Don't touch a word! A practical input eavesdropping attack against mobile touchscreen devices

Technical Report TR-2010-59, Politecnico di Milano

Federico Maggi, Alberto Volpatto
{fmaggi,volpatto}@elet.polimi.it
DEI, Politecnico di Milano

Simone Gasparini
simone.gasparini@gmail.com
INRIA Grenoble, Rhone-Alpes

Giacomo Boracchi, Stefano Zanero
{boracchi,zanero}@elet.polimi.it
DEI, Politecnico di Milano

***Abstract*—Spying on a person is a subtle, yet easy and reliable method to obtain sensitive information. Even if the victim is well protected from digital attacks, spying may be a viable option. In addition, the pervasiveness of mobile devices increases an attacker's opportunities to observe the victims while they are accessing or entering sensitive information. This risk is exacerbated by the remarkable user-friendliness of modern, mobile graphical interfaces, which, for example, display visual feedback to improve the user experience and make common tasks, *e.g.*, typing, more natural. Unfortunately, this turns into the well-known trade-off between usability and security.**

In this work, we focus on how usability of modern mobile interfaces may affect the users' privacy. In particular, we describe a practical eavesdropping attack, able to recognize the sequence of keystrokes from a low-resolution video, recorded while the victim is typing on a touchscreen. Our attack exploits the fact that modern virtual keyboards, as opposed to mechanical ones, often display magnified, virtual keys in predictable positions. To demonstrate the feasibility of this attack we implemented it against 2010's most popular smart-phone, *i.e.*, the iPhone. Our approach works under realistic conditions, because it tracks and rectifies the target screen according to the victim's natural movements, before performing the keystroke recognition. On real-world settings, our attack can automatically recognize up to 97.07% (91.03% on average) of the keystrokes, with a 1.15% error rate and a speed between 37 and 51 keystrokes per minute. This work confirms that touchscreen keyboards that magnify keys make automatic eavesdropping attacks easier than in classic mobile keyboards.

I. Introduction

A recent survey [1] on a sample of 2,252 individuals reports that, in 2010, 72% of the Americans uses a mobile phone to send or receive text messages, 38% accesses the Internet on mobile devices, and 30% chat on the go. Modern mobile devices rely on touchscreen technology, which has evolved from its humble beginnings (in 1970 [2]) into a $5 billion market product, now growing at a tremendous rate (around 159% [3]): 417 million mobile devices were sold worldwide in the third quarter of 2010 [4], touchscreen smart-phones (*i.e.*, iPhone and Android) are driving the sales. This success also stems from the continuous efforts in human-computer interaction research and industry, which devised intuitive, highly-usable interfaces, with full-fledged, on-screen keyboards and much more.

Unfortunately, since more and more people rely on mobile devices to work on the go, there is an increasing risk of inadvertently moving sensitive information outside the security perimeter of the workplaces. For instance, since people often connect to public wireless networks (*e.g.*, at cafes, airports), attackers may break into mobile devices by exploiting vulnerabilities exposed via the wireless interface. Highly-motivated attackers may simply spy on the victim, probably obtaining more information than from a break-in. For example, researchers have shown that automatic input eavesdropping is feasible and large portions of text can be extracted from a video of a user typing on a regular keyboard [5]. Fortunately, violating a user's privacy in such a way needs quite unrealistic assumptions (*e.g.*, a fixed cam pointed toward the keyboard), and works well against desktop-sized keyboards only (with large, mechanical keys that move vertically). On one hand, it is indeed very difficult to adapt this type of attacks to spy on traditional mobile phones (with small, tactile keys, safely hidden under the user's fingers), especially in dynamic and noisy conditions. On the other hand, modern touchscreen graphical interfaces provide visual hints to overcome their complete lack of tactile feedback, thus increasing the user experience in error-prone tasks such as typing on virtual keyboards. For example, the Apple iPhone's keyboard magnifies each pressed key above the user's finger.

The main motivation of this work is that usability enhancements may pose sensitive information at risk, by simply making it visible not only to the users, but also to shoulder surfers. These risks are clearly exacerbated by the immense popularity of (mobile) touchscreen devices, which are often adopted in publicly accessible areas, also to enter *sensitive* information (*e.g.*, a debit-card PIN at an ATM). These issues have risen the vendors' concerns and some countermeasures have been proposed (*e.g.*, a patented privacy-preserving tilting screen [6] with shallow viewing angle, dynamically adjusted according to the user's sight, spy-resistant touchscreen keyboards [7], or gaze-based passwords), which, unfortunately are not definitive nor suitable for mainstream mobile devices.

For the above motivations, we investigated thoroughly the feasibility of automating shoulder surfing attacks. Specifically, our goal was to show that by exploiting *only* the visual feedback provided by modern, touchscreen keyboards, keystrokes can be easily detected. To this end, we designed a completely automatic, black-box system that recognizes keystroke sequences in a fraction of the time needed to perform the same, tedious task by manually analyzing a video recoded while the victim is typing. Differently from the approach that inspired us [5], which is very difficult to adapt for mobile scenarios, our system

Note: This work was submitted for review to the PC of the IEEE Symposium on Security and Privacy in November 2010. A newer and improved version of this work is currently under review.

is very easy to implement, requires no unrealistic assumptions, and works in "natural" conditions.

Our attack relies on computer vision and image processing techniques and is divided into three sequential phases. *Phase 1* analyzes the input video searching for a (possibly) tilted, distorted, or rotated image of the screen. When a screen is detected its image is tracked along the subsequent frames, following the natural movement of the user or of the spying camera. Then, a geometrical transformation is estimated to rectify the image of the screen thus eliminating distortions such as rotations or perspective deformations. The resulting image is almost equivalent to an image taken as if the camera were placed on a tripod, just in front of the target screen. *Phase 2* subtracts the background (*i.e.*, an image of the virtual keyboard with no keys pressed) from each frame, to highlight the variations. These variations are either fingers, removed with appropriate image filtering techniques, or the visual feedback we want to capture. In *Phase 3*, the recognition phase, the center of each highlighted area is computed, and matched to the keyboard layout to determine the general area of the pressed key. Then, the templates of the letters neighboring the target region are exploited to find the best-matching areas, thus recognizing the keystrokes (if any).

To the best of our knowledge, we are the first to study the concrete risks brought forth by mobile touchscreen keyboards and provide a practical attack that works against mainstream devices. Even though we demonstrate our attack on a specific device (*i.e.*, the iPhone) and layout (*i.e.*, QWERTY), its generality depends upon very simple requirements and thus can be extended with minor modifications.

In our evaluation, the system we implemented recognizes up to 97.07% (91.03% on average) of the keystrokes typed, with a 1.15% error rate, at a speed between 37 and 51 keystrokes per minute, generally faster than tedious manual analysis. This proves that our method can successfully recover keystroke sequences by simply relying on the feedback displayed. If even more precision is needed, syntax or grammar corrections tools can be easily attached as a post-processing step, but this would not represent a novel contribution.

The contributions of this paper are summarized as follows:

- in Section II we discuss the privacy issues brought forth by user-friendly mobile devices. In our opinion, this is a real-world example of the well-known trade off between usability and security [8];
- in Section III we propose a technique for accurately recognizing keystrokes from a video taken while a user is typing on a mobile touchscreen device;
- in Section IV we detail the criteria we designed to precisely select the keys based on the output produced by leveraging computer vision and imaging algorithms.
- in Section V we discuss the results of three experiments that show that our attack is feasible and, in realistic conditions, we achieve remarkably high precision.

We conclude by discussing the limitations of our approach, which are measurable and precisely described.

II. Eavesdropping touchscreen keyboards

Eavesdropping is the (commonly unethical) practice of secretly listening to a communication, with the goal of stealing sensitive information. In some situations, eavesdropping may be performed simply by observation (*e.g.*, of a computer screen). The most notable, low-tech example is known as *shoulder surfing*, *i.e.*, the practice of looking over a victim's shoulder while (s)he is visualizing or typing the target information. Shoulder surfing is simple and effective, in particular to steal someone's PINs (*e.g.*, at ATMs) or passwords (*e.g.*, at a public cafe which offers Internet access). However, when the target information is ample, *e.g.*, long e-mails or chat conversations, real-time shoulder surfing becomes quite tedious if not unfeasible. In fact, the attacker would need to observe the user for extended periods of time, and possibly leverage recording mechanisms (*e.g.*, photographs or video-taping) to overcome the limitations of short-term human memory, and then analyze these materials offline to obtain the target information — which, in the meanwhile, may have lost its value already.

Eavesdropping can also be performed through several types of malware, such as trojan horses that record what a user is typing, and subsequently send this data to the attacker. This can be easily perpetrated remotely by leveraging the huge malware arsenal available today [9]. In these cases attackers are *not* required to be in the same physical environment of the victim.

In what follows, we explain how the remarkable and improved usability of mobile touchscreen keyboards with the aforesaid feedback mechanisms comes at the cost of making direct eavesdropping stealthier, easier and time-efficient. In fact, to make a system more usable for end users (who are not supposed to be security experts), some security requirements are relaxed.

A. Usability and privacy in mobile devices

A crucial factor that favored the spreading of mobile touchscreen devices was certainly the improved interfaces, in particular the ones that give some form of feedback to the users while they type. For example, some BlackBerry's touchscreen keyboards vibrate each time a key is pressed, giving tactile feedback to the user. But the most notable example is the iPhone and Android keyboard. As exemplified in Figure 1, both devices "assist" the user with an effective visual feedback.

As explained in the remainder of this section, previous research demonstrated that, even in absence of such visual feedback, it is possible to keep track of the victim's fingers and reconstruct large portions of texts from a video. However, according to our preliminary study described in the following section, it is very difficult even for a human to perform the same task on mobile non-touchscreen keyboards, which do not expose such details as their small keys are safely covered by the victim's fingers.

B. Preliminary study: Spying on classic mobile keyboards

The goal of this qualitative study is to support the intuition that classic mobile keyboards are inherently less "privacy-leaking" than touchscreen ones, because shoulder surfing is harder to perform, even with the aid of a video camera. To do this, we recorded a video depicting a BlackBerry keyboard while a hypothetical victim is typing on it (1) a long English

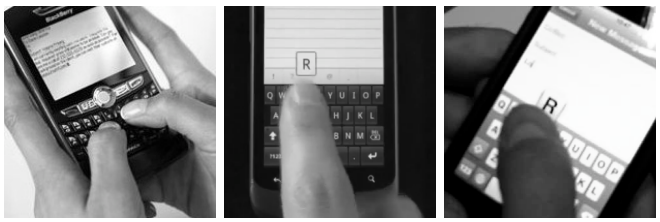


Figure 1: Different mobile QWERTY keyboards. Classic, BlackBerry-like devices feature tactile or mechanical keyboards with small keys, while iPhone-like devices feature touchscreen, feedback-rich keyboards.

TIME	LENGTH	ATTEMPT 1	ATTEMPT 2	ATTEMPT 3
0'23"	*35*	*100%, 7'00"*	*12%, 30'00"*	*0%, -*
4'22"	**444**	**0%, -**	**0%, -**	**1.35%, -**

Table I: Attempts to manually recognize a context-free and a brief text typed on a BlackBerry keyboard. Hit rate and error rate are reported along with the time required for recognition. The time for typing each text and its length are reported too. The sign "-" indicates that the volunteer gave up for excessive fatigue.

text with no linguistic context[1], and (2) a brief text, *i.e.*, *"Hello world, how are you today? I am very fine, and you?"* (41 letters plus spaces and symbols, total 12 words).

Then we asked six volunteers to analyze the videos offline, giving them "unlimited" time and the possibility of stopping, slowing-down and restarting the videos as needed. Only one volunteer was able to recognize, initially, some bits of the brief text. Most of the keystrokes were not actually visible, and thus the volunteer resorted to the rich linguistic context (*e.g.*, *"Hello w..."* is likely to be *"Hello world"*) as the only chance to reconstruct also the whole text (yet this took 7 minutes, for just 23 seconds of video). Besides this exception, as summarized in Table I, even with the help of a recorded video, a shoulder surfer would be able to recognize only small portions (or none) of brief texts. None of the volunteers was practically able to reconstruct the longer, context-free text.

C. Related work

Previous research focused on eavesdropping personal computer's mechanical keyboards, and since our goal is to automatically recognize what a victim is typing using a video as the only information source, the closest work in previous literature is [5]. It proposes a sophisticated attack with unprecedented precision, which searches the hands' contour and spots occlusions of the key-caps to identify keystrokes. An advanced grammar analyzer then corrects the quite large amount of detection errors. Despite its accuracy, such attack assumes that camera and keyboard are aligned in a fixed relative position. While the latter assumption is realistic (at least for the non-mobile world), the former is not, especially because the authors mention that the webcam — previously exploited to take the video — has to be aimed at the keyboard, a really uncommon setup.

[1]the same context-free text utilized in our experimental evaluation, described in Section V

Our attack makes no assumptions on the relative positions of the target device and the camera; it even allows for victim movements (not sudden ones, but most natural ones). Additionally, the recording can be done with a handheld, low-end camera. Clearly, as discussed thoroughly in Section VI, our attack needs the mobile device to be visible to some extent, *i.e.*, the screen may be partially occluded by hands or other objects but yet there is a minimum amount of the screen that can be recognized. Additionally, as the attack precision is superior, we do not strictly require post-processing to clean the output. Therefore we can detect with high accuracy also keystrokes that do not belong in a dictionary or have no context. Also, it must be noted that, instead of exploiting large keys, which move vertically, our attack exploits the visual feedback provided by virtual keyboards (an insightful example is shown in Figure 1).

D. Shoulder surfing mitigation mechanisms

To prevent information leakage from mobile touchscreen devices, two strategies have been proposed. The first one consists in reducing the viewing angle of the screen, thus limiting the chances for an attacker to see what a victim is typing. The second strategy consists in designing the touchscreen interfaces in such a way that users are forced to input sensitive information in a secure way.

A notable example of the first type of strategy is a technology recently patented by HTC [6]. The principle is very simple: a wide viewing angle allows shoulder surfers to easily read what a user is seeing. Thus, the patent proposes screens with a very shallow viewing angle (*i.e.*, 30 degrees *vs*. 130 degrees, which is the human eye's viewing angle). A similar idea is implemented in the so called "privacy screen filters"[2]. As a consequence, users would need to re-adjust the screen position continuously as they move to keep it aligned to their sight. HTC proposes to solve this issue by orienting the screen dynamically, according to the position of the user's eyes estimated from gyroscopes and front-facing cameras. Although this mechanism is not yet on the general market, it may limit the chances for a shoulder surfer to take a usable video of a user while typing.

The method described in [7] falls in the second type of protection strategies. The authors propose a methodology for designing secure touchscreen interfaces. In particular, they focus on virtual keyboards and, more specifically, those used to enter sensitive information such as PINs on public terminals (*e.g.*, automated teller machines), which typically contain a limited character set (*e.g.*, numbers or letters only). For this reason, it is feasible for a human to map their credentials onto a different, temporary, mnemonic alphabet (*e.g.*, colors or simple shapes). This mapping is dynamically chosen by users *before* each authentication session. Then, credentials are entered using the new, temporary alphabet. This approach mitigates casual shoulder surfers, but the authors explicitly mention that no protection is guaranteed against attackers armed with video cameras, who can record, rewind, and review the entire interaction. Therefore, the general approach we propose cannot be effectively mitigated by this type of techniques, unless the interaction involves feedback that is hard to track.

[2]*e.g.*, http://www.case-mate.com/Privacy-Screens/Case-Mate-iPhone-3G-Full-Face-Privacy-Screen.asp

A different example of the second type of mitigation strategy is described in [10], which consists in tracking the user's pupil movements and map them onto a grid layout to implement a gaze-based keyboard. It must be noticed, however, that the proposed threat model is quite unrealistic, because it assumes that a motivated attacker would not be able to hide a micro-camera on public terminals, while the high success of card skimming [11] shows that devices that typically comprise a micro-camera and a skimmer are routinely hidden into ATMs. In addition, these countermeasures are not suitable for mobile devices conditions, because the quality of the eye-tracking decrease in mobile scenarios.

Given the state of the art of the research in computer vision applied to gesture recognition and provided that no definitive protection solutions exist, we conclude that the extent to which usability of mobile touchscreen keyboards affects the users' privacy needs to be assessed thoroughly. To this end, in what follows we demonstrate that a malicious attacker can efficiently violate a user's privacy by automatically eavesdropping keystroke sequences by exploiting the visual feedback displayed by modern, mobile touchscreen devices.

III. Exploiting highly-usable touchscreens

In this section we describe a practical attack against touchscreen interfaces. Before describing the algorithm, its threat model and assumptions are detailed.

Threat model and attack requirements

In our threat model the attacker is allowed only to record a video, at any viewing angle, of the target device while the victim types on it. No remote nor local access to the device is required. The reconstruction of the text is based *solely* on the feedback displayed on the screen, and no visibility of the typed text is assumed. For example, the victim may type text on a "password" input field, where the letters are replaced with asterisks or dots. Our attack requires no high-end cameras, tripods, nor any other special equipment. For example, an hypothetical attacker may just stand behind the victim (*e.g.*, waiting at a bus station) and point a camera (possibly, a very small one embedded in a smartphone) towards the victim's device.

Our attack is extremely general, depending *exclusively* on the following simple and realistic requirements:

Requirement 1: the target virtual keyboard must display feedback whenever a key is pressed. From now on, we refer to such feedback as the *magnified key*. Magnified keys must be partially visible (at least in one frame after each keystroke). The attack works even if fingers partially cover the magnified keys, as this typically happens while typing. iPhone and Android devices, the two most popular mobile touchscreen phones, both display magnified keys.

Requirement 2: the attacker must know the model of the target device, in order to compute the following static information a priori:

- *screen template*, a screenshot or a photograph of the target device and application used by the victim, including the most significant parts of the target screen (*i.e.*, the virtual keyboard);
- *key template*, the appearance (*i.e.*, sizes and font family or symbol set) of each magnified key;
- *magnified layout*, a set containing the coordinates of the central pixels of the magnified keys. In what follows it is represented by $ML = \{c_1, \cdots, c_L\}$; note that these central pixels can be easily mapped onto a regular grid. For example, in the US English keyboard (Figure 2d), the magnified layout contains the coordinates of $L = 26$ magnified keys, and c_1 is the coordinate of the central pixel of the magnified key letter 'Q', while c_2 corresponds to 'W'.

For example, an attacker who wants to eavesdrop the input from iPhone devices just needs to buy (or borrow) an iPhone, install the target application(s) (*e.g.*, Mail, Twitter) and take a screenshot (or a cropped photograph with a camera on a tripod, and the iPhone screen parallel to the camera's sensor). In many cases, one may conveniently search significant screenshots via the Web. The attacker then can easily build the grid of the magnified layout by merely measuring the distance in pixels between each magnified key's barycenter. The key templates can be automatically generated with a scriptable typesetting tool.

Our proof-of-concept implementation (detailed in Section IV and thoroughly evaluated as described in Section V) has been tested (using a low-end handheld camera) against the Apple iPhone, yet, in principle, it can be adapted to capture keystrokes from different devices, provided that they adopt a similar visual feedback mechanism.

System overview

The screen template and the key templates are both static data, *i.e.*, computed offline only once. The *actual input* of our system is a video of the victim typing on a touchscreen keyboard. This is processed frame by frame as follows:

Phase 1: each frame is analyzed to detect the device screen, by using a *feature-based template-matching* method against the screen template. When the template successfully matches the device in the current frame, the screen area of the device is selected and rectified. A successful match is used to improve matches in the next frame(s).

Output: a rectified image of the current frame containing only the device screen. This image is similar to the image that a camera set on a tripod would acquire when the device is at a fixed distance, with the screen parallel to the camera's sensor.

Phase 2: the magnified-key candidates are identified as high-contrast areas of the rectified image that are different from the template and the previous frames. The core technique used in this phase is *background subtraction*.

Output: a segmented image (*i.e.*, a map of the image areas) identifying the magnified-key candidates. Typically, there is more than one magnified key candidate per frame.

Phase 3: each magnified-key candidate is validated by direct comparison with the corresponding template of the magnified key, thus identifying the best-matching key.

Output: the recognized, typed symbol.

This workflow can be described by means of its intermediate outputs. As exemplified in Figure 2, the acquired frame in (a)

(a) Input of Phase 1. (b) Input of Phase 2. (c) Input of Phase 3. (d) Output (*i.e.*, the 'R' key).

Figure 2: Intermediate outputs on a sample capture. Phase 1: the device screen is detected in each frame $\mathtt{I}_t$ (a), cropped and rectified, yielding $\mathtt{Z}_t$ (b). Phase 2: the magnified-key candidates are selected within the foreground, *i.e.*, the image areas shown in (c). Phase 3: according to the coordinates of the magnified layout *ML* (d), each candidate is compared to its template to identify the typed key. The template of 'R' is selected as it shows the best match.

is rectified obtaining the frame in (b), which shows several differences with respect to the screen template and the previous frames. In particular, such differences are not limited to the 'R' magnified key only, and three magnified-key candidates are identified by Phase 2, as shown in (c). The validation of Phase 3 consist in comparing each candidate with the corresponding template. The best match is given by the candidate magnified-key containing the 'R', because an area that is very similar to its corresponding template is identified in the frame. Thus the typed symbol is successfully recognized. Note that, some frames can be "empty", *i.e.*, with no keystrokes. Our algorithm handles this case as detailed in Section III-C, IV-C.

Notation From hereinafter we consider grayscale images. An image $\mathtt{I}$ is a matrix of real values in $[0,1]$, and $\mathtt{I}(x,y)$ indicates the intensity of its pixel at coordinate (x,y). Images are often frames of a video sequence: in these cases, we use the subindex t to indicate the frame at time t.

We indicate with $\mathtt{I}_t$ and $\mathtt{Z}_t$ the acquired frame and the corresponding rectified screen at time t, respectively. Image rectification encloses image resizing and scaling, and yields $\mathtt{Z}_t$ that has the same size of the screen template, and minimum and maximum value set to 0 and 1.To ease the notation, we use a vector to indicate the 2D pixel coordinates $\mathbf{x} = (x,y)$ and, where not specified, we assume that $\mathbf{x}$ belongs to the domain of $\mathtt{Z}_t$.

A. Phase 1: Screen detection and rectification

This phase is divided into two sub-tasks executed in cascade: *screen detection*, that searches for any occurrence of the screen in the input video, and *image rectification*, which estimates the perspective deformation of the detected screen and rectifies its image. Both methods rely on feature extraction and matching: an image feature is a small image patch centered on a peculiar point of the image, usually where the image presents a discontinuity, *e.g.*, a corner or an edge. Given two images and their features, the features can be matched in order to find image correspondences, *i.e.*, two features representing the same object in the scene. In our work we use the so-called SURF features [12], which are invariant to rotation, scaling and skew transformations.

For the sake of clarity, we first explain the *rectification* task and then the *detection* algorithm.

Image rectification: Since the spying camera is looking at the device from a skewed position, the resulting image of the screen is perspectively distorted: typically, the rectangular shape of the screen is imaged as a trapezoid. This effect can be corrected by generating a (synthetic) rectified image that preserves the screen's geometry. In general, the distorted image of a planar surface is related to its rectified version by a linear transformation $\mathtt{H}$ called *homography* [13]. The homography maps corresponding points of the two images according to the following equation:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \sim \mathtt{H} \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix}, \tag{1}$$

where (x',y') and (x,y) are the image coordinates of the points of the acquired images before and after rectification, respectively. 3×3 matrix $\mathtt{H}$ represents the homography relating the two images: it is a full rank matrix (hence the relation is invertible) and it is defined up to scale, *i.e.*, it has 9 elements but only 8 of them are independent [14], and they can be estimated from the distorted and rectified images using a minimum of 4 corresponding points on the two images. This is implemented via Direct Linear Transformation as detailed in Section IV-A.

In our case, since there are many invariant parts on the screen, *e.g.*, the keyboard and other graphical elements, we use the screen template as a reference rectified image, and we exploit the common parts to find corresponding points and estimate, at any time t, the matrix $\mathtt{H}_t$. Therefore, at any time step t, the rectified image $\mathtt{Z}_t$ can be obtained by applying the estimated $\mathtt{H}_t$ to each pixel belonging to the device screen in the distorted image $\mathtt{I}_t$:

$$\mathtt{Z}_t(x,y) = \mathtt{I}_t(x',y'), \tag{2}$$

where (x,y) and (x',y') are related by (1). The rectified image $\mathtt{Z}_t$ contains only the device screen, and has the same size of the screen template (thanks to image interpolation). Finally, $\mathtt{Z}_t$ is scaled to guarantee that the darkest area correspond to 0 and the lightest to 1; in such a way $\mathtt{Z}_t$ can be easily compared with the screen template.

Screen Detection: This is a challenging task because the degree of (perspective) distortion and the position of the screen in the frame can vary as the camera moves. Therefore, the whole

frame must be searched for the screen image. Also, the screen can be (dynamically) occluded by fingers or other objects a priori unknown.

For these reasons, we use a feature-based template matching algorithm [15]. The SURF features of the template are matched with the features of the current frame, in order to find corresponding points and detect the region of the image where the screen appears. In order to have a reliable detection, false correspondences must be ruled out: indeed some features may be mismatched if the corresponding patch is similar but belonging to different objects in the scene. Therefore, we exploit the additional constraint provided by the homography relating the template and the current frame: all the correspondences are used to estimate the homography $\mathtt{H}_t$ in a RANSAC [16] process, which allows to discriminate inliers and outliers, *i.e.*, good and false corresponding points: if the number of inliers is sufficiently larger than the number of outliers, then the screen is considered detected and the estimated homography is used to rectify the image of the screen. Otherwise, no screen is detected and the frame is discarded.

This approach is faster than a pixel-wise comparison of the two images and it can be easily extended to any other device just by using the proper template image.

B. Phase 2: Magnified Keys Detection

Magnified keys are dynamic elements of the rectified frame sequence, thus, they can be effectively detected with *background subtraction* techniques [17], often adopted, for instance, to identify intruders in videos taken from surveillance cameras.

1) Background Subtraction: An estimate of the *background model* that describes the depicted scene in stationary conditions is needed. Each frame is compared with said background model to identify possible dynamic objects, *i.e.*, the scene's *foreground*. More formally, the background model is composed of the background image $\mathtt{B}_0(\mathbf{x})$, typically estimated as the pixel-wise average of a training sequence that contains no foreground elements, and by $\Sigma_0(\mathbf{x})$, an estimate of the pixel-wise standard deviation of the training sequence. For the sake of clarity, we assume here that a short sequence of frames depicting the device before the victim starts typing is available, and can be used as a training sequence to estimate $\mathtt{B}_0$ and Σ_0, while in Section IV-A we detail how our implementation estimates them when no training sequence is available.

Let $\mathtt{Z}_t$ be the rectified frame at time t, containing the screen area. According to one of the most straightforward background-subtraction technique [17], $\mathtt{F}_t$, the foreground at time t, is

$$\mathtt{F}_t(\mathbf{x}) = \begin{cases} 1, & \text{if } |\mathtt{Z}_t(\mathbf{x}) - \mathtt{B}_{t-1}(\mathbf{x})| > k\,\Sigma_{t-1}(\mathbf{x}) \\ 0, & \text{otherwise} \end{cases}, \tag{3}$$

where $k > 0$ is a tuning parameter, $\mathtt{B}_{t-1}$ and Σ_{t-1} are the estimates, at $t-1$, of the background image and its standard deviation, respectively. These are recursively computed as follows:

$$\mathtt{B}_t(\mathbf{x}) = \begin{cases} \mathtt{B}_{t-1}(\mathbf{x}), & \text{if } \mathtt{F}_t(\mathbf{x}) \neq 0 \\ \alpha\,\mathtt{F}_{t-1}(\mathbf{x}) + (1-\alpha)\,\mathtt{B}_{t-1}(\mathbf{x}), & \text{otherwise} \end{cases}, \tag{4}$$

and

$$\Sigma_t(\mathbf{x}) = \begin{cases} \Sigma_{t-1}(\mathbf{x}), & \text{if } \mathtt{F}_t(\mathbf{x}) \neq 0 \\ \sqrt{\alpha(\mathtt{F}_t(\mathbf{x}) - \mathtt{B}_t(\mathbf{x}))^2 + (1-\alpha)\,\Sigma^2_{t-1}(\mathbf{x})}, & \text{otherwise} \end{cases}, \tag{5}$$

where $\alpha \in [0,1]$ is an update parameter. The update relies on the assumption that each pixel in the background image is distributed according to a Gaussian, whose mean and standard deviation are updated at time t when the pixel does not belong to the foreground.

2) Magnified keys identification: Foreground $\mathtt{F}_t$ discloses parts of $\mathtt{Z}_t$ that have changed with respect to the background $\mathtt{B}_t$: such changes can be due to occlusions (most probably typing fingers), light changes, rectification errors, and magnified keys. In order to disambiguate magnified keys, we exploit the following priors:

- key magnification lasts for few frames, and typically less than other occlusions. Thus, the short-term foreground $\mathtt{S}_t$ has been introduced to highlight image parts that have recently changed:

$$x\mathtt{S}_t(\mathbf{x}) = \begin{cases} 1, & \text{if } |\mathtt{F}_t(\mathbf{x}) - \overline{[\mathtt{F}_t(\mathbf{x})]}_n| > 0 \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

  with $\overline{[\mathtt{F}_t(\mathbf{x})]}_n = \frac{1}{n}\sum_{i=1}^{n} \mathtt{F}_{t-i}(\mathbf{x})$, and $n \in \mathbb{N}$ corresponds to the minimum number of frames a magnification lasts.
- Magnified keys (black characters over a white key area) are characterized by higher contrast than other occlusions and the background. These provide a high response when $\mathtt{Z}_t$ is processed by high-pass filters: therefore we compute the gradient $\mathtt{G}_t$ and the Laplacian $\mathtt{L}_t$ magnitudes by means of convolutional filters:

$$\mathtt{G}_t(\mathbf{x}) = [(\mathtt{Z}_t \circledast g_x)(\mathbf{x})]^2 + [(\mathtt{Z}_t \circledast g_y)(\mathbf{x})]^2, \tag{7}$$

  and

$$\mathtt{L}_t(\mathbf{x}) = \left[\left(\mathtt{Z}_t \circledast g_x^2\right)(\mathbf{x})\right]^2 + \left[\left(\mathtt{Z}_t \circledast g_y^2\right)(\mathbf{x})\right]^2, \tag{8}$$

  where g_x and g_y denote the first-order derivative filters, g_x^2 and g_y^2 the second-order ones, and $\circledast$ the discrete convolution.

The average of short-term foreground and high-frequency information is $\mathtt{M}_t(\mathbf{x})$, and provides a heuristic measure to indicate of how likely the foreground contains a magnified key in each pixel:

$$\mathtt{M}_t(\mathbf{x}) = \begin{cases} \frac{1}{3}\left(\frac{\mathtt{G}_t(\mathbf{x})}{\max \mathtt{G}_t(\mathbf{x})} + \mathtt{S}_t(\mathbf{x}) + \frac{\mathtt{L}_t(\mathbf{x})}{\max \mathtt{L}_t(\mathbf{x})}\right), & \text{if } \mathtt{F}_t(\mathbf{x}) \neq 0 \\ 0, & \text{otherwise.} \end{cases} \tag{9}$$

Note that, $\mathtt{M}_t(\mathbf{x}) \in [0,1]$, as the terms in (9) are in the range $[0,1]$. Also note that, $\mathtt{G}_t(\mathbf{x})$ and $\mathtt{L}_t(\mathbf{x})$ in (7) and (8) are taken into account only in the foreground pixels (*i.e.*, where $\mathtt{F}_t(\mathbf{x}) \neq 0$).

3) Segmentation: Values of $\mathtt{M}_t$ can be considered as generated by two classes: (1) the magnified keys, which yields high values, and (2) other foreground elements (*e.g.*, fingers, other occlusions, screen displacements due to rectification errors) which indeed results in low values. Thresholding is then a viable solution to distinguish magnified keys: the threshold $\Gamma > 0$ can

be determined by the Otsu method [18], which is widely-used in image binarization. The thresholded image

$$\mathtt{K}_t = \begin{cases} 1, & \text{if } \mathtt{M}_t(\mathbf{x}) > \Gamma \\ 0, & \text{otherwise} \end{cases} \tag{10}$$

is non-zero in regions containing a possible magnified key, according to the measure $\mathtt{M}_t$. The binary image $\mathtt{K}_t$ is then segmented, using conventional morphological image processing techniques [19], to identify its connected components (blobs). Each blob is indeed a set of pixels coordinates and Figure 2 (c) shows the image values in these blob's areas. In what follows, the set of blobs is denoted by $\mathcal{B}_t$.

C. Phase 3: Validation via template matching

The third phase identifies the magnified key that has been pressed, if any: such key is selected by analyzing the blobs in $\mathcal{B}_t$. Each blob must be validated because Phase 1-2 may introduce errors or spurious objects (*e.g.*, a finger's contour) that do not correspond to a magnified key. Validation is performed exploiting the magnified layout, *ML*, and the key templates, both defined in Requirement 2.

Each blob $b \in \mathcal{B}_t$, yields one or more magnified-key candidates, which belong to the magnified layout *ML*. Section IV-C2 describes simple yet effective criteria to select magnified key candidates for each blob. Let $\mathcal{C}_t \subset ML$ be the set of the magnified-key candidates identified by all the blobs in $\mathcal{B}_t$. The *best-matching key*, at time step t, is denoted by $c_t^\star$ and is looked up $\mathcal{C}_t$ by maximizing the *key similarity*. As detailed in Section IV-C4, the key similarity measures the degree of matching between key templates and *Regions of Interest* (ROI), *i.e.*, squared crops of $\mathtt{Z}_t$. Figure 3(a) and (c) give an insightful example of a template and a ROI, respectively.

D. Keystroke Sequence Recognition

The phases 1-3 determine $c_t^\star$, the best-matching key *i.e.*, the magnified key that has been most likely pressed in frame t. Nevertheless, keystroke sequence recognition is not straightforward as key magnifications typically last longer than one frame, and there are frames that do not contain magnified keys. These issues are addressed by analyzing how the key similarity of $c_t^\star$ varies with t. Section IV-C details a simple yet effective mechanism to carry out such analysis.

IV. System details

In this section we focus on the details of the system overviewed in Section III, and in particular on the key-recognition criteria. Phase 1 has been entirely implemented in C++ with the OpenCV library [20] (the same used in [5]), while Phase 2-3 have been implemented in MATLAB, using the image processing toolbox functions, and then compiled in C++ for integration with Phase 1.

A. Phase 1: Screen detection and rectification

The state-of-the-art algorithms required by this phase are already implemented in the OpenCV library. More precisely:

- the SURF features are extracted by the `cvExtractSURF` function set.
- The homography estimation is performed using the function `cvHomography`, which implements the classical Direct Linear Transformation approach [21], embedded in a RANSAC [16] framework to improve robustness.
- The function `cvWarpPerspective` applies the homography $\mathtt{H}_t$ to the source image and generate the rectified image.

B. Phase 2: Magnified key detection

The background subtraction described in Section III-B requires a short video sequence (about 2 seconds) capturing the device when the victim is *not* typing, to estimate $\mathtt{B}_0$ and Σ_0. In practical applications this assumption is not always met. Therefore, we tested our system in these challenging cases and showed that $\mathtt{B}_0$ and Σ_0 can be successfully estimated from the screen template taken offline. Thus, $\mathtt{B}_0$ is initialized with the screen template, and $\Sigma_0(\mathbf{x}) \sim \sigma \ \forall \mathbf{x}$, where σ is the standard deviation of the image noise that is estimated from $\mathtt{Z}_1$ (the first rectified frame) using the technique proposed in [22]. Although these estimates are quite a naïve approximation, the update process of (4) and (5) guarantees satisfactory recognition performance (as shown in Section V).

For magnified key identification, Sobel filters [19] were used in our implementation for (7) and (8).

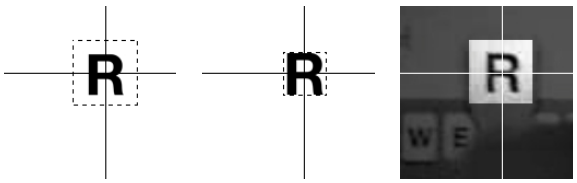
C. Phase 3: Validation via template matching

As described in Section III-C, the input of this phase is the set of blobs $\mathcal{B}_t$ at time t, while the output is the best matching key symbol $c_t^\star$, and its associated key-similarity measure $\Phi_t(c_t^\star)$.

To make the whole validation more robust we need to compensate possible errors of Phase 2 (typically, blobs displaced with respect to the magnified key), Thus, as stated in Section III-C, each blob yields a *neighborhood* of candidates. This, unfortunately, has the side effect of slowing down the computation. To alleviate this, the best-matching key is looked up within those candidates that have a *percentage of black and white pixels* similar to their templates.

1) Key templates and Region Of Interest: For each key $c \in ML$, we define the *full key template*, $T^f(c)$, the *cropped key template*, $T^r(c)$, and the key *Region of Interest*, $ROI(c)$, as exemplified in Figure 3.

The key templates are the a-priori models of each magnified key. Our system automatically generates them offline with a scriptable typesetting tool (*e.g.*, Adobe Illustrator or LaTeX) to



(a) Full key template $T^f(c_4)$ (b) Cropped key template $T^r(c_4)$ (c) Region of Interest $ROI(c_4)$

Figure 3: The full key template, (a), used for computing $\mathrm{d_{bw}}$, defined in (13), and the cropped key template, (b) used for computing ncc, defined in (15), both performed with respect to the ROI, (c), for a given key that, in this example, corresponds to 'R'.

render each symbol according as it appears in ideal conditions on the device. In our implementation, both full and cropped key templates are squared images: for the iPhone, $T^f(c)$ includes the full white background of the magnified key, while $T^r(c)$ is cropped to $2/3$ of the area of $T^f(c)$ (see Figure 3a-b). As highlighted in Figure 3c, the $ROI(c)$ is the area of the rectified frame where the magnification of the key corresponding to c is expected. More formally, $ROI(c) \subset \mathtt{Z}_t, \forall t$ is an image area centered in c, of the same size of its full template $T^f(c)$.

2) Key neighborhoods: For each blob $b_i \in \mathcal{B}_t$ the barycenter $\dot{b}_i$ is computed and the closest key, c_i, of the magnified layout, ML, is computed as

$$c_i = \text{argmin}_{c \in ML} \, \mathrm{d}(c, \dot{b}_i), \tag{11}$$

where $\mathrm{d}(c, \dot{b}) = \|c - \dot{b}\|$ is the Euclidean distance. Then, for each c_i, we define its *neighborhood* $\mathcal{N}(c_i)$ as the set composed by c_i and the coordinates of adjacent keys. Specifically, in our implementation $\mathcal{N}(c_i) = \{c_i^l, c_i, c_i^r\}$ is used, where labels 'l' and 'r' indicate the magnified keys at the left and the right of c_i, respectively. The neighborhood of 'E' is exemplified in Figure 4. Note that, c^l, c^r do not exist for certain keys. More precisely, 'Q', 'A', 'Z' have no corresponding left neighbor, while 'P', 'L', 'M' have no right neighbor. Thus $\mathcal{N}(c), \forall c$ contains either 2 or 3 elements.

Then we define the set of *candidate keys* as

$$\mathcal{C}_t = \bigcup_{i=1}^{|\mathcal{B}_t|} \mathcal{N}(c_i), \tag{12}$$

where c_i is defined as in (11). Experimentally, we observed that key validation yields better results when the neighbors of each magnified key are considered.

3) Black and white percentage of pixels: The key similarity $\Phi_t(c)$ is a computationally expensive measure, as it involves several pixel-by-pixel comparisons. For this reason, it is evaluated only when $ROI(c)$ is likely to contain the corresponding magnified key (as opposed to, *e.g.*, a finger, occlusion, or spurious blob). For example, in Figure 4, some candidate keys (*e.g.*, c_2^l, c_2^r) can be safely discarded as they are not resemble any possible magnified key. A simple yet effective criteria to discard these blobs is to compare, for each $c \in \mathcal{C}_t$, the *percentage of black and white pixels* in $ROI(c)$ and in the corresponding *full* template, $T^f(c)$. When they are similar enough, we put c in a set of *selected candidates*, $\mathcal{C}_t^\star \subseteq \mathcal{C}_t$, otherwise we discard c.

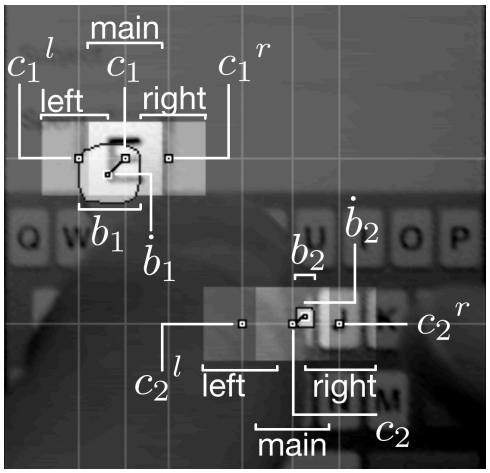


Figure 4: At frame $\mathtt{Z}_t$, for each blob b_1, b_2 the candidate keys c_1, c_2 are found as the keys that are closer to each blob's centroid $\dot{b}_1, \dot{b}_2$. The corresponding neighborhood are $\{c_1^l, c_1, c_1^r\}$, and $\{c_2^l, c_2, c_2^r\}$, respectively.

More precisely, given an image region $\mathtt{A}$ (*e.g.*, a ROI or a full key template), we define the function $\mathrm{bw}(\mathtt{A}) = (A_b, A_w)$, where A_b and A_w are the percentage of black and white pixels, respectively. The *black-white distance* is then calculated as

$$\mathrm{d}_{\mathrm{bw}}(c) = \mathrm{d}\left(\mathrm{bw}(ROI(c)) - \mathrm{bw}(T^f(c))\right)/\sqrt{2}, \tag{13}$$

where $\sqrt{2}$ guarantees that $\mathrm{d}_{\mathrm{bw}} \in [0,1], \forall c$. This distance is fast to compute and is leveraged to build the subset $\mathcal{C}_t^\star$ as

$$\mathcal{C}_t^\star = \{c \in \mathcal{C}_t \mid \mathrm{d}_{\mathrm{bw}}(c) \leq \Gamma_{\mathrm{bw}}\} \tag{14}$$

that indeed contains only those keys which full template matches the corresponding ROI, with respect to bw. The threshold Γ_{bw} is determined as described in Section IV-E.

Note that, in our implementation, d_{bw} considers as black pixels with intensity lower than 0.3, and white those with intensity above 0.5. These thresholds have been experimentally defined, because in the iPhone keyboard pixels not belonging to magnified keys (that must to be ruled out) typically have values in $[0.3, 0.5]$.

4) Key similarity maximization: The *key similarity* of the magnified-key candidate $c \in \mathcal{C}_t^\star$ is proportional to the maximum value of the *normalized cross-correlation*, ncc, between the *cropped* key template $T^r(c)$ and the $ROI(c)$:

$$\Phi_t(c) := \frac{\max\left(\mathrm{ncc}(T^r(c), ROI(c))\right)}{1 + d_c}, \tag{15}$$

where d_c is the distance between the candidate c and the barycenter of the corresponding blob b_i that yielded $c \in \mathcal{N}(c_i)$, and $1 + d_c$ is always non-zero. Recall that, maximizing $\mathrm{ncc}(T^r(c), ROI(c))$ means considering different displacements of $T^r(c)$ to determine the best match with $ROI(c)$. In our system, ncc is calculated using the fast algorithm in [23].

The *best-matching key* at time t is then identified as

$$c_t^\star := \text{argmax}_{c \in \mathcal{C}_t^\star} \Phi_t(c), \tag{16}$$

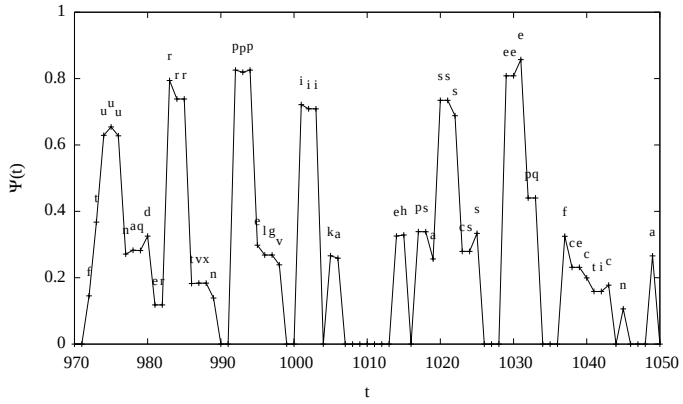
and the corresponding *key-similarity measure* is

$$\Psi(t) := \Phi_t(c_t^\star). \tag{17}$$

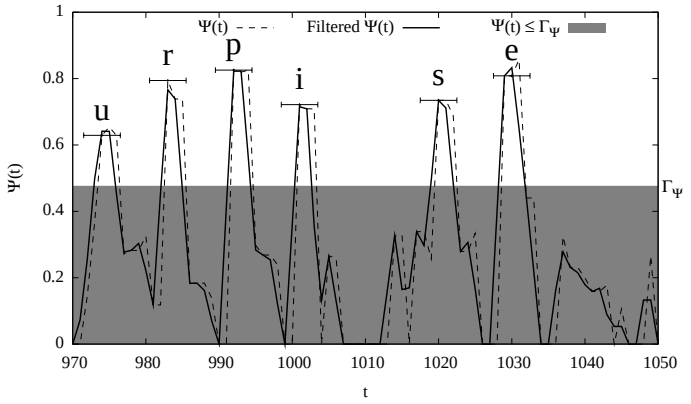
Summarizing, $c_t^\star$ identifies the magnified key selected as the most likely to appear in frame $\mathtt{Z}_t$, while $\Psi(t)$ represents the measure of the similarity between the template, $T^r(c_t^\star)$, and the corresponding region of interest $ROI(c_t^\star)$.

D. Keystroke Sequence Recognition

In order to recognize the keystroke sequence, it is insufficient to identify the best-matching key at each time step. In fact, key magnifications typically last longer than one frame, and there are frames that contains no magnified keys. These issues are addressed by analyzing $\Psi(t)$ when t varies (an example of Ψ is plotted in Figure 5a). We stress that key magnification is typically smooth, thus the measure $\Psi(t)$ reaches its maximum



(a) $\Psi(t)$ and letters at coordinates $c_t^\star$.



(b) $\Psi(t)$, dashed, and low-pass filtered $\Psi(t)$, solid.

Figure 5: (a): $\Psi(t)$, the key similarity measure of the best matching key, $c_t^\star$. (b): low-passed $\Psi(t)$, threshold Γ_Ψ and local maxima extracted. Frames providing values of $\Psi(t)$ below Γ_Ψ are discarded. The brackets above each local maximum indicate the minimum distance between two local-maxima, which can be considered as the minimum number of frames the key magnification lasts. The selected keys, given by (16) are also displayed. Note that selected magnified keys last longer than one frame, and that the zero values in $\Psi(t)$ correspond to frames where phase 2 does not detect any blob.

and then decreases every time a key magnifies. Therefore, the first issue is successfully solved by considering only the best-matching keys corresponding to local maxima of $\Psi(t)$.

The second issue, frames without magnified keys, is solved by exploiting the fact that such frames exhibit lower values of $\Psi(t)$. Thus, these can be discarded by introducing a threshold on $\Psi(t)$. Such threshold Γ_Ψ is computed as described in Section IV-E, and, experimentally, corresponds to a typical value of $\Psi(t)$ when there is no key magnification.

In order to reduce fluctuations of the key similarity measure, we preliminarily process $\Psi(t)$ with a low-pass filter (*e.g.*, an averaging filter), and we impose a minimum distance of 5 frames between adjacent local maxima. Figure 5b illustrates the local-maxima extraction, threshold Γ_Ψ, the minimum distance between the local maxima, and the corresponding keystrokes recognized. Note that, at 25fps a typing rate of one stroke every five frames is a high typing speed; we underline that this choice only influences the *maximum* typing speed handled by our system. Therefore, even when victims type *slower* than one stroke every five frames, magnified keys are recognized correctly.

E. Parameters estimation

Both thresholds Γ_{bw} and Γ_Ψ are determined by the following statistical analysis. Few videos of different users mimicking text typing without actually pressing any key are acquired. Then, Γ_{bw} and Γ_Ψ are set to 1 and 0, respectively, and each acquired video is processed by our system to record $\mathrm{d}_{\text{bw}}(\cdot)$ and $\Psi(\cdot)$ into two sequences, Y_{d} and Y_Ψ. These sequences thus contain the values assumed by d_{bw} and $\Psi(\cdot)$ when there are no keys magnified on the screen. We consider them as random variables whose distribution is unknown. Hence, according to Chebyshev's inequality, the thresholds are $\Gamma_{\text{bw}} = \hat{\mu}_{\text{d}} + \nu\hat{\sigma}_{\text{d}}$ and $\Gamma_\Psi = \hat{\mu}_\Psi + \eta\hat{\sigma}_\Psi$, where $\nu, \eta \in \mathbb{R}$ are tuning parameters, while $\hat{\mu}$ and $\hat{\sigma}$ indicate the sample mean and sample standard deviation, respectively. Preliminary experiments on iPhone devices revealed that $\nu = 0$ and $\eta = 3$ yield best results, therefore we use such values in our evaluation.

V. Evaluation

The most important goal of this evaluation is to show that our system implements an automatic, faster alternative to tedious manual inspection of a video while achieving comparable accuracy in real-world settings. The second goal is to measure the maximum recognition capabilities achievable under ideal conditions, to better understand the possible errors that it may exhibit when these are not met. The third goal is to evaluate the robustness of our system to extreme working conditions, to assess its limitations (detailed in Section VI).

Evaluation procedure, data and criteria

As summarized in Figure 6, our evaluations follow three logical steps. The Typing step is performed by volunteers (referred to as "victims") and takes the ground truth text as input. The Recording step is performed with a low-resolution camera (*i.e.*, 640 by 480 pixels @ 25fps) and produces a video depicting the iPhone screen while the victim types in the input text. Unless differently stated, we kept the *handheld* camera at an angle such that our system can recognize the screen. This simply means that Requirement 1 must hold. On the other hand, fingers are allowed to cover the keyboard as part of the regular typing actions. As detailed in the remainder of this section, the Recognition step is performed by manual volunteers (referred to as "attackers") and also, automatically, with our system. In both cases, the sequence of letters recognized is output.

In the remainder of this section we define precisely the evaluation criteria and the ground truth text used as input.

Symbols: We evaluated our implementation on a peculiar use case (*i.e.*, the iPhone), which displays no feedback for the spacebar (as explained in Section VI); however, we opted for this device because of its vast popularity. For such reason, the output text is the concatenation of words. Therefore, from

Figure 6: Our evaluations follow three logical steps. The target text, *e.g.*, *"loremipsum"*, is typed in by the victim, who may introduce errors, thus obtaining, for example, *"oremepsum"*. Given the visible (according to Requirement 1) magnified keys (*e.g.*, *"orempsum"*), our system is run and $\mathcal{D}$ is output, *e.g.*, *"rempsum"*.

hereinafter, when referring to "letters" (or "symbols") we imply excluding spaces. More precisely:

- I, is the sequence of symbols typed in, which might differ from the symbols of original text, because victims in our tests often introduce mistakes or skip words.
- $\mathcal{V} \subseteq I$, is the sequence of visible magnified keys. Fingers may cover magnified keys *entirely* and for many frames, hence our system might miss it.
- $\mathcal{D}$, is the sequence of symbols recognized by either our system or an attacker.

Notation For convenience, we indicate with "$\sqcap$" the intersection between keystroke sequences. For example, 'spaghetti' $\sqcap$ 'pdagjetti' = 'pagetti'.

Hits: Given the aforesaid measures, we are interested in two types of hit rates.

- *actual* rates are calculated with respect to $\mathcal{V}$ and thus express the *real* hit rate of our system, because take into account only the letters that the system could possibly recognize (because Requirement 1 holds). The *hits* are $\mathcal{H} := \mathcal{D} \sqcap \mathcal{V}$, hence the *hit rate* is $h := \frac{|\mathcal{H}|}{|\mathcal{V}|}$.
- *perceived* rates are calculated with respect to I and thus express the "recall" of our system as perceived by considering it as a black box. Using the same formalism described above, the *perceived hits* are $\tilde{\mathcal{H}} := \mathcal{D} \sqcap I$, hence the *perceived hit rate* is $\tilde{h} := \frac{|\tilde{\mathcal{H}}|}{|I|}$.

Notation For convenience, with "$\neg$" we indicate the sequence of symbols not marked as hits with respect to another sequence. More precisely, $\mathcal{A}\neg\mathcal{H}$ is a sequence that contains all the symbols in $\mathcal{A}$ that are not in hits, with respect to $\mathcal{H}$. Note that, given the way $\mathcal{H}$ is defined, this cannot be expressed with a simple subtraction operation. For example, given $\mathcal{H}$ = 'spaghetti' $\sqcap$ 'pdagjetti' = 'pagetti', 'pdagjetti'$\neg\mathcal{H}$ = '~~pdagjetti~~' = 'dj'.

Errors: Given the aforesaid measures, we are also interested in calculating the *errors*, $\mathcal{E} := \mathcal{D}\neg\mathcal{H}$, and the *error rate*, $\varepsilon := \frac{|\mathcal{E}|}{|\mathcal{D}|}$.

Note that, the error rate is *independent* from the visibility of the typed symbols, and quantifies the meaningless symbols in $\mathcal{D}$. For instance, in a video of a user mimicking text typing without actually pressing any key, ε would express whether spurious blobs are mis-detected as keystrokes.

Speed: We are interested in three speed measures:

- *typing speed* $s_t := \frac{|I|}{T}$, in *symbols per second* (sps), where T is the elapsed *typing time*.
- *recognition speed* $s_r := \frac{|\mathcal{D}|}{T_r}$, in *symbols per second* (sps), where T_r is the execution time of Phase 1-3.
- *processing speed* s_p is simply the maximum frame-rate (in fps) achieved by Phase 1-3.

Ground truth input: Throughout our experiments, we utilized three types of inputs:

- *context-free* text[3] with poor context (63 English words, 444 symbols plus spaces). This ensures that manual attackers in our experiments cannot simply reconstruct a word by simply guessing based on the linguistic context.
- *Context-rich* text[4], the first 65 words of the lyrics of Dream Theater's *"Regression"* song, which is rich of context (total 278 symbols plus spaces).
- *Brief text*, used to evaluate specific features and limitations:. *"Pellentesque habitant morbi tristique senectus et netus et malesuada fames ac turpis egestas"* (80 letters plus spaces and symbols, total 13 words).

A. Experiment 1: Precision and speed comparison

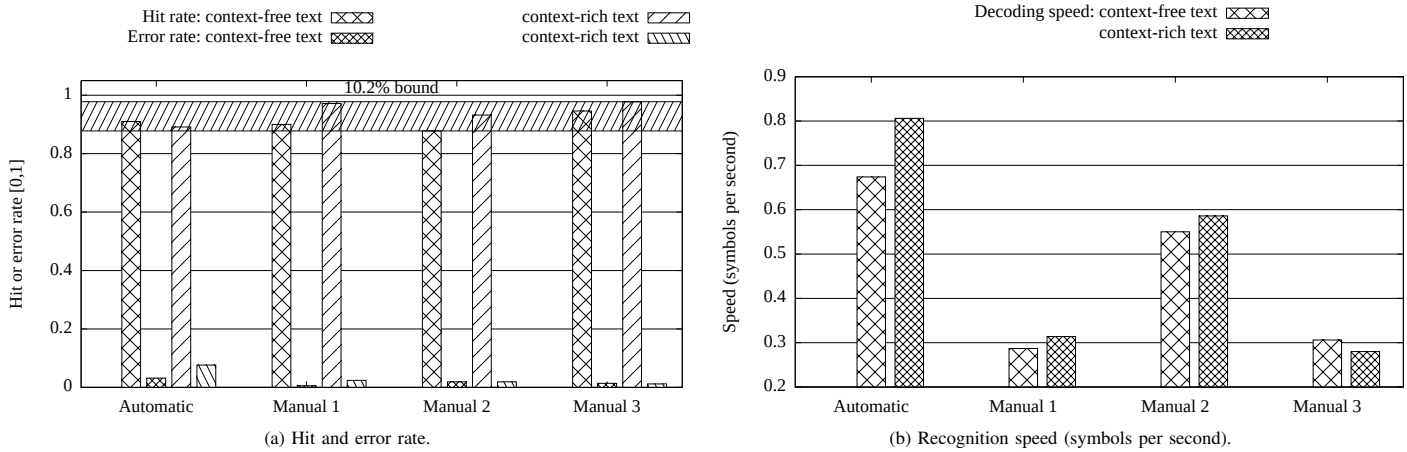
To demonstrate that our attack works in practice we recorded 3 videos with context-free text and 3 videos with context-rich text, each featuring a different victim. In particular, we asked two iPhone owners to type naturally, as they would do in their daily activities. The third victim is not used to type on an iPhone, but is used to work with mobile touchscreen devices.

Each video was processed by our system and, to compare its performance, we also assigned one *different* attacker to each victim. Without any prior knowledge on the ground truth, each attacker was asked to manually reconstruct the keystrokes by stopping, rewinding or slowing-down the video as needed, in order to recognize as many symbols as possible.

The average performance are summarized in Figure 7, and detailed in Table II (context-free text) and Table IV (context-rich text). Each table cell shows, for each victim and attacker combination, $|\mathcal{H}|$ and $|\mathcal{E}|$, the corresponding hit and error rates h, $\tilde{h}$, ε, and the recognition speed, s_r, (in sps). The s_p columns refer to the automatic attacker only. The "AVERAGE" columns report the average (plotted in Figure 7) of the punctual values obtained on each victim/attacker. As shown in Figure 7, regardless of the text's context, manual recognition is notably slower than our system. For example, our system can recognize, on average, up to 0.803 sps, about one third of the maximum typing speed, and 0.864 in the best case, about one half of the average typing speed. Only two attackers were able to surpass such speeds.

As expected, with manual inspection an attacker can recognize symbols with slightly higher precision than our system. This is more evident in the context-rich text experiment (Table IV), where our system is outperformed by about 8 percentage points. Without any a-posteriori correction, for context-free text for example, our system is just 3 percentage points

[3]publicly available at http://sqze.it/qMNwy
[4]publicly available at http://sqze.it/SGTu-



(a) Hit and error rate.

(b) Recognition speed (symbols per second).

Figure 7: Comparison of average precision (a) and speed (b) of our automatic detector *vs*. human attackers. While precision (*i.e*., hit and error rates) remain within shallow bounds, the speed of manual recognition is significantly low with respect to our automated attack.

below the best average attacker in our experiments. Hence, our system is comparable with manual inspection, and as a plus, is faster and, more importantly, tireless. Indeed, all the volunteers described this session as an extremely tedious task, certainly not doable for extended periods of time.

The analysis of these results must be "scaled" by taking into account the typing speed. For example, Victim 3 types at slow speed and, as expected, humans were able to recognize keystrokes very efficiently. In case of faster users, manual analysis is more tedious and prone to error, whereas our system is not influenced by these factors (as long as magnified keys are partially visible).

To the best of our knowledge, no state-of-the-art comparison baseline exists for automated eavesdropping on *mobile*, touch-screen keyboards. However, it is noteworthy that the state-of-the-art system [5], which has the considerable advantage to work on large, desktop-sized keyboards, can recognize at a speed of only 0.101 symbols per second, with a maximum precision of 82%.

B. Experiment 2: Recognition in ideal vs. real-world conditions

In the previous experiment we assessed the feasibility of the attack in natural conditions: the camera was *handheld* and the victim was allowed to *move naturally*. The goal of this second experiment is instead to evaluate Phase 1, and Phase 2-3, separately, thus allowing us to measure the maximum achievable performance of Phase 2-3 in ideal (unrealistic) conditions.

We recorded one victim while typing the context-free text in ideal conditions (*i.e*., with a camera on a tripod and the device fixed on a table, with its screen perfectly aligned and parallel to the camera's sensor), and run our system with Phase 1 *disabled*. Values of hit and error rates are summarized in Table III. As one may expect, under ideal conditions Phase 2-3 alone reach remarkable hit rate and error rate. In particular, we notice that the perceived hit rate, $\tilde{h}$, exhibits a higher improvement, 8.25%, since in these ideal conditions magnified keys are not occluded (*e.g*., the victim's finger is always below each magnified key), a less common case in real-world settings (*e.g*., because of the camera's skewed viewing angle) As a consequence, in ideal settings $|I| \simeq |\mathcal{V}|$, whereas in real-world settings $|I| > |\mathcal{V}|$. However, in Section V-A we showed that, in real-world settings, Phase 1-3 together can reach up to 97.07% hit rate (91.03% on average).

C. Experiment 3: Resilience to excessive aberrations

The goal of this third experiment is to stress the robustness of Phase 1. To this end, we performed a series of brief typing sessions and included several significant aberrations. In practice, we asked a volunteer to type the brief text B.2 under the following conditions:

1) we attached a piece of gray tape on the screen to emulate a permanent occlusion,
2) we asked the victim to shake the device while typing,
3) the camera was shaken while recording, and
4) we asked the victim to shake the device while typing, and the camera was shaken while recording.

As shown in Table V, Phase 1 was able to rectify parts of the video and thus recognize the screen at some point of the video. In the best case we were able to recognize 96% of the symbols with as low as 4% errors. In the worst case, we are still able to detect part of the text, 44.44%, although the errors caused by the difficulty of dealing with objects that cover the screen permanently are quite high. However, users seldom hold touchscreen devices with permanent occlusions, especially while typing. As shown in Figure 8, our system can handle sudden movements of either camera or device, while Phase 1 fails when both camera and device move excessively, causing intra-frame, motion-blurring side effects that are computationally expensive to remove and affects the features extraction task described in Section III-A. However, some offline manual intervention at the beginning of Phase 1 would help to extract the significant features that would help to bootstrap the rectification, even under challenging conditions, yet these techniques fall outside the purpose of this paper.

Context-free experiment				VICTIM 1			VICTIM 2			VICTIM 3		
Typing time				4'19"			3'54"			5'54"		
Symbols typed, $\|I\|$				440			442			445		
Typing speed, s_t				1.698			1.888			1.257		
Frames processed				5200			4686			7098		
Processing speed, s_p				9.6498			10.3450			9.6697		
ATTACKERS	AVERAGE			$\|\mathcal{H}\|$	$\|\mathcal{E}\|$	s_r						
	$h\%, \tilde{h}\%$	$\varepsilon\%$	s_r	$h\%$, $\tilde{h}\%$	$\varepsilon\%$							
Automatic	**91.03, 85.95**	3.16	**0.674**	355 87.01, 80.68	20 5.33	0.673	356 89.00, 80.54	11 2.99	0.729	430 97.07, 96.63	5 1.15	0.619
Manual 1	96.09, 90.71	0.75	0.327	401 98.28, 91.14	7 1.71	0.374	360 90.00, 81.44	2 0.55	0.287	443 100.00, 99.55	0 0.00	0.321
Manual 2	87.78, 82.88	1.90	0.550	288 70.58, 65.16	15 4.95	0.974	372 93.00, 84.16	2 0.53	0.208	442 99.77, 99.33	1 0.22	0.469
Manual 3	94.61, 98.01	1.39	0.306	383 93.87, 87.04	1 0.26	0.292	378 94.49, 84.94	5 1.32	0.304	423 95.48, 95.06	11 2.60	0.323

Table II: Context-free text (444 letters). Average hit rate, $h\%$ (and $\tilde{h}\%$), error rate, $\varepsilon\%$, and recognition speed, s_r (symbols per second) of our automatic recognizer vs. manual recognition. The text typed is the same yet each victim mistyped some letters and skipped some words randomly. Each text was reviewed by a different attacker (total 9). **Note:** below each value of $|\mathcal{H}|$, the corresponding $h\%$ and $\tilde{h}\%$ are displayed; below each value of $|\mathcal{E}|$, the corresponding $\varepsilon\%$ is displayed.

VI. LIMITATIONS

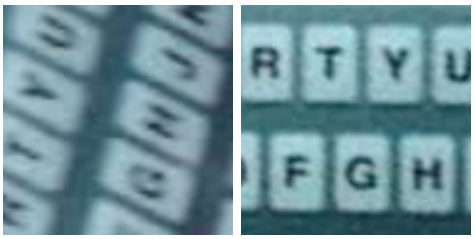
Our system's main limitation revolves around the fact that keystrokes are recognizable as long as visual feedback, such as magnified keys, is displayed. Therefore, if some keys are not magnified our current implementation would not be able to detect them. One example is the iPhone's spacebar key, which is not magnified but it simply changes color from light to dark when pressed. To deal with this problem, we tried to develop an ad-hoc solution leveraging the aforesaid color change. However, preliminary validation revealed that this technique fails frequently because fingers often cover entirely the spacebar, thus generating many errors. For this reason, this feature was not included in the experimental evaluation as it will be further investigated in future work.

Similarly, punctuation symbols on the iPhone keyboard are selected on a different magnified layout, which is chosen by pressing a non-magnifying key. To deal with this problem, our system would need to support multiple templates, although this may increase the computational cost. In Phase 3, for example, the number of magnified-key candidates would increase proportionally with the number of different magnified layouts, because the validation step would need to lookup the best-matching key by cycling through several alternative layouts. Consequently, the system would rely on multiple key templates and multiple screen templates. Thus also Phase 2 is affected.

As minor limitation, we do not take into account automatic corrections performed by the device's typing system. Fortunately, since many automatic spell checkers and correctors are available for free in public domain, this poses no actual limitations to our attack. In fact, as shown in previous work [5], if the raw output of a system like ours contains many errors, it is very easy to correct them automatically. Nonetheless, we have shown that our system is quite accurate as no significant errors are introduced other than those committed by the victim. Thus, we believe that a simple automatic spell corrector would suffice.

Last, since the system we propose is basically an automated shoulder surfing attack, its applicability might be reduced by screen-protection methods as those described in Section II-D

(a) Both camera and device sudden movements.



(b) Device movements only.

Figure 8: Phase 1 is affected by sudden movements. In (a), a quick relative movement generates a high blur level that prevents the detection of the screen, which is instead successfully detected when only the device moves (b).

MEASURES	IDEAL	REAL-WORLD (AVG.)	DIFFERENCE
	PRECISION %		
$h\%$	95.12	91.03	4.09
$\tilde{h}\%$	94.20	85.95	8.25
$\varepsilon\%$	1.01	3.16	-2.15

Table III: Ideal conditions vs. real-world conditions.

Context-rich experiment				VICTIM 1			VICTIM 2			VICTIM 3		
Typing time				2'47"			2'10"			2'23"		
Symbols typed, $\|I\|$				270			250			277		
Typing speed, s_t				1.616			1.923			1.937		
Frames processed				3341			2607			2877		
Processing speed, s_p				10.8883			10.6351			10.8174		
ATTACKERS	AVERAGE			$\|\mathcal{H}\|$	$\|\mathcal{E}\|$	s_r						
	$h\%, \tilde{h}\%$	ε%	s_r	$h\%$, $\tilde{h}\%$	ε%							
Automatic	**89.11, 85.83**	7.64	**0.803**	220 86,27, 81.48	40 15.38	0.708	218 87.55, 87.20	10 4.38	0.838	246 93.53, 88.80	8 3.15	0.864
Manual 1	97.11, 93.92	1.07	0.290	243 95.29, 90.00	2 0.82	0.348	242 97.18, 96.80	6 2.41	0.314	263 98.87, 94.95	0 0.00	0.209
Manual 2	93.20, 90.14	1.90	0.586	231 90.58, 85.55	8 3.35	0.802	232 93.17, 92.80	1 0.43	0.485	255 95.86, 92.06	5 1.92	0.471
Manual 3	97.77, 94.53	1.18	0.280	249 97.64, 92.22	3 1.20	0.389	242 97.19, 96.80	2 0.82	0.265	262 98.49, 94.58	4 1.53	0.185

Table IV: Context-rich text (278 letters). Average hit rate, h (and $\tilde{h}$), error rate, ε, and recognition speed, s_r (symbols per second) of our automatic recognizer vs. manual recognition. The text typed is the same yet each victim mistyped some letters and skipped some words randomly. Each text was reviewed by a different attacker (total 9). **Note:** below each value of $|\mathcal{H}|$, the corresponding $h\%$ and $\tilde{h}\%$ are displayed; below each value of $|\mathcal{E}|$, the corresponding ε is displayed.

simply because it would be more challenging to take a video, not specifically because of a limitation of our recognition approach.

VII. CONCLUSIONS

To the best of our knowledge, we were the first to thoroughly demonstrate the feasibility of automatically recognizing keystrokes on a touchscreen by exploiting the graphical interface usability.

We have shown that our system can recognize keystroke sequences nearly as accurately as a human attacker, yet significantly faster, and in a fully automated way, without the need of a postprocessing phase. In addition, our system works under more realistic hypotheses than previous work [5], and works under realistic conditions for mobile devices. Our experimental validation confirms that our attack is feasible and work in real-world scenarios, with a few well-defined limitations.

We conclude that modern touchscreen keyboards that magnify the pressed keys to help the user while typing make shoulder surfing easier and more efficient than in tactile or mechanical, small-factor keyboard. Therefore, such interfaces must be deemed unsuitable for scenarios that demand high privacy.

REFERENCES

[1] A. Smith, "Mobile access 2010," Available online, Pew Research Center's Internet & American Life Project, Tech. Rep., July 2010, http://www.pewinternet.org/Reports/2010/Mobile-Access-2010.aspx.

ABERRATION	PHASE 1	PHASE 2-3	
		$h\%$	ε%
1) Permanent occlusion	difficult	44.44	33.33
2) Shake device	feasible	67.74	8.70
3) Shake camera	feasible	96.00	4.00
4) Shake device + camera	unfeasible	0.00	-

Table V: Detection results under different working conditions.

[2] G. S. Hurst and J. E. Parks, "Electrical sensor of plane coordinates," Available online, The University of Kentucky Research Foundation, Patent 3662105, May 1970, http://www.google.com/patents/about?id=UUovAAAAEBAJ.

[3] comScore, Inc, "Touchscreen mobile phone adoption grows at blistering pace in u.s. during past year," Available online, November 2009, http://www.comscore.com/Press_Events/Press_Releases/2009/11/Touchscreen_Mobile_Phone_Adoption_Grows_at_Blistering_Pace_in_U.S._During_Past_Year.

[4] R. Cozza, C. Milanesi, A. Gupta, H. J. D. L. Vergne, A. Zimmermann, C. Lu, A. Sato, and T. H. Nguyen, "Competitive landscape: Mobile devices, worldwide, 3q10," Excerpt available online, Gartner, Inc., Tech. Rep., November 2010, http://www.gartner.com/it/page.jsp?id=1466313.

[5] D. Balzarotti, M. Cova, and G. Vigna, "ClearShot: Eavesdropping on Keyboard Input from Video," in *Proceedings of the IEEE Symposium on Security and Privacy*, Oakland, CA, May 2008.

[6] Y.-M. Tsuei, "Method and apparatus for preventing on-screen keys from being accidentally touched using the same," US Patent 12427767, HTC Corporation, April 2009. [Online]. Available: http://www.google.com/patents?id=VU_TAAAAEBAJ

[7] D. S. Tan, P. Keyani, and M. Czerwinski, "Spy-resistant keyboard: more secure password entry on public touch screen displays," in *OZCHI '05: Proceedings of the 17th Australia conference on Computer-Human Interaction*. Narrabundah, Australia: CHISIG of Australia, 2005, pp. 1–10.

[8] R. Anderson and T. Moore, "Information security: where computer science, economics and psychology meet," *Philosophical Transactions of The Royal Society*, no. 367, pp. 2717–2727, 2009.

[9] U. Bayer, I. Habibi, D. Balzarotti, E. Kirda, and C. Kruegel, "A view on current malware behaviors," in *Proc. of the 2nd USENIX conference on Large-scale exploits and emergent threats (LEET '09)*. USENIX Association, 2009.

[10] M. Kumar, T. Garfinkel, D. Boneh, and T. Winograd, "Reducing shoulder-surfing by using gaze-based password entry," in *SOUPS '07: Proc. of the 3rd Symposium On Usable Privacy and Security*. New York, NY, USA: ACM, 2007, pp. 13–19.

[11] D. Russell, "2009 - skimming review," Jan 2010. [Online]. Available: http://www.atmsecurity.com/monthly-digest/atm-security-monthly-digest/2009-skimming-review.html

[12] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-up robust features (surf)," *Comput. Vis. Image Underst.*, vol. 110, no. 3, pp. 346–359, 2008.

[13] J. G. Semple and G. T. Kneebone, *Algebraic Projective Geometry*. Oxford Classic Texts, 1998.

[14] R. I. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed. Cambridge University Press, ISBN: 0521540518, 2004.

[15] R. Brunelli, *Template Matching Techniques in Computer Vision: Theory and Practice*. Wiley, May 2009.
[16] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communication of ACM*, vol. 24, no. 6, pp. 381–395, 1981.
[17] M. Piccardi, "Background subtraction techniques: a review," in *Systems, Man and Cybernetics, 2004 IEEE International Conference on*, vol. 4, 2004, pp. 3099 – 3104 vol.4.
[18] N. Otsu, "A threshold selection method from gray-level histograms," *IEEE Transactions on Systems, Man and Cybernetics*, vol. 9, no. 1, pp. 62 –66, jan. 1979.
[19] R. C. Gonzalez and R. E. Woods, *Digital Image Processing (3rd Ed.)*. Prentice-Hall, Inc., 2006.
[20] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, vol. 25, no. 11, pp. 122–125, November 2000.
[21] Y. Abdel-Aziz and H. Karara, "Direct linear transformation from comparator coordinates into object space coordinates in close-range photogrammetry," in *Proceedings of the Symposium on Close-Range Photogrammetry*, 1971, pp. 1–18.
[22] D. L. Donoho and I. M. Johnstone, "Ideal spatial adaptation by wavelet shrinkage," *Biometrika*, vol. 81, no. 3, pp. 425–455, 1994.
[23] J. Lewis, "Fast normalized cross-correlation," in *Vision Interface*, vol. 10, 1995, pp. 120–123.