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

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. With ScreenshotMatcher, we present 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 matching screen region and sends it back to the smartphone. Comparing four feature matching algorithms and multiple parameters, we identified a combination of ORB keypoint detection (feature limit 2000) and a brute force feature matcher using Hamming distance as the best solution for this task (success rate: 85%, processing time: 90 ms). This raw performance results in a real-world success rate of 47% and a mean response time per screenshot of 878 ms as measured in a remote user study (N=19). 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. 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, 5 pages. https://doi.org/10.1145/3473856.3474014

MuC '21, September 5–8, 2021, Ingolstadt, Germany

ACM ISBN 978-1-4503-8645-6/21/09.
https://doi.org/10.1145/3473856.3474014

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 systems offer the ability to take a screenshot by retrieving the screen content from an internal graphics buffer and saving it as a digital file. Typically, users 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. 1). Furthermore, the file size of screen photos is

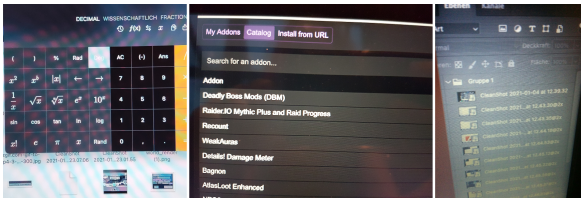

Figure 1: Degradations that occur in screen photos. From left to right: Moiré pattern, reflections, perspective distortion.

significantly higher than that of screenshots with the same content because of modern smartphone's high resolution cameras.

To combine the advantages of screen photos and screenshots, we developed ScreenshotMatcher, an extensible interaction technique for capturing impeccable screenshots of screen regions by taking a photo with a smartphone camera. A smartphone app 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. Our research adds to the existing body of knowledge by demonstrating a new use case, systematically evaluating suitable combinations of keypoint detection and feature matching algorithms, and providing an extensible open-source platform on which further applications can be developed.

2 RELATED WORK

ScreenshotMatcher extends a large body of research on interaction with displays via the smartphone camera. In 2004, Madhavepeddy et al. [13] showed how a mobile phone camera could be used to interact with content on a computer screen. They displayed markers encoding information about a Bluetooth service on a screen and decoded them on the mobile phone, which in term connected to said service. An example application was a world map where airports were represented by such markers. By scanning them with a phone camera, users could book flights. In *Sweep and Point & Shoot* [2] Ballagas et al. present an application that uses optical flow image processing on a phone camera image so it can track relative movement similar to an optical mouse. They combined this with a grid of markers encoding absolute coordinates on a large screen so users could interact with the displayed content by pointing at the screen with a phone camera. *Shoot&Copy* by Boring et al. [5] is a method to capture content of a public display with a phone camera without relying on markers. Users take a photo of a region on a large display. The photo is sent to a host computer which searches for the corresponding region within the image on the display. The region is calculated by identifying icons displayed on the screen within the photograph. Each of those icons represents a URL which is sent back to the phone. To access the content associated with the icon, received URLs can be sent to a PC via Bluetooth. Baur et al. [3] propose multi-device interaction techniques that use a smartphone as a magic lens. Users can interact with content displayed on computer screens by touching them in a live camera image on their smartphone. This way, they can for example move content from one screen to another or open applications like web browsers or video players on a secondary screen. To detect screens within the camera image, *SURF* [4] is used to find features in screenshots of all connected displays and the camera image of the smartphone. If a match is found, a homography between both images is calculated. With *DeepShot* [7], Chang and Li present a method to transfer application context from one device to another by scanning a screen with a phone camera. Applications can either be "picked up" with the phone *(Deep Shooting)* or "projected" from the phone onto the screen *(Deep Posting)*. The system automatically recognizes the captured context and opens it with an appropriate application on the other device. To identify the region of interest on the computer screen, a screenshot of the screen's content and a photo with the phone's camera are captured. *SURF* is used to detect keypoints and matches are calculated by finding the nearest neighbor for each point by computing cosine similarity. By calculating a homography, the region visible in the camera image can be determined within the screenshot of the computer screen.

Even though several application use computer vision to match camera images to screen content, to our knowledge, this method has not been applied to the use case of capturing screenshots yet.

3 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 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.

The ScreenshotMatcher smartphone application resembles a typical camera app (Fig. 2, left). The top right area indicates the connection status and doubles as a button which opens a list of available PCs running the ScreeshotMatcher desktop application. 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

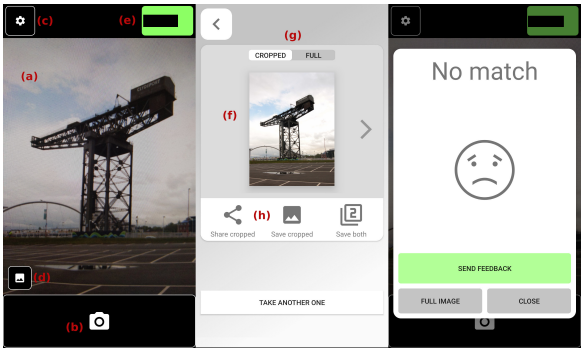

Figure 2: The three screens of the Android app: Main Screen (left): *(a)* live view of the smartphone's main camera, *(b)* capture button, *(c)* change settings, *(d)* display recently saved screenshots, *(e)* indicator for the current connection status. Result Screen (center): *(f)* result image, *(g)* 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.

passed to a feature matching algorithm implemented with *OpenCV 4* [6]. 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. 2, center). Should the matching process not find a result, the app displays a message box indicating a matching failure (Fig. 2, 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* [15] 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 keypoints detected within screen photo and screenshot. A subsequent Lowe's ratio test [12] discards any bad matches. The homography between both images is calculated via *RANSAC* [9], 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.

4 OPTIMIZATION AND EVALUATION

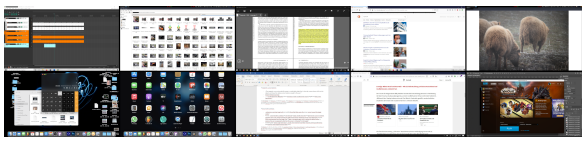

Figure 3: Optimization data set. Each category is represented by two images (from left: GUI, Icons, Text, Article, Image).

For ScreenshotMatcher to be a usable in practice, the matching algorithm should deliver the correct screenshots reliably and as quickly as if the user had only taken a normal photo. Therefore, we systematically compared keypoint detectors, feature matchers, and associated parameters in terms of recognition rate and computation time. We then evaluated the performance of the resulting algorithm with a data set of screen photos and corresponding screenshots.

To compare the different matching algorithms and later evaluate the system, a data set of 68 screenshots in 1080p resolution were captured. They were categorized as GUI, icons, text, article (combination of text and images) or image. The data set was split up into a optimization data set and an evaluation data set to avoid bias towards the data set. Two screenshots of each content category were used to compare the different algorithms (Fig. 3), the remaining 58 screenshots were used for the evaluation of the final system. We then asked nine colleagues to display each of the screenshots full-screen on a computer screen and take a photo of an interesting region as if they were about to share the content with a friend or colleague. As some of them owned multiple phones or monitors, 16 data sets of 68 photographs (1088 total) could be collected this way.

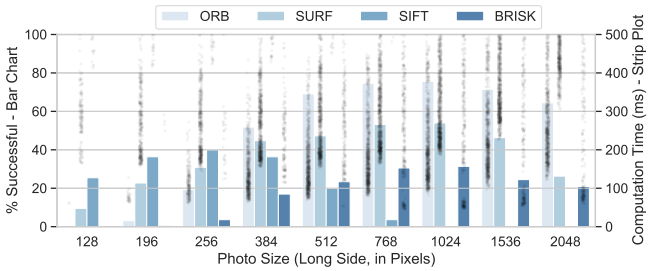

Figure 4: Comparison of keypoint detectors for different image sizes in terms of success rate and computation time. As responsiveness of the system is important, processing times of over 500 ms were considered as unsuccessful.

To find a matching algorithm suitable for ScreenshotMatcher, we compared multiple keypoint detection and feature matching algorithms. Our goal was to achieve a short processing time while keeping a high success rate. Even though feature extraction with artificial neural networks is commonplace [10], we restricted ourselves to comparing standard computer vision algorithms as the training set requirements and the computational effort are not justified if an approach with less overhead can deliver sufficiently accurate results. Shneiderman et al. [17] suggest a system response time of below one second for "simple frequent tasks" and Seow [16] categorizes response times of 0.5 to one second as "immediate". As the the total response time also includes the round trip time from capturing the photo until receiving the result on the smartphone, we aimed for a processing time of below 100 ms.

We included the keypoint detectors *SIFT* [12], *SURF* [4] because of their high accuracy and acceptable speed [14], as well as *BRISK* [11] and *ORB* [15] because of their good balance between computation time and accuracy [18]. The first variable of interest was the image size of the scaled-down photograph which should be minimized while keeping an acceptable success rate, as the transfer of images between devices is the most time-consuming part of the whole process. Because of the similar aspect ratios of different smartphone cameras, we use the length of an image's long edge in pixels as a measure for image size and compared sizes between 128 and 2048 pixels (Fig. 4). For each keypoint detector, all suitable feature matching algorithms included in *OpenCV* were compared. Furthermore, different thresholds were tested for each detector/matcher combination. The success rate and processing time of all matcher/parameter combinations were compared by computing matches between the photographs and screenshots in our test data set (Fig. 3) on an *HP EliteBook 850 G4* (*Intel i7* CPU with 2.7 GHz, *Intel HD Graphics 620*, 16 GB RAM).

We found an image size of 512 pixels (long edge) to be the sweet spot as smaller sizes lead to low success rates and larger sizes hardly increase success rate. The fastest matching algorithm (mean: 95 ms, sd: 16 ms) for this image size is an *ORB* keypoint detector (feature limit: 2000), together with a brute force matcher using Hamming distance. This combination achieves a success rate of 89% which we consider accurate enough for use in an interactive application where users can repeat the process until they get a positive result.

The ORB+Hamming algorithm was evaluated using the evaluation data set described above. As we excluded the ten images that we already had used for selecting the best algorithm, the data set contains 16 sets of photos of 58 screenshots (928 photos in total). The evaluation was run on the same hardware as the optimization. Before the evaluation, all photographs were converted to grayscale and scaled down so the long side was 512 pixels wide. No further pre-processing was applied to the images. Each of those images was then passed to the matching algorithm together with the corresponding original screenshot. We measured how well the matching algorithm performs on a realistic data set **(1)**, and how processing time and success rate are affected by the content of the image **(2)** and the phone/screen combination used to capturing the photo **(3)**.

(1) Success Rate and Processing Time. The system could detect matches between screenshot and photograph for 86.9% of the complete data set. Mean computation time was 90 milliseconds (range: 57 – 336 ms, sd: 24 ms). This confirms the results from the optimization step. While not sufficient for applications such as real-time optical tracking, the computation time of the matching process is short enough to be perceived as responsive [8].

(2) Effect of Image Content. As the success of keypoint detection algorithms is dependent on the content of the image, we investigated whether the algorithm selected for ScreenshotMatcher is suitable for all real-world use cases. The screenshots in the evaluation data set were divided up into five categories: graphical user interfaces, text, articles (combination of text an images, e.g. most websites), icons (e.g. a file explorer) and images (Fig. 3). Both processing time (mean: 84 – 93 ms, sd: 11 – 22 ms) and success rate (84% – 93%) were in a similar range for all categories. For seven individual screenshots, a success rate below 75% was found. Those screenshots were spread across all categories but had in common that they were either very cluttered or contained very few recognizable elements.

(3) Effect of Phone and Screen. The evaluation data set contains photos of nine different screens (laptop and desktop monitors) captured with nine different smartphones (total phone/screen combinations: 16). Mean computation time was similar for all combinations with values between 80 and 90 milliseconds (sd: 9 – 15 ms). For the success rate of the matching process, a bigger influence of the phone/screen combination could be observed. Success rates ranged from 71% (*Samsung Galaxy A3 + Sony Vaio* 17" laptop with glossy display) to 97% (*OnePlus One + Dell* 24" matte monitor).

To test ScreenshotMatcher in a real-world context, we asked 19 participants from our computer science department (13 male, 6 female) to use the application over the course of one week however they wanted. During the study, metadata about each screenshot taken (participant ID, timestamps, used matching algorithm, and match success) with ScreenshotMatcher was sent to a log server. No image data was logged to preserve participant's privacy. We also did not collect any feedback during use of the application in order to not affect how participants used it. After the week of use, 14 of the 19 participants answered a questionnaire about their usage of ScreenshotMatcher, their personal assessment of its performance and usability, and which problems occurred during the study. Eleven participants used it to send screenshots from the PC using the phone's instant messenger. Four participants had problems with an unstable connection between phone and PC. Seven participants reported that the wrong region of the screenshot was extracted on some occasions. Suggested improvements were the possibility to further crop the screenshot within the app, annotating the screenshot, recording animations, and integrating ScreenshotMatcher in the default camera app. Twelve of the 14 participants stated that they would continue using ScreenshotMatcher after the study.

A total of 635 images were captured with ScreenshotMatcher over the course of the study. However, one participant alone captured 326 screenshots whereas ten participants captured less than ten screenshots. Mean processing time (from pressing the capture button to the result being displayed on the phone) for successful matches was 878 ms (sd: 806 ms, range: 287 – 6588 ms). Mean processing time of the matching algorithm was 178 ms (sd: 235 ms, range: 41 – 1964 ms), indicating that participants' computers had on average less processing power than our reference hardware. Only 47.4% of screen photos were recognized successfully, much less than in the technical evaluation (85%). As we did not store the screen photos, we do not know for sure what the reasons are.

5 DISCUSSION AND FUTURE WORK

While previous approaches for smartphone-screen interactions mostly employed SURF for feature matching, our evaluation indicates that ORB outperforms SURF significantly for finding features from screen photos in screenshots. A technical evaluation of the matching algorithm used for this application has resulted in a success rate of over 85% and an average processing time of 90 milliseconds which we consider to be sufficient in practice, especially together with the options to re-take the photo or to retrieve a screenshot of the whole screen. In our preliminary real-world evaluation, average response time for ScreenshotMatcher was less than one second, making it suitable for everyday use. The evaluation also showed that real-world recognition rate and processing time are only half as good as results from the technical evaluation would indicate. This indicates room for further improving the processing pipeline and exploring how real-world screen photos differ from our test data set. For example, the success rate for regions with a lot of text could be improved by extending the matching algorithm with text based features such as described by Tsai et al. [19]. Furthermore, applying feature-matching algorithms based on artificial neural networks might be worth considering. One technical limitation of our current implementationis 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. We also plan to further optimize processing time so the system can be used for real-time applications. Source code, data set, and performance measurements are published under an open source license at https://hci.ur.de/projects/screenshotmatcher. 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] Rafael Ballagas, Michael Rohs, and Jennifer G. Sheridan. 2005. Sweep and point and shoot: phonecam-based interactions for large public displays. In *CHI '05 Extended Abstracts on Human Factors in Computing Systems*. Association for Computing Machinery, New York, NY, USA, 1200–1203. https://doi.org/10.1145/1056808.1056876

[3] 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

[4] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. 2006. SURF: Speeded Up Robust Features. In *Computer Vision – ECCV 2006*, Aleš Leonardis, Horst Bischof, and Axel Pinz (Eds.). Vol. 3951. Springer Berlin Heidelberg, Berlin, Heidelberg, 404–417. https://doi.org/10.1007/11744023_32

[5] Sebastian Boring, Manuela Altendorfer, Gregor Broll, Otmar Hilliges, and Andreas Butz. 2007. Shoot & copy: phonecam-based information transfer from public displays onto mobile phones. In *Proceedings of the 4th international conference on mobile technology, applications, and systems and the 1st international symposium on Computer human interaction in mobile technology - Mobility '07*. ACM Press, Singapore, 24. https://doi.org/10.1145/1378063.1378068

[6] G. Bradski. 2000. The OpenCV Library. *Dr. Dobb's Journal of Software Tools* (2000).

[7] 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

[8] Rina A. Doherty and Paul Sorenson. 2015. Keeping Users in the Flow: Mapping System Responsiveness with User Experience. *Procedia Manufacturing* 3 (2015), 4384–4391. https://doi.org/10.1016/j.promfg.2015.07.436

[9] 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

[10] Asifullah Khan, Anabia Sohail, Umme Zahoora, and Aqsa Saeed Qureshi. 2020. A survey of the recent architectures of deep convolutional neural networks. *Artificial Intelligence Review* 53, 8 (Dec. 2020), 5455–5516. https://doi.org/10.1007/s10462-020-09825-6

[11] Stefan Leutenegger, Margarita Chli, and Roland Y. Siegwart. 2011. BRISK: Binary Robust invariant scalable keypoints. In *2011 International Conference on Computer Vision*. IEEE, Barcelona, Spain, 2548–2555. https://doi.org/10.1109/ICCV.2011.6126542

[12] 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

[13] Madhavapeddy, Anil, Scott, David, Sharp, Richard, and Upton, Eben. 2004. Using Camera-Phones to Enhance Human–Computer Interaction. In *Sixth International Conference on Ubiquitous Computing (Adjunct Proceedings: Demos)*. 2.

[14] Moritz Riegler. 2015. *A study and comparison of feature matching*. Master's thesis. Hochschule München. https://elib.dlr.de/100301/

[15] 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

[16] Seow, Steven C. 2008. *Designing and Engineering Time: The Psychology of Time Perception in Software*. Addison-Wesley Professional.

[17] Ben Shneiderman, Catherine Plaisant, Maxine Cohen, Steven Jacobs, Niklas Elmqvist, and Nicholas Diakopoulos. 2016. *Designing the User Interface: Strategies for Effective Human-Computer Interaction* (6th ed.). Pearson.

[18] Shaharyar Ahmed Khan Tareen and Zahra Saleem. 2018. A comparative analysis of SIFT, SURF, KAZE, AKAZE, ORB, and BRISK. In *2018 International Conference on Computing, Mathematics and Engineering Technologies (iCoMET)*. IEEE, Sukkur, 1–10. https://doi.org/10.1109/ICOMET.2018.8346440

[19] 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