Demonstrating ScreenshotMatcher: Taking Smartphone Photos to Capture Screenshots

Andreas Schmid
University of Regensburg
Regensburg, Germany
andreas.schmid@ur.de

Thomas Fischer
University of Regensburg
Regensburg, Germany
thomas1.fischer@stud.uni-regensburg.de

Alexander Weichart
University of Regensburg
Regensburg, Germany
alexander.weichart@stud.uni-regensburg.de

Alexander Hartmann
University of Regensburg
Regensburg, Germany
alexander@hartmann-it.de

Raphael Wimmer
University of Regensburg
Regensburg, Germany
raphael.wimmer@ur.de

Figure 1: ScreenshotMatcher can be used to capture high quality screenshots by photographing a PC screen with a smartphone camera.

ABSTRACT

Taking screenshots is a common way of capturing screen content to share it with others or save it for later. Even though all major desktop operating systems come with a screenshot function, a lot of people also use smartphone cameras to photograph screen contents instead. While users see this method as faster and more convenient, image quality is significantly lower. This paper is a demonstration of ScreenshotMatcher, a system that allows for capturing a high-fidelity screenshot by taking a smartphone photo of (part of) the screen. A smartphone application sends a photo of the screen region of interest to a program running on the PC which retrieves the corresponding screen region with a feature matching algorithm. The result is sent back to the smartphone. As phone and PC communicate via WiFi, ScreenshotMatcher can also be used together with any PC in the same network running the application – for example to capture screenshots from a colleague's PC. Released as open-source code, ScreenshotMatcher may be used as a basis for

applications and research prototypes that bridge the gap between PC and smartphone.

CCS CONCEPTS

• **Human-centered computing** → *Ubiquitous and mobile computing systems and tools.*

KEYWORDS

mobile, computer vision, cross device interaction

ACM Reference Format:

Andreas Schmid, Thomas Fischer, Alexander Weichart, Alexander Hartmann, and Raphael Wimmer. 2021. Demonstrating ScreenshotMatcher: Taking Smartphone Photos to Capture Screenshots. In *Mensch und Computer 2021 (MuC '21)*, September 5–8, 2021, Ingolstadt, Germany. ACM, New York, NY, USA, 4 pages. <https://doi.org/10.1145/3473856.3474029>

1 INTRODUCTION

By taking a screenshot, the content of a computer screen can be captured to share it with others or to archive it for later use. Originally, screenshots - physical photographs of computer screens - were used by researchers working on the first CAD applications at MIT to show the capabilities of their newly developed system to its designated users [1]. Today, all major operating system offer the ability to take a screenshot by retrieving the screen content from

This work is licensed under a Creative Commons Attribution International 4.0 License.

MuC '21, September 5–8, 2021, Ingolstadt, Germany
© 2021 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-8645-6/21/09.
<https://doi.org/10.1145/3473856.3474029>

an internal graphics buffer and saving it as a digital file. Typically, the user can also elect to capture only a certain region or window.

However, many people use their mobile phones for sharing screenshots with others. Instead of transferring a screenshot to the phone, they photograph the computer screen with their smartphone's camera and share this photo. By adjusting the camera's field of view, the image can already be cropped to a region of interest while taking the photo.

In a 2020 survey among 66 university students and employees (31 male, age 19–39), we found that 97% regularly took screenshots – mostly of pictures, web pages, text documents, and program code. 52% used only the screenshot function of their device, 6% only ever took photos of their screen, and 42% did both, depending on the situation. Whereas screenshots were often used for personal documentation, screen photos were seen as faster and more convenient when sharing information with others.

However, screen photos' image quality suffers from reflections, perspective distortion, moiré artifacts, and interference between the camera's and screen's refresh rates (Fig. 2). Furthermore, the file size of screen photos is significantly higher than that of screenshots with the same content because of modern smartphone's high resolution cameras.

In order to combine the advantages of screen photos and screenshots, we developed ScreenshotMatcher [8], an extensible interaction technique for capturing impeccable screenshots of screen regions by taking a photo with a smartphone camera. A smartphone application takes screen photos which are then sent to an application running on the host computer. The host application applies feature matching to find the photographed region within the screen contents, extracts the region of interest and sends it back to the smartphone where it can be shared with others or stored in the gallery – just like with a normal camera application. Similar techniques have been used in Baur et al.'s *Virtual Projection* project [2] to detect a screen visible in a live video feed, and in Chang and Li's *DeepShot* [4] to share application context between devices. Our research adds to the existing body of knowledge by demonstrating a new use case and providing an extensible open-source platform on which further applications can be developed.

2 SCREENSHOTMATCHER

With ScreenshotMatcher, phone and host PC communicate via an existing local WiFi network; neither explicit device pairing nor an internet connection are required. The smartphone application finds available host PCs via UDP broadcasts. Privacy and security are

Figure 2: Degradations that occur in screen photos. From left to right: Moiré pattern, reflections, perspective distortion.

ensured as no third party servers or cloud services are used and the connection to the WiFi network serves as implicit authentication for capturing screen content. Furthermore, in the secure default setting, users can only request screenshots of screen regions of which they have taken a photograph, i.e. of screen content to which they have physical access. Our architecture supports multiple smartphones being connected to the same PC, as well as multi monitor setups.

2.1 Usage and Interaction

Before ScreenshotMatcher can be used, both smartphone and desktop application have to be installed on the corresponding devices. The smartphone application actively searches for PCs running ScreenshotMatcher within the WiFi network and displays found devices in a list. This way, users can quickly switch between different devices, for example to capture a screenshot from a colleague's or roommate's PC.

The ScreenshotMatcher smartphone application resembles a typical camera app (Fig. 3, left). To capture a screenshot, the region of interest is selected with the app's viewfinder (Fig. 1, left). Therefore, using ScreenshotMatcher works the same way as photographing any other physical object (or computer screen). After the image has been processed successfully (section 2.2), the resulting high quality screenshot is displayed and can be saved to the gallery or shared to other applications, such as instant messengers (Fig. 3, center). In case of a failure, users can request a full screenshot of the PC's screen content (Fig. 3, right). However, this feature can be deactivated in the PC application to prevent other users in the same network from taking unauthorized screenshots.

Figure 3: The three screens of the Android app: Main Screen (left): (a) live video feed of the smartphone's main camera, (b) capture button, (c) button to change settings, (d) button to display recently saved screenshots, (e) area indicating the current connection status to the desktop application. Result Screen (center): (f) result image, (g) buttons to switch between cropped and full screenshot, (h) buttons to share or save the image. The Failed-Screen (right) is shown, if the matching process was unsuccessful.

Figure 4: Processing pipeline of ScreenshotMatcher. A feature matching algorithm searches for the photographed region of interest within a screenshot. This region is then cropped and sent back to the smartphone as a result.

2.2 Image Processing

Once the user presses the capture button, the app extracts the current image from the live feed, scales it down in resolution, converts it to grayscale and sends it to the connected PC via HTTP.

On the PC, a cross-platform daemon written in *Python 3.9* then matches the received photo against the contents on its screen and sends back the corresponding screenshot. Whenever the daemon receives a screen photo, a screenshot is captured and both images are passed to a feature matching algorithm implemented with *OpenCV 4* [3]. The matching algorithm calculates a homography between both images, extracts the region of the computer screen visible in the photograph from the actual screenshot, and returns the resulting image to the app, where it gets displayed to the user (Fig. 3, center). Should the matching process not find a result, the app displays a message box indicating a matching failure (Fig. 3, right) and users can then either try again or request a full screenshot from the PC. A full screenshot of all screens can also be obtained by swiping left or pressing the arrow on the right. As retrieving a full screenshot via network may leak information, this feature should only be enabled in trusted environments, however.

We compared different algorithms for keypoint detection and feature matching and selected the *ORB* [7] keypoint detection algorithm as it performed faster and more accurately than comparable ones. A brute force feature matcher using Hamming distance and the *k*-nearest neighbours algorithm finds matches between key-points detected within screen photo and screenshot. A subsequent Lowe’s ratio test [6] discards any bad matches. The homography between both images is calculated via *RANSAC* [5], and the photograph of the screen is aligned with the screenshot via a perspective transformation. Finally, an axis-aligned bounding box around the resulting region is extracted from the screenshot and the final image’s dimensions and size are validated to discard false positives.

The whole process from capturing the photo until the result image is displayed on the smartphone takes between 300 milliseconds and one second and is therefore similar to only taking a photo with a normal camera app.

3 DISCUSSION AND FUTURE WORK

One technical limitation of our current implementation is the requirement for all devices to be in the same WiFi network. This may restrict the system’s use in some public settings. This issue could be addressed e.g., by using a STUN server in future versions of ScreenshotMatcher. Additionally, while the feature matcher works very well for images with clear borders, it oftentimes fails for images that mainly contain text. This could be improved by extending the matching algorithm with text based features such as described by Tsai et al. [9]. We also plan to further optimize processing time so the system can be used for real-time applications.

ScreenshotMatcher’s source code is published under an open source license at our project website¹. This allows for researchers and software developers to develop research prototypes and custom applications that require that a smartphone knows about screen contents.

ACKNOWLEDGMENTS

This project is funded by the Bavarian State Ministry of Science and the Arts and coordinated by the Bavarian Research Institute for Digital Transformation (bidt).

REFERENCES

- [1] Matthew Allen. 2016. Representing Computer-Aided Design: Screenshots and the Interactive Computer circa 1960 (Perspectives on Science, 24:6). *Perspectives on Science* 24, 6 (2016).
- [2] Dominikus Baur, Sebastian Boring, and Steven Feiner. 2012. Virtual projection: exploring optical projection as a metaphor for multi-device interaction. In *Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems - CHI '12*. ACM Press, Austin, Texas, USA, 1693. <https://doi.org/10.1145/2207676.2208297> 00081.
- [3] G. Bradski. 2000. The OpenCV Library. *Dr. Dobbs’ Journal of Software Tools* (2000). 07375.
- [4] Tsung-Hsiang Chang and Yang Li. 2011. Deep shot: a framework for migrating tasks across devices using mobile phone cameras. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '11)*. ACM, Vancouver, BC, Canada, 2163–2172. <https://doi.org/10.1145/1978942.1979257> 00091.
- [5] Martin A. Fischler and Robert C. Bolles. 1981. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM* 24, 6 (June 1981), 381–395. <https://doi.org/10.1145/358669.358692> 25386.

¹<https://hci.ur.de/projects/screenshotmatcher>

- [6] David G. Lowe. 2004. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision* 60, 2 (Nov. 2004), 91–110. <https://doi.org/10.1023/B:VISI.0000029664.99615.94>
- [7] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. 2011. ORB: An efficient alternative to SIFT or SURF. In *2011 International Conference on Computer Vision*. IEEE, Barcelona, Spain, 2564–2571. <https://doi.org/10.1109/ICCV.2011.6126544> 07466.
- [8] Andreas Schmid, Thomas Fischer, Alexander Weichart, Alexander Hartmann, and Raphael Wimmer. 2021. ScreenshotMatcher: Taking Smartphone Photos to Capture Screenshots. In *Proceedings of Mensch und Computer 2021*. <https://doi.org/10.1145/3473856.3474014>
- [9] Sam S. Tsai, Huizhong Chen, David Chen, Vasu Parameswaran, Radek Grzeszczuk, and Bernd Girod. 2012. Visual Text Features for Image Matching. In *2012 IEEE International Symposium on Multimedia*. IEEE, Irvine, CA, USA, 408–412. <https://doi.org/10.1109/ISM.2012.84>