DynaKey: Dynamic Keystroke Tracking Using a Head-Mounted Camera Device

Hao Zhang, *Student Member, IEEE*, Yafeng Yin, *Member, IEEE*, Lei Xie, *Member, IEEE*, Tao Gu, *Senior Member, IEEE*, Minghui You, and Sanglu Lu, *Member, IEEE*

***Abstract*—Mobile and wearable devices have become more and more popular. However, the tiny touch screen leads to inefficient interaction with these devices, especially for text input. In this article, we propose *DynaKey*, which allows people to type on a virtual keyboard printed on a piece of article or drawn on a desk, for inputting text into a head-mounted camera device (e.g., smart glasses). By using the built-in camera and gyroscope, we capture image frames during typing and detect possible head movements, then track keys, detect fingertips, and locate keystrokes. To track the changes of keys' coordinates in images caused by natural head (i.e., camera) movements, we introduce perspective transformation to transform keys' coordinates among different frames. To detect and locate keystrokes, we utilize the variation of fingertip's coordinates across multiple frames to detect possible keystrokes for localization. To reduce the time cost, we combine gyroscope and camera to adaptively track the keys, and introduce a series of optimizations, such as keypoint detection, frame skipping, multithread processing, etc. Finally, we implement DynaKey on Android-powered devices. The extensive experimental results show that our system can efficiently track and locate the keystrokes in real time. Specifically, the average tracking deviation of the keyboard layout is less than 3 pixels and the Intersection over Union (IoU) of a key in two consecutive images is above 93%. The average keystroke localization accuracy reaches 95.5%.**

***Index Terms*—Camera, dynamic keystroke tracking, head-mounted device, inertial sensor.**

I. Introduction

RECENT years have witnessed an ever-growing popularity of mobile and wearable devices, such as smartphones, smart watches, and smart glasses. These devices usually impose a small form factor design so that they can be carried by users everywhere conveniently. The portable design brings much mobility to these devices, but on the other hand it creates many challenges for human–computer interaction, especially for text input. Some of these devices adopt an on-screen virtual keyboard [1], [2] for text input, but others may require intelligent methods due to the tiny screen or even no screen.

Based on the observation that each finger's typing movement is associated with a unique keystroke, recognizing finger movements has been proposed as a novel text input method, which is achieved by additional wearable sensors (e.g., finger-mounted sensors [3]–[7]) and incurs an additional cost. Considering the users' habits in typing on a common QWERTY keyboard layout, a projection keyboard [24], [27] generated by casting the standard keyboard layout onto a surface via a projector has been proposed, which is used to recognize keystrokes based on light reflection and depends on the dedicated equipment for projection. Recently, with the advance of contactless sensing, recognition of keystrokes can be done via WiFi signals or acoustic signals. For example, WiFi channel state information (CSI) signals have been explored in [8] and [19] to capture keystrokes' typing patterns, the built-in microphone of a smartphone has been used in [16] and [20] to infer keystrokes on a solid surface. However, contactless sensing is usually vulnerable to environmental noises, hence limiting its performance in real-world applications. Therefore, the camera-based approaches [26], [29], [30] have also been proposed to recognize keystrokes on a predefined keyboard layout using image processing. However, existing camera-based text input methods assume a fixed camera and the coordinates of a keyboard layout keep unchanged in the fixed camera view. In reality, the camera of a head-mounted device can hardly keep still. Existing methods may not work in such *dynamic moving scenes* where the camera will suffer from unavoidable movements. Specifically, as shown in Fig. 1, head movements cause camera jitters which lead to changes of keyboard's coordinate in image frames, and eventually cause the mismatch between fingertip and key. The limitation of existing camera-based text input methods strongly motivate the work in this article.

In this article, we propose a novel scheme named DynaKey using camera and gyroscope for text input on a virtual keyboard in *dynamic moving scenes*. DynaKey does not impose a fixed camera, hence it works in more realistic scenarios. Fig. 1 illustrates a typical scenario where a user wears a head-mounted camera device (e.g., smart glasses), while a standard keyboard layout can be printed on a piece of paper or drawn on a desk surface. DynaKey combines the embedded camera and gyroscope to track finger movements and recognize keystrokes

Manuscript received April 1, 2021; revised August 7, 2021; accepted September 4, 2021. Date of publication September 21, 2021; date of current version April 25, 2022. This work was supported in part by the National Key Research and Development Program of China under Grant 2018AAA0102302; in part by the National Natural Science Foundation of China under Grant 61802169, Grant 61872174, Grant 61832008, and Grant 61906085; in part by the JiangSu Natural Science Foundation under Grant BK20180325; and in part by the Collaborative Innovation Center of Novel Software Technology and Industrialization. *(Corresponding author: Yafeng Yin.)*

Hao Zhang, Yafeng Yin, Lei Xie, Minghui You, and Sanglu Lu are with the State Key Laboratory of Novel Software Technology, Nanjing University, Nanjing 210023, China (e-mail: yafeng@nju.edu.cn).

Tao Gu is with the Department of Computing, Macquarie University, Sydney, NSW 2109, Australia.

Digital Object Identifier 10.1109/JIOT.2021.3114224

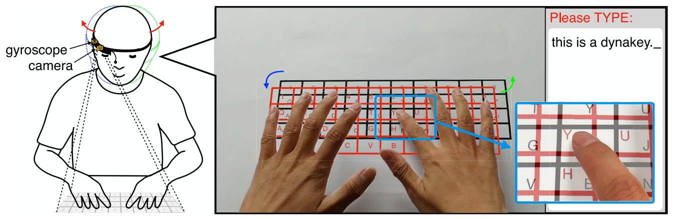


Fig. 1. Typing on a virtual keyboard in dynamic scenarios. Camera movements change the keys' coordinates in image frames and lead to the mismatch between fingertip and key.

in real time. Specifically, while the user types on a virtual keyboard, DynaKey utilizes camera to capture image frames continuously, then detects fingertips and locates keystrokes using image processing techniques. During the tying process, when the head movement is detected by gyroscope, DynaKey needs to track the changes of keyboard coordinate caused by camera movements. This keyboard tracking is crucial due to natural head movements in real application scenarios.

The design of DynaKey creates three key challenges that we aim to address in this article.

The first challenge is how to track changes of keyboard's coordinate accurately so that DynaKey is able to adapt to dynamic moving scenes. In reality, the camera moves naturally along with the head. Such movements will cause dynamic changes of the camera coordinate system. The different camera views and unavoidable image distortion eventually result in changes of keyboard coordinate in image frames. An intuitive solution is to re-extract keyboard layout from each image, but it is costly. In addition, we may not be able to obtain the keyboard layout from each image properly due to unavoidable occlusion by hands. Our intuitive idea asks a fundamental question—can we build a fixed coordinate system no matter how the keyboard coordinate changes? In DynaKey, we propose a *perspective transformation*-based technique that converts any previous coordinate to the current coordinate system. To obtain appropriate feature point pairs for facilitating transformation, we propose a keypoint selection method to dynamically select appropriate cross point pairs from the keyboard layout, while tolerating the occlusion of keyboard.

The second challenge is how to detect and locate keystrokes efficiently and accurately from a single camera view. This is a nontrivial task due to the lack of depth information of fingertips from single-camera view. In the setting of a head-mounted camera and a keyboard located in the front of and below the camera, the camera view from top and behind can hardly get the perpendicular distance between the fingertip and the keyboard plane, i.e., it is difficult to determine whether a finger is typing and which finger is typing. To address this challenge, we utilize the variation of a fingertip's coordinate across multiple frames to detect a keystroke, i.e., whether a finger is typing. In addition to the fingertip movement, we further match a key's coordinate with the fingertip's coordinate to locate which finger is typing.

The third challenge is how to tradeoff between dynamic tracking of keyboard and tracking cost for resource-constrained devices. If the camera does not move or has negligible movements, tracking keyboard's coordinate is unnecessary. To achieve the best tradeoff for resource-constrained head-mounted devices, we introduce a gyroscope-based lightweight method to detect non-negligible camera movements, including short-time sharp movement and long-time accumulated micro movement. Only the detected non-negligible camera movements will trigger the keyboard tracking module to ensure DynaKey work dynamically in real time.

In summary, we make three main contributions in this article.

1) To the best of our knowledge, this article appears the first work focusing on efficient text input using the built-in camera of a head-mounted device (e.g., smart glasses) in dynamic moving scenes. To adapt to the dynamic camera views, we propose a perspective transformation-based technique to track the changes of keyboard's coordinate. Besides, without the depth information of fingertips in a single camera view, we utilize the variation of fingertip's coordinate across multiple frames for keystroke detection.
2) To ensure the real-time response, DynaKey proposes a gyroscope-based lightweight design to adaptively detect the camera movement and remove unnecessary image processing for keyboard tracking. Besides, we introduce a series of optimizations, such as keypoint selection, frame skipping, and multithread processing for image processing.
3) We implement DynaKey on off-the-shelf Android devices, and conduct comprehensive experiments to evaluate the performance of DynaKey. Results show that the average tracking deviation of keyboard layout is less than 3 pixels and the Intersection over Union (IoU) [25] of a key in two consecutive images is above 93%. The accuracy of keystroke localization reaches 95.5% on average. The time response is 63 ms and such latency is below human response time [23].

II. Related Work

Virtual keyboards have been used as an alternative of on-screen keyboards [1], [2] to support text input for mobile or wearable devices with small or no screen. These virtual keyboards can be mainly classified into five categories, i.e., wearable sensor-based, projection-based, WiFi-based, acoustic-based, and camera-based keyboards.

Wearable Sensor-Based Keyboards: Wearable sensors have been used to capture the movements of fingers for text input. iKey [4] utilizes a wrist-worn piezoelectric ceramic sensor to recognize keystrokes on the back of hand. DigiTouch [5] introduces a glove-based input device which enables thumb-to-finger touch interaction by sensing touch position and pressure. MagBoard [3] leverages the triaxial magnetometer embedded in mobile phones to locate a magnet on a printed keyboard. FingerSound [6] utilizes a thumb-mounted ring which consists of a microphone and a gyroscope, to recognize unistroke thumb gestures for text input. These approaches introduce additional hardwares to capture typing behaviors.

Projection-Based Keyboards: Projection keyboards [24], [27] have been proposed for mobile devices, by adopting a conventional QWERTY keyboard layout. They usually require a light projector to cast a keyboard layout onto a flat surface, and then recognize keystrokes based on light reflection. This approach requires dedicated equipment. Microsoft Hololens [13] provides a projection keyboard in front of a user using a pair of mixed-reality smart glasses. During text input, the user needs to move her/his head to pick a key and then make a specific "tap" gesture to select the character. This tedious process may slow down text input and affect user experience.

WiFi-Based Keyboards: By utilizing the unique pattern of CSI in time series, WiFinger [19] is designed to recognize a set of finger-grained gestures to input text for off-the-shelf WiFi devices. Similarly, when a user types on a keyboard, WiKey [8] recognizes the typed keys based on how the CSI value changes at the WiFi signal receiver. However, the WiFi-based approach can be easily affected by environments, such as changes of transceiver's orientation or location, and unexpected human motions in surrounding areas. They are often expected to work in controlled environments, rather than real-world scenarios.

Acoustic-Based Keyboards: By utilizing the built-in microphones of mobile and wearable devices, acoustic-based keyboards have been recently proposed. UbiTap [16] presents an input method by turning the solid surface into a touch input space, based on the sound collected by the microphones. To infer the keystroke's position, it requires three phones to estimate the arrival time of acoustic signals. KeyListener [20] infers keystrokes on the QWERTY keyboard of the touch screen by leveraging the microphones of a smartphone, while it is designed for indirect eavesdropping attacks, the accuracy of keystroke inference is usually not sufficient for text input. UbiK [28] leverages the microphone of a mobile device to locate the keystrokes, while it requires the user to click a key with the fingertip and nail margin, which may be not typical. Some auto speech recognition (ASR) tools [31] are also designed for text input by decoding the speaker's voice, but they can be vulnerable to environmental sounds and not suitable to work in public space needing to keep quiet.

Camera-Based Keyboards: By using a built-in camera, TiPoint [18] detects keystrokes for interactions with smart glasses, it requires a finger to move and click on the mini-trackball to input a character. However, its input speed and user experience need further improvement for real applications. Chang *et al.* [11] designed a text input system for HMDs by cutting a keyboard into two parts. Its performance is almost comparable to that of single-hand text input on tablet computers. Sun *et al.* [26] proposed a depth-aware tapping scheme for VR/AR devices by combining a microphone with a COTS mono camera. It enables tracking of user's fingers based on ultrasound and image frames. Yin *et al.* [29] leveraged a built-in camera in mobile device to recognize keystrokes by comparing the fingertip's location with a key's location in image frames. However, these methods assume that the text input space has a fixed location in the camera view, i.e., the coordinates of the keyboard or keys keep unchanged.

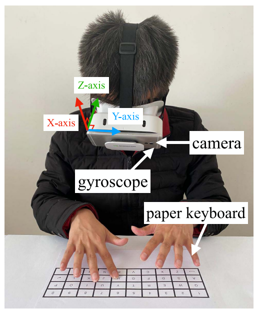
Our work is motivated by the recent advance of camera-based text input methods. We move an important step toward dynamic scenarios where the camera moves naturally with user's head. In our work, the keyboard coordinate in the camera's view changes dynamically, creating more challenges in achieving high accuracy in keystroke localization, and low latency for resource limited head-mounted devices.

III. Observations

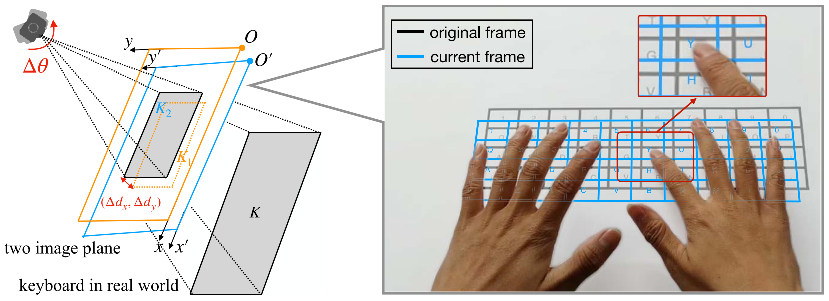
We first conduct our preliminary experiments to study how the changes of keyboard coordinate affect key tracking and keystroke localization in a dynamic scenario. In our experiments, we use a Samsung Galaxy S9 smartphone as a head-mounted camera device, as shown in Fig. 2(a). We use a A4-sized paper keyboard with the Microsoft Hololens [13] keyboard layout and keep its location unchanged. Unless otherwise specified, the frame rate of camera is set to 30 fps. The sampling rate of gyroscope is set to 200 Hz.

Observation 1 (Unconscious Head Movements Can Lead to Large Coordinate Deviations of the Keyboard): As shown in Fig. 2(a), the head-mounted camera moves along with the head. The head movements will lead to the dynamic changes of camera view. When the location of keyboard keeps unchanged, the camera view changes will lead to the changes of keyboard coordinate in the image frames. As shown in Fig. 2(b), the article keyboard is represented as K, and the captured keyboard from the camera view is K_1. We take the case of rotating around the y-axis [marked in Fig. 2(a)] as an example of head (i.e., camera) movements. When the camera slightly rotates $\Delta\theta = 5^\circ$ around y-axis anticlockwise, the image frame changes from $x - y$ plane to $x' - y'$ plane, and the captured keyboard in the image frame changes to K_2. Correspondingly, the location offset of the keyboard achieves $(\Delta d_x, \Delta d_y) = (78, 27)$ pixels, which can lead to the mismatch between coordinates and keys. As shown in the right part of Fig. 2(b), due to the camera movement, the captured keyboard in the current image is shown in blue, while that in the original image is shown in black. In the current frame, i.e., blue keyboard, the user types letter "y." When using the coordinates of keys in the original frame, it may mismatch letter "h" with the keystroke.

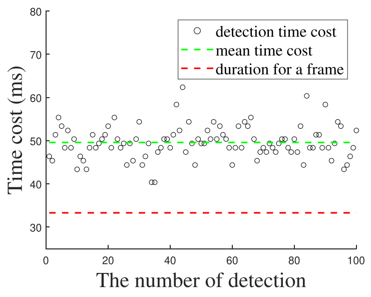
Observation 2 (Extracting All Keys From Each Image Suffers From Unavoidable Occlusion of Hands and Has an Unacceptable Cost of Processing): To track the coordinate changes of keys, an intuitive solution is to extract keys from each image frame. However, considering the hand occlusion which is unavoidable, as shown in Fig. 2(b), it is difficult to extract each key from the image frame accurately. Besides, considering the limited resources of a head-mounted device and the real-time requirement of text input, the processing cost of extracting keys from each image frame is expensive. Specifically, we use Δt to represent the processing cost of key extraction from an image frame, i.e., processing an input image and extracting all keys from the image. In Fig. 2(c), we show the cost of key extraction in 100 different frames. The result shows that the processing cost Δt ranges from 40 to 60 ms, while the average cost is 49 ms, which is larger than



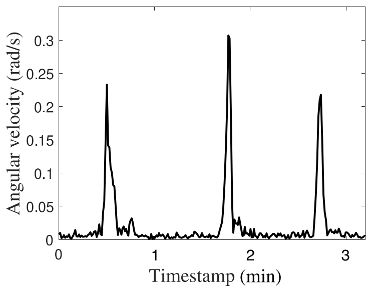
(a)



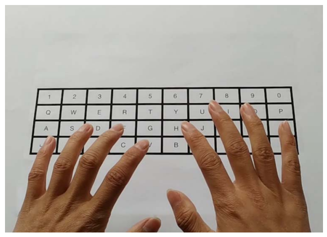
(a)



(c)



(d)



(e)

Fig. 2. Observations about coordinate changes of keys, captured frames for keystrokes, and time cost in image processing. (a) Experimental setup. (b) Unconscious head movements can lead to the large coordinate deviations of keys. (c) Extracting all keys from each image leads to unacceptable time cost for real-time systems. (d) Head movements occur occasionally and last for several frames instead of all frames. (e) Frame from camera view can hardly detect the depth information of fingertips.

the interframe duration (i.e., 33 ms). Therefore, extracting all keys from each image frame to track the coordinates of keys may be unacceptable for real applications. More time-efficient key tracking methods are expected.

Observation 3 (Head Movements Occur Occasionally and Last for Several Frames Instead of All Frames): According to Observation 2, extracting all keys in each image can hardly work. In fact, we find that performing key extraction in each frame is unnecessary. Although the user's head moves during typing, the ratio of head movement duration to the whole typing duration is small. Fig. 2(d) shows that the head movements cause the peaks in gyroscope data during a typing process (i.e., 3 min), the total duration of the three head movements is less than 1 min. It implies that during the typing process, the coordinates of keys in the image frames keep unchanged for more than 67% of the time. Consequently, we only need to re-extract the coordinates of keys when detecting head movements, rather than performing key extraction in each frame.

Observation 4 (A Frame From Camera View Is Insufficient to Detect the Depth Information of Fingertips): To decide whether a keystroke is occurring or not, it is critical to determine whether the fingertip is pressing on a key. However, different from the front camera view, the camera view from top and behind can hardly detect the depth of an object, i.e., the perpendicular distance between the fingertip and the keyboard plane. As shown in Fig. 2(e), all fingers hover above the keyboard. Some fingertips appear above the keys from camera view, hence it is easy to recognize nonkeystrokes as keystrokes by mistake. To address the confusion, we may dynamically track the moving patterns of a fingertip and detect a keystroke from several frames instead of a single frame. We also need an

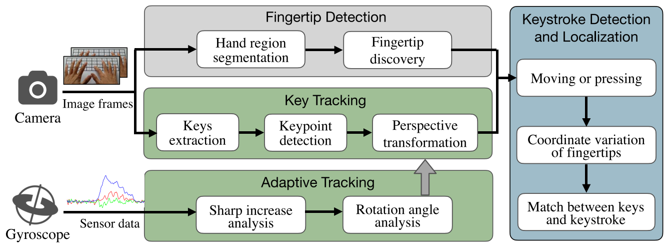


Fig. 3. Architecture of DynaKey.

efficient way to distinguish the fingertip pressing a key from other fingertips.

IV. System Design

We now present the design of DynaKey, which provides a text-input scheme for a head-mounted camera device in dynamic scenes, as shown in Fig. 1. DynaKey works in realistic scenarios where a user types on a virtual keyboard with natural head movement. The keyboard layout can be printed on a piece of paper or drawn on a desk surface. Unless otherwise specified, we use an Android smartphone as the head-mounted camera device, where the embedded camera is used to capture user's typing behaviors, then track and locate keystrokes. The embedded gyroscope is used to detect head movements. In regard to the keyboard layout, it is printed on a piece of paper, as shown in Fig. 2(a).

A. System Overview

Fig. 3 shows the framework of DynaKey. The inputs are image frames captured by camera and the angular velocity collected by gyroscope, while the output is the character of the pressed key. Initially, the user keeps the head unchanged and moves the hand out of the camera view for about 3 s, while using *Key Tracking* to detect the keyboard and extract each key from the initial image frame. When the screen shows "Please TYPE," the user begins typing. During the typing process, we use *Key Tracking* to select keypoints of images to transform the coordinates of keys among different frames. At the same time, we use *Adaptive Tracking* to analyze the angular velocity of gyroscope to detect head (i.e., camera) movements, and then determine whether to update the coordinates of keys or not. In addition, DynaKey uses *Fingertip Detection* to segment the hand region from the frame and detect the fingertips. After that, we use *Keystroke Detection and Localization* to detect the keystroke occurred and locate the keystroke. To ensure DynaKey work in real time, we adopt three threads to implement the image capturing, image processing (i.e., key tracking, fingertip detection, keystroke detection, and localization), and adaptive tracking in parallel.

B. Key Tracking

Before typing, we first need to extract keys from the image. With possible head movements, i.e., camera view changes, we then need to track the coordinates of keys in the following frames, as mentioned in Observation 1 of Section III. Key tracking in DynaKey consists of key extraction and coordinate transformation, as described in the following.

1) Key Extraction: We adopt a common QWERTY keyboard layout, which is printed in black and white on a piece of paper, as shown in Fig. 4(a). Given an input image in Fig. 4(a), we use the Canny edge detection algorithm [10], [29] to obtain all edges, and then find all possible contours from detected edges, as shown in Fig. 4(b) and (c) respectively. The largest contour [i.e., the green contour shown in Fig. 4(c)] with four corners corresponds to the keyboard, where the corners are detected based on the angles formed by the consecutive contour segments, as the red points shown in Fig. 4(d). When the keyboard location is fixed, i.e., four corner points are fixed, as shown in Fig. 4(e), we can detect the keys from the keyboard. Specifically, with small contours [i.e., the red contours shown in Fig. 4(c)] located in the keyboard, we utilize the area of a key to eliminate pitfall contours and then extract each key from the keyboard, as shown in Fig. 4(f). Finally, we map the extracted keys with characters based on the relative locations among keys, i.e., the known keyboard layout.

2) Coordinate Transformation: Due to head movements, it is essential to track the coordinates of keys among different frames. Besides, the camera view changes also bring in the distortion of keyboard in images, as the two captured quadrilaterals $P_0P_1P_3P_2$ and $Q_0Q_1Q_3Q_2$ shown in Fig. 5. To tolerate the camera movement and image distortion, we propose a perspective transformation-based method to track the coordinates of keys.

Perspective Transformation: As shown in Fig. 5, for a fixed point G_i in the physical space, when we obtain its projection point (X_i, Y_i) in the jth frame, perspective

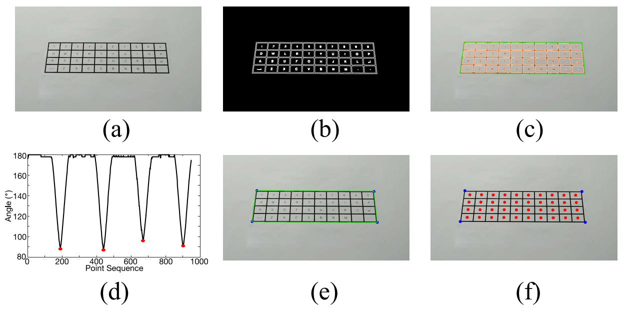


Fig. 4. Process of extracting keys. (a) Input frame. (b) Edge detection result. (c) All detected contours. (d) Corner point detection. (e) Keyboard with corner points. (f) Key extraction result.

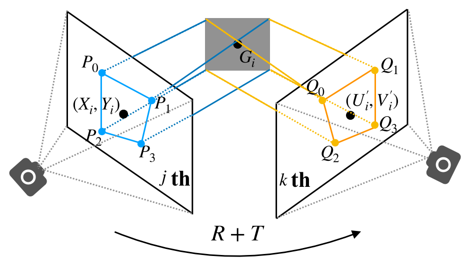


Fig. 5. Principle of perspective transformation.

transformation [21] can use a transformation matrix $C = (C_{00}, C_{01}, C_{02}; C_{10}, C_{11}, C_{12}; C_{20}, C_{21}, C_{22})$ to calculate its projection (U_i', V_i') in the kth frame. Therefore, when the paper keyboard is fixed, we can use the known keyboard/key locations in the previous frames to infer the keyboard/key locations in the following frames, without keyboard detection and key extraction. Specifically, with the known projection point (X_i, Y_i) in the jth frame, we first use C to calculate the 3-D coordinate (U_i, V_i, W_i) related to (X_i, Y_i) in the physical space, as described in (1). We then introduce a division operation to obtain its corresponding projection point (U_i', V_i') in the kth frame, as described as follows:

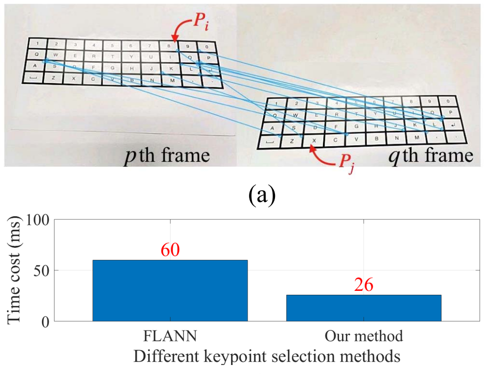
$$\begin{bmatrix} U_i \\ V_i \\ W_i \end{bmatrix} = \begin{bmatrix} C_{00} & C_{01} & C_{02} \\ C_{10} & C_{11} & C_{12} \\ C_{20} & C_{21} & C_{22} \end{bmatrix} \cdot \begin{bmatrix} X_i \\ Y_i \\ 1 \end{bmatrix} \tag{1}$$

$$U_i' = \frac{U_i}{W_i} = \frac{C_{00} \cdot X_i + C_{01} \cdot Y_i + C_{02}}{C_{20} \cdot X_i + C_{21} \cdot Y_i + C_{22}}$$

$$V_i' = \frac{V_i}{W_i} = \frac{C_{10} \cdot X_i + C_{11} \cdot Y_i + C_{12}}{C_{20} \cdot X_i + C_{21} \cdot Y_i + C_{22}}. \tag{2}$$

Here, the projection points of the keyboard or keys in the previous frame can be obtained through key extraction, as mentioned in Section IV-B1. Thus, the main challenge lies in the calculation of transformation matrix C, which will be described below.

Keypoint Selection: In the transformation matrix C, C_{22} is a scale factor and usually set to $C_{22} = 1$, thus we only need to calculate the other eight variables, which can be solved by selecting four nonlinear feature point pairs [e.g., $P_i(X_i, Y_i)$ and $Q_i(U_i', V_i')(i \in [0, 3])$ shown in Fig. 5]. The specific formula for calculating C with four feature point pairs



(b)

Fig. 6. Feature points selection and time cost of FLANN based matcher. (a) Keypoint selection by FLANN-based matcher. (b) Time cost of two keypoint selection methods.

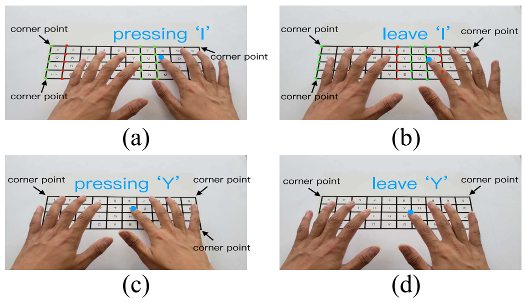


Fig. 7. Hands move on the keyboard and lead to the different occlusion in the process of pressing "I" and "Y." (a) Pressing "I." (b) Leaving "I." (c) Pressing "Y." (d) Leaving "Y."

is shown in (3)

$$\begin{bmatrix} X_0 & Y_0 & 1 & 0 & 0 & 0 & -X_0*U_0' & -Y_0*U_0' \\ X_1 & Y_1 & 1 & 0 & 0 & 0 & -X_1*U_1' & -Y_1*U_1' \\ X_2 & Y_2 & 1 & 0 & 0 & 0 & -X_2*U_2' & -Y_2*U_2' \\ X_3 & Y_3 & 1 & 0 & 0 & 0 & -X_3*U_3' & -Y_3*U_3' \\ 0 & 0 & 0 & X_0 & Y_0 & 1 & -X_0*V_0' & -Y_0*V_0' \\ 0 & 0 & 0 & X_1 & Y_1 & 1 & -X_1*V_1' & -Y_1*V_1' \\ 0 & 0 & 0 & X_2 & Y_2 & 1 & -X_2*V_2' & -Y_2*V_2' \\ 0 & 0 & 0 & X_3 & Y_3 & 1 & -X_3*V_3' & -Y_3*V_3' \end{bmatrix} \cdot \begin{bmatrix} C_{00} \\ C_{01} \\ C_{02} \\ C_{10} \\ C_{11} \\ C_{12} \\ C_{20} \\ C_{21} \end{bmatrix} = C_{22} \cdot \begin{bmatrix} U_0' \\ U_1' \\ U_2' \\ U_3' \\ V_0' \\ V_1' \\ V_2' \\ V_3' \end{bmatrix}. \quad (3)$$

To get the feature point pairs, FLANN-based matcher [22] was often adopted, which finds an approximate (may be not the best) nearest neighbor point in image p for the point in image q, and then pairs the two points. Considering the possible wrongly selected feature point pair, such as P_i and P_j in Fig. 6(a), the FLANN-based method often needs to detect a large number of feature point pairs, and then selects the top-k (k is usually larger than 4) feature point pairs to calculate the transformation matrix with the least square method. However, selecting a larger number of feature points will lead to non-negligible time latency (e.g., 60 ms), which is larger than interframe duration (i.e., 33 ms) and unacceptable in real-time systems, as shown in Fig. 6(b). Therefore, it is necessary to quickly and the accurately select appropriate number of feature point pairs for transformation matrix calculation.

Algorithm 1: Keypoint Selection

```
Input: An image frame.
Using skin segmentation to extract hand regions.
Using Canny edge detector to get the top, leftmost, rightmost, and bottom lines $\{L_k\}, k \in [1,4]$.
Corner point set $P = \emptyset$.
while $k \le 4$ & $P = \emptyset$ do
    for $P_l' \in L_k$ & $P = \emptyset$ do
        A square area containing point $P_l$ is $S_l' = \{(x_i, x_l') \le \delta x, |y_i - y_l'| \le \delta y\}$.
        The ratio of black pixels in $S_l'$ is $\rho_l$.
        if $\rho_l > \Delta\rho_c$ then
            Fitting a line $l_y$ for black pixels satisfying $|x_i - x_l'| \le \delta x$.
            Fitting a line $l_x$ for black pixels satisfying $|y_i - y_l'| \le \delta y$.
            The angle between $l_y$ and $l_x$ is $\gamma$.
            if $|\gamma - 90°| < \Delta\epsilon$ then
                $P_l'$ is the corner point and $P = \{P_l'\}$.
Detecting cross points.
Matching common cross point pairs in two frames.
Selecting four noncolinear cross point pairs as keypoint pairs $\{(X_i, Y_i), (U_i', V_i')|i \in [0,3]\}$.
Output: The keypoint pairs.
```

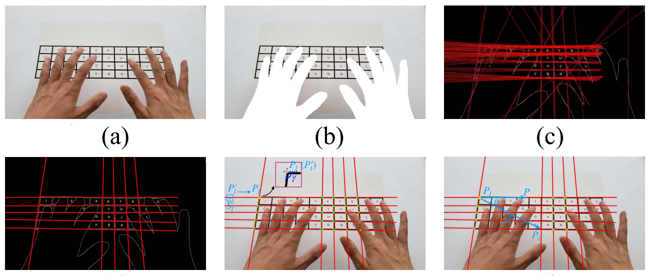


Fig. 8. Process of selecting keypoints. (a) Input frame. (b) Hand segmentation. (c) Line detection. (d) Optimized line detection. (e) Corner point. (f) Keypoint selection.

To achieve the above goal, we introduce *keypoint selection* to calculate C with only four keypoint pairs, where keypoints mean cross points of lines in the keyboard. As shown in Fig. 7, due to the size differences of the keyboard and hands, whatever the location of the occlusion is, the cross points in the keyboard will not be occluded completely at the same time. In addition, during a typing process, we observe that the hand movements between two consecutive frames are not violent, i.e., there often exist several common cross points for the two frames, as the green points shown in Fig. 7(a) and (b). Therefore, we can detect the common cross points appearing on both of the two consecutive frames (i.e., cross point pairs), and select four noncolinear keypoint pairs for perspective transformation, as shown in Algorithm 1.

a) Line detection: With an input image as shown in Fig. 8(a) which equals to Fig. 7(b), we first utilize skin segmentation [29] to segment the hand region from the image, shown as the white region in Fig. 8(b). Then, we get the edges in Fig. 8(b) using the Canny edge detector [10], as shown in Fig. 8(c). After that, to detect the lines of keyboard and reduce the interference of other edges, we use the Hough transformation [15] to detect the long lines in image, shown as the red lines in Fig. 8(c). However, there are too many lines, which may confuse the cross point detection. Therefore, we merge the detected lines. For convenience, we represent each line in polar coordinates with a vector (ρ, θ). For the lines close to each other, which satisfy $\Delta\rho < 50$ pixels and $\Delta\theta < 5.7°$, we only select one of them. The optimized line detection result for Fig. 8(c) is shown in Fig. 8(d).

b) Corner point detection: As shown in Fig. 8(d), not all lines of the keyboard (i.e., not all cross points) can be detected, due to the occlusion of hands. Correspondingly, the location of a detected cross point can not be directly inferred. To solve this problem, we introduce the corner point of keyboard to infer the location of a detected cross point, based on the relative position between corner point and other cross points. Specifically, we observe that there usually exist one or more corner points in captured images during typing, as shown in Fig. 7. Particularly, the top left or the top right corner often exists. Therefore, we utilize the top, leftmost, rightmost, and bottom detected line by priority to detect possible corner points, until one corner point is detected.

Take the top line as an example, we trace the points of the top line from the leftmost point to right, to detect the top left corner point. As shown in Fig. 8(e), for a point $P_l'(x_l', y_l')$ in the top line, we use a square area $S_l' = \{(x_i, y_i)\,|\,|x_i - x_l'| \le \delta x, |y_i - y_l'| \le \delta y\}$ to verify whether P_l' is a corner point. When the ratio of the number of black pixels (i.e., the possible contour of a corner) to the number of all pixels in S_l' is larger than $\Delta\rho_c$, P_l' can be a candidate corner point. After that, we fit a line for the black pixels satisfying $|x_i - x_l'| < \delta x$ and the black pixels satisfying $|y_i - y_l'| < \delta y$, respectively, as shown in Fig. 8(e). If the angle γ between the two fitted lines satisfies $|\gamma - 90| < \Delta\epsilon$, P_l' will be selected as the top left corner, (i.e., P_l). Based on extensive experiments, we set $\Delta\rho_c = 0.25$, $\Delta\epsilon = 6°$, $\delta x = \delta y = 5$ by default. It is worth noting that if all borders (i.e., all corner points) of the keyboard are not detected, we will skip this frame. This is because there usually have no valid keystrokes, when all borders are blocked. Otherwise, if any border of the keyboard is detected, we will then detect the corner points for key tracking.

c) Common cross point detection: For other detected lines, we extend the length of each line to detect the cross points, as the green points shown in Fig. 8(e). To extract the common cross point set detected in two frames, we first utilize the detected top-left corner point P_l to infer the location of a cross point. Specifically, we represent the location of a cross point P_i with a distance d_i and an angle θ_i. As shown in Fig. 8(f), the distance d_i is measured as the Euclidean distance between P_i and P_l, and the θ_i is computed as the angle between $\overrightarrow{P_lP_i}$ and $\overrightarrow{P_lP}$. Here, the point P is a randomly selected point on the right of P_l in top line. By comparing d_i and θ_i of each point in two frames, we pair two keypoints with similar distance and angle, i.e., the distance difference in two frames satisfies $\delta d < 20$ pixels while the angle difference satisfies $\delta\theta < 4°$. In Fig. 8(f), the yellow and the green keypoints are selected as common cross points.

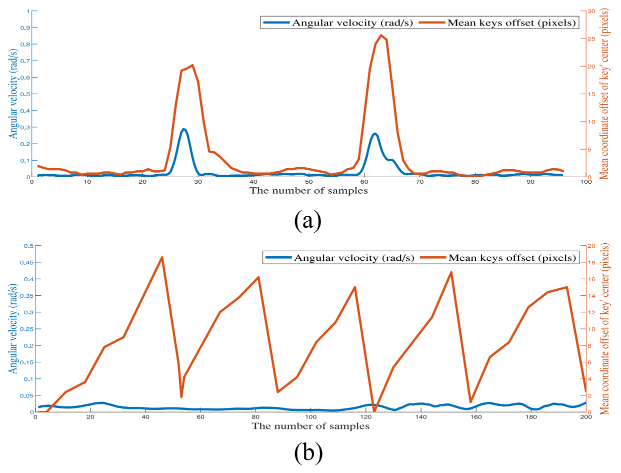


Fig. 9. Sharp and micro camera movements versus coordinate changes of keys. (a) Short-time sharp camera movements versus coordinate changes of keys. (b) Long-time micro camera movements versus coordinate changes of keys.

d) Keypoint pair determination: Finally, we select four noncolinear cross point pairs as keypoint pairs, $\{(X_i, Y_i), (U_i', V_i')|i \in [0, 3]\}$, which will be used for calculating the transformation matrix. As shown in Fig. 8, by only detecting several intersection points instead of a large number of feature points, we can reduce the time of processing one image for key tracking from 60 to 26 ms, as shown in Fig. 6(b), which is smaller than the interframe interval and satisfies the real-time requirement.

C. Adaptive Tracking

Based on Observation 3, head movements occur occasionally, thus it is unnecessary to track keys from each image. To reduce the unnecessary computation overhead in image processing, we present an adaptive key tracking scheme by introducing a gyroscope, and only activate the previous *Key Tracking* module when the head movement is detected.

In Fig. 9(a), we show the gyroscope data and coordinate changes of keys during a typing process. When the head movement occurs, there is a sharp increase of angular velocity. Considering that the size of a key in an image is only about 45×25 pixels, when the head movement occurs, the coordinate offset can achieve more than half of a key's height, which can probably lead to the mismatch between the locations and keys, thus we need to activate *Key Tracking* module (Section IV-B) to get the new coordinates of keys in the current frame. Specifically, we use $\omega(t) = \sqrt{\omega_x^2(t) + \omega_y^2(t) + \omega_z^2(t)}$ to represent the angular velocity at time t, where $\omega_x(t)$, $\omega_y(t)$, and $\omega_z(t)$ represent the angular velocity in x-axis, y-axis, and z-axis, respectively. When $\omega(t) \ge \epsilon_g$, we activate *Key Tracking* module, where we set $\epsilon_g = 2.9°/\mathrm{s}$ by default.

In fact, in addition to the sharp increase of angular velocity caused by non-negligible head movements, the long-time micro camera movements will also lead to the coordinate changes of keys. As shown in Fig. 9(b), there is no sharp increase of gyroscope data, while the accumulated rotation

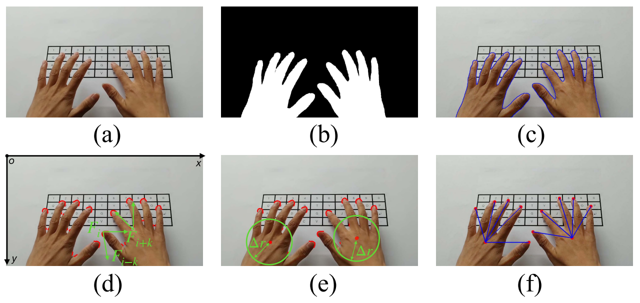


Fig. 10. Process of detecting fingertips. (a) Input frame. (b) Hand segmentation. (c) Hand contour. (d) Possible fingertips. (e) Remove pitfall fingertips. (f) Final fingertips.

angle from the last time of key tracking can lead to the non-negligible coordinate changes. In this case, we introduce $\Delta\theta_r = \int_{t=t_0}^{t=t_1} \omega(t)dt$, where t_0 means the last time of key tracking and t_1 means the current time. If $\Delta\theta_r \geq \epsilon_r$, we activate the *Key Tracking* module, where we set $\epsilon_r = 3.1^\circ$ by default. By introducing adaptive tracking, we can remove unnecessary image processing for key tracking.

D. Fingertip Detection

After we obtain the coordinate of each key, we need to detect the fingertips for further keystroke detection and localization. Given an input image, as shown in Fig. 10(a), we first utilize skin segmentation [29] to extract the hand region from the image, as shown in Fig. 10(b). We then use the hand contours shown in Fig. 10(c) and the shape feature [29] of a fingertip to detect the possible fingertips, as shown in Fig. 10(d). After that, we move along the hand's contour in Fig. 10(c) to remove the pitfall points corresponding to fingerwebs. Specifically, we use F_i to represent the possible fingertip point, while using F_{i-k} and F_{i+k} to represent the points visited before and after F_i. As shown in Fig. 10(d), if $\overrightarrow{F_iF_{i-k}} \times \overrightarrow{F_iF_{i+k}} > 0$, F_i can be treated as a fingertip. Otherwise, it is a pitfall point in the fingerweb and will be eliminated, as shown in Fig. 10(e). Besides, we introduce the distance between the possible fingertip and the center of the hand, to further remove the pitfall points with distances smaller than Δr. Finally, for each cluster of points related to a fingertip, we choose the middle point to represent the final detected fingertip, as shown in Fig. 10(f). Unless otherwise specified, we set $k = 50$ and $\Delta r = 100$ pixels by default.

E. Keystroke Detection and Localization

After obtaining coordinates of keys and detecting fingertips, we will detect and locate keystrokes. Specifically, we first determine whether a typing operation occurs, i.e., keystroke detection, and then determine which fingertip is pressing the key, i.e., keystroke localization, as shown in Algorithm 2.

1) Keystroke Detection: According to Observation 4, the depth information of fingertips is hardly obtained through a single image , thus we detect a keystroke from multiple consecutive frames. Specifically, a keystroke operation involves several steps, first the fingertip moves toward the key, then stays on the key for a short duration, and finally moves away from it. An example is shown in Fig. 11 (i.e., the seventh fingertip). Therefore, the coordinate changes of fingertips can be used to detect possible keystrokes. Additionally, to reduce the processing cost, we introduce a *frame-skipping* scheme for keystroke detection, instead of detecting the coordinates of fingertips from each image frame.

Algorithm 2: Keystroke Detection and Localization

```
Input: The consecutive frames.
The $i$th fingertip in the $j$th frame is $(x_i^{(j)}, y_i^{(j)})$, which is
transformed to $(x_i^{(j)'}, y_i^{(j)'})$ in the $k$th frame, $k = j + 5$.
if $\sqrt{(x_i^{(j)'} - x_i^{(k)})^2 + (y_i^{(j)'} - y_i^{(k)})^2} < \epsilon_d$ then
    if The jth frame has no keystroke then
        Detecting a new keystroke.
else if $\sqrt{(x_i^{(k-1)'} - x_i^{(k)})^2 + (y_i^{(k-1)'} - y_i^{(k)})^2} < \epsilon_d$ then
    Detecting a new keystroke.
if A new stroke is detected then
    Selecting fingertip with largest coordinate variation.
    Matching fingertip with key by coordinates.
Output: The located keystroke.
```

To capture enough information for keystroke detection and localization, we first set the frame rate of camera to 30 fps, which is the maximum/default frame rate of off-the-shelf mobile devices. According to [9], the duration of a keystroke usually lasts 185 ms, which is about the duration of capturing five or six frames. Therefore, we first process every five image frames and compare each fingertip's coordinate. For convenience, we use $T_i^{(j)}(x_i^{(j)}, y_i^{(j)})$ and $T_i^{(k)}(x_i^{(k)}, y_i^{(k)})$ to represent the ith fingertip's coordinate in the jth and the kth frames, where $k = j + 5$. Considering that camera movement may happen between the jth frame and the kth frame, $T_i^{(j)}$ is transformed to the coordinate system of the kth frame as $T_i^{(j)'}(x_i^{(j)'}, y_i^{(j)'})$, based on perspective transformation. If the coordinate change $\delta d = \sqrt{(x_i^{(j)'} - x_i^{(k)})^2 + (y_i^{(j)'} - y_i^{(k)})^2}$ is less than ϵ_d, the fingertip is considered unchanged, otherwise it is moving. We set $\epsilon_d = 15$ pixels by default.

After obtaining the coordinate changes of a fingertip from every five frames, we further need to determine whether a fingertip is pressing a key. As mentioned before, the duration of a keystroke usually lasts for 185 ms. If the coordinates of a fingertip $T_i^{(j)'}$ and $T_i^{(k)}$ from the jth to the kth ($k = j + 5$) frames keep unchanged, it implies that the fingertip keeps staying on the pressed key during the last five frames because the duration for pressing a key in the jth frame, then moving out and coming back to the same key is usually larger than 185 ms (i.e., more than the duration of five frames). At this time, if we have processed a keystroke in the jth frame, we will not process the keystroke in the kth frame repeatedly. Otherwise, we detect a new possible keystroke in the kth frame. Differently, if the coordinates of fingertip $T_i^{(j)'}$ and $T_i^{(k)}$ from the jth to the kth frames change, we need to further determine whether there is a keystroke in the kth frame. At this time, we introduce the $(k-1)$th frame, detect the coordinate of the fingertip as $T_i^{(k-1)}$,

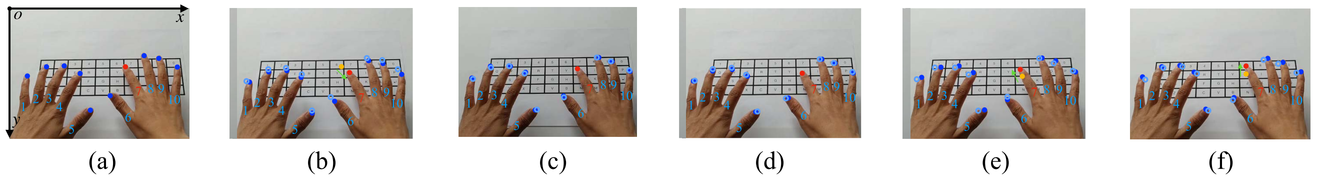


Fig. 11. Sampled frames in the process of pressing “U” with fingertip 7. The yellow and red points are the locations of the fingertip 7 in previous and current sampled frame, respectively. The green arrow indicates the trend of the fingertip’s movement. The blue circles and points are the locations of other fingertips in previous and current sampled frame, respectively. (“Fm” is short for frame.) (a) Fm 1: moving to “U.” (b) Fm 4:moving to “U.” (c) Fm 7: pressing “U.” (d) Fm 10: pressing “U.” (e) Fm 13: leaving “U.” (f) Fm 16: leaving “U.”

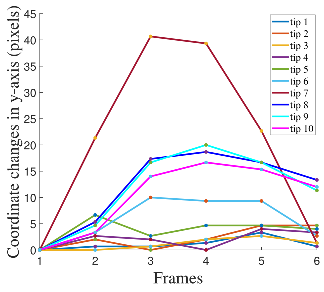


Fig. 12. Coordinate changes in y-axis of ten fingertips.

and transform $T_i^{(k-1)}$ to the coordinate system of the kth frame as $T_i^{(k-1)'}$. Then, we calculate the coordinate changes of fingertip $\delta d' = \sqrt{(x_i^{(k-1)'} - x_i^{(k)})^2 + (y_i^{(k-1)'} - y_i^{(k)})^2}$ between the $(k-1)$th and the kth frames. If $\delta d' > \epsilon_r$, the fingertip keeps moving, there is no keystroke. Otherwise, we detect a possible keystroke in the kth frame, and keystroke localization will be described in the following section.

2) Keystroke Localization: Keystroke localization is to detect which finger is typing. As shown in Fig. 11, although all fingertips move together during a keystroke, the fingertip T_k pressing a key often has the largest coordinate changes, especially in y-axis. This is because T_k needs to move toward the target key, stay on the key, and then move away, while other fingertips often keep hovering or staying on the keyboard without a large variation of coordinates. As shown in Fig. 12, the “fingertip 7” pressing a key has the largest variation of coordinates in y-axis. For the detected fingertip pressing a key, we further match the coordinate of the fingertip and the location of a key to locate the keystroke.

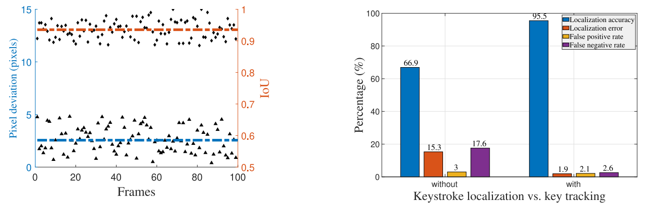
3) Adaptive Calibration: However, considering the possible errors in keystroke detection and localization, we introduce the adaptive calibration scheme for a better typing experience. First, in the user interface, we keep the “ADD” and “DELETE” operations. If the typing operation is not detected, the user can use “ADD” button in the top right corner of user interface to input the character by screen. If the typing operation is wrongly detected/located, the “DELETE” button in the top-right corner of user interface can be used to remove the character. Second, considering the language rules in regular text, we introduce the Bayesian method [33] to correct the wrong keystroke sequence. That is to say, given the keystroke sequence, we calculate the likelihood of each possible word and finally select the word with largest likelihood. In this way, we can tolerate the errors like false-negative keystrokes, false-positive keystrokes, and wrongly detected keystrokes.

V. Performance Evaluation

We deploy DynaKey on a Samsung Galaxy S9 smartphone which is used as a head-mounted camera device, as shown in Fig. 2(a). The smartphone runs Android OS 9.0. We use a Microsoft Hololens [13] keyboard layout and print it on a piece of A4-sized paper. Unless otherwise specified, the frame rate of camera is set to 30 fps, the sampling rate of gyroscope is set to 200 Hz, the image size is set to 800×480 pixels. We conduct our experiments in an office environment. We recruit twelve volunteers to participate in the experiments and each subject types a set of predefined 1600 characters. Data sanitized is done to ensure no private and identity information. We first evaluate the performance of key tracking and keystroke localization. Then we evaluate how camera jitters, frame sizes, and frame rates affect the performance of key tracking and keystroke localization. We also evaluate the performance of DynaKey in complex scenarios to explore its usage modes. After that, we evaluate the latency and energy consumption of DynaKey. Finally, we evaluate DynaKey on text input, and compare DynaKey with the state-of-the-art text input methods.

A. Performance Metrics

To measure the accuracy of key tracking, we use $E_r = (1/z)\sum_{i=1}^{z}\sqrt{(x_{m_i} - x_{g_i})^2 + (y_{m_i} - y_{g_i})^2}$ to represent the average pixel deviation between the calculated cross points’ coordinates forming the keyboard layout and the ground truth, and $IoU = (1/n)\sum_{i=1}^{n} IoU_i$ to represent the average IoU [25] between the calculated keys’ areas and the ground truth. The smaller E_r the better, and the larger IoU the better. Here, (x_{m_i}, y_{m_i}) and (x_{g_i}, y_{g_i}) represent the calculated coordinate and the ground truth of the ith cross point, respectively, and IoU_i represents the IoU between the calculated ith key’s area and the ground truth. $z = 55$ represents the number of cross points, while $n = 40$ represents the number of keys, as shown in Fig. 4(a). We obtain the ground truth by manually detecting the premarked coordinates of cross points and keys from each image frame. To measure the performance of keystroke localization, we use several metrics–localization accuracy, localization error, false-positive rate (FPR) and false-negative rate (FNR). The localization accuracy is the ratio of correctly located keystrokes to the number of keystrokes performed by subject. The localization error is the ratio of falsely located keystrokes to the number of keystrokes performed by subject.



(a) (b)

Fig. 13. Performance of key tracking and keystroke localization. (a) Key tracking accuracy in 100 frames. (b) Keystroke localization performance with and without key tracking.

FPR and FNR are defined as the ratio of falsely detected keystrokes and missed keystrokes to the number of keystrokes performed by subject, respectively.

B. Accuracy of Key Tracking and Keystroke Localization

In the experiment, a subject is instructed to type on the keyboard in her/his own way. She/he may move her/his head naturally during the typing process. We evaluate the accuracy of key tracking by the aforementioned pixel deviation E_r and the average intersection over union IoU in 100 frames. As shown in Fig. 13(a), the pixel deviation E_r in an image ranges from 0 to 5 pixels, and the average pixel deviation E_r among the frames is less than 3 pixels. When comparing with the key size, i.e., 45×25 pixels, the deviation less than 3 pixels can be neglected. Meanwhile, the average IoU achieves above 93%, indicating that the area of the calculated key coincides with the ground truth in a high degree. To conclude, DynaKey accurately tracks the coordinate changes of keys in different frames while tolerating head movements during the typing process.

To evaluate the performance of keystroke localization and tracking in dynamic scenes, we instruct a subject to press all the keys on the keyboard without and with the key tracking module. Fig. 13(b) shows that the keystroke localization accuracy without key tracking module is only about 66.9%, while the localization error and false-negative rate are also high. This may be mainly due to the mismatch between the key's location and its coordinates in dynamic camera views. With the key tracking module, the keystroke localization accuracy increases significantly, i.e., from 66.9% to 95.5%, and localization error, false-positive rate, and false-negative rate are 1.9%, 2.1%, and 2.6%, respectively. The results demonstrate that DynaKey accurately locates the keystrokes, and the key tracking module plays a critical role in keystroke localization in dynamic scenes.

C. Effect of Camera Jitters

In this experiment, we evaluate the performance of key tracking and keystroke localization under different camera jitters. First, we change the range of camera jitters, i.e., from stationary, slight ($1.28° \pm 0.26°$), obvious ($6.8° \pm 4.4°$) to large ($18.4° \pm 3.7°$). Here, *stationary* means the device keeps unchanged during typing, while other jitters mean different ranges of camera movements, which are controlled by attaching the device to a motor. The performance of key tracking and keystroke localization are shown in Figs. 14(a) and Fig.15(a), respectively. The results show good performances of key tracking and keystroke localization under slight and obvious range of jitters. When the camera jitter is obvious, the average pixel deviation is less than 3 pixels while the average IoU achieves 92.3%, and the localization accuracy reaches 93.7%. When the camera suffers from large jitters, the performance of key tracking reduces clearly, the localization accuracy drops to 89.1%. This may be caused by the mismatch between the detected fingertip and the key's coordinate during large jitters.

In addition, we evaluate the performance of DynaKey by changing the frequency of jitters, i.e., keeping stationary and moving in low ($0.04° \pm 0.03°$/s), medium ($0.09° \pm 0.06°$/s), and high ($0.2 \pm 0.15°$/s) speed, respectively. The subject types the same text as the above experiment. As shown in Figs. 14(b) and Fig. 15(b), DynaKey can tolerate low and medium camera jitters well. When camera moves in medium speed, the average pixel deviation is 3.2 pixels and the IoU is 92.2%, and the keystroke localization accuracy reaches 93.3%, respectively. In the case of high-frequency jitters, it is hard to guarantee that the updated coordinates of keys perfectly match with the fingertip pressing the key, the tracking and localization performance decreases. The pixel deviation increases to 4.5 pixels while the IoU decreases to 88.9%, and the false-negative rate for keystroke localization increases to 7.5%. However, considering the normal or unconscious head/camera movements during a typing process, the large or high-frequency jitters are rare, thus DynaKey performs well in typical cases. Besides, when using the camera with higher frame rates to capture fine-grained camera movements, it is possible to mitigate the effect from large or high-frequency jitters.

D. Effect of Frame Sizes and Frame Rates

In this experiment, we evaluate how image sizes affect the performance of DynaKey. When the frame size is small, e.g., 480×320 pixels, the keyboard in the captured frame involves too few pixels to be extracted accurately, leading to poor performance. When the frame size increases to 800×480 pixels, the performance shows good results. When the frame size keeps increasing to 1280×720 pixels, the performance has a little decrease. This may be because the higher image resolution leads to the keyboard containing more pixels, resulting in a larger pixel deviation for key tracking. Besides, the higher image resolution also causes higher image processing cost, which may be too slow to process each keystroke and leads to higher false-negative rate. In practice, to minimize latency and power consumption while guaranteeing the keystroke localization performance, the frame size is set to 800×480 pixels.

In addition, to show the efficiency of our frame-skipping scheme in Section IV-E1, we evaluate how the frame rates affect the system performance. Specifically, the default frame rate of the camera is 30 fps, which is usually the maximal/default frame rate of off-the-shelf Android smartphones. Then, we change the interval of processing an image, i.e., we process every N_d images and $N_d \in [1, 10]$. Fig. 14(d) shows that the change of N_d has little effect on the performance

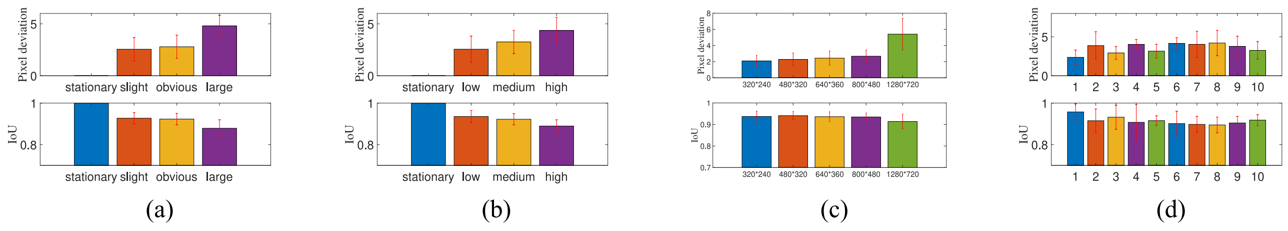


Fig. 14. Performance of key tracking under different ranges and speeds of jitters, frame sizes, and rates. (a) Key tracking versus range of jitters. (b) Key tracking versus speed of jitters. (c) Key tracking versus frame sizes. (d) Key tracking versus sampling interval.

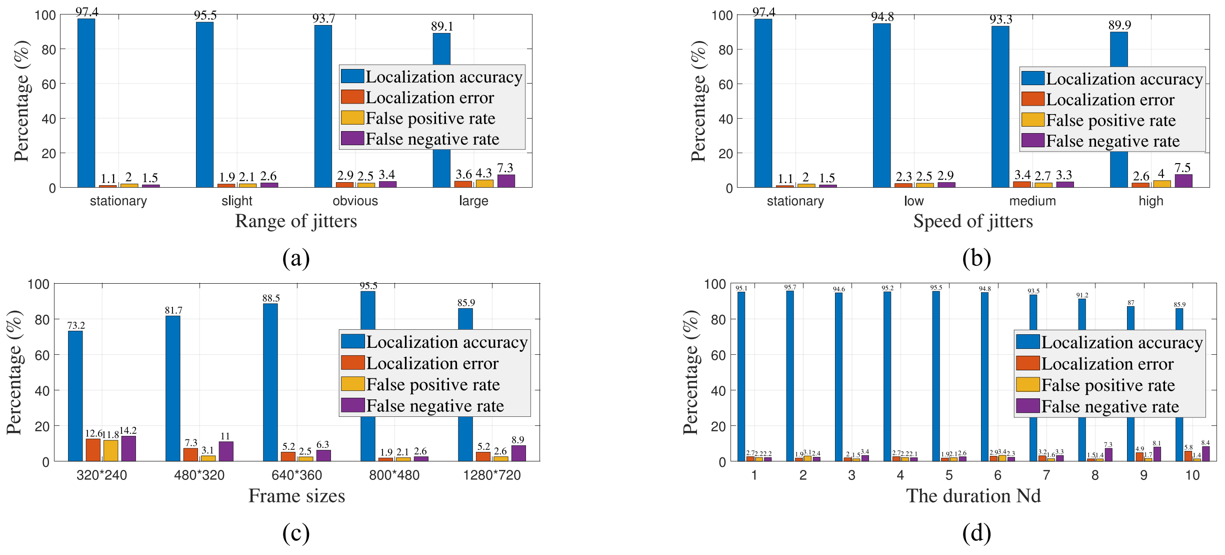


Fig. 15. Performance of keystroke localization under different ranges and speeds of jitters, frame sizes and rates. (a) Keystroke localization versus range of jitters. (b) Keystroke localization versus speed of jitters. (c) Keystroke localization versus different frame sizes. (d) Keystroke localization versus different sample intervals.

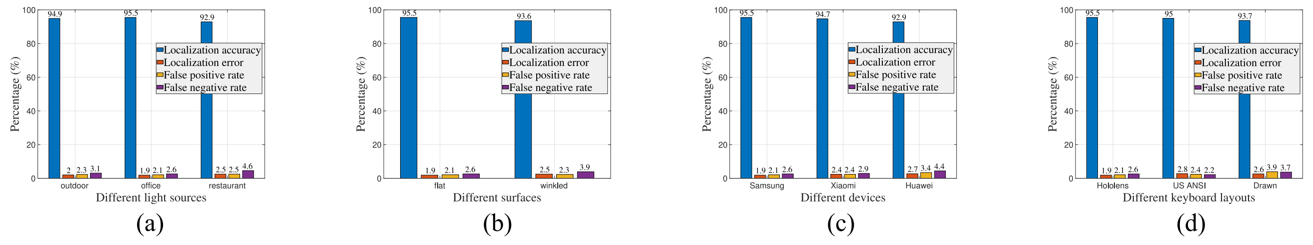


Fig. 16. Keystroke localization performance under different complex scenarios. (a) Keystroke localization versus different light sources. (b) Keystroke localization versus different surfaces. (c) Keystroke localization versus different devices. (d) Keystroke localization versus different keyboard layouts.

of key tracking. However, N_d affects the frequency of key tracking and keystroke localization performance. Fig. 15(d) shows that when the interval $N_d <= 5$, DynaKey performs well in keystroke localization. When the interval is 5, the localization accuracy reaches 95.5%. However, when the interval keeps increasing, it may miss some keystrokes, leading to a lower localization accuracy and a higher false-negative rate. To achieve a better tradeoff between the keystroke localization performance and computation overhead, we set the interval $N_d = 5$, i.e., DynaKey processes every five frames.

E. Effect of Complex Scenarios

1) Different Light Sources: In this experiment, we evaluate whether DynaKey can locate the keystrokes efficiently in the environments with different light conditions. We conduct the experiments in three typical scenarios: 1) an office environment (light color is close to white); 2) outdoors (basic light); and 3) a restaurant (light is a bit warm). A subject is instructed to type the same set of characters in these three scenarios. As show in Fig. 16(a), DynaKey achieves good performance in all the three scenarios, i.e., the localization accuracy is 94.4% on average, while the average localization error, false-positive rate, and false-negative rate are 2.1%, 2.3%, and 3.4%, respectively. In the office scenario, DynaKey achieves the best keystroke localization accuracy, i.e., 95.5%.

2) Different Surfaces: In real-world applications, repeatedly handling a printed paper keyboard may easily cause

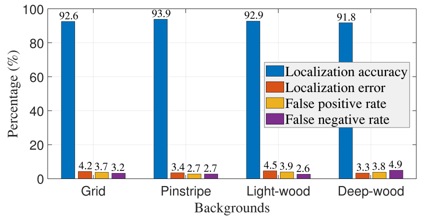


Fig. 17. Keystroke localization versus backgrounds with different stripes.

warping. In this experiment, we use a flat paper keyboard and a wrinkled one to evaluate the effect of wrinkled paper on keystroke localization. Fig. 16(b) shows that when typing on the flat keyboard layout, the accuracy of keystroke localization achieves 95.5% on average. When typing on the wrinkled one, the localization accuracy is 93.6% on average. Wrinkled paper may affect the detection of long lines and corner points of the keyboard layout, thus the keystroke localization performance for a wrinkled paper keyboard is slightly worse than that of the flat one.

3) Different Backgrounds: When placing the keyboard on object surface, the texture of surface (i.e., background) may affect the detection of keyboard, especially for surfaces with stripes which are similar to lines in keyboard. Therefore, we use four kinds of backgrounds with stripes (i.e., grid, pinstripe, wood with light color, and wood with deep color) to evaluate DynaKey. As shown in Fig. 17, the stripes in backgrounds only have a little effect on keystroke detection and localization. In fact, the lines in keyboard have fixed rules (e.g., distance and length), while the rules in stripes of backgrounds are usually different from that in keyboard. Thus DynaKey can work well for keyboard/key tracking. In regard to the performance decrease in the fourth bar, it is mainly caused by the color of background, which is closer to skin color and can affect hand/finger extraction.

4) Different Devices: In addition to the Samsung Galaxy S9 smartphone, we also evaluate the performance of keystroke localization in DynaKey using two other smartphones—XiaoMi Note 3 (Android OS 8.1) and Huawei Honor 7i (Android OS 6.0). As shown in Fig. 16(c), the average localization accuracy is 95.5% for Samsung phone, 94.7% for XiaoMi phone, and 92.9% for Huawei phone. The performance difference may come from the aspects like the location of camera, viewing angle of camera, the size of device, etc. Nevertheless, DynaKey can work well in different devices.

5) Different Keyboard Layouts: In this experiment, we use two common keyboard layouts—Hololens [13] and U.S. ANSI [17], to evaluate the performance of keystroke localization. Each layout is printed on a piece of A4-sized paper. Fig. 16(d) shows that whatever the keyboard layout is, DynaKey has good performance in keystroke localization, i.e., the accuracy achieves above 94.3%. Besides, to further explore the scalability of DynaKey, we draw a A4-sized Hololens keyboard layout on the surface of a table, and the subject is instructed to type a set of characters as in previous experiments. As shown in Fig. 16(d), even if we replace the article keyboard with a drawn keyboard layout, the accuracy of keystroke localization still reaches above 93%, indicating that DynaKey can work with a simple keyboard layout.

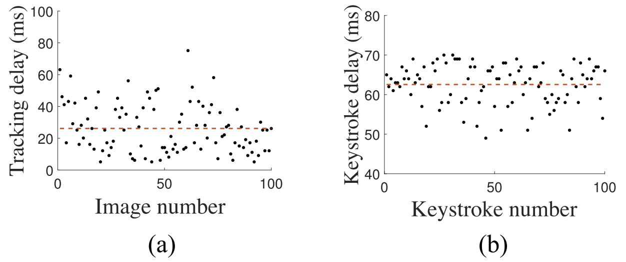


Fig. 18. Latency of key tracking and keystroke localization. (a) Latency of key tracking. (b) Latency of keystroke localization.

TABLE I
POWER CONSUMPTION

	Power
Idle	39.2 ± 14.7 mW (CPU only)
Backlight	464.7 ± 6.7 mW (CPU + Screen)
Camera-on	3186.8 ± 12.6 mW (CPU+Screen+Camera)
DynaKey	4278.6 ± 25.9 mW (Total)

F. Latency and Power Consumption

We first evaluate the time delay of key tracking for a frame. Specifically, we compute the time duration from getting an input image to obtaining the updated coordinates of keys in 100 images. Fig. 18(a) shows a large variation in the distribution of latency. This may be due to the difference in the number of pixels involved in keypoint selection. However, the average time cost for tracking keys for one frame is 26 ms, which is smaller than the interframe duration, i.e., 33 ms. It indicates that our key tracking scheme satisfies the real-time requirement for real-world applications. To evaluate the latency of processing a keystroke, we conduct the experiment by randomly pressing 100 keystrokes, and calculate the time duration from detecting a possible keystroke to locating the pressed key. Fig. 18(b) shows that the mean latency is 63 ms, which is below human response time [23]. Overall, with low latency for key tracking and keystroke localization, DynaKey provides a real-time text input method for head-mounted devices.

To measure the power consumption of DynaKey on the Samsung Galaxy S9 smartphone, we use Battery Historian [12] by Google. For comparison, we measure the average power consumption in four different states: 1) idle, with the screen off; 2) idle, with the screen on; 3) keeping the camera on the preview mode; and 4) running our system for text input. As shown in Table I, by comparing the case of "Backlight," "Camera-on," and "DynaKey," more than 64% of power consumption comes from camera, which is essential for camera-based applications. The additional power consumption by DynaKey is 1092 mW, i.e., DynaKey consumes about 25% additional power. In the future, we will optimize the approach to further reduce the power consumption.

G. Evaluation on Text Input

To further evaluate the performance of DynaKey on text input, we invite twelve volunteers and instruct them to type

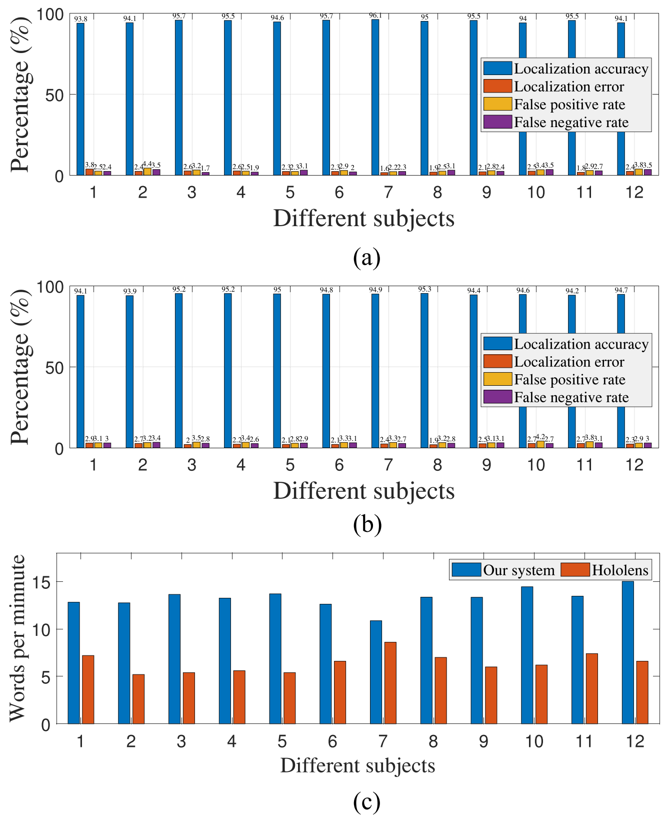


Fig. 19. Performance of different subjects. (a) Localization performance versus provided text input for different subjects. (b) Localization performance versus self-determined input for different subjects. (c) Input speed versus keyboard for different subjects.

on the virtual keyboard. The typed word sequences are picked from the top-5000 words of the word frequency corpus [32], which well represents the usage frequency of English words. Before evaluation, all subjects have 30-min trial runs. In an experiment, a subject types a set of predefined text and the self-determined text naturally. Each experiment lasts for 2 h with three breaks in due course.

1) Text Input: Each subject is first instructed to type a predefined words (1600 characters). Fig. 19(a) shows that DynaKey performs well, i.e., achieving 95.0% localization accuracy on average, while the localization accuracy of "subject 7" achieves 96%. The average localization error, false-positive rate, and false-negative rate are 2.3%, 2.9%, and 2.7%, respectively. They are then allowed to type a self-determined words (2000 characters). Fig. 19(b) shows that the average localization accuracy is 94.7%, and the average localization error, false-positive rate, and false-negative rate are 2.4%, 3.3%, and 2.9%, respectively. The performance is comparable to that of typing predefined text.

2) Input Speed: We also evaluate the input speeds of different subjects in terms of WPM (words per minute). Subjects are instructed to type the predefined characters with the proposed DynaKey system and Microsoft Hololens 1 [13], respectively. Fig. 19(c) shows that the average text input speed of DynaKey is 13.8 WPM while that of Hololens 1 is 6.4 WPM. Although the result of DynaKey is slower than that of typing on a physical keyboard [9], it still achieves 2X typing speedup compared to Hololens 1. In addition, typing with Hololens usually requires head movements for key selection, which may lead to the fatigue of head.

TABLE II
USER EXPERIENCE

Score	complexity	accuracy	latency	user friendliness
DynaKey	4.8	4.2	4.5	4.0

3) Difference With the State-of-the-Art Text Input Methods: In stationary scenarios, UbiK [28] and Camk [29] are typical state-of-the-art text input methods for mobile devices. In terms of text input performance, these two approaches and our DynaKey have comparable keystroke localization accuracy and time latency. However, in regard to working scenarios, UbiK fixes the device and utilizes the microphone to locate keystrokes. CamK fixes the embedded camera to capture the typing process. Both of the two methods are not suitable for text input in dynamic scenarios. Differently, our proposed DynaKey aims to move a step toward dynamic scenarios, where the camera/device can move naturally. The dynamic movement of device is natural and realistic for wearable devices, especially for head-mounted devices (e.g., smart glasses).

4) User Experience: Finally, we evaluate the user experience among 12 participants via questionnaire, including 1) technical complexity; 2) accuracy; 3) latency; and 4) user friendliness (1 = strong negative and 5 = strong positive). Table II shows the results. For the technical complexity and latency, most of the participants hold positive attitudes, because identifying the fingertip and small-size key are challenging. For the accuracy, some participants have a little negative evaluation, we will try our best to improve it in future. For the user friendliness, some participants have a negative evaluation for the adopted head-mounted equipment. Overall, as a new technology for text-input in dynamic scenes, our DynaKey can benefit a lot of head-mounted devices (e.g., smart glasses) and even work in many other human–computer interaction scenarios.

VI. Discussion

This section discusses the limitations of the work and points out our future direction.

Multiple Cameras: DynaKey uses one camera of a head-mounted device. Many off-the-shelf mobile devices have been equipped with more than one camera. With multiple cameras, we may obtain the depth information of a target, thus will significantly improve the keystroke localization performance. We will extend DynaKey with multiple cameras.

Virtual Keyboard: In DynaKey, we print the keyboard layout on a piece of paper to represent the virtual keyboard by default. In fact, DynaKey can also work with other virtual keyboards, e.g., keyboard drawn on the desk, as demonstrated in Section V-E5. The virtual keyboard allows the user to type on the common keyboard layout with two hands, even the mobile or wearable device has tiny or no screen. Besides, as a technique working with off-the-shelf devices in dynamic moving scene, DynaKey can complement to existing text input methods using dedicated equipments [11], [13], working in fixed scenarios [26], [29], etc. In our future work, we will investigate the use of a head-mounted virtual reality (VR) device to generate a true virtual keyboard layout, and further improve the design of DynaKey.

Keyboard Tracking: We have also considered placing four noncollinear markers outside the keyboard to track keys' coordinates, which may prevent the hand occlusion cases. However, we observe that small markers (e.g., diameter is close to the keyboard's line thickness) are difficult to be detected and are vulnerable to light conditions, while big makers (e.g., with a larger diameter) may confuse the selection of feature point (i.e., determine the position of feature point in a large area) and affect the calculation of transformation matrix. Therefore, we select cross points of lines in the keyboard to calculate the transformation matrix and track the keys' coordinates.

Keyboard Movement: DynaKey focuses on the scene where camera moves dynamically. In fact, when the keyboard moves, DynaKey can also work by relocating the keyboard with the detected cross points in Section IV-B2. Besides, when the user looks back or too close to the keyboard, DynaKey keeps sleeping because of no entire keyboard or fingers in the camera view, and restarts working once detecting the keyboard.

Possibility of Using Deep Learning-Based Methods: Recently, a lot of object tracking and semantic segmentation methods were proposed. Intuitively, these methods can be used to track keyboard and extract fingertips for keystroke detection and localization. However, due to the lack of large-scale training samples, the methods can not achieve pixel-level accuracy in keyboard tracking or fingertip extraction. Besides, these methods can hardly achieve real-time tracking or segmentation on resource-limited smartphone. More exploration about deep learning-based methods are expected.

VII. Conclusion

In this article, we propose DynaKey that deploys a head-mounted camera device to enable users type on a paper-based virtual keyboard in realistic and dynamic scenarios. DynaKey dynamically tracks keys, detects fingertips, and locates keystrokes. We implement DynaKey on Android devices and our experiment results show that DynaKey performs well in key tracking and keystroke localization and achieves low latency for text input.

References

[1] J. Gugenheimer, D. Dobbelstein, C. Winkler, G. Haas, and E. Rukzio, "FaceTouch: Enabling touch interaction in display fixed UIs for mobile virtual reality," in *Proc. 29th Annu. Symp. User Interface Softw. Technol.*, 2016, pp. 49–60.

[2] R. Xiao, J. Schwarz, N. Throm, A. D. Wilson, and H. Benko, "MRTouch: Adding touch input to head-mounted mixed reality," *IEEE Trans. Vis. Comput. Graphics*, vol. 24, no. 4, pp. 1653–1660, Apr. 2018.

[3] H. Abdelnasser, M. Youssef, and K. A. Harras, "MagBoard: Magnetic-based ubiquitous homomorphic off-the-shelf keyboard," in *Proc. 13th Annu. IEEE Int. Conf. Sens. Commun. Netw. (SECON)*, London, U.K., 2016, pp. 1–9.

[4] W. Chen, Y. Lian, L. Wang, R. Ruby, W. Hu, and K. Wu, "Virtual keyboard for wearable wristbands," in *Proc. 15th ACM Conf. Embedded Netw. Sens. Syst.*, 2017, pp. 1–2.

[5] E. Whitmire *et al.*, "DigiTouch: Reconfigurable thumb-to-finger input and text entry on head-mounted displays," *Proc. ACM Interact. Mobile Wearable Ubiquitous Technol.*, vol. 1, no. 3, pp. 1–21, 2017.

[6] C. Zhang *et al.*, "FingerSound: Recognizing unistroke thumb gestures using a ring," *Proc. ACM Interact. Mobile Wearable Ubiquitous Technol.*, vol. 1, no. 3, pp. 1–19, 2017.

[7] (2021). *Tap Strap*. [Online]. Available: https://www.tapwithus.com/

[8] K. Ali, A. X. Liu, W. Wang, and M. Shahzad, "Keystroke recognition using WiFi signals," in *Proc. 21st Annu. Int. Conf. Mobile Comput. Netw.*, 2015, pp. 90–102.

[9] (2020). *An Average Professional Typist Types Usually in Speeds of 50 to 80 WPM*. [Online]. Available: https://en.wikipedia.org/wiki/Words-per-minute

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

[11] M.-W. Chang, T.-C. Chiueh, and C.-M. Chang, "Virtual keyboard for head mounted display-based wearable devices," in *Proc. 20th IEEE Int. Conf. Parallel Distrib. Syst. (ICPADS)*, Hsinchu, Taiwan, 2014. pp. 225–232.

[12] (2017). *Battery Historian*. [Online]. Available: https://github.com/google/battery-historian

[13] (Jan. 2016). *Microsoft Hololens*. [Online]. Available: https://www.microsoft.com/en-us/hololens

[14] R. Horaud, B. Conio, O. Leboulleux, and B. Lacolle, "An analytic solution for the perspective 4-point problem," *Comput. Vis. Graph. Image Process.*, vol. 47, no. 1, pp. 33–44, 1989.

[15] J. Illingworth and J. Kittler, "A survey of the hough transform," *Comput. Vis. Graph. Image Process.*, vol. 44, no. 1, pp. 87–116, 1988.

[16] H. Kim, A. Byanjankar, Y. Liu, Y. Shu, and I. Shin, "UbiTap: Leveraging acoustic dispersion for ubiquitous touch interface on solid surfaces," in *Proc. 16th ACM Conf. Embedded Netw. Sens. Syst.*, 2018, pp. 211–223.

[17] (2020). *Keyboard Layout*. [Online]. Available: https://en.wikipedia.org/wiki/Keyboard-layout

[18] L. H. Lee, T. Braud, F. H. Bijarbooneh, and P. Hui, "TiPoint: Detecting fingertip for mid-air interaction on computational resource constrained smartglasses," in *Proc. 23rd Int. Symp. Wearable Comput.*, 2019, pp. 118–122.

[19] H. Li, W. Yang, J. Wang, Y. Xu, and L. Huang, "WiFinger: Talk to your smart devices with finger-grained gesture," in *Proc. ACM Int. Joint Conf. Pervasive Ubiquitous Comput.*, 2016, pp. 250–261.

[20] L. Lu *et al.*, "KeyListener: Inferring keystrokes on QWERTY keyboard of touch screen through acoustic signals," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, Paris, France, 2019, pp. 775–783.

[21] J. Mezirow, "Perspective transformation," *Adult Educ.*, vol. 28, no. 2, pp. 100–110, 1978.

[22] F. K. Noble, "Comparison of OpenCV's feature detectors and feature matchers," in *Proc. 23rd Int. Conf. Mechatronics Mach. Vis. Pract. (M2VIP)*, Nanjing, China, 2016, pp. 1–6.

[23] (2017). *Human Reponse Time Benchmark*. [Online]. Available: https://www.humanbenchmark.com/tests/reactiontime/statistics

[24] 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. CHI Extended Abstracts Hum. Factors Comput. Syst.*, 2003, pp. 712–713.

[25] A. Rosebrock. (2016). *Intersection Over Union (IoU) for Object Detection*. [Online]. Available: http://www.pyimagesearch.com/2016/11/07/intersection-overunion-iou-for-objectdetection

[26] K. Sun, W. Wang, A. X. Liu, and H. Dai, "Depth aware finger tapping on virtual displays," in *Proc. 16th Annu. Int. Conf. Mobile Syst. Appl. Serv.*, 2018, pp. 283–295.

[27] C. Tomasi, A. Rafii, and I. Torunoglu, "Full-size projection keyboard for handheld devices," *Commun. ACM*, vol. 46, no. 7, pp. 70–75, 2003.

[28] 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. 12th Annu. Int. Conf. Mobile Syst. Appl. Serv.*, 2014, pp. 14–27.

[29] Y. Yin, Q. Li, L. Xie, S. Yi, E. Novak, and S. Lu, "CamK: Camera-based keystroke detection and localization for small mobile devices," *IEEE Trans. Mobile Comput.*, vol. 17, no. 10, pp. 2236–2251, Oct. 2018.

[30] (2020). *Selfie Type by Samsung*. [Online]. Available: https://www.digitaltrends.com-/computing/samsung-selfie-type-invisible-keyboard-ces-2020/

[31] S. Benkerzaz, Y. Elmir, and A. Dennai, "A study on automatic speech recognition," *J. Inf. Technol. Rev.*, vol. 10, no. 3, pp. 77–85, Aug. 2019.

[32] (2020). *Word Frequency Corpus*. [Online]. Available: https://www.wordfrequency.info/samples.asp

[33] H. Zhang, Y. Yin, L. Xie, and S. Lu, "AirTyping: A mid-air typing scheme based on leap motion," in *Proc. ACM Int. Joint Conf. Pervasive Ubiquitous Comput. Int. Symp. Wearable Comput.*, 2020, pp. 168–171.

Hao Zhang (Student Member, IEEE) received the B.S. and M.S. degrees in computer science and technology from Nanjing University, Nanjing, China, in 2018 and 2021, respectively.

His research interests include mobile sensing and wearable computing.

Yafeng Yin (Member, IEEE) received the B.E. degree in network engineering from Nanjing University of Science and Technology, Nanjing, China, in 2011, and the Ph.D. degree in computer science from Nanjing University, Nanjing, in 2017.

She is currently an Associate Researcher with the Department of Computer Science and Technology, Nanjing University. Her research interests include human activity recognition, mobile sensing, and wearable computing.

Lei Xie (Member, IEEE) received the B.S. and Ph.D. degrees in computer science from Nanjing University, Nanjing, China, in 2004 and 2010, respectively.

He is currently a Professor with the Department of Computer Science and Technology, Nanjing University. He has published over 100 papers in IEEE/ACM TRANSACTIONS ON NETWORKING, IEEE TRANSACTIONS ON MOBILE COMPUTING, IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, *ACM Transactions on Sensor Networks*, ACM MobiCom, ACM UbiComp, ACM MobiHoc, IEEE INFOCOM, IEEE ICNP, and IEEE ICDCS.

Tao Gu (Senior Member, IEEE) received the bachelor's degree from Huazhong University of Science and Technology, Wuhan, China, in 1990, the M.Sc. degree from Nanyang Technological University, Singapore, in 2001, and the Ph.D. degree in computer science from the National University of Singapore, Singapore, in 2005.

He is currently a Professor with the Department of Computing, Macquarie University, Sydney, NSW, Australia. His research interests include mobile computing, ubiquitous computing, wireless sensor networks, sensor data analytics, and the Internet of Things.

Minghui You received the B.S. degree in computer science and technology from Nanjing University, Nanjing, China, in 2020, where he is currently pursuing the M.S. degree in computer science and technology with the Department of Computer Science and Technology.

His research interests include mobile sensing, and wearable computing.

Sanglu Lu (Member, IEEE) received the B.S., M.S., and Ph.D. degrees in computer science from Nanjing University, Nanjing, China, in 1992, 1995, and 1997, respectively.

She is currently a Professor with the Department of Computer Science and Technology, Nanjing University. Her research interests include distributed computing and pervasive computing.