CamK: a Camera-based Keyboard for Small Mobile Devices

Yafeng Yin†, Qun Li‡, Lei Xie†, Shanhe Yi‡, Ed Novak‡, Sanglu Lu†
†State Key Laboratory for Novel Software Technology, Nanjing University, China
‡College of William and Mary, Williamsburg, VA, USA
Email: †yyf@dislab.nju.edu.cn, †{lxie, sanglu}@nju.edu.cn ‡liqun@cs.wm.edu, ‡{syi, ejnovak}@cs.wm.edu

***Abstract*—Due to the smaller size of mobile devices, on-screen keyboards become inefficient for text entry. In this paper, we present CamK, a camera-based text-entry method, which uses an arbitrary panel (e.g., a piece of paper) with a keyboard layout to input text into small devices. CamK captures the images during the typing process and uses the image processing technique to recognize the typing behavior. The principle of CamK is to extract the keys, track the user's fingertips, detect and localize the keystroke. To achieve high accuracy of keystroke localization and low false positive rate of keystroke detection, CamK introduces the initial training and online calibration. Additionally, CamK optimizes computation-intensive modules to reduce the time latency. We implement CamK on a mobile device running Android. Our experimental results show that CamK can achieve above 95% accuracy of keystroke localization, with only 4.8% false positive keystrokes. When compared to on-screen keyboards, CamK can achieve 1.25X typing speedup for regular text input and 2.5X for random character input.**

I. Introduction

Recently, mobile devices have converged to a relatively small form factor (e.g., smartphones, Apple Watch), in order to be carried everywhere easily, while avoiding carrying bulky laptops all the time. Consequently, interacting with small mobile devices involves many challenges, a typical example is text input without a physical keyboard.

Currently, many visual keyboards are proposed. However, wearable keyboards [1], [2] introduce additional equipments. On-screen keyboards [3], [4] usually take up a large area on the screen and only support single finger for text entry. Projection keyboards [5]–[9] often need an infrared or visible light projector to display the keyboard to the user. Audio signal [10] or camera based visual keyboards [11]–[13] remove the additional hardware. By leveraging the microphone to localize the keystrokes, UbiK [10] requires the user to click keys with their fingertips and nails to make an audible sound, which is not typical of typing. For existing camera based keyboards, they either slow the typing speed [12], or should be used in controlled environments [13]. They can not provide a similar user experience to using physical keyboards [11].

In this paper, we propose CamK, a more natural and intuitive text-entry method, in order to provide a PC-like text-entry experience. CamK works with the front-facing camera of the mobile device and a paper keyboard, as shown in Fig. 1. CamK takes pictures as the user types on the paper keyboard, and uses image processing techniques to detect and localize keystrokes. CamK can be used in a wide variety of scenarios, e.g., the office, coffee shops, outdoors, etc.

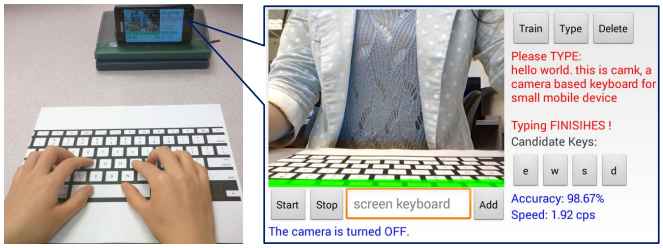


Fig. 1. A typical use case of CamK.

There are three key technical challenges in CamK. **(1) High accuracy of keystroke localization:** The inter-key distance in the paper keyboard is only about two centimeters [10]. While using image processing techniques, there may exist a position deviation between the real fingertip and the detected fingertip. To address this challenge, CamK introduces the initial training to get the optimal parameters for image processing. Besides, CamK uses an extended region to represent the detected fingertip, aiming to tolerate the position deviation. In addition, CamK utilizes the features (e.g., visually obstructed area of the pressed key) of a keystroke to verify the validity of a keystroke. **(2) Low false positive rate of keystroke detection:** A false positive occurs when a non-keystroke (i.e., a period in which no fingertip is pressing any key) is treated as a keystroke. To address this challenge, CamK combines keystroke detection with keystroke localization. If there is not a valid key pressed by the fingertip, CamK will remove the possible non-keystroke. Besides, CamK introduces online calibration to further remove the false positive keystrokes. **(3) Low latency:** When the user presses a key on the paper keyboard, CamK should output the character of the key without any noticeable latency. Usually, the computation in image processing is heavy, leading to large time latency in keystroke localization. To address this challenge, CamK changes the sizes of images, optimizes the image processing process, adopts multiple threads, and removes the operations of writing/reading images, in order to make CamK work on the mobile device.

We make the following contributions in this paper.

- We propose a novel method CamK for text-entry. CamK only uses the camera of the mobile device and a paper

keyboard. CamK allows the user to type with all the fingers and provides a similar user experience to using physical keyboards.

- We design a practical framework for CamK, which can detect and localize the keystroke with high accuracy, and output the character of the pressed key without any noticeable time latency. Based on image processing, CamK can extract the keys, track the user's fingertips, detect and localize keystrokes. Besides, CamK introduces the initial training to optimize the image processing result and utilizes online calibration to reduce the false positive keystrokes. Additionally, CamK optimizes the computation-intensive modules to reduce the time latency, in order to make CamK work on the mobile devices.
- We implement CamK on a smartphone running Google's Android operating system (version 4.4.4). We first measure the performance of each module in CamK. Then, we invite nine users[1] to evaluate CamK in a variety of real-world environments. We compare the performance of CamK with other methods, in terms of keystroke localization accuracy and text-entry speed.

II. Observations of A Keystroke

In order to show the feasibility of localizing the keystroke based on image processing techniques, we first show the observations of a keystroke. Fig. 2 shows the frames/images captured by the camera during two consecutive keystrokes. The origin of coordinates is located in the top left corner of the image, as shown in Fig. 2(a). We call the hand located in the left area of the image the *left hand*, while the other is called the *right hand*, as shown in Fig. 2(b). From left to right, the fingers are called finger i in sequence, $i \in [1, 10]$, as shown in Fig. 2(c). The fingertip pressing the key is called *StrokeTip*. The key pressed by *StrokeTip* is called *StrokeKey*.

- The *StrokeTip has the largest vertical coordinate* among the fingers on the same hand. An example is finger 9 in Fig. 2(a). However this feature may not work well for thumbs, which should be identified separately.
- The *StrokeTip stays on the StrokeKey for a certain duration*, as shown in Fig. 2(c) - Fig. 2(d). If the positions of the fingertip keep unchanged, a keystroke may happen.
- The *StrokeTip is located in the StrokeKey,* as shown in Fig. 2(a), Fig. 2(d).
- The *StrokeTip obstructs the StrokeKey* from the view of the camera, as shown in Fig. 2(d). The ratio of the visually obstructed area to the whole area of the key can be used to verify whether the key is pressed.
- The *StrokeTip has the largest vertical distance between the remaining fingertips of the corresponding hand.* As shown in Fig. 2(a), 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. Considering the difference caused by the distance between the camera and the fingertip, sometimes this feature may not be satisfied. Thus this feature is used to assist in keystroke localization, instead of directly determining a keystroke.

[1]All data collection in this paper has gone through the IRB approval

III. System Design

As shown in Fig. 1, CamK works with a mobile device (e.g., a smartphone) with the embedded camera, a paper keyboard. The smartphone uses the front-facing camera to watch the typing process. The paper keyboard is placed on a flat surface. The objective is to let the keyboard layout be located in the camera's view, while making the keys in the camera's view look as large as possible. CamK does not require the keyboard layout is fully located in the camera's view, because some user may only want to input letters or digits. Even if the user only place the concerned part of keyboard in the camera's view, CamK can still work. CamK consists of the following four components: key extraction, fingertip detection, keystroke detection and localization, and text-entry determination.

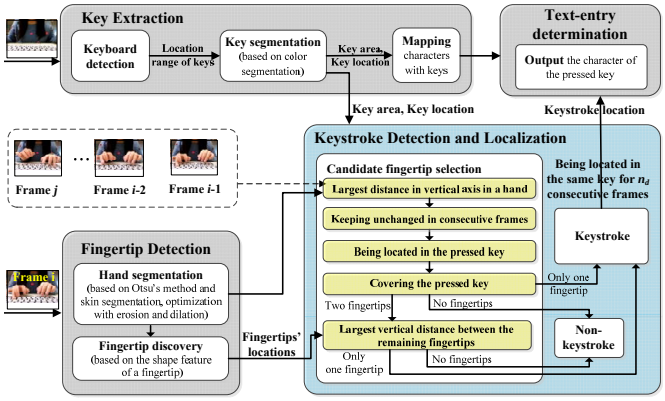


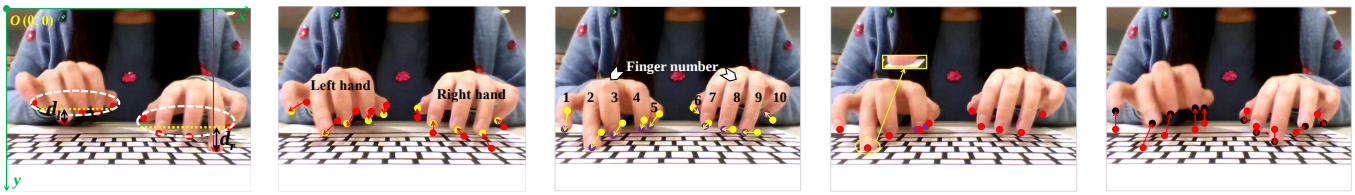
Fig. 3. Architecture of CamK.

A. System Overview

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 the fingertip based on the shape of a finger, in order to track the fingertips. Based on the movements of fingertips, CamK uses **Keystroke Detection and Localization** to detect a possible keystroke and localize the keystroke. Finally, CamK uses **Text-entry Determination** to output the character of the pressed key.

B. Key Extraction

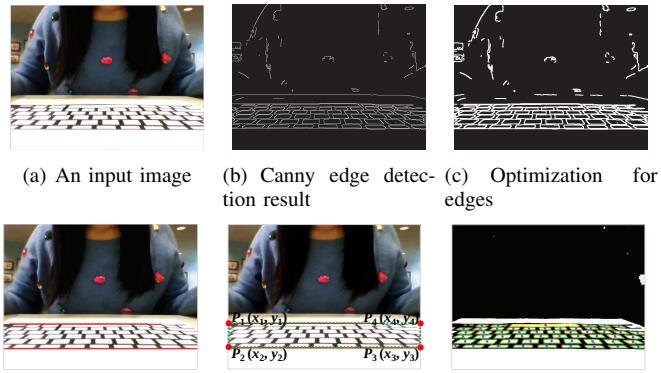
Without loss of generality, CamK adopts the common QWERTY keyboard layout, which is printed in black and white on a piece of paper, as shown in Fig. 1. In order to eliminate background effects, we first detect the boundary of the keyboard. Then, we extract each key from the keyboard. Therefore, key extraction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



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

Fig. 2. Frames during two consecutive keystrokes

1) Keyboard detection: We use Canny edge detection algorithm [14] to obtain the edges of the keyboard. Fig. 4(b) shows the edge detection result of Fig. 4(a). However, the interference edges (e.g., the paper's edge / longest edge in Fig. 4(b)) should be removed. Based on Fig. 4(b), the edges of the keyboard should be close to the edges of the keys. We use this feature to remove pitfall edges, the result is shown in Fig. 4(c). Additionally, we adopt the dilation operation [15] to join the dispersed edge points which are close to each other, in order to get better edges/boundaries of the keyboard. After that, we use the Hough transform [12] to detect the lines in Fig. 4(c). Then, we use the uppermost line and the bottom line to describe the position range of the keyboard, as shown in Fig. 4(d). Similarly, we can use the Hough transform [12] 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. 4(e), we extend the four edges (lines) to get four intersections $P_1(x_1, y_1), P_2(x_2, y_2), P_3(x_3, y_3), P_4(x_4, y_4)$, which are used to describe the boundary of the keyboard.



(a) An input image (b) Canny edge detection result (c) Optimization for edges

(d) Position range of keyboard (e) Keyboard boundary (f) Key segmentation result

Fig. 4. Keyboard detection and key extraction

2) Key segmentation: With the known location of the keyboard, we can extract the keys based on color segmentation. 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 can only use the difference of 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. ε_y can be estimated in the initial training (see section IV-A). The initial/default value of ε_y is $\varepsilon_y = 50$.

When we obtain the white pixels, we need to get the contours of the keys and separate the keys from one another. While considering the pitfall areas such as small white areas which do not belong to any key, we estimate the area of a key at first. Based on Fig. 4(e), we use P_1, P_2, P_3, P_4 to calculate the area S_b of the keyboard as $S_b = \frac{1}{2} \cdot (|\overrightarrow{P_1P_2} \times \overrightarrow{P_1P_4}| + |\overrightarrow{P_3P_4} \times \overrightarrow{P_3P_2}|)$. 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, different distances between the camera and the keys can also affect the area of a key in the image. Therefore, we introduce α_l, α_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. 4(e) is shown in Fig. 4(f). Then, we use the location of the space key (biggest key) to locate other keys, based on the relative locations between keys.

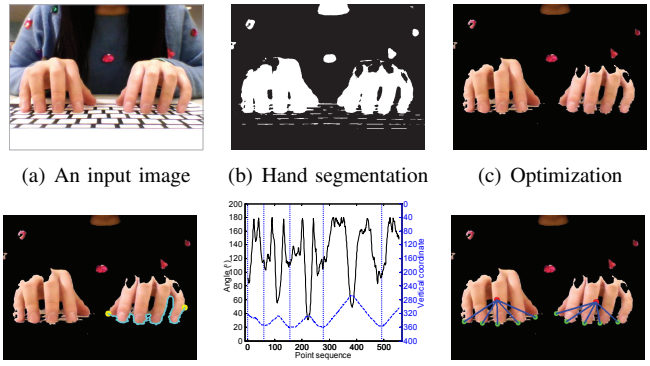
C. Fingertip Detection

In order to detect keystrokes, CamK needs to detect the fingertips and track the movements of fingertips. Fingertip detection consists of hand segmentation and fingertip discovery.

1) Hand segmentation: Skin segmentation [15] is a common method used for hand detection. In 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 [16] and the red channel in YCrCb color space for skin segmentation.

In YCrCb color space, the red channel Cr is essential to human skin coloration. Therefore, for a captured image, we use the grayscale image that is split based on Cr channel as an input for Otsu's method. Otsu's method [16] can automatically perform clustering-based image thresholding, i.e., it can calculate the optimal threshold to separate the foreground and background. Therefore, this skin segmentation approach can tolerate the effect caused by environments such as lighting conditions. For the input image Fig. 5(a), the hand segmentation result is shown in Fig. 5(b), where the white regions represent the hand regions, 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 fingertips. Thus, CamK introduces the erosion and dilation operations [17]. 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. 5(c) shows the optimized result of hand segmentation. Intuitively, if the color of the user's clothes is close to his/her skin color, the hand segmentation result will become worse. At this time, we only focus on the hand region located in the keyboard area. Due to the color difference between the keyboard and human skin, CamK can still extract the hands efficiently.



(a) An input image (b) Hand segmentation (c) Optimization

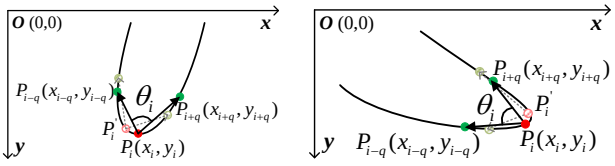
(d) Fingers'contour (e) Fingertip discovery (f) Fingertips

Fig. 5. Fingertip detection

2) Fingertip discovery: After we extract the fingers, we need to detect the fingertips. As shown in Fig. 6(a), the fingertip is usually a convex vertex of the finger. 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 θ_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 θ_i, we map θ_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. 6(a), we choose the vertex which has the largest vertical coordinate, as P_i shown in Fig. 6(a). Because this point has the largest probability to be a fingertip. Based on extensive experiments, we set $\theta_l = 60°$, $\theta_h = 150°$, $q = 20$ in this paper.

$$\theta_i = \arccos \frac{\overrightarrow{P_iP_{i-q}} \cdot \overrightarrow{P_iP_{i+q}}}{|\overrightarrow{P_iP_{i-q}}| \cdot |\overrightarrow{P_iP_{i+q}}|} \tag{1}$$

Considering the specificity of thumbs, which may press the key (e.g., space key) in a different way from other fingers, the relative positions of P_{i-q}, P_i, P_{i+q} may change. Fig. 6(b) shows the thumb in 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 which has the largest vertical coordinate as the fingertip.



(a) Fingertips (excluding thumbs) (b) A thumb

Fig. 6. Features of a fingertip

In fingertip detection, we only need to detect the points located in 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. 5(d). The shape feature θ_i and the positions in vertical coordinates y_i along the bottom edge are shown Fig. 5(e). If we can detect five fingertips in a hand with θ_i and y_{i-q}, y_i, y_{i+q}, we will not detect the thumb specially. 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 θ_i and x_{i-q}, x_i, x_{i+q}. The detected fingertips of Fig. 5(a) are marked in Fig. 5(f).

D. Keystroke Detection and Localization

When CamK detects the fingertips, it will track the fingertips to detect a possible keystroke and localize it. The keystroke localization result can be used to remove false positive keystrokes. We illustrate the whole process of keystroke detection and localization together.

1) Candidate fingertip in each hand: CamK allows the user to use all the fingers for text-entry, thus the keystroke may be caused by the left or right hand. According to the observations (see section II), the fingertip (i.e., *StrokeTip*) pressing the key usually has the largest vertical coordinate in that hand. 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 all points in C_l, if a point $P_l(x_l, y_l)$ satisfies $y_l \geq y_j, l \neq j, P_j, P_l \in C_l$, 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 to know the moving states of hands. It is unnecessary to detect other fingertips.

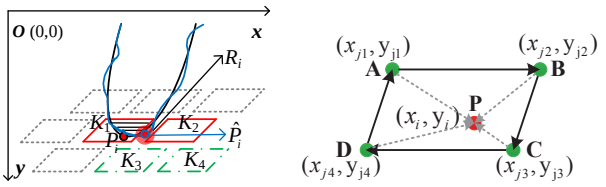
2) Moving or staying: As described in the observations, when the user presses a key, the fingertip will stay at that key for a certain duration. Therefore, we can use the location variation of the candidate fingertip to detect a possible keystroke. In Frame i, we use $P_{l_i}(x_{l_i}, y_{l_i})$ and $P_{r_i}(x_{r_i}, y_{r_i})$ to represent the candidate fingertips in the left hand and right hand, respectively. Based on Fig. 5, the interference regions around a fingertip may affect the contour of the fingertip, there may exist a position deviation between the real fingertip and the detected fingertip. Therefore, if the candidate fingertips in frame $i-1$, i satisfy Eq. (2), the fingertips can be treated as static, i.e., a keystroke probably happens. Based on extensive experiments, we set $\Delta r = 5$ empirically.

$$\begin{aligned} \sqrt{(x_{l_i} - x_{l_{i-1}})^2 + (y_{l_i} - y_{l_{i-1}})^2} \leq \Delta r, \\ \sqrt{(x_{r_i} - x_{r_{i-1}})^2 + (y_{r_i} - y_{r_{i-1}})^2} \leq \Delta r. \end{aligned} \tag{2}$$

3) Discovering the pressed key: For a keystroke, the fingertip is located at the key and a part of the key will be visually

obstructed by that fingertip, as shown in Fig. 2(d). 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 localize the keystroke by using Alg. 1.

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) Candidate keys (b) Locating a fingertip

Fig. 7. Candidate keys and Candidate fingertips

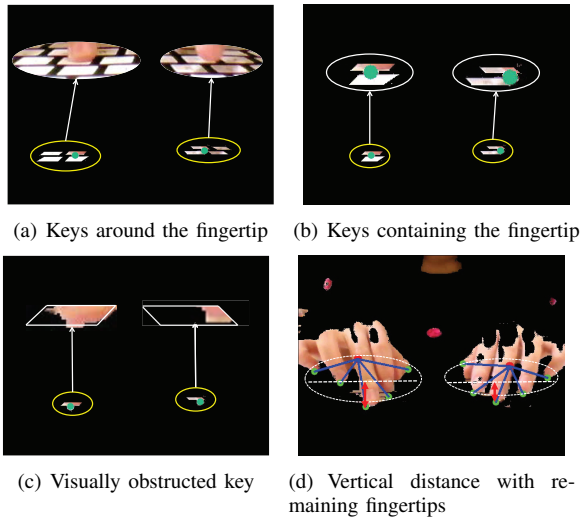
Selecting the nearest candidate keys: For each candidate fingertip P_i, we first search the candidate keys which are probably pressed by P_i. As shown in Fig. 7(a), although the real fingertip is P_i, the detected fingertip is $\hat{P}_i$. We use $\hat{P}_i$ to search the candidate keys. We use $K_{cj}(x_{cj},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

Keeping 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, in order to remove the invalid candidate keys. As shown in Fig. 7(a), 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. 7(a). 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. 7(b), a key is represented as a quadrangle $ABCD$. If a point is located in $ABCD$, when we move around $ABCD$ clockwise, the point will be located in the right side of each edge in $ABCD$. As shown in Fig. 2(a), the origin of coordinates is located in the top left corner of the image. Therefore, if the fingertip $P \in R_i$ satisfy Eq. (3), 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. 7(a), K_1, K_2 are the remaining candidate keys. The candidate keys contain the fingertip in Fig. 8(a) is shown in Fig. 8(b).

$$\begin{aligned} \overrightarrow{AB} \times \overrightarrow{AP} \geq 0, \overrightarrow{BC} \times \overrightarrow{BP} \geq 0, \\ \overrightarrow{CD} \times \overrightarrow{CP} \geq 0, \overrightarrow{DA} \times \overrightarrow{DP} \geq 0. \end{aligned} \tag{3}$$

Calculating the coverage ratios of candidate keys: For the pressed key, it is visually obstructed by the fingertip, as the dashed area of key K_1 shown in Fig. 7(a). We use the coverage ratio to measure the visually obstructed area of a candidate key, in order to remove the wrong candidate keys. For a candidate key K_j, whose area is S_{kj}, the visually obstructed area is D_{kj}, then 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 the ρ_{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 $f_j = S_{K_j}/S_k$. Here, S_k means the average area of a key, as described in section III-B2. 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 $\rho_l = 0.25$ in this paper. 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 candidate fingertip will be eliminated. Fig. 8(c) shows each candidate fingertip and its associated key.



(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

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 we infer that no keystroke happens. If there is only one candidate fingertip, then the fingertip is the *StrokeTip*, and the associated candidate key is *StrokeKey*. 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. 2(a).

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 *left hand*, and the distance d_r between P_r and the remaining fingertips in *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. 8(d), we choose fingertip 3 in the left hand as the *StrokeTip*. However, based on the observations, the

distance between the camera and hands may affect the value of d_l (d_r). Therefore, for the unselected candidate fingertip (e.g., fingertip 8 in Fig. 8(d)), we do not discard it. We display its associated key as the candidate key. The user can select the candidate key for text input (see Fig. 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} with four nearest keys around P_i.
  for $K_j \in C_{key}$ **do**
    if P_i *is located in* K_j **then**
      Calculate the coverage ratio ρ_{k_j} of K_j.
      if $\rho_{k_j} < \rho_l$ **then**
        Remove K_j from C_{key}.
  if $C_{key} \neq \emptyset$ **then**
    Select K_j with largest ρ_{k_j} from C_{key}.
    P_i and K_j form a combination $< P_i, K_j >$.
  else Remove P_i from C_{tip} ;
if $C_{tip} = \emptyset$ **then** No keystroke occurs, return ;
if $|C_{tip}| = 1$ **then**
  Return the associated key of the only fingertip.
For each hand, select $< P_i, K_j >$ with largest ratio ρ_{k_j}.
Use $< P_l, K_l >$ ($< P_r, K_r >$) to represent the fingertip and its associated key in left (right) hand.
Calculate d_l (d_r) between P_l (P_r) with the remaining fingertips in left (right) hand.
if $d_l > d_r$ **then** Return K_l ;
else Return K_r;
Output: The pressed key.

IV. Optimizations for keystroke localization and image processing

A. Initial Training

Optimal parameters for image processing: For key segmentation (see section III-B2), ε_y is used for tolerating the change of Y caused by environments. 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 ε_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 ε_{y_i} is selected as the optimal value for ε_y.

In hand segmentation, CamK uses erosion and dilation operations, which respectively use a kernel B [17] to process images. In order to get the suitable size of B, the user first puts his/her hands on the home row of the keyboard, as shown in Fig. 5(a). 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 $z = z_i$.

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 frame rate of many smartphones. For the ith keystroke, the number of frames containing the keystroke is represented as n_{0_i}. 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 V-C), we set $\bar{n_f} = 3$, thus $f = \left\lceil f_0 \cdot \frac{\bar{n_f}}{\bar{n_0}} \right\rceil$.

B. Online Calibration

Removing false positive keystrokes: Sometimes, the fingers may keep still, even the user does not type any key. CamK may treat the non-keystroke as a keystroke by chance, leading to an error. 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 the $\bar{n_f}$ consecutive frames, it will display 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 very short time during 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 are kept at stable positions during its usage life-cycle. For best results, we recommend the user tape the paper keyboard on the panel. However, to alleviate the effect caused by the movements of the mobile device or the keyboard, we offer a simple solution. If the user uses the *Delete* key on the screen multiple times (e.g., larger than 3 times), it may indicate CamK can not output the character correctly. The movements of the device/keyboard may happen. Then, CamK informs the user to move his/her hands away from the keyboard for relocation. After that, the user can continue the typing process.

C. Real Time Image Processing

Because image processing is rather time-consuming, it is difficult to make CamK work on the mobile device. Take the Samsung GT-I9100 smartphone as an example, when the image size is $640 * 480$ pixels, it needs 630ms to process this image to localize the keystroke. When considering the time cost for taking images, processing consecutive images to track fingertips for keystroke detection, the time cost for localizing a keystroke will increase to 1320ms, which will lead to a very low input speed and a bad user experience. Therefore, we introduce the following optimizations for CamK.

Adaptively changing image sizes: We use small images (e.g., $120 * 90$ pixels) during two keystrokes to track the fingertips, and use a large image (e.g., $480 * 360$ pixels) for keystroke localization. **Optimizing the large-size image processing:** When we detect a possible keystroke in (x_c, y_c) of frame $i-1$, then we focus on a small area $S_c = \{P_i(x_i, y_i) \in S_c | \ |x_i - x_c| \leq \Delta x, |y_i - y_c| \leq \Delta y\}$ of frame i to localize the keystroke. We set $\Delta x = 40$, $\Delta y = 20$ by default. **Multi-thread Processing:** CamK adopts three threads to detect and localize the keystroke in parallel, i.e., capturing thread to take images, tracking thread for keystroke detection, and localizing thread for keystroke localization. **Processing without writing and reading images:** CamK directly stores the bytes of the source data to the text file in binary mode, instead of writing/reading images.

V. Performance Evaluation

We implement CamK on the Samsung GT-I9100 smartphone running Google's Android operating system (version 4.4.4). Samsung GT-I9100 has a 2 million pixels front-facing camera. We use the layout of AWK (Apple Wireless Keyboard) as the default keyboard layout, which is printed on a piece of US Letter sized paper. Unless otherwise specified, the frame rate is 15fps, the image size is $480*460$ pixels. CamK works in the office. We first evaluate each component of CamK. Then, we invite 9 users to use CamK and compare the performance of CamK with other text-entry methods.

A. Localization accuracy for known keystrokes

In order to verify whether CamK has obtained the optimal parameters for image processing, we measure the accuracy of keystroke localization, when CamK knows a keystroke is happening. The user presses the 59 keys (excluding the PC function keys: first row, five keys in last row) on the paper keyboard sequentially. We press each key fifty times. The localization result is shown in Fig. 9. the localization accuracy is close to 100%. It means that CamK can adaptively select suitable values of the parameters used in image processing.

B. Accuracy of keystroke localization and false positive rate of keystroke detection

In order to verify whether CamK can utilize the features of a keystroke and online calibration for keystroke detection and localization. We conduct the experiments in three typical scenarios; an office environment, a coffee shop, and outdoors. Usually, in the office, the color of the light is close to white. In the coffee shop, the red part of light is similar to that of human skin. In outdoors, the sunlight is basic/pure light. In each test, a user randomly makes $N_k = 500$ keystrokes. Suppose CamK localizes N_a keystrokes correctly and treats N_f non-keystrokes as keystrokes wrongly. We define the accuracy as $p_a = \frac{N_a}{N_k}$, and the false positive rate as $p_f = \min(\frac{N_f}{N_k}, 1)$. We show the results of these experiments in Fig. 10, which shows that CamK can achieve high accuracy (larger than 90%) with low false positive rate (about 5%). In the office, the localization accuracy can achieve 95%.

C. Frame rate

As described in section IV-A, the frame rate affects the number of images $\bar{n_f}$ during a keystroke. Obviously, with the larger value of $\bar{n_f}$, CamK can easily detect the keystroke and localize it. On the contrary, CamK may miss the keystrokes. Based on Fig. 11, when $\bar{n_f} \geq 3$, CamK has good performance. When $\bar{n_f} > 3$, there is no obvious performance improvement. However, increasing $\bar{n_f}$ means introducing more images for processing. It may increase the time latency. While considering the accuracy, false positive, and time latency, we set $\bar{n_f} = 3$.

Besides, we invite 5 users to test the duration Δt of a keystroke. Δt represents the time when the *StrokeTip* is located in the *StrokeKey* from the view of the camera. Based on Fig. 12, Δt is usually larger than 150ms. When $\bar{n_f} = 3$, the frame rate is less than the maximum frame rate (30fps). CamK can work under the frame rate limitation of the smartphone. Therefore, $\bar{n_f} = 3$ is a suitable choice.

D. Impact of image size

We first measure the performance of CamK by adopting a same size for each image. Based on Fig. 13, 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 cause obvious improvement. However, increasing the image size will increase the image processing time and power consumption (measured by a Monsoon power monitor [18]) for processing an image, as shown in Fig. 14. Based on section IV-C, CamK adaptively change the sizes of the images. In order 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 $480 * 380$ pixels.

In Fig. 15, the size of the small image decreases from $480 * 360$ to $120 * 90$, CamK keeps the high accuracy with low false rate. When the size of small images continuously changes, the accuracy decreases a lot, and the false positive rate increases a lot. When the image size decreases, the time cost / power consumption for locating a keystroke keeps decreasing, as shown in Fig. 16. Combining Fig. 15 and Fig. 16, the size of the small image is set $120 * 90$ pixels.

E. Time latency and power consumption

Based on Fig. 16, the time cost for locating a keystroke is about 200ms, which is comparable to the duration of a keystroke, as shown in Fig. 12. It means when the user stays in the pressed key, CamK can output the text without noticeable time latency. The time latency is within 50ms, or even smaller, which is well below human response time [10]. In addition, we measure the power consumption of Samsung GT-I9100 smartphone in the 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 516mW, 1189mW, 1872mW, 2245mW. The power consumption of CamK is a little high. Yet as a new

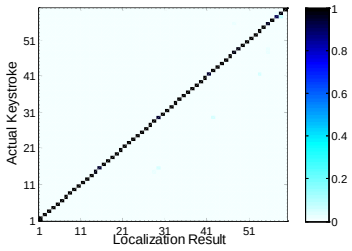


Fig. 9. Confusion matrix of the 59 keys

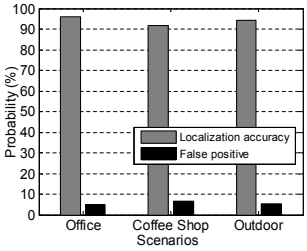


Fig. 10. Three scenarios

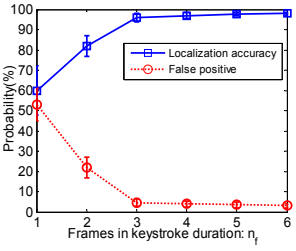


Fig. 11. Accuracy/false positive vs. frames in a keystroke

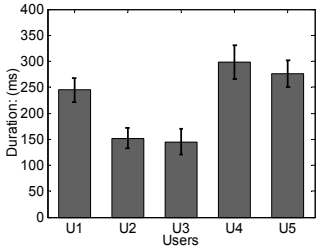


Fig. 12. Duration for a keystroke

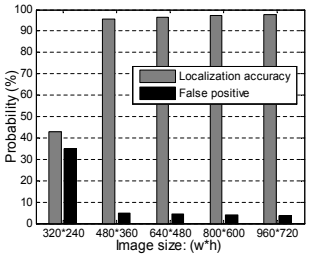


Fig. 13. Accuracy/fase positive vs. image sizes

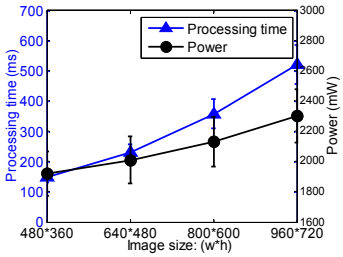


Fig. 14. Processing time/power vs. image sizes

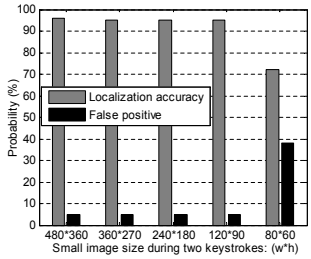


Fig. 15. Accuracy/fase positive by changing sizes of small images

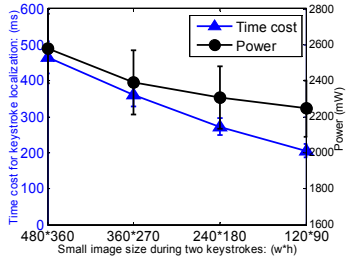


Fig. 16. Processing time/power by changing sizes of small images

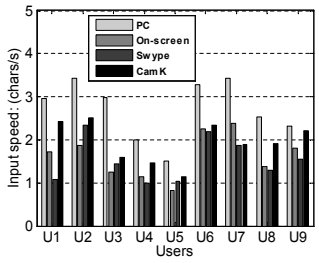


Fig. 17. Input speed with regular text input

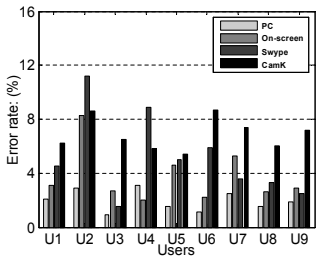


Fig. 18. Error rate with regular text input

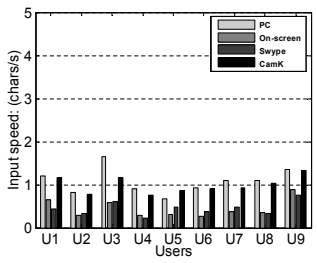


Fig. 19. Input speed with random character input

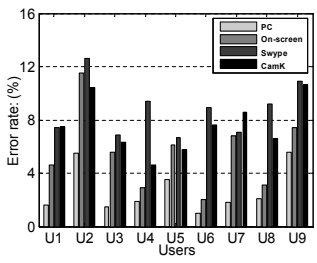


Fig. 20. Error rate with random character input

technique, the power consumption is acceptable. In future, we will try to reduce the energy cost.

F. User study

In order to evaluate the usability of CamK in practice, we invite 9 users to test CamK in different environments. We use the input speed and the error rate $p_e = (1 - p_a) + p_f$ as metrics. Each user tests CamK by typing regular text sentences and random characters. We compare CamK with the following three input methods: typing with an IBM style PC keyboard, typing on Google's Android on-screen keyboard, and typing on Swype keyboard [19], which allows the user to slide a finger across the keys and use the language mode to guess the word. For each input method, the user has ten minutes to familiarize with the keyboard before using it.

1) Regular text input: Fig. 17 shows the input speed of each user when they input the regular text. Each user achieves the highest input speed when he/she uses the PC keyboard. This is because the user can locate the keys on a physical keyboard by touch, while the user tends to look at the paper keyboard to find a key. 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 [10], which requires the user to type with the finger nail (which is not typical), CamK improves the input speed about 20%. Fig. 18 shows the error rate of each method. Although CamK is relatively more erroneous than other methods, as a new technique, the error rate is comparable and tolerable. Usually, the error rate of CamK is between $5\% - 9\%$, which is comparable to that of UbiK (about $4\% - 8\%$).

2) Random character input: Fig. 19 shows the input speed of each user when they input the 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 to the on-screen keyboard and Swype. Because the latter two keyboards need to switch between different screens to find letters, digits and punctuations. For random character input, UbiK [10] achieves 2X typing speedup, compared to that of on-screen keyboards. Therefore, our solution can improve more input speed, when compared to UbiK. Fig. 20 shows the error rate of each method. Due to the randomness of the characters, the error rate increases, especially for typing with the on-screen keyboard and Swype. The error rate of CamK does not increase much, because the user can input the characters just like he/she uses the PC keyboard. The error rate in CamK ($6\% - 10\%$) is comparable to that of UbiK [10] (about $4\% - 10\%$).

VI. Related Work

Due to small sizes of mobile devices, existing research work has focused on redesigning visual keyboards for text entry, such as wearable keyboards, modified on-screen keyboards, projection keyboards, camera based keyboard, and so on.

Wearable keyboards: Among the wearable keyboards, FingerRing [1] puts a ring on each finger to detect the finger's movement to produce a character based on the accelerometer. Similarly, Samsung's Scurry [20] works with the tiny gyroscopes. Thumbcode method [21], finger-Joint keypad [22] work with a glove equipped with the pressure sensors for each finger. The Senseboard [2] consists of two rubber pads which slip onto the user's hands. It senses the movements in the palm to get keystrokes.

Modified on-screen keyboards: Among the modified on-screen keyboards, BigKey [3] and ZoomBoard [4] adaptively change the size of keys. ContextType [23] leverages the information about a user's hand posture to improve mobile touch screen text entry. While considering using multiple fingers, Sandwich keyboard [24] affords ten-finger touch typing by utilizing a touch sensor on the back side of a device.

Projection keyboards: Considering the advantages of the current QWERTY keyboard layout, projection keyboards are proposed. However, they either need a visible light projector to cast a keyboard [5], [6], [7], or use the infrared projector to produce a keyboard [8] [9]. They use optical ranging or image recognition methods to identify the keystroke.

Camera based keyboards: Camera based visual keyboards do not need additional hardware. In [11], the system gets the input by recognizing the gestures of user's fingers. It needs users to remember the mapping between the keys and the fingers. In [12], the visual keyboard is printed on a piece of paper. The user can only use one finger and needs to wait for one second before each keystroke. Similarly, the iPhone app paper keyboard [25] only allows the user to use one finger in a hand. In [13], the system detects the keystroke based on shadow analysis, which is easy affected by light conditions.

In addition, Wang et al. [10] propose UbiK, which leverages the microphone on a mobile device to localize the keystrokes. However, it requires the user to click the key with fingertip and nail margin, which is not typical.

VII. Conclusion

In this paper, we propose CamK for inputting text into small mobile devices. By using image processing techniques, CamK can achieve above 95% accuracy for keystroke localization, with only 4.8% false positive keystrokes. 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 to on-screen keyboards.

Acknowledgment

This work is supported in part by National Natural Science Foundation of China under Grant Nos. 61472185, 61373129, 61321491, 91218302, 61502224; JiangSu Natural Science Foundation, No. BK20151390; Key Project of Jiangsu Research Program under Grant No. BE2013116; EU FP7 IRSES MobileCloud Project under Grant No. 612212; CCF-Tencent Open FundChina Postdoctor Science Fund under Grant No. 2015M570434. This work is partially supported by Collaborative Innovation Center of Novel Software Technology and Industrialization. This work was supported in part by US National Science Foundation grants CNS-1320453 and CNS-1117412.

References

[1] M. Fukumoto and Y. Tonomura, "Body coupled fingerring: wireless wearable keyboard," in *Proc. of ACM CHI*, 1997.

[2] M. Kolsch and M. Turk, "Keyboards without keyboards: A survey of virtual keyboards," University of California, Santa Barbara, Tech. Rep., 2002.

[3] K. A. Faraj, M. Mojahid, and N. Vigouroux, "Bigkey: A virtual keyboard for mobile devices," *Human-Computer 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. of ACM CHI*, 2013.

[5] H. Du, T. Oggier, F. Lustenberger, and E. Charbon1, "A virtual keyboard based on true-3d optical ranging," in *Proc. of the British Machine Vision Conference*, 2005.

[6] M. Lee and W. Woo, "Arkb: 3d vision-based augmented reality keyboard," in *Proc. of ICAT*, 2003.

[7] C. Harrison, H. Benko, and A. D. Wilson, "Omnitouch: Wearable multitouch interaction everywhere," in *Proc. of ACM UIST*, 2011.

[8] J. Mantyjarvi, J. Koivumaki, and P. Vuori, "Keystroke recognition for virtual keyboard," in *Proc. of IEEE ICME*, 2002.

[9] 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. of ACM CHI EA*, 2003.

[10] 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. of ACM MobiSys*, 2014.

[11] T. Murase, A. Moteki, N. Ozawa, N. Hara, T. Nakai, and K. Fujimoto, "Gesture keyboard requiring only one camera," in *Proc. of ACM UIST*, 2011.

[12] 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. of ACM PUI*, 2001.

[13] Y. Adajania, J. Gosalia, A. Kanade, H. Mehta, and N. Shekokar, "Virtual keyboard using shadow analysis," in *Proc. of ICETET*, 2010.

[14] R. Biswas and J. Sil, "An improved canny edge detection algorithm based on type-2 fuzzy sets," *Procedia Technology*, vol. 4, pp. 820–824, 2012.

[15] S. A. Naji, R. Zainuddin, and H. A. Jalab, "Skin segmentation based on multi pixel color clustering models," *Digital Signal Processing*, vol. 22, no. 6, pp. 933–940, 2012.

[16] "Otsu's method," https://en.wikipedia.org/wiki/Otsu%27s_method, 2015.

[17] "Opencv library," http://opencv.org/, 2015.

[18] "Monsoon power monitor," http://www.msoon.com/, 2015.

[19] "Swype," http://www.swype.com/, 2015.

[20] Y. S. Kim, B. S. Soh, and S.-G. Lee, "A new wearable input device: Scurry," *IEEE Transactions on Industrial Electronics*, vol. 52, no. 6, pp. 1490–1499, December 2005.

[21] V. R. Pratt, "Thumbcode: A device-independent digital sign language," in *http://boole.stanford.edu/thumbcode/*, 1998.

[22] M. Goldstein and D. Chincholle, "The finger-joint gesture wearable keypad," in *Second Workshop on Human Computer Interaction with Mobile Devices*, 1999.

[23] 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. of ACM CHI*, 2013.

[24] 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. of ACM MobileHCI*, 2013, pp. 175–178.

[25] "iphone app: Paper keyboard," http://augmentedappstudio.com/support.html, 2015.