key as $S_k \in [\alpha_l \cdot \frac{S_b}{N_{avg}}, \alpha_h \cdot \frac{S_b}{N_{avg}}]$. We set $\alpha_l = 0.15$, $\alpha_h = 5$ in CamK, based on extensive experiments. The key segmentation result of Fig. 4e is shown in Fig. 4f. Then, we use the location of the space key (biggest key) to locate other keys, based on the relative locations between keys.

### 4.2 Fingertip Detection

After extracting the keys, we need to track the fingertips to detect and locate the keystrokes. To achieve this goal, we should first detect the fingertip with hand segmentation and fingertip discovery, as shown below.

#### 4.2.1 Hand Segmentation

Skin segmentation [30] is often used for hand segmentation. In the YCrCb color space, a pixel (Y, Cr, Cb) is determined to be a skin pixel, if it satisfies $Cr \in [133, 173]$ and $Cb \in [77, 127]$. However, the threshold values of $Cr$ and $Cb$ can be affected by the surroundings such as lighting conditions. It is difficult to choose suitable threshold values for $Cr$ and $Cb$. Therefore, we combine Otsu's method [31] and the red channel in YCrCb color space for skin segmentation.

In the YCrCb color space, the red channel $Cr$ is essential to human skin color. Therefore, with a captured image, we use the grayscale image that is split based on the $Cr$ channel as an input for Otsu's method [31]. Otsu's method can automatically perform clustering-based image thresholding, i.e., calculate the optimal threshold to separate the foreground and background. The hand segmentation result of Fig. 5a is shown in Fig. 5b, where the white regions represent the hand regions with high value in $C_r$ channel, while the black regions represent the background. However, around the hands, there exist some interference regions, which may change the contours of fingers, resulting in detecting wrong


(a) An input image (b) Hand segmentation (c) Optimization (d) Fingers'contour (e) Fingertip discovery (f) Fingertips

Fig. 5. Fingertip detection.

fingertips. Thus, CamK introduces the following erosion and dilation operations [32]. We first use the erosion operation to isolate the hands from keys and separate each finger. Then, we use the dilation operation to smooth the edge of the fingers. Fig. 5c shows the optimized result of hand segmentation. After that, we select the top two segmented areas as hand regions, i.e., left hand and right hand, to further reduce the effect of inference regions, such as the red areas in Fig. 5c.

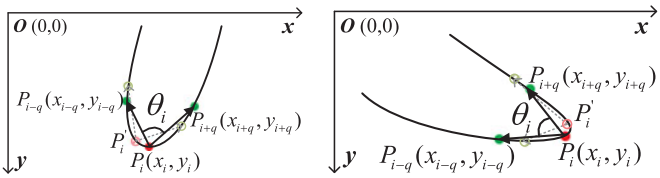
#### 4.2.2 Fingertip Discovery

After we extract the fingers, we will detect the fingertips. We can differentiate between the thumbs (i.e., finger 5-6 in Fig. 2c) and non-thumbs (i.e., finger $1-4$, $7-10$ in Fig. 2c) in shape and typing movement, as shown in Fig. 6.

In a non-thumb, the fingertip is usually a convex vertex, as shown in Fig. 6a. For a point $P_i(x_i, y_i)$ located in the contour of a hand, by tracing the contour, we can select the point $P_{i-q}(x_{i-q}, y_{i-q})$ before $P_i$ and the point $P_{i+q}(x_{i+q}, y_{i+q})$ after $P_i$. Here, $i, q \in \mathbb{N}$. We calculate the angle $\theta_i$ between the two vectors $\overrightarrow{P_iP_{i-q}}$, $\overrightarrow{P_iP_{i+q}}$, according to Eq. (1). In order to simplify the calculation for $\theta_i$, we map $\theta_i$ in the range $\theta_i \in [0°, 180°]$. If $\theta_i \in [\theta_l, \theta_h], \theta_l < \theta_h$, we call $P_i$ a candidate vertex. Considering the relative locations of the points, $P_i$ should also satisfy $y_i > y_{i-q}$ and $y_i > y_{i+q}$. Otherwise, $P_i$ will not be a candidate vertex. If there are multiple candidate vertexes, such as $P_i^{'}$ in Fig. 6a, we will choose the vertex having the largest vertical coordinate, because it has higher probability of being a fingertip, as $P_i$ shown in Fig. 6a. Here, the largest vertical coordinate means the local maximum in a finger's contour, such as the red circle shown in Fig. 5e. The range of a finger's contour can be limited by Eq. (1), i.e., the angle feature of a finger. Based on extensive experiments, we set $\theta_l = 60°$, $\theta_h = 150°$, $q = 20$ in this paper

$$\theta_i = \arccos \frac{\overrightarrow{P_iP_{i-q}} \cdot \overrightarrow{P_iP_{i+q}}}{|\overrightarrow{P_iP_{i-q}}| \cdot |\overrightarrow{P_iP_{i+q}}|}. \quad (1)$$

In a thumb, the "fingertip" also means a convex vertex of the finger. Thus we still use Eq. (1) to represent the shape of the fingertip in a thumb. However, the position of the convex vertex can be different from that of a non-thumb. As




(a) Fingertips (non-thumbs) (b) A thumb

Fig. 6. Features of a fingertip.

shown in Fig. 6b, the relative positions of $P_{i-q}$, $P_i$, $P_{i+q}$ are different from that in Fig. 6a. In Fig. 6b, we show the thumb of the left hand. Obviously, $P_{i-q}$, $P_i$, $P_{i+q}$ do not satisfy $y_i > y_{i-q}$ and $y_i > y_{i+q}$. Therefore, we use $(x_i - x_{i-q}) \cdot (x_i - x_{i+q}) > 0$ to describe the relative locations of $P_{i-q}$, $P_i$, $P_{i+q}$ in thumbs. Then, we choose the vertex with largest vertical coordinate in a finger's contour as the fingertip, as mentioned in the last paragraph.

In fingertip detection, we only need to detect the points located on the bottom edge (from the left most point to the right most point) of the hand, such as the blue contour of right hand in Fig. 5d. The shape feature $\theta_i$ and the positions in vertical coordinates $y_i$ along the bottom edge are shown Fig. 5e. If we can detect five fingertips in a hand with $\theta_i$ and $y_{i-q}$, $y_i$, $y_{i+q}$, we assume that we have also found the thumb. At this time, the thumb presses a key like a non-thumb. Otherwise, we detect the fingertip of the thumb in the right most area of left hand or left most area of right hand according to $\theta_i$ and $x_{i-q}$, $x_i$, $x_{i+q}$. The detected fingertips of Fig. 5a are marked in Fig. 5f.

### 4.3 Keystroke Detection and Localization

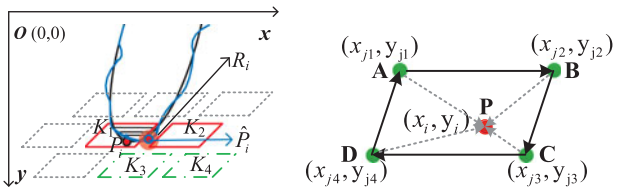
After detecting the fingertip, we will track the fingertip to detect a possible keystroke and locate it for text entry. The keystroke is usually correlated with one or two fingertips, therefore we first *select the candidate fingertip* having a high probability of pressing a key, instead of detecting all fingertips, to reduce the computation overhead. Then, we track the candidate fingertip *to detect the possible keystroke*. Finally, we correlate the candidate fingertip with the pressed key to *locate the keystroke*.

#### 4.3.1 Candidate Fingertip Selection in Each Hand

CamK allows the user to use all of their fingers for text-entry, thus the keystroke may come from the left or right hand. Based on the observations (see Section 3.1), the fingertip (i.e., *StrokeTip*) pressing the key usually has the largest vertical coordinate in that hand, such as finger 9 shown in Fig. 2a. Therefore, we first select the candidate fingertip with the largest vertical coordinate in each hand. We respectively use $C_l$ and $C_r$ to represent the points located in the contour of left hand and right hand. For a point $P_l(x_l, y_l) \in C_l$, if $P_l$ satisfies $y_l \geq y_j (\forall P_j(x_j, y_j) \in C_l, j \neq l)$, then $P_l$ will be selected as the candidate fingertip in the left hand. Similarly, we can get the candidate fingertip $P_r(x_r, y_r)$ in the right hand. In this step, we only need to get $P_l$ and $P_r$, instead of detecting all fingertips.

#### 4.3.2 Keystroke Detection Based on Fingertip Tracking

As described in the observations, when the user presses a key, the fingertip will stay at that key for a certain duration.




(a) Candidate keys (b) Locating a fingertip

Fig. 7. Candidate keys and Candidate fingertips.

Therefore, we can use the location variation of the candidate fingertip to detect a possible keystroke. In Frame $i$, we use $P_{l_i}(x_{l_i}, y_{l_i})$ and $P_{r_i}(x_{r_i}, y_{r_i})$ to represent the candidate fingertips in the left hand and right hand, respectively. If the candidate fingertips in frame $[i-1, i]$ satisfy Eq. (2) in left hand or Eq. (3) in right hand, the corresponding fingertip will be treated as static, i.e., a keystroke probably happens. Based on extensive experiments, we set $\Delta r = 5$ empirically

$$\sqrt{(x_{l_i} - x_{l_{i-1}})^2 + (y_{l_i} - y_{l_{i-1}})^2} \leq \Delta r, \tag{2}$$

$$\sqrt{(x_{r_i} - x_{r_{i-1}})^2 + (y_{r_i} - y_{r_{i-1}})^2} \leq \Delta r. \tag{3}$$

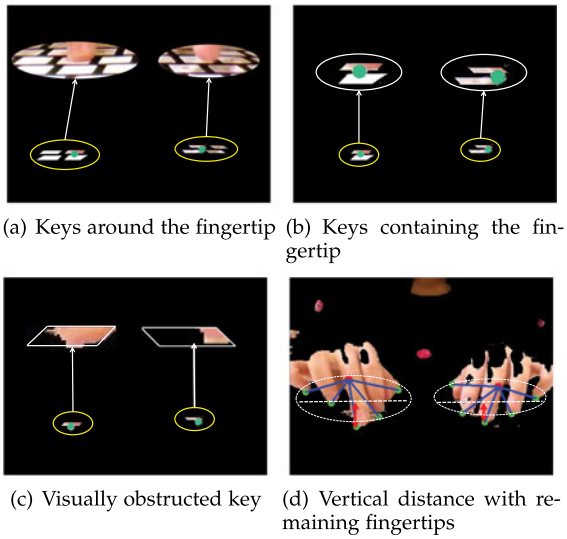
#### 4.3.3 Keystroke Localization by Correlating the Fingertip with the Pressed Key

After detecting a possible keystroke, we correlate the candidate fingertip and the pressed key to locate the keystroke, based on the observations of Section 3.1. In regard to the candidate fingertips, we treat the thumb as a special case, and also select it as a candidate fingertip at first. Then, we get the candidate fingertip set $C_{tip} = \{P_{l_i}, P_{r_i}, \text{left thumb in frame } i, \text{right thumb in frame } i\}$. After that, we can locate the keystroke by using Algorithm 1.

**Algorithm 1.** Keystroke Localization

**Input:** Candidate fingertip set $C_{tip}$ in frame $i$.
Remove fingertips out of the keyboard from $C_{tip}$ .
**for** $P_i \in C_{tip}$ **do**
  Obtain candidate key set $C_{key}$ around $P_i$.
  **for** $K_j \in C_{key}$ **do**
    **if** $P_i$ *is located in* $K_j$ **then**
      Calculate the coverage ratio $\rho_{k_j}$ of $K_j$.
      **if** $\rho_{k_j} < \rho_l$ **then**
        Remove $K_j$ from $C_{key}$.
    **else** Remove $K_j$ from $C_{key}$.
  **if** $C_{key} \neq \emptyset$ **then**
    Select $K_j$ with largest $\rho_{k_j}$ from $C_{key}$.
    $< P_i, K_j >$ forms a possible keystroke.
  **else** Remove $P_i$ from $C_{tip}$.
**if** $C_{tip} = \emptyset$ **then** No keystroke occurs, return.
**if** $|C_{tip}| = 1$ **then** Return the pressed key.
Select $< P_i, K_j >$ with largest ratio $\rho_{k_j}$ in each hand.
Obtain $< P_l, K_l >$ ($< P_r, K_r >$) in left (right) hand.
Calculate relative distance $d_l$ ($d_r$) in left (right) hand.
**if** $d_l > d_r$ **then** Return $K_l$. **else** Return $K_r$.
**Output:** The pressed key.

*Eliminating Impossible Fingertips.* For convenience, we use $P_i$ to represent the fingertip in $C_{tip}$, i.e., $P_i \in C_{tip}, i \in [1, 4]$. If a fingertip $P_i$ is not located in the keyboard region, CamK eliminates it from the candidate fingertips $C_{tip}$.



(a) Keys around the fingertip (b) Keys containing the fingertip

(c) Visually obstructed key (d) Vertical distance with remaining fingertips

Fig. 8. Candidate fingertips/keys in each step.

*Selecting the Nearest Candidate Keys.* For each candidate fingertip $P_i$, we first search the candidate keys which are probably pressed by $P_i$. As shown in Fig. 7a, although the real fingertip is $P_i$, the detected fingertip is $\hat{P}_i$. We use $\hat{P}_i$ to search the candidate keys. We use $K_{cj}(x_{cj}, y_{cj})$ to represent the centroid of key $K_j$. Then we get two rows of keys nearest the location $\hat{P}_i(\hat{x}_i, \hat{y}_i)$ (i.e., the rows with two smallest $|y_{cj} - \hat{y}_i|$). For each row, we select the two nearest keys (i.e., the keys with two smallest $|x_{cj} - \hat{x}_i|$). In Fig. 7a, the candidate key set $C_{key}$ is consisted of $K_1, K_2, K_3, K_4$. Fig. 8a shows the candidate keys of each fingertip.

*Retaining Candidate Keys Containing the Candidate Fingertip.* If a key is pressed by the user, the fingertip will be located in that key. Thus we use the location of the fingertip $\hat{P}_i(\hat{x}_i, \hat{y}_i)$ to verify whether a candidate key contains the fingertip, to remove the invalid candidate keys. As shown in Fig. 7a, there exists a small deviation between the real fingertip and the detected fingertip. Therefore, we extend the range of the detected fingertip to $R_i$, as shown in Fig. 7a. If any point $P_k(x_k, y_k)$ in the range $R_i$ is located in a candidate key $K_j$, $\hat{P}_i$ is considered to be located in $K_j$. $R_i$ is calculated as $\{P_k \in R_i | \sqrt{(\hat{x}_i - x_k)^2 + (\hat{y}_i - y_k)^2} \leq \Delta r\}$. We set $\Delta r = 5$ empirically.

As shown in Fig. 7b, a key is represented as a quadrangle $ABCD$. If a point is located in $ABCD$, when we traverse $ABCD$ clockwise, the point will be located in the right side of each edge in $ABCD$. As shown in Fig. 2a, the origin of coordinates is located in the top left corner of the image. Therefore, if the fingertip $P \in R_i$ satisfies Eq. (4), it is located in the key. CamK will keep it as a candidate key. Otherwise, CamK removes the key from the candidate key set $C_{key}$. In Fig. 7a, $K_1, K_2$ are the remaining candidate keys. The candidate keys contain the fingertip in Fig. 8a is shown in Fig. 8b

$$\begin{aligned} &\overrightarrow{AB} \times \overrightarrow{AP} \geq 0, \overrightarrow{BC} \times \overrightarrow{BP} \geq 0, \\ &\overrightarrow{CD} \times \overrightarrow{CP} \geq 0, \overrightarrow{DA} \times \overrightarrow{DP} \geq 0. \end{aligned} \tag{4}$$

*Calculating the Coverage Ratios of Candidate Keys.* When a key is pressed, it is visually obstructed by the fingertip, as

the dashed area of key $K_1$ shown in Fig. 7a. We use the coverage ratio to measure the visually obstructed area of a candidate key, in order to remove wrong candidate keys. For a candidate key $K_j$, whose area is $S_{k_j}$, the visually obstructed area is $D_{k_j}$, and its coverage ratio is $\rho_{k_j} = \frac{D_{k_j}}{S_{k_j}}$. For a larger key (e.g., the space key), we update $\rho_{k_j}$ by multiplying a key size factor $f_j$, i.e., $\rho_{k_j} = \min(\frac{D_{k_j}}{S_{k_j}} \cdot f_j, 1)$, where $f_j = S_{k_j}/\bar{S_k}$. Here, $\bar{S_k}$ means the average area of a key, i.e, $\bar{S_k} = S_b/N_{avg}$. If $\rho_{k_j} \geq \rho_l$, the key $K_j$ is still a candidate key. Otherwise, CamK removes it from the candidate key set $C_{key}$. We set $\rho_l = 0.25$ by default. For each hand, if there is more than one candidate key, we will keep the key with largest coverage ratio as the *final candidate key*. For a candidate fingertip, if there is no candidate key associated with it, the fingertip will be eliminated. Fig. 8c shows each candidate fingertip and its associated key.

#### 4.3.4 Vertical Distance with Remaining Fingertips

Until now, there is one candidate fingertip in each hand at most. If there are no candidate fingertips, then no keystroke is detected. If there is only one candidate fingertip, then the fingertip is the *StrokeTip* while the associated key is *StrokeKey*, they represent the keystroke. However, if there are two candidate fingertips, we will utilize the vertical distance between the candidate fingertip and the remaining fingertips to choose the most probable *StrokeTip*, as shown in Fig. 2a.

We use $P_l(x_l, y_l)$ and $P_r(x_r, y_r)$ to represent the candidate fingertips in the left hand and right hand, respectively. Then, we calculate the distance $d_l$ between $P_l$ and the remaining fingertips in the *left hand*, and the distance $d_r$ between $P_r$ and the remaining fingertips in the *right hand*. Here, $d_l = |y_l - \frac{1}{4} \cdot \sum_{j=1}^{j=5} y_j, j \neq l|$, while $d_r = |y_r - \frac{1}{4} \cdot \sum_{j=6}^{j=10} y_j, j \neq r|$. Here, $y_j$ represents the vertical coordinate of fingertip $j$. If $d_l > d_r$, we choose $P_l$ as the *StrokeTip*. Otherwise, we choose $P_r$ as the *StrokeTip*. The associated key for the *StrokeTip* is the pressed key *StrokeKey*. In Fig. 8d, we choose fingertip 3 in the left hand as *StrokeTip*. However, considering the effect of camera's view, sometimes $d_l$ ($d_r$) may fail to locate the keystroke accurately. Therefore, for the unselected candidate fingertip (e.g., fingertip 8 in Fig. 8d), we will not discard its associated key directly. Specifically, we sort the previous candidate keys which contain the candidate fingertip based on the coverage ratio in descending order. Finally, we select top four candidates keys and show them on the screen. The user can press the candidate key for text input (see Fig. 1), to tolerate the localization error.

## 5 Optimizations for Keystroke Localization and Image Processing

Considering the deviation caused by image processing, the influence of light conditions, and other factors, we introduce the initial training to select the suitable values of parameters for image processing and utilize online calibration to improve the performance of keystroke detection and localization. In addition, considering the limited resources of small mobile devices, we also introduce multiple optimization techniques to reduce the time latency and energy cost in CamK.

### 5.1 Initial Training

*Optimal Parameters for Image Processing*. For key segmentation (see Section 4.1.2), $\varepsilon_y$ is used for tolerating the change of $Y$ caused by the environment. Initially, $\varepsilon_y = 50$. CamK updates $\varepsilon_{y_i} = \varepsilon_{y_{i-1}} + 1$, when the number of extracted keys decreases, it stops. Then, CamK sets $\varepsilon_y$ to 50 and updates $\varepsilon_{y_i} = \varepsilon_{y_{i-1}} - 1$, when the number of extracted keys decreases, it stops. In the process, when CamK gets maximum number of keys, the corresponding value $\varepsilon_{y_i}$ is selected as the optimal value for $\varepsilon_y$.

In hand segmentation, CamK uses erosion and dilation operations, which respectively use a kernel $B$ [32] to process images. To get a suitable size of $B$, the user first puts his/her hands on the home row of the keyboard (see Fig. 5a). For simplicity, we set the kernel sizes for erosion and dilation to be equal. The initial kernel size is $z_0 = 0$. Then, CamK updates $z_i = z_{i-1} + 1$. When CamK can localize each fingertip in the correct key with $z_i$, then CamK sets the kernel size as $z = z_i$. In initial training, the user puts on the hands based on the on-screen instructions, it usually spends less than 10s.

*Frame Rate Selection.* CamK sets the initial/default frame rate of the camera to be $f_0 = 30$fps (frames per second), which is usually the maximal possible frame rate. We use $n_{0_i}$ to represent the number of frames containing the $i$th keystroke. When the user has pressed $u$ keys, we can get the average number of frames during a keystroke as $\bar{n_0} = \frac{1}{u} \cdot \sum_{i=1}^{i=u} n_{0_i}$. In fact, $\bar{n_0}$ reflects the duration of a keystroke. When the frame rate $f$ changes, the number of frames in a keystroke $\bar{n_f}$ changes. Intuitively, a smaller value of $\bar{n_f}$ can reduce the image processing time, while a larger value of $\bar{n_f}$ can improve the accuracy of keystroke localization. Based on extensive experiments (see Section 7.3), we set $\bar{n_f} = 3$, thus $f = \lceil f_0 \cdot \frac{\bar{n_f}}{\bar{n_0}} \rceil$.

### 5.2 Online Calibration

*Removing False Positive Keystrokes.* Under certain conditions, the user does not type any key while keeping the fingers stationary, CamK may misclassify the non-keystroke as a keystroke. Thus we introduce a *temporary character* to mitigate this problem.

In the process of pressing a key, the *StrokeTip* moves towards the key, stays at that key, and then moves away. The vertical coordinate of the *StrokeTip* first increases, then pauses, then decreases. If CamK has detected a keystroke in $\bar{n_f}$ consecutive frames, it displays the current character on the screen as a temporary character. In the next frame(s), if the position of the *StrokeTip* does not satisfy the features of a keystroke, CamK will cancel the temporary character. This does not have much impact on the user's experience, because of the short time between two consecutive frames. Besides, CamK also displays the candidate keys around the *StrokeTip*, the user can choose them for text input.

*Movement of Smartphone or Keyboard*. CamK presumes that the smartphone and the keyboard do not move while in use. For best results, we recommend the user to tape the paper keyboard on a flat surface. Nevertheless, to alleviate the effect caused by the movements of the mobile device or the keyboard, we offer a simple solution. If the user continuously uses the *Delete* key on the screen multiple times (e.g., larger than 3 times), CamK will inform the user to move


Fig. 9. Changing image sizes and focusing on the target area.

his/her hands away from the keyboard for relocation. After that, the user can continue the typing process. The relocation process just needs the user to move away the hands and it is usually less than 10s.

### 5.3 Real Time Image Processing

As a text-entry method, CamK needs to output the character without noticeable latency. According to Section 4, in order to output a character, we need to capture images, track fingertips, and finally detect and locate the keystroke. The large time cost in image processing leads to large time latency for text output. To solve this problem, we first profile the time cost of each stage in Camk, and then introduce four optimization techniques to reduce the time cost. Unless otherwise specified, the frame rate is set to 30fps by default.

#### 5.3.1 Time Cost in Different Stages

There are three main stages in CamK, i.e., capturing the images, tracking the fingertips, and locating the keystroke. The stages are respectively called 'Cap-img', 'Tra-tip' and 'Loc-key' for short. We frist set the image size to $640 * 480$ pixels which is supported by many smartphones, and then measure the time cost of producing or processing one image in each stage with a Samsung GT-I9100 smartphone (GT-I9100 for short). According to the measurement, the time cost in 'Cap-img', 'Tra-tip', 'Loc-key' is 99, 118, 787 ms, respectively. Here, 'Cap-img' means the time for capturing an image, 'Tra-tip' means the time for processing the image to select the candidate fingertips, 'Loc-key' means the time for processing an image to locate the keystroke. We repeats the measurement for 500 times to get the average time cost.

According to Section 5.1, we need to capture/process three images during a keystroke to guarantee the localization accuracy. Thus we can estimate the minimal time $T_{k1}$ of detecting and locating a keystroke with Eq. (5). Obviously, 1,320 ms is a very large latency, thus more optimizations are expected for CamK to realize real time processing

$$T_{k1} = (99 + 118 + 99 + 118 + 99 + 787)\text{ms} = 1320\ \text{ms}. \quad (5)$$

#### 5.3.2 Adaptively Changing Image Sizes

As descried before, it is rather time-consuming to process the image with $640 * 480$ pixels. Intuitively, if CamK operates on smaller images, it would reduce the time cost. However, to guarantee the keystroke localization accuracy, we can not use a very small image. Therefore, we adaptively adopts different sizes of images for processing, as shown in Fig. 9. We use smaller images to track the fingertips during two keystrokes and use larger images to locate the detected keystroke. Based on extensive experiments, we set the size of the small image to be $120 * 90$ pixels, while the large image size is 480*360 pixels. As shown in Fig. 9, when a keystroke will happen, CamK adaptively changes frame $i$ to be a large image (i.e., 480*360 pixels). After that, CamK changes the following frames to small images (i.e., 120*90 pixels) until the next keystroke is detected. The time cost in 'Cap-img (120*90 pixels)', 'Cap-img (480*360 pixels)' 'Tra-tip (120*90 pixels)', 'Loc-key (480*360 pixels)' is 32, 75, 9, 631 ms, respectively. Then, we can estimate the minimal time cost $T_{k2}$ to detect and locate a keystroke with Eq. (6). Here, $T_{k2}$ is 59.7 percent of $T_{k1}$. However, more optimizations are expected for large-size image processing

$$T_{k2} = (32 + 9 + 32 + 9 + 75 + 631)\text{ms} = 788\ \text{ms}. \quad (6)$$

TABLE 1
Time Cost in GT-I9100 Smartphone
(Image is Described in Pixels)

| | Cap-img (120*90) | Cap-img (480*360) | Tra-tip (120*90) | Loc-key (480*360) |
|---|---|---|---|---|
| Image | 32 ms | 75 ms | 9 ms | 339 ms |
| Reference | 0.02 ms | 0.02 ms | 13 ms | 106 ms |

#### 5.3.3 Optimizing Large-Size Image Processing

Based on Fig. 8, the keystroke is only related to a small area of the large-size image. Thus we optimize CamK by only processing the small area around the *StrokeTip*: the red region of frame $i$ shown in Fig. 9. Suppose the position of the candidate fingertip in frame $i-1$ (small image) is $P_{i-1}(x_c, y_c)$, then we scale the position to $P'_{i-1}(x'_c, y'_c)$ (corresponding position in large image), according to the ratio $\rho_{sl}$ of the small image to the large image in width, i.e, $\frac{x_c}{x'_c} = \frac{y_c}{y'_c} = \rho_{sl}$. Here, $\rho_{sl} = \frac{120}{480}$. Then, CamK only processes the area $S'_c$ in frame $i$ (large image) to reduce time cost, as shown in Eq. (7). We set $\Delta x = 40$, $\Delta y = 20$ by default

$$S'_c = \{P_i(x_i, y_i) \in S'_c | \ |x_i - x'_c| \leq \Delta x, |y_i - y'_c| \leq \Delta y\}. \quad (7)$$

Currently, the processing time for the large-size image is 339 ms. As 'Image' row shown in Table 1, we estimate the minimal time to detect and locate a keystroke with Eq. (8), which is 37.6 percent of that in Eq. (5). However, the processing time 339 ms for the large-size image is still a little high. If CamK works with a single thread, it may miss the next keystroke, due to the large processing time. Thus, CamK is expected to work with multiple threads in parallel

$$T_{k3} = (32 + 9 + 32 + 9 + 75 + 339)\text{ms} = 496\ \text{ms}. \quad (8)$$

#### 5.3.4 Multi-Thread Processing

According the above conclusion, CamK uses three threads to capture, detect and locate the keystrokes in parallel. As shown in Fig. 10, the 'Cap-img' thread captures the images, the 'Tra-tip' thread processes the small images for keystroke detection, and the 'Loc-key' thread processes the large image to locate the keystroke. In this way, CamK will not miss the frames of next keystroke, because the 'Cap-img' thread does not stop taking images. As shown in Fig. 10, Camk utilizes consecutive frames to determine the keystroke and also introduces the online calibration to improve the performance.

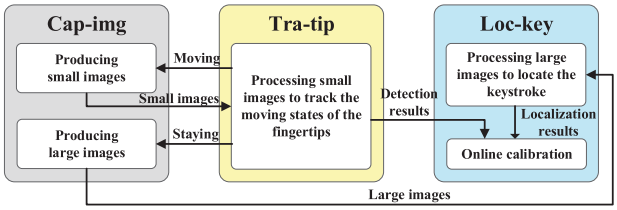



Fig. 10. Multi-thread processing.

By adopting multiple threads, we can estimate the minimal time to detect and locate the keystroke with Eq. (9), which is 36.4 percent of that in Eq. (5). Because the frame rate is 30fps, the interval between two frames is 33 ms. Therefore, we use 33 ms to replace 32 ms ('Cap-img' time). In regard to the 'Tra-tip' time (9 ms), which is simultaneous with the 'Cap-img' time of the next frame, thus not being added in Eq. (9). When comparing with Eq. (8), Eq. (9) does not reduce much processing time. This is mainly caused by the time-consuming operations for writing and reading each image. Therefore, it is better eliminate the operations of writing/reading images

$$T_{k4} = (33 + 33 + 75 + 339)\ \text{ms} = 480\ \text{ms}. \tag{9}$$

#### 5.3.5 Processing without Writing and Reading Images

To remove the operations of writing and reading images, we store the image data captured by the camera in the RAM. Then, CamK accesses the data based on pass-by-reference. It indicates that different functions access the same image data. In this way, we remove the operations of reading and writing images. The corresponding time cost in each stage is shown in the 'Reference' row of Table 1. Here, the time cost for capturing/storing the source data of a small image and a large image is the same. This is because the limitation of hardwares, in preview mode, the size of the source data in each frame is the same (e.g., $480 * 360$ pixels). If we want to get the image with size $120 * 90$ pixels, we will resize the image during image processing.

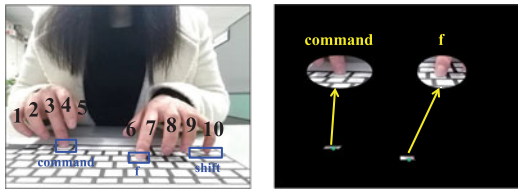
In Table 1, the time cost of image processing includes the time of reading image data and processing the image data. When processing images with $120 * 90$ pixels, CamK only needs to detect the candidate fingertips with hand segmentation and fingertip discovery. When processing images with $480 * 360$ pixels, CamK not only needs to detect the fingertip, but also needs to select the candidate keys, calculate the covered area of the pressed key and correlate the candidate fingertip with the pressed key to locate the keystroke. Thus the time cost of processing the image with $120 * 90$ pixels is smaller than that of the image with $480 * 360$ pixels. According to Table 1, we can estimate the minimal time to detect and locate a keystroke with Eq. (10), which is only 15.5 percent of that in Eq. (5). Here, 33 ms means the waiting time between two images, because the maximum frame rate is 30fps. Unit now, $T_k$ is comparable to the duration of a keystroke, i.e., the output time latency is usually within 50 ms and below human response time [7]

$$T_k = (33 + 33 + 33 + 106)\ \text{ms} = 205\ \text{ms}. \tag{10}$$

### 5.4 Reduction of Power Consumption

To make Camk practical for small mobile devices, we need to reduce the power consumption in CamK, especially in image processing. Based on the definition of *Camera.Parameters* [33] in Android APIs, the adjustable parameters of camera are picture size, preview frame rate, preview size, and camera view size (i.e., window size of camera).

Among the parameters, the *picture size* has no effect on CamK. The *preview frame rate* and *preview size* (i.e., image sizes) have already been optimized in Sections 5.1 and 5.3. Therefore, we only observe how camera view size affect the power consumption by a Monsoon power monitor [34]. When we respectively set the camera view size to 120*90, 240*180, 360*270, 480*360, 720*540, 960*720, 1200*900 pixels, the corresponding power consumption is 1204.6, 1228.4, 1219.8, 1221.8, 1222.9, 1213.5, 1222.2 mW. The camera view size has little effect on power consumption. In this paper, the camera view size is set to 480*360 pixels by default.




(a) An input image (b) Multi-touch

Fig. 11. Multi-touch function in CamK.

## 6 Extension: Multi-Touch and Word Prediction

To provide a better user experience, we add multi-touch to allow the user to use a key combination, e.g., pressing 'shift' and 'a' at the same time. Besides, we introduce *word prediction* to guess the possible word to be typed to improve the text-input speed.

### 6.1 Multi-Touch Function

In Section 4.3, CamK determines one efficient keystroke. However, considering the actual typing behavior, we may use key combinations for special purposes. Take the Apple Wireless Keyboard as an example, we can press 'command' and 'c' at the same time to "copy". Therefore, we introduce the *multi-touch* for CamK, to allow the user to use special key combinations.

In CamK, we consider the case that user presses the key combination by using left hand and right hand at the same time, in order to eliminate ambiguity. In Fig. 11a, we show an example of ambiguity, the 7th finger in right hand presses the key 'f', while the 10th fingertip in the image is located in 'shift', it seems that the user aims to get the capital letter 'F'. In fact, the 3rd finger in left hand presses the key 'command', i.e., the user wants to call the search function instead of getting the letter 'F', as shown in Fig. 11b. Therefore, CamK considers that the user uses key combination with two hands at the same time, i.e., one hand correlates with one keystroke. With the located keystroke in each hand, we verify whether the two keystrokes forms a special key combination. If it is true, CamK calls the corresponding function. Otherwise, CamK determines the only efficient keystroke based on Section 4.3.

### 6.2 Word Prediction

When the user has input one or more characters, CamK will predict the word the user is probably going to type, by

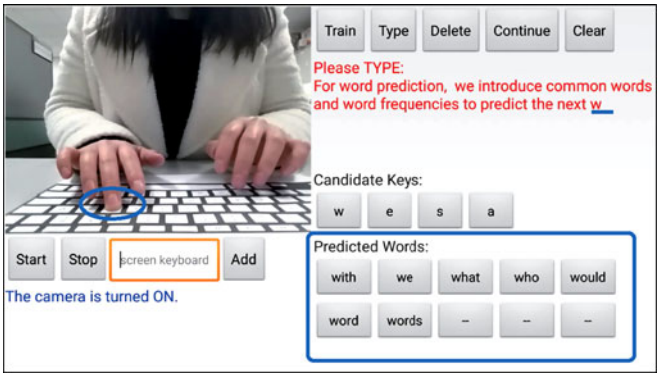



Fig. 12. Word prediction.

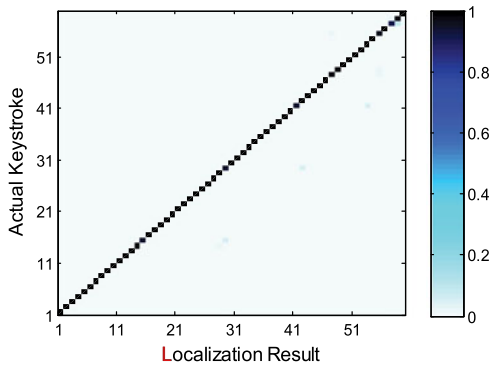



Fig. 13. Confusion matrix of 59 keys.

using the common-word set and word frequencies [35] of the typed words. In regard to the *common-word set*, we introduce $N_c$ most common words [36], which are sorted by frequency of use in descending order. We use $E_i, i \in [1, N_c]$ to represent the priority level of the common word $W_i$, $E_i = \frac{N_c - i + 1}{N_c + 1}$, $E_i \in (0, 1)$. In regard to *word frequencies of the typed words*, we use $F_j$ to represent the frequency of word $W_j$ typed by the user in CamK. Initially, $F_j$ is set to zero. Everytime the user types a word $W_j$, the frequency of $W_j$ increases by one. Whatever the larger priority level or the larger word frequency, it indicates the word has a higher probability to be typed. By matching the prefix $S_{p_k}$ of word $W_k$ with that of common words, we get the $\text{top} - m_c$ candidate words $l_{c_1}$ with the highest $E_i$. Similarly, we get the $\text{top} - m_u$ candidate words $l_{c_2}$ with the highest $F_j$. After that, we merge $l_{c_1}$ and $l_{c_2}$ to get the candidate words in $\{l_{c_1} \cup l_{c_2}\}$. We set $N_c = 1000$, $m_c = m_u = 5$ by default.

As shown in Fig. 12, when the user types 'w', we get $l_{c_1}$ as $\{with, we, what, who, would\}$, while $l_{c_2}$ is $\{word, words, we\}$. Then, we merge the candidate words as $\{with, we, what, who, would, word, words\}$. By pressing the button corresponding to the candidate word (e.g., "word"), the user can omit the following keystrokes (e.g., 'o', 'r' and 'd'). Then, CamK can improve the input speed and reduce the computation/energy overhead.

## 7 Performance Evaluation

We implement CamK on smartphones running Android. We use the layout of Apple Wireless Keyboard (AWK) as the default keyboard layout, which is printed on a piece of US Letter sized paper. Unless otherwise specified, we use the Samsung GT-I9100 smartphone whose Android version

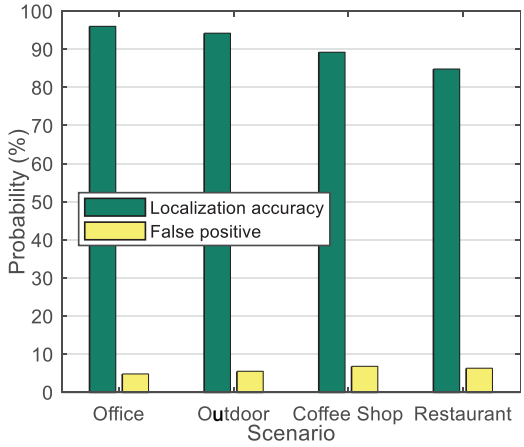



Fig. 14. Four scenarios.

is 4.4.4, the frame rate is 15fps, the image size is $480 * 360$ pixels, and CamK works in the office. We first evaluate each component of CamK. Then, we test the performance of CamK in different environments. Finally, we recruit 9 participants to use CamK and compare the performance of CamK with other text-entry methods.

### 7.1 Localization Accuracy for Known Keystrokes

To verify whether CamK has obtained the optimal parameters for image processing, we first measure the accuracy of keystroke localization with a Samsung SM-N9109W smartphone, when CamK knows a keystroke is happening. The user presses 59 keys (excluding the PC function keys: first row, five keys in last row) on the paper keyboard. Specifically, we let the user press the sentences/words from the standard MacKenzie set [37]. Besides, we introduce some random characters to guarantee that each key will be pressed with fifty times. We record the typing process with a camera. When the occurrence of a keystroke is known, the localization accuracy is close to 100 percent, as shown in Fig. 13. It indicates that CamK can adaptively select suitable values of the parameters used in image processing.

### 7.2 Accuracy in Different Environments

To verify whether CamK can detect and locate the keystroke accurately, we conduct the experiments in four typical scenarios: an office environment (light's color is close to white), outdoors (basic/pure light), a coffee shop (light's color is a little bit closer to that of human skin), and a restaurant (light is a bit dim). In each test, the user types words from the MacKenzie set [37] and makes $N_k = 500$ keystrokes. Suppose CamK locates $N_a$ keystrokes correctly and treats $N_f$ non-keystrokes as keystrokes wrongly. We define the localization accuracy as $p_a = \frac{N_a}{N_k}$, while the false positive rate as $p_f = \min(\frac{N_f}{N_k}, 1)$. As shown in Fig. 14, CamK can achieve high accuracy (close to or larger than 85 percent) with low false positive rate (about 5 percent). In the office, the localization accuracy can achieve above 95 percent.

### 7.3 Effect of Frame Rate

As described in Section 5.1, the frame rate affects the number of images $\bar{n_f}$ during a keystroke. If the value of $\bar{n_f}$ is too small, CamK may miss the keystrokes. On the contrary, more frames

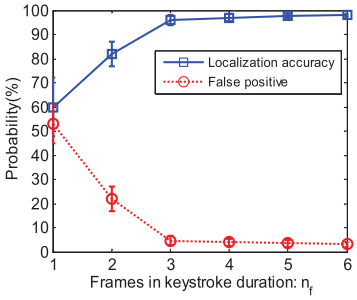



Fig. 15. Frames in a keystroke.

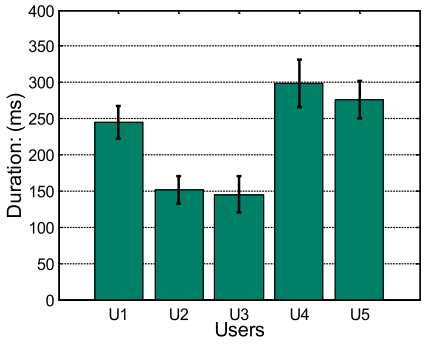



Fig. 16. Duration for a keystroke.

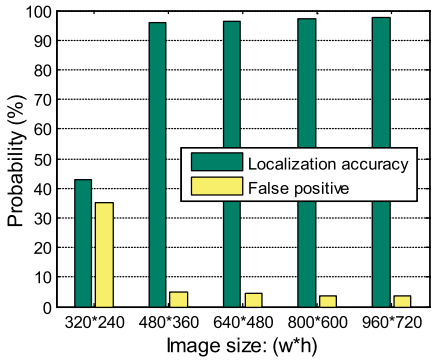



Fig. 17. Accuracy/fase positive versus image sizes.

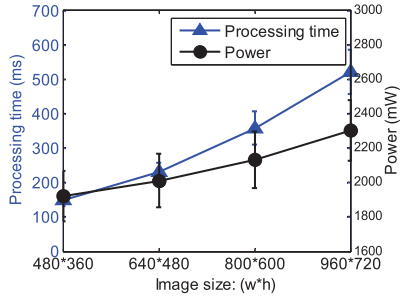



Fig. 18. Processing one image versus image sizes.

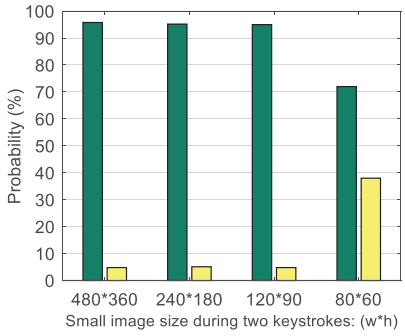



Fig. 19. Changing sizes of small images.

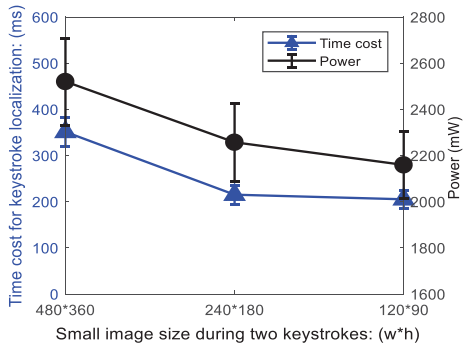



Fig. 20. Locating a keystroke by changing sizes of small images.

will increase the time latency. Based on Fig. 15, when $\bar{n_f} \geq 3$, CamK has good performance. When $\bar{n_f} > 3$, there is no obvious performance improvement. While considering the accuracy, false positive, and time latency, we set $\bar{n_f} = 3$.

Besides, we invited 5 users to test the duration $\Delta t$ of a keystroke. $\Delta t$ represents the time when the *StrokeTip* is located in the *StrokeKey* from the view of the camera. Based on Fig. 16, $\Delta t$ is usually larger than 150 ms. When $\bar{n_f} = 3$, the frame rate is less than the maximum frame rate (30fps), i.e., CamK can work under the frame rate limitation of the smartphone. Therefore, $\bar{n_f} = 3$ is a suitable choice.

### 7.4 Effect of Image Size

At first, we choose a constant image size. Based on Fig. 17, as the size of image increases, the performance of CamK becomes better. When the size is smaller than $480 * 360$ pixels, CamK can not extract the keys correctly, the performance is rather bad. When the size of image is $480 * 360$ pixels, the performance is good. Keeping increasing the size does not bring obvious improvement. However, increasing the image size will increase the time cost and power consumption ( measured by a Monsoon power monitor [34]) for processing an image, as shown in Fig. 18. Based on Section 5.3, CamK adopts both large images and small images. To guarantee high accuracy and low false positive rate, and reduce the time latency and power consumption, the size of the large image is set to $480 * 360$ pixels. In regard to small images, when the image size decreases from $480 * 360$ to $120 * 90$ pixels, CamK has high accuracy and low false positive rate, as shown in Fig. 19. If the size of small images continuously decreases, the accuracy decreases a lot, and the false positive rate increases a lot. However, decreasing the image size means decreasing the time cost/power consumption, as shown in Fig. 20. Combining Figs. 19 and 20, the size of small image is set to $120 * 90$ pixels.

### 7.5 Time Latency and Power Consumption

Based on Fig. 20, the time cost for locating a keystroke is about 200 ms, which is comparable to the duration of a keystroke, as shown in Fig. 16. Thus CamK can output the text without noticeable time latency. The time latency is usually within 50 ms, which is below the human response time [7]. In addition, we measure the power consumption of a

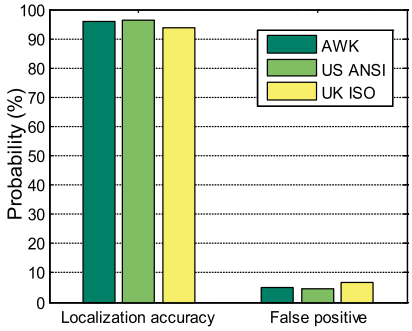



Fig. 21. Accuracy/fase positive versus keyboard layouts.

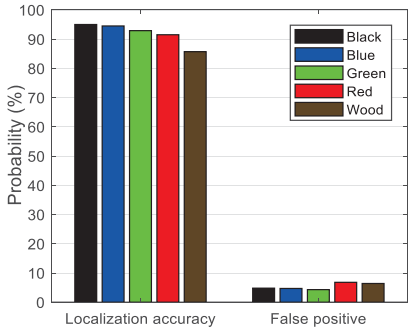



Fig. 22. Accuracy/fase positive versus keyboard colors.

Samsung GT-I9100 smartphone in following states: (1) idle with the screen on; (2) writing an email; (3) keeping the camera on the preview mode (frame rate is 15fps); (4) running CamK (frame rate is 15fps) for text-entry. The power consumption in each state is 505, 1118, 1189, 2159 mW. The power consumption of CamK is a little high. However, using a smartphone with better hardwares can change multiple threads into one thread for power saving.

### 7.6 Effect of Keyboards with Different Layouts, Colors, and Textures

*Different Layouts*. We use three common keyboard layouts, i.e., AWK [38], US ANSI [39], and UK ISO [39], to verify the scalability of CamK. Each keyboard layout is scaled on a piece of US Letter sized paper with similar inter-key distances. Based on Fig. 21, whatever the keyboard layout is, CamK has good performance. It can achieve above 93 percent accuracy of keystroke localization, while the false positive rate is usually less than 7 percent.

*Different Colors/Textures*. We use keyboards with different colors and textures to test CamK. As show in Fig. 22, the background of the keyboard is set to black, blue, green, red and brown wood texture. When there is a large difference between the colors of the keyboard and the skin, e.g., the keyboard with black, blue and green background, the localization accuracy is usually larger than 90 percent, while the false positive rate is lower than 5 percent. In regard to the keyboard with red color or brown wood texture, the color of keyboard is close to that of human skin, it is harder to extract the contour of the fingertip from the keyboard, thus the localization accuracy of CamK decreases.

### 7.7 Effect of Different Phones

In addition to the Samsung GT-I9100 smartphone, we also test CamK in Samsung SM-G9009W (Android version 5.0) and SM-N9109W (Android version 6.0.1) smartphones, in terms of the accuracy of keystroke localization, the false positive rate of keystroke detection, time cost and power consumption. According to Fig. 23, whatever the phone model is, the keystroke localization accuracy can achieve above 90 percent, while the false positive rate is usually less than 6 percent. According to Fig. 24, which shows the time cost of capturing an image, processing a small-size image, processing a large-size image in different phones, SM-G9009 and SM-N9109W smartphone greatly reduce the time cost of the same task. In regard to the total time of detecting and locating a keystroke, i.e., 'Keystroke' bar in Fig. 24, the time cost in SM-G9009W and SM-N9109W smartphone are 54.3 and 58.6 percent of that in GT-I9100 smartphone, respectively. In Fig. 25, we test power consumption in four states: (1) idle with the screen on; (2) writing an email; (3) keeping the camera in the preview mode (the preview size is $480 * 360$ pixels and frame rate is 15fps); (4) running CamK (frame rate is 15pfs). Usually, the power consumption of CamK is a little high, because we use the power-consuming camera module and adopt multiple threads. In future, when the smartphone is configured with better hardwares, we can change the three threads into one thread to reduce power consumption.

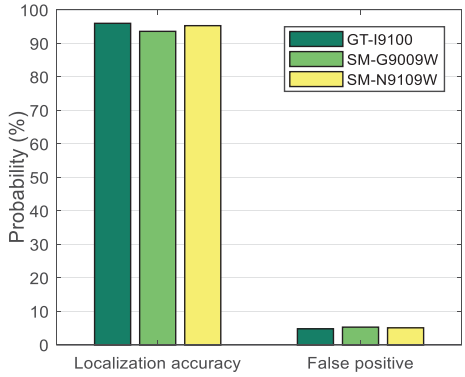



Fig. 23. Accuracy/fase positive versus phone models.

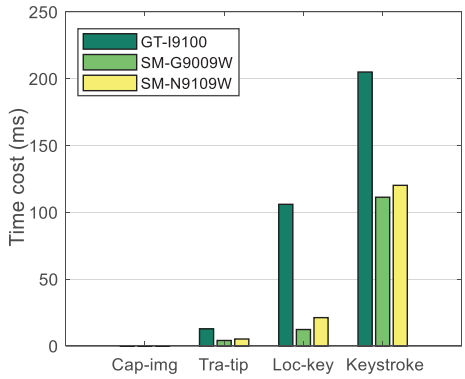



Fig. 24. Time cost versus phone models.

### 7.8 Effect of Skin Colors

We invited four users with different skin tones to observe how the skin color affects the performance of CamK. The skin color of each user is represented with the color in Fig. 26. Intuitively, if the skin color is closer to the color of keys or the keyboard's background, it will be more difficult to extract the finger's contour. However, even if the skin color changes, CamK can still work. As shown Fig. 26, the

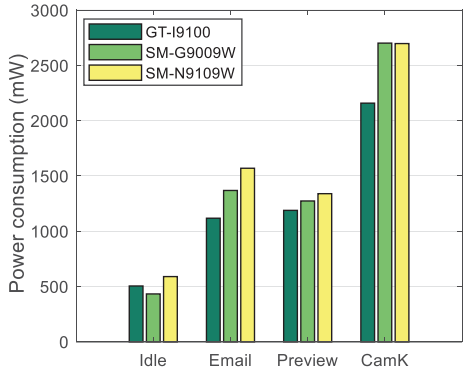



Fig. 25. Power consumption versus phone models.

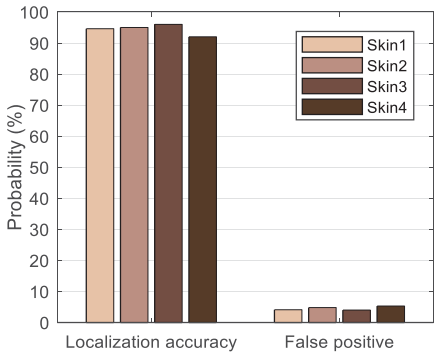



Fig. 26. Accuracy/fase positive versus skin colors.

false positive rate is usually lower than 6.0 percent while the localization accuracy is usually larger than 90.0 percent.

### 7.9 Working in Complex Environments

*Keyboard-Device Distance*. While considering the effect of the camera's view, we test the performance of CamK by changing the distance between the keyboard and the device (Samsung SM-N9109W smartphone). According to Fig. 27, when the distance is smaller than 17 cm, the keystroke localization accuracy is larger than 90 percent. If the distance keeps increasing, the keys look smaller in the camera's view, thus the small deviation of the detected fingertip can lead to keystroke localization error, i.e., the performance of CamK decreases.

*Keyboard's Displacement*. To test whether CamK can tolerate the slight movement of the keyboard, we respectively move the keyboard left, right, up, down with 5 and 8 mm. According to Fig. 28, when the keyboard is moved within 5 mm, the keystroke accuracy is still larger than 90.0 percent. However, when the keyboard is moved with a large distance (e.g., 8 mm), CamK may locate the keystroke wrongly. When moving the keyboard down with 8 mm, the keystroke localization accuracy drops to 63.8 percent.

*Different Surfaces*. To test CamK on different surfaces, we respectively put the paper keyboard on a flat surface on a desk, a matte surface on a windowsill, and a soft surface on a sofa. According to Fig. 29, CamK can work well on flat surface and matte surface. However, on the soft surface, when the user presses a key, her/his fingertip will change the location of the key, then CamK will wrongly match the fingertip with another key originally located in this place, thus the localization accuracy drops under 90.0 percent.

*Dynamic Backgrounds*. To test Camk in dynamic backgrounds, we perform the tests in a room, on the metro

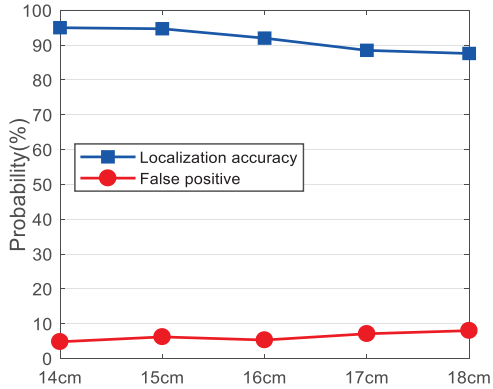



Fig. 27. Accuracy/fase positive versus distance between keyboard and smartphone.

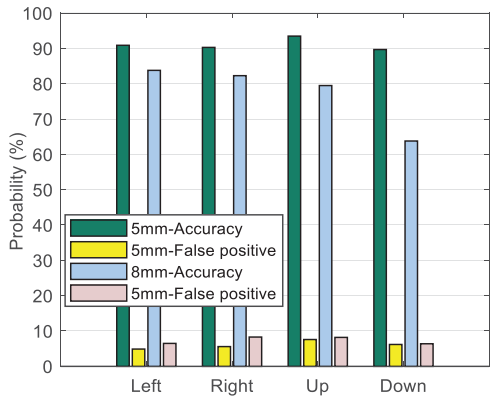



Fig. 28. Accuracy/fase positive versus movements of keyboard.

(subway train), and on a campus bus using a portable lapdesk. When CamK works in a bus, the slowing down and speeding up of the bus will lead to the movement of the panel. When pressing a key, the moved panel will make the finger be located in another key, leading to an error. When CamK works in the environment with a steady moving state, the keystroke localization accuracy is usually close to or larger than 90 percent, as shown in Fig. 30.

## 8 User Study

To verify the efficiency of CamK, we compare CamK with the following three input methods: IBM style PC keyboard, Android on-screen keyboard, and Swype keyboard [40], which allows the user to slide a finger across the keys and use the language mode to guess the word. We recruit 9 participants (3 female and 6 male, $\mu = 24.4$ years old) to test the above input methods in the office. Each participant types the same regular text sentences and random characters in each method. The regular text sentences are picked from the standard MacKenzie set [7], [37]. The random characters are randomly selected from the characters in the AWK keyboard. Before using each keyboard, the user has 10 minutes to familiarize with it. After that, each user respectively spends 10 minutes to type regular text and random characters with each input method. In the experiments, the participant deploys the system and presses the key according to his/her habits. In the trails, the participants can correct the erroneous input as they want. However, if they do not find the mistake until several characters later, they should ignore the mistake and continue the typing

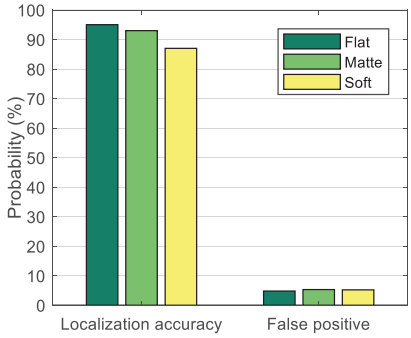



Fig. 29. Accuracy/fase positive versus different surfaces.

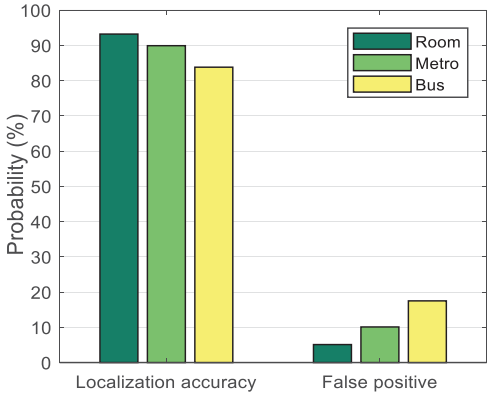



Fig. 30. Accuracy/fase positive under moving environments.

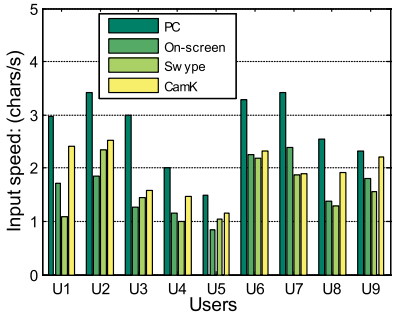



Fig. 31. Input speed with regular text input.

process. We use the input speed (characters per second) and the error rate $p_e = (1 - p_a) + p_f$ as metrics for comparison.

### 8.1 Typing without Word Prediction

*Regular Text Input*. Fig. 31 shows the input speed of each user while inputting regular text. Each user achieves the highest input speed when using the PC keyboard. In regard to other virtual keyboards, CamK can achieve 1.25X typing speedup, when compared to the on-screen keyboard. In CamK, the user can type 1.5-2.5 characters per second. When compared with UbiK [7], CamK improves the input speed about 20 percent. Fig. 32 shows the error rate of each method. Usually, the error rate of CamK is between $5\% - 9\%$, which is comparable to that of UbiK (about $4\% - 8\%$).

*Random Character Input*. Fig. 33 shows the input speed of each user when inputting random characters, which contain a lot of digits and punctuations. The input speed of CamK is comparable to that of a PC keyboard. CamK can achieve 2.5X typing speedup, when compared with the on-screen keyboard and Swype, which need to switch between

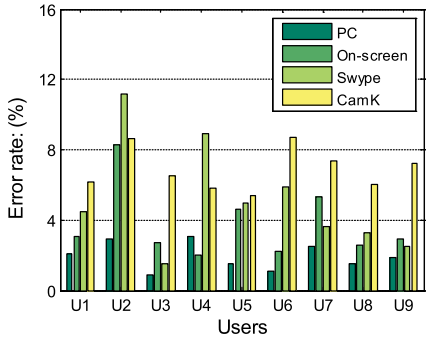



Fig. 32. Error rate with regular text input.

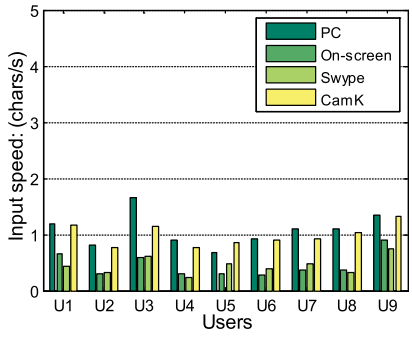



Fig. 33. Input speed with random character input.

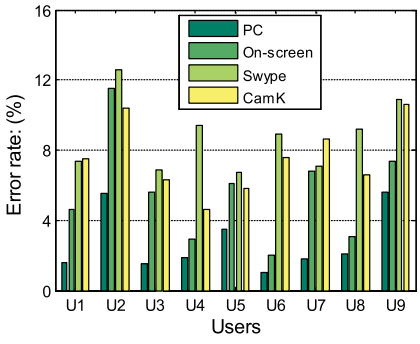
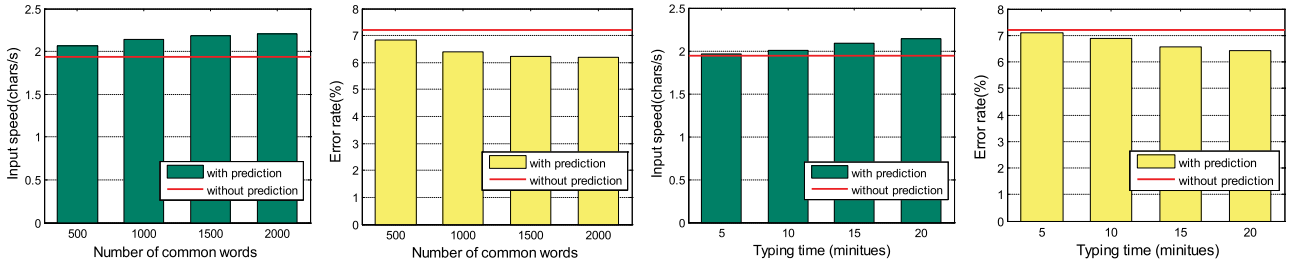



Fig. 34. Error rate with random character input.

different screens to find letters, digits and punctuations. When inputting random characters, UbiK [7] achieves about 2X typing speedup, compared to that of on-screen keyboards. It implies that CamK can improve more input speed compared to UbiK. Fig. 34 shows the error rate of each method. The error rate in CamK ($6\% - 10\%$) is comparable to that of UbiK [7] (about $4\% - 10\%$).

### 8.2 Text Input with Word Prediction

Fig. 35 shows the input speed and error rate while using word prediction, we average the experiment results of 9 users. As shown in Fig. 35a, when the number of common words is 2,000, the input speed is increased by 13.4 percent, when compared with that without word prediction. At the same time, the error rate also decreases, because the user presses less keys for the same amount of words. As shown in Fig. 35b, when the number of common words is 2,000, the error rate is decreased by 14.1 percent, when compared with that without word prediction. In addition, as the typing time increases, the number of candidate words from the typed words will increase, the input speed increases (shown in Fig. 35c), while the error rate decreases (shown in Fig. 35d).




(a) Input speed vs. number of common words (b) Error rate vs. number of common words (c) Input speed vs. typing time (d) Error rate vs. typing time

Fig. 35. Input speed and error rate with word prediction.

In the user study, we compare CamK with other three input methods, i.e., PC keyboard, Android on-screen keyboard, Swype keyboard. When compared with PC keyboards, CamK is a little inferior than the PC keyboard, due to the lack of the tactile feedback of pressing keys. However, unlike the bulky PC keyboard, CamK only uses a portable paper keyboard. When compared with the on-screen keyboard and Swype keyboard, CamK has better performance than these two input methods in input speed, especially when the text contains many digits and punctuations. In regard to UbiK and our CamK, they have comparable performances in input speed and error rate. However, UbiK requires the user to click keys with their fingertips and nails, while CamK provides a similar user experience to using PC keyboards.

## 9 Discussion

Considering the effect of camera settings, environments and diverse user behaviors, CamK still bears some limitations. *Field of view*: We set the aspect ratio of the captured image to $4:3$, which is supported by a lot of phones. In fact, if the camera works with an aspect ratio of $16:9$, it can capture larger images in the same location. However, different aspect ratios also mean different image sizes, which affect the efficiency of image processing. In future, we will implement CamK by capturing images in different aspect ratios and test the performance of CamK. *White balance error*: CamK introduces a light training before typing, to tolerate the effect of environments. In future, we aim to iteratively adjust the extracted hand regions with the learned parameters in previous rounds, to enhance the tolerance of white balance errors. *Typing postures*: When the user puts the hands on the keyboard and presses the key with slight movements of fingertips, CamK may wrongly recognize the *StrokeTip* or miss the keystroke. In future, we try to combine the movements of multiple fingers to infer the keystroke. Besides, we try to make a sound for a keystroke to provide a feedback to the user in text entry.

## 10 Conclusion

In this paper, we propose CamK for inputting text into small mobile devices. By using image processing techniques, CamK can achieve above 95 percent accuracy for keystroke localization, with only 4.8 percent false positives. Based on our experiment results, CamK can achieve 1.25X typing speedup for regular text input and 2.5X for random character input, when compared with on-screen keyboards. Besides, we introduce word prediction to further improve the input speed for regular text by 13.4 percent.

## Acknowledgments

This work is supported by the National Key R&D Program of China under Grant No. 2017YFB1001801, National Natural Science Foundation of China under Grant Nos. 61472185, 61321491, 61502224, and JiangSu Natural Science Foundation under Grant No. BK20151390. This work is partially supported by the Collaborative Innovation Center of Novel Software Technology and Industrialization. Qun Li was supported in part by US National Science Foundation grant CNS-1320453.

## References

[1] M. Fukumoto and Y. Tonomura, "Body coupled FingerRing: Wireless wearable keyboard," in *Proc. ACM SIGCHI Conf. Human Factors Comput. Syst.*, 1997, pp. 147–154.
[2] M. Kölsch and M. Turk, "Keyboards without keyboards: A survey of virtual keyboards," Univ. California, Santa Barbara, CA, USA, UCSB Tech. Rep. 2002-21, Jul. 2002.
[3] K. A. Faraj, M. Mojahid, and N. Vigouroux, "BigKey: A virtual keyboard for mobile devices," *Human-Comput. Interaction*, vol. 5612, pp. 3–10, 2009.
[4] S. Oney, C. Harrison, A. Ogan, and J. Wiese, "ZoomBoard: A diminutive qwerty soft keyboard using iterative zooming for ultra-small devices," in *Proc. ACM SIGCHI Conf. Human Factors Comput. Syst.*, 2013, pp. 2799–2802.
[5] C. Harrison, H. Benko, and A. D. Wilson, "OmniTouch: Wearable multitouch interaction everywhere," in *Proc. ACM Symp. User Interface Softw. Technol.*, 2011, pp. 441–450.
[6] C. Tomasi, A. Rafii, and I. Torunoglu, "Full-size projection keyboard for handheld devices," *Commun. ACM*, vol. 46, no. 7, pp. 70–75, 2003.
[7] J. Wang, K. Zhao, X. Zhang, and C. Peng, "Ubiquitous keyboard for small mobile devices: Harnessing multipath fading for fine-grained keystroke localization," in *Proc. ACM Annu. Int. Conf. Mobile Syst. Appl. Serv.*, 2014, pp. 14–27.
[8] Z. Zhang, Y. Wu, Y. Shan, and S. Shafer, "Visual panel: Virtual mouse, keyboard and 3D controller with an ordinary piece of paper," in *Proc. ACM Workshop Perceptive User Interfaces*, 2001, pp. 1–8.
[9] Y. Adajania, J. Gosalia, A. Kanade, H. Mehta, and N. Shekokar, "Virtual keyboard using shadow analysis," in *Proc. 3rd Int. Conf. Emerging Trends Eng. Technol.*, 2010, pp. 163–165.
[10] Y. Yin, Q. Li, L. Xie, S. Yi, E. Novak, and S. Lu, "CamK: A camera-based keyboard for small mobile devices," in *Proc. IEEE INFOCOM*, 2016, pp. 1–9.
[11] Y. S. Kim, B. S. Soh, and S.-G. Lee, "A new wearable input device: Scurry," *IEEE Trans. Ind. Electron.*, vol. 52, no. 6, pp. 1490–1499, Dec. 2005.
[12] V. R. Pratt, "Thumbcode: A device-independent digital sign language." 1998. [Online]. Available: http://boole.stanford.edu/thumbcode/
[13] S. Nirjon, J. Gummeson, D. Gelb, and K. H. Kim, "TypingRing: A wearable ring platform for text input," in *Proc. ACM Annu. Int. Conf. Mobile Syst. Appl. Serv.*, May 2015, pp. 227–239.
[14] M. Goldstein and D. Chincholle, "The finger-joint gesture wearable keypad," in *Proc. 2nd Workshop Human Comput. Interaction Mobile Devices*, 1999, pp. 9–18.
[15] M. Funk, A. Sahami, N. Henze, and A. Schmidt, "Using a touch-sensitive wristband for text entry on smart watches," in *Proc. Extended Abstracts Human Factors Comput. Syst.*, 2014, pp. 2305–2310.

[16] M. Goel, A. Jansen, T. Mandel, S. N. Patel, and J. O. Wobbrock, "ContextType: Using hand posture information to improve mobile touch screen text entry," in *Proc. ACM SIGCHI Conf. Human Factors Comput. Syst.*, 2013, pp. 2795–2798.
[17] S. Kwon, D. Lee, and M. K. Chung, "Effect of key size and activation area on the performance of a regional error correction method in a touch-screen qwerty keyboard," *Int. J. Ind. Ergonom.*, vol. 39, no. 5, pp. 888–893, Sep. 2009.
[18] S. Zhai, et al., "ShapeWriter on the iPhone: From the laboratory to the real world," in *Proc. ACM Extended Abstracts Human Factors Comput. Syst.*, Apr. 2009, pp. 2667–2670.
[19] O. Schoenleben and A. Oulasvirta, "Sandwich keyboard: Fast ten-finger typing on a mobile device with adaptive touch sensing on the back side," in *Proc. ACM Int. Conf. Human-Comput. Interaction Mobile Devices Serv.*, 2013, pp. 175–178.
[20] J. Hu, G. Li, X. Xie, Z. Lv, and Z. Wang, "Bare-fingers touch detection by the button's distortion in a projector–camera system," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 24, no. 4, pp. 566–575, Apr. 2014.
[21] H. Roeber, J. Bacus, and C. Tomasi, "Typing in thin air: The canesta projection keyboard - a new method of interaction with electronic devices," in *Proc. ACM Extended Abstracts Human Factors Comput. Syst.*, 2003, pp. 712–713.
[22] T. Murase, A. Moteki, N. Ozawa, N. Hara, T. Nakai, and K. Fujimoto, "Gesture keyboard requiring only one camera," in *Proc. ACM Symp. User Interface Softw. Technol.*, 2011, pp. 9–10.
[23] J. Sun, X. Jin, Y. Chen, J. Zhang, Y. Zhang, and R. Zhang, "VISIBLE: Video-assisted keystroke inference from tablet backside motion," in *Proc. Netw. Distrib. Syst. Security Symp.*, NDSS'16, 21–24 Feb. 2016, San Diego, CA, USA, http://dx.doi.org/10.14722/ndss.2016.23060
[24] S. Malik and J. Laszlo, "Visual touchpad: A two-handed gestural input device," in *Proc. 6th Int. Conf. Multimodal Interfaces*, 2004, pp. 289–296.
[25] M. Hagara and J. Pucik, "Fingertip detection for virtual keyboard based on camera," in *Proc. 23rd Int. Conf. Radioelektronika*, 2013, pp. 356–360.
[26] iPhone app: Paper keyboard, 2015. [Online]. Available: http://augmentedappstudio.com/support.html
[27] I. S. MacKenzie and R. W. Soukoreff, "Text entry for mobile computing: Models and methods, theory and practice," *Human–Comput. Interaction*, vol. 17, no. 2/3, pp. 147–198, 2002.
[28] C. Zhang, J. Tabor, J. Zhang, and X. Zhang, "Extending mobile interaction through near-field visible light sensing," in *Proc. 21st Annu. Int. Conf. Mobile Comput. Netw.*, 2015, pp. 345–357.
[29] R. Biswas and J. Sil, "An improved canny edge detection algorithm based on type-2 fuzzy sets," *Procedia Technol.*, vol. 4, pp. 820–824, 2012.
[30] S. A. Naji, R. Zainuddin, and H. A. Jalab, "Skin segmentation based on multi pixel color clustering models," *Digit. Signal Process.*, vol. 22, no. 6, pp. 933–940, 2012.
[31] N. Otsu, "A threshold selection method from gray-level histograms," *IEEE Trans. Syst. Man Cybern.*, vol. 9, no. 1, pp. 62–66, Jan. 1979.
[32] R. M. Haralick, S. R. Sternberg, and X. Zhuang, "Image analysis using mathematical morphology," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. PAMI-9, no. 4, pp. 532–550, Jul. 1987.
[33] Camera parameters, 2016. [Online]. Available: https://developer.android.com/reference/android/hardware/Camera.Parameters.html
[34] Monsoon power monitor, 2017. [Online]. Available: http://www.msoon.com/
[35] I. S. MacKenzie and X. Zhang, "Eye typing using word and letter prediction and a fixation algorithm," in *Proc. Symp. Eye Tracking Res. Appl.*, 2008, pp. 55–58.
[36] Word Frequency Data: Corpus of contemporary American English, 2017. [Online]. Available: http://www.wordfrequency.info/free.asp
[37] I. S. MacKenzie and R. W. Soukoreff, "Phrase sets for evaluating text entry techniques," in *Proc. Extended Abstracts Human Factors Comput. Syst.*, 2003, pp. 754–755.
[38] Apple wireless keyboard, 2017. [Online]. Available: http://www.apple.com/us/search/magic-keyboard-us-english?src=serp
[39] Keyboard layout, 2017. [Online]. Available: https://en.wikipedia.org/wiki/Keyboard_layout
[40] Swype, 2015. [Online]. Available: http://www.swype.com/

**Yafeng Yin** received the PhD degree in computer science from Nanjing University, China, in 2017. She is currently an assistant professor in the Department of Computer Science and Technology, Nanjing University. Her research interests include human activity recognition, mobile sensing, wearable computing, etc. She is a member of the IEEE.

**Qun Li** received the PhD degree in computer science from Dartmouth College. He is a professor in the Department of Computer Science, College of William and Mary. His research interests include wireless networks, sensor networks, RFID, and pervasive computing systems. He received a US NSF Career Award in 2008. He is a fellow of the IEEE.

**Lei Xie** received the BS and PhD degrees from Nanjing University, China, in 2004 and 2010, respectively, all in computer science. He is currently an associate professor in the Department of Computer Science and Technology, Nanjing University. He has published more than 50 papers in the *IEEE Transactions on Mobile Computing*, the *IEEE/ACM Transactions on Networking*, the *IEEE Transactions on Parallel and Distributed Systems*, the *ACM Transactions on Sensor Networks*, ACM UbiComp, ACM MobiHoc, IEEE INFOCOM, IEEE ICNP, IEEE ICDCS, etc. He is a member of the IEEE.

**Shanhe Yi** is working toward the PhD degree in computer science under the supervision of Prof. Qun Li at the College of William and Mary. His research interests include mobile/wearable computing and edge computing, with the emphasis on the usability, security and privacy of applications and systems.

**Ed Novak** is an assistant professor of computer science at Franklin and Marshall College. His research focus is in digital privacy and security on mobile devices. Recently, he has been interested in security and privacy in the new Internet of Things (IoT) paradigm.

**Sanglu Lu** received the BS, MS, and PhD degrees from Nanjing University, China, in 1992, 1995, and 1997, respectively, all in computer science. She is currently a professor in the Department of Computer Science and Technology, Nanjing University. Her research interests include distributed computing and pervasive computing. She is a member of the IEEE.