https://doi.org/10.2352/ISSN.2470-1173.2020.8.IMAWM-309

Real-time Whiteboard Coding on Mobile Devices

Xunyu Pan, Colin Crowe, Toby Myers, and Emily Jetton
Department of Computer Science and Information Technologies, Frostburg State University, Frostburg, Maryland, USA

Abstract

Mobile devices typically support input from virtual keyboards or pen-based technologies, allowing handwriting to be a potentially viable text input solution for programming on touchscreen devices. The major problem, however, is that handwriting recognition systems are built to take advantage of the rules of natural languages rather than programming languages. In addition, mobile devices are also inherently restricted by the limitation of screen size and the inconvenient use of a virtual keyboard. In this work, we create a novel handwriting-to-code transformation system on a mobile platform to recognize and analyze source code written directly on a whiteboard or a piece of paper. First, the system recognizes and further compiles the handwritten source code into an executable program. Second, a friendly graphical user interface (GUI) is provided to visualize how manipulating different sections of code impacts the program output. Finally, the coding system supports an automatic error detection and correction mechanism to help address the common syntax and spelling errors during the process of whiteboard coding. The mobile application provides a flexible and user-friendly solution for real-time handwriting-based programming for learners under various environments where the keyboard or touchscreen input is not preferred.

Introduction

The fast-growing information technology industry in the United States is expected to outpace the overall economy's growth in the coming years. Meanwhile, the IT job market powered by advances in emerging technologies such as AI, big data, cloud computing, cybersecurity, blockchain, and machine learning, creates tens of thousands of jobs each month across the country. Today, college students and working professionals have been learning to improve their programming skills to fit for those high paying IT careers. Moreover, many U.S. states have ramped up efforts to increase K-12 education on computer science subjects, including computer programming. Thanks to the rapid advance of mobile computing technologies, high quality and low-cost mobile devices such as laptops and smartphones are increasingly used in everyday activities, making the study of programming extremely convenient these days. Mobile devices typically support the input from a virtual keyboard or a pen-based technology such as a stylus, allowing handwriting to be a potentially viable text input solution for programming on a touchscreen, which is especially convenient for children or adults who have no coding experience.

Many research projects on handwriting recognition with touchscreen have been conducted recently. The major problem, however, is that handwriting recognition systems are built to take advantage of the rules of natural languages rather than programming languages. In addition, mobile devices are also inherently restricted by the limitation of screen size and the inconvenient use of a virtual keyboard.

To address these issues, we explore the use of handwriting input for human-computer interaction and address a particular problem of the recognition of source code written directly on a whiteboard or a piece of paper. To this end, we create a novel handwriting-to-code transformation system on a mobile platform to recognize and analyze handwritten code. Unlike most handwritten code recognition systems reading from the touchscreen device, the ultimate goal of the proposed system is to support whiteboard coding as a means of programming on mobile devices. First, the system recognizes and further compiles the handwritten Java code into an executable program in real-time using Optical Character Recognition (OCR) technologies. A two-stage diagnostic process is conducted to enhance the source code recognition performance. Second, a friendly graphical user interface (GUI) is provided to visualize how manipulating different sections of code impacts the program output and hence demonstrates a more interactive method of programming learning. Finally, given the high possibility of syntax errors against grammar rules for handwritten code, the coding transformation system supports an automatic error suggestion and correction mechanism to help address the common syntax and spelling errors that occurred during the process of whiteboard coding.

We evaluate the performance of the proposed handwritten code processing system based on the Character Recognition Rate and Error Correction Number over a dataset of Java source code samples collected from participating students. Experimental results demonstrate that the visualization of code transformation and the mechanism of error correction significantly enhance the user experience with mobile programming. The presented mobile application provides a flexible and user-friendly solution to the real-time whiteboard handwriting-to-code transformation under various environments where keyboard or touchscreen input is not preferred.

Related Work

Writing source code into Integrated Development Environments (IDEs) is one of the major steps for software development. However, keyboard input may not always fit in every situation. Alternative methods for source code input have been intensively studied over the years. Traditional approaches involve the *Spoken Programming* based on specially designed programming languages [1, 2, 3] or some earlier proposals based on natural languages [4, 5]. However, the input from speech is not always an appropriate solution, especially in various situations where the surrounding environments are complicated or privacy is required [6]. Due to these constraints, *Spoken Programming* is implemented more frequently to enhance the programming accessibility for developers with visual disabilities [7].

More recently, *Handwriting Programming* has attracted increasing attention from both academia and industry. The task for converting written or printed text into digital text is generally called OCR for optical character recognition. OCR works better with high-quality printed materials than with handwriting. Many recent techniques [8, 9, 10] on source code recognition are implemented using existing natural language handwriting recognition engines with additional post-processing. However, these applications typically perform on touchscreen devices and do not support automatic error correction on input source code.

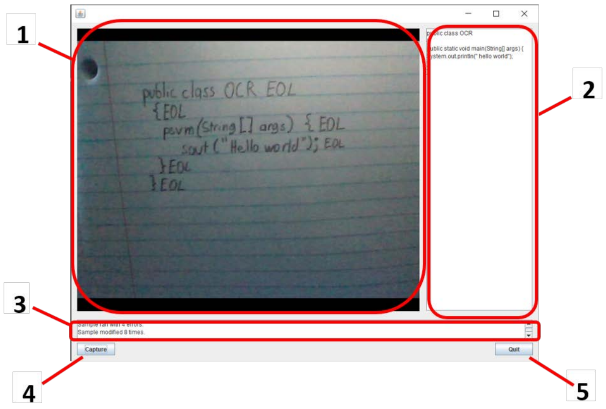
Methods

Nowadays, high quality and low-cost mobile devices with powerful integrated cameras such as laptops and smartphones are increasingly used in everyday activities [11, 12], making the study of programming extremely convenient for children and adults who have no coding experience. Source code input from a virtual keyboard or pen-based technology is restricted by the limitation of screen size and the inconvenient use of a virtual keyboard. Due to the fact that computer programs are written using standard syntax with limited words, the performance of handwriting programming can be greatly improved based on the highly formatted input such as source code.

In this work, we introduce a novel handwriting programming system implemented on a mobile platform to recognize and analyze handwritten Java source code. Different from existing handwritten code recognition systems that read directly from the touchscreen, the proposed system is specifically designed to transform a piece of source code written on whiteboard or paper into an executable software program using mobile devices. The application also supports an automatic error suggestion and correction mechanism to help address the common syntax and spelling errors when users write source code. The purpose of the system is to allow users to quickly convert their program design into viewable execution results.

When a user selects to transform handwritten source code into an executable program, he or she points a mobile device at the source code on the whiteboard or paper and interacts with the mobile system through a control panel as shown in Figure 1. The integrated camera captures a rectangle region around the source code and creates a bitmap image from the camera feed. The newly generated image and its related information are saved locally and further processed through a set of functional components of the proposed system.

The handwriting-to-code transformation system consists of four major functional components: (a). A *Handwriting Recognition* module to convert handwritten source code into a string of readable characters called *OCR Text*. The system communicates with Google's Cloud Vision API OCR service to retrieve the text information from the captured handwriting image; (b). A *Text Modification* module to replace any custom abbreviations of the retrieved OCR Text. Depending on user customization, the system can define various abbreviations for frequently used programming keywords; (c). An *Error Correction* module to fix common syntax and spelling errors in the source code written by inexperienced learners. The final version of the input text after this process is referred to as *Revised Text*; (d). A *Code Execution* module to compile and execute the Revised Text-based



```
public class OCR EOL
 {EOL
  psvm(String[] args) {EOL
    sout("Hello world"); EOL
  }EOL
 }EOL
```

```
blic class OCR

ublic static void main(String[] args) {
ystem.out.println(" hello world");
```

```
Sample ran with 4 errors.
Sample modified 8 times.
```

Figure 1. The Graphical User Interface (GUI) of the proposed handwriting-to-code transformation system: (1) A live camera preview of a piece of paper with handwritten source code, (2) A view box to display the recognized source code, (3) A view box to display the execution results of the recognized code, (4) A button allowing the user to capture and process an image of the paper with handwritten source code, and (5) A button to exit the application.

source code file to output program results. The system GUI helps to display both the recognized source code and the execution results. Shown in Figure 2 is the high-level logic overview of the described handwriting-to-code transformation system.

Handwriting Recognition

Figure 1 demonstrates the main GUI of the proposed system. Users are allowed to use this interface to view the target piece of paper/whiteboard with handwritten source code, the recognized source code represented by the Revised Text, and the program execution results. Using an external Webcam API, the GUI creates camera-view objects that allow the application to initialize and display live camera feed. The main interface has two notable options: *Capture* and *Quit*. The *Capture* option creates a new sample image object. This object holds numerous details about the handwriting image taken from the live camera feed. The *Quit* option, on the other hand, performs cleanup work to remove temporary files that were created during the program's previous executions to prevent overlap errors on future tasks.

The captured handwriting image is saved locally and then transmitted to Google's Cloud Vision API OCR Service. This service loads the image, searching for blocks of handwritten text that it can recognize. Upon recognition of dense text, the OCR service examines the text blocks to parse individual characters, constructing the initial result containing one string of all the parsed characters. We refer to this string as *OCR Text*. The recognized OCR Text results are reliable on typed text and precise for uniform proportions, signs, and other digitally-manufactured textural contents. However, the recognition results are sometimes inaccurate for the average user's handwriting due to varying degrees of legibility, readability, and flavor. Hence, additional enhancement processing is required to obtain more accurate translation between handwritten code and OCR Text.

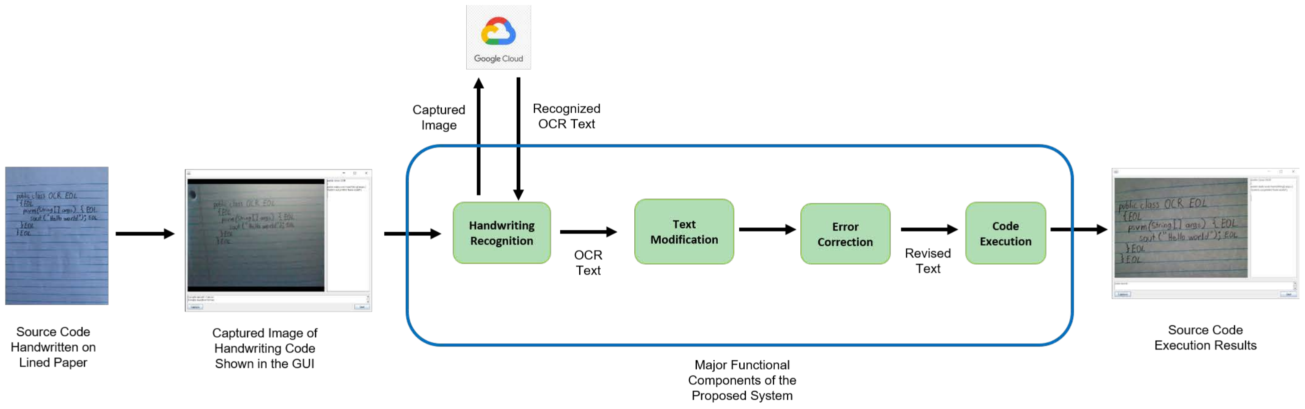

Google Cloud
Captured Image
Recognized OCR Text
Handwriting Recognition
OCR Text
Text Modification
Error Correction
Revised Text
Code Execution
Source Code Handwritten on Lined Paper
Captured Image of Handwriting Code Shown in the GUI
Major Functional Components of the Proposed System
Source Code Execution Results

Figure 2. *The overview of the proposed handwriting-to-code transformation system: A user first writes source code on a paper/whiteboard. The integrated camera then captures an image of the target paper/whiteboard with handwritten code. The captured image is sent to Google's Cloud Vision API OCR service to retrieve recognized OCR Text. The system then performs a two-stage text diagnostic process consisting of Text Modification stage and Error Correction stage to generate the final Revised Text. The Revised Text is used to create a source code file which can be compiled and executed to output results displayed to the user. In addition, the user can view captured image, recognized code, and execution results all together in system GUI.*

Text Modification

Upon retrieval of the OCR Text, the proposed system performs an additional diagnostic process to clean up the OCR Text and infer the correct input from the user. This process involves two stages. The first one is called *Text Modification*. In this stage, the program will parse the OCR Text and fix any user-defined abbreviations that have been customized in the system. For example, *"psvm"* is short for *"public static void main"* and *"EOL" signifies an "End of Line"*. The Text Modification step has no intention to handle innate errors which could prevent the input source code from successful compilation, but the employment of programming keywords integrated into the system provides more reliable results.

Error Correction

The second stage is called *Error Correction*. In this stage, the developed program scans the input lines of modified text obtained from the first stage and attempts to recognize what Java command is supposed to be occurring based on unique details. For example, *"public class main E"* should be converted into *"public class main {"*. The proposed system can identify and correct syntax errors such as capitalization errors and incorrect class names, as well as common spelling errors. Misspelled abbreviations from the first stage can be fixed as well. The final version of this text is referred to as *Revised Text*.

Now that the OCR Text has been converted to Revised Text, the assumption becomes that it should be prepared to endure the Java compilation process and run correctly. A new file is created for the Revised Text called "OCR.java". In the current system, it is entirely possible that this newly generated file is not properly compiled for some reason.

Code Execution

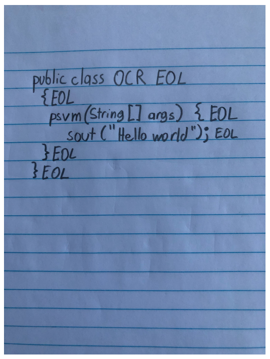
A new Java source code (.java) file is created using the Revised Text. The default Java compiler integrated in the system builds the source code and executes the corresponding program. Both the Revised Text and compilation result are stored in the system. At this point, the entire handwriting-to-code transformation process is completed. The GUI now takes control and displays the Revised Text and compilation fields on two panes of the front-end. If the compilation was successful, the execution result is displayed at the bottom of the screen. Otherwise, a simple error message is displayed. The GUI remains in a live state, allowing the user to capture the other source code sample as desired.

Results

The handwriting-to-code transformation system was written in Java language with the support of an external Camera API [13]. The proposed system was developed on Windows 10(X64) operating system. The standard Java JDK 8 is used, and all programming is performed within the IntelliJ Idea Development Environment. The system is run on a Dell XPS 15 2-in-1 laptop which has an Intel Core i5-8305G processor running at 2.8 GHz with 8GB of memory.

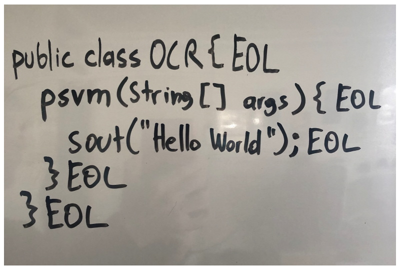
We evaluate the system performance from two perspectives: *Character Recognition Rate* and *Error Correction Number*. The Character Recognition Rate measures the average rate of the characters successfully recognized in the written source code. The Error Correction Number measures the average number of syntax and spelling errors successfully identified and corrected.

With default system settings, we tested the performance of our system on successful code recognition and automatic error correction in different source code input scenarios. Character Recognition Rate (CRR) and Error Correction Number (ECN) highly relies on the specific type of paper and the local lighting conditions. Testing samples used for our experiments were collected in 4 indoor scenarios under various local lighting conditions: (a). Typed Paper: the source code is typed on white paper; (b). Typed Paper (varied text fonts): the source code is typed with the combination of varied text fonts on white paper; (c). Lined Paper: the source code is handwritten on lined paper, and (d). Whiteboard: the source code is handwritten on whiteboard. For testing purposes, we collected 100 samples of source code handwritten or typed by high school students and amateur program-



```
public class OCR EOL
 {EOL
  psvm(String[] args) { EOL
   sout("Hello world"); EOL
  }EOL
 }EOL
```

(a)



```
public class OCR{ EOL
  psvm(String[] args){ EOL
    sout("Hello World"); EOL
  }EOL
}EOL
```

(b)

Figure 3. *Two samples of source code in our testing collection: (a) A sample of source code handwritten on a piece of lined paper, and (b) A sample of source code handwritten on a whiteboard.*

mers in 4 indoor scenarios. Figure 3 shows two samples of source code in our collection, which are handwritten on a piece of lined paper and on a whiteboard, respectively.

Experimental results shown in Table 1 demonstrate that the proposed system achieves very high CRR values for all indoor writing scenarios. As a comparison, the two scenarios for typed code have the highest CRR values since the typed characters are easier for OCR recognition. Meanwhile, the other two scenarios for handwritten code also obtain satisfactory CRR values. These results show that the proposed system is useful in recognition of both typed and handwritten source code in various indoor scenarios. To address the syntax and spelling errors introduced by both the users and the OCR service, we further tested the system performance on error detection and correction. As shown in Table 1, the system can detect and correct the errors for all writing scenarios. Again, more errors are corrected in the handwritten scenarios than those corrected in the typed scenarios. This is expected, as more errors can be introduced during the process of OCR recognition for the handwritten scenarios.

Conclusions

Today, learning to code on mobile platforms is becoming more popular for younger generations. Mobile devices, however, are inherently restricted by the limitation of screen size and the inconvenient use of virtual keyboards. Alternative source code input methods such as spoken programming are not appropriate for all situations. In this work, we introduce an effective application developed on a mobile platform aiming to support real-time whiteboard coding. The proposed application first captures an image of handwriting source code using an integrated camera. The mobile system then delivers the image to a remote OCR cloud service and retrieves a string of identified characters. To enhance the recognition performance, a two-stage diagnostic process is conducted on the retrieved text string to remove various errors: (a). Text Modification stage to replace any custom abbreviations and (b). Error Correction stage to fix common syntax and spelling errors. Finally, the Revised Text is used to create a source code file which can be further compiled and executed to output program results. The described application is portable and user-friendly for young students and working professionals to improve their programming skills. Experimental results have demonstrated that the proposed system is helpful to recognize, correct, and execute source code written on paper or whiteboard.

Though having achieved promising performance in converting handwritten source code into executable programs, our method relies on the recognition performance of the remote OCR service. Some potential extensions are achievable for the proposed system in the near future: (a). Create an integrated OCR module which can be trained using handwriting samples over time to achieve better recognition results; (b). Build a local database storing past syntax and spelling errors for future error detection cases; (c). Extend the application to smartphones and other mobile platforms to serve more users. The ultimate goal of this work is to demonstrate the feasibility and the potential for converting any mobile device into a flexible tool that can support real-time and reliable whiteboard coding and execution.

Acknowledgments

This work was partially supported by the Frostburg State University Provost's Experiential Learning Enhancement Fund Program (PELEF) and by a FSU Foundation Boxley Faculty Research Award (# 36046).

Special thanks to Kent County High School of Maryland and Beth H. Jelinek, M.D., for assistance with the collection of handwriting samples.

References

[1] B. M. Gordon, "Developing a language for spoken programming," in *Proceedings of the Twenty-Fifth AAAI Conference on Artificial Intelligence*, San Francisco, CA, 2011.

[2] A. Désilets, D. C. Fox, and S. Norton, "Voicecode: An innovative speech interface for programming-by-voice," in *ACM CHI 06 Extended Abstracts on Human Factors in Computing Systems*, Montréal, Canada, 2006.

[3] A. Begel and S. L. Graham, "Spoken programs," in *IEEE Symposium on Visual Languages and Human-Centric Computing*, Dallas, TX, 2005.

[4] S. C. Arnold, L. Mark, and J. Goldthwaite, "Programming by voice, vocalprogramming," in *Proceedings of the Fourth International ACM Conference on Assistive Technologies*, Arlington, VA, 2000.

[5] D. Price, E. Rilofff, J. Zachary, and B. Harvey, "Naturaljava: A natural language interface for programming in java," in *Proceedings of the 5th International Conference on Intelligent User Interfaces*, New Orleans, LA, 2000.

[6] X. Pan, X. Zhang, and S. Lyu, "Detecting splicing in digital audios using local noise level estimation," in *IEEE Interna-*

Table 1. Comparison of the Character Recognition Rate (in percentage) and Error Correction Number of the proposed system in four indoor coding scenarios under various local lighting conditions.

	Character Recognition Rate (CRR)	Error Correction Number (ECN)
Typed Paper	96.91	1.00
Typed Paper (varied text fonts)	94.30	1.25
Lined Paper	88.86	2.13
Whiteboard	89.37	3.45

tional Conference on Acoustics, Speech and Signal Processing (ICASSP), Kyoto, Japan, 2012.

[7] V. Potluri, P. Vaithilingam, S. Iyengar, Y. Vidya, M. Swaminathan, and G. Srinivasa, "Codetalk: Improving programming environment accessibility for visually impaired developers," in *Proceedings of the ACM CHI 18 Conference on Human Factors in Computing Systems*, Montréal, Canada, 2018.

[8] MyScript, "Myscript programs," http://www.myscript.com.

[9] Q. Zhi and R. Metoyer, "Recognizing handwritten source code," in *Proceedings of the 43rd Graphics Interface Conference*, Edmonton, Canada, 2017.

[10] M. Kherallah, N. Tagougui, A. M. Alimi, H. E. Abed, and V. Margner, "Online arabic handwriting recognition competition," in *International Conference on Document Analysis and Recognition*, Beijing, China, 2011.

[11] X. Pan and C. Gill, "Multimedia instant messaging with real-time attribute-based encryption," in *IS&T Symposium on Electronic Imaging (IS&T-EI)*, Burlingame, CA, 2017.

[12] X. Pan and S. Lyu, "Region duplication detection using image feature matching," *IEEE Transactions on Information Forensics and Security (TIFS)*, vol. 5, no. 4, pp. 857–867, 2010.

[13] B. Firyn, "Webcam capture api," https://github.com.

Author Biography

Xunyu Pan received the B.S. degree in Computer Science from Nanjing University, China, in 2000, and the M.S. degree in Artificial Intelligence from the University of Georgia in 2004. He received the Ph.D. degree in Computer Science from the State University of New York at Albany (SUNY Albany) in 2011. From 2000 to 2002, he was an instructor with Department of Computer Science and Technology, Nanjing University, China. In August 2012, he joined the faculty of Frostburg State University (FSU), Maryland, where he is currently an Associate Professor of Computer Science and the Director of Laboratory for Multimedia Communications and Security. Dr. Pan is the recipient of 2011∼2012 SUNY Albany Distinguished Dissertation Award and 2016 FSU Faculty Achievement Award in Teaching. His publications span peer-reviewed conferences, journals, and book chapters in the research fields of multimedia security, image analysis, medical imaging, communication networks, computer vision and machine learning. He is a member of the ACM, IEEE, and SPIE. (Corresponding Author: xpan@frostburg.edu)

Colin Crowe is a senior at Frostburg State University currently pursuing a B.S. degree in Computer Science and minor in Secure Computing and Information Assurance, as well as planning to further his education with FSUs Computer Science Masters Program. Colin participates in many extracurricular activities, including participating in the Phi Kappa Phi honor society and serving as Vice President of the FSU Computer Club. In addition, he works as a Computer Science Tutor and is an intern at a local Web-Development company. He has accepted a position as a full-time Software Engineer post-graduation.

Toby Myers is a veteran of the U.S. Army. In 2018, he received an A.S. degree in Computer Science from Allegany College of Maryland in Cumberland, MD. Toby is currently pursuing a B.S. degree in Computer Science at Frostburg State University in Frostburg, MD. He is a member of Upsilon Pi Epsilon Computer Honor Society and Salute Veterans National Honor Society.

Emily Jetton is a senior in the Computer Science program at Frostburg State University. After graduating, she has accepted a position working in the Johns Hopkins Applied Physics Lab while pursuing a Masters Degree through the Johns Hopkins Professionals Program. She is a member of FSUs Computer Club, a leading officer in the Women in Computer Science Club and is a member of the Phi Kappa Phi Honor Society. Her work experience includes working as a Computer Science and Statistics Tutor as well as an intern for the United States Department of Justice.

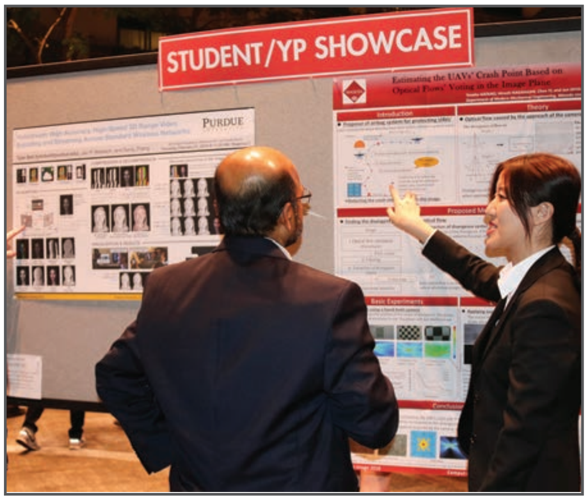
JOIN US AT THE NEXT EI!

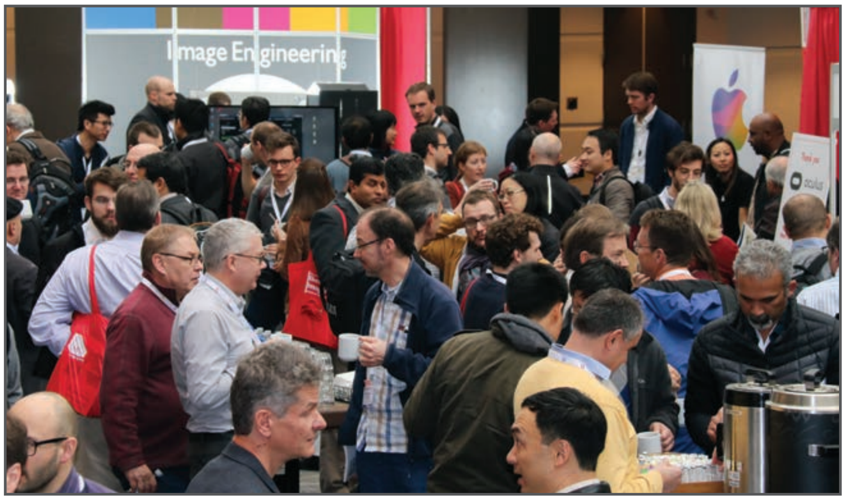
IS&T International Symposium on

Electronic Imaging

SCIENCE AND TECHNOLOGY

Imaging across applications . . . Where industry and academia meet!





AR is a fundamental and transformative interface and platform.
20+ years from my initial working demo to today's AR displays
Optimistic that it won't take another 20 to reach ubiquitous consumer AR!

• SHORT COURSES • EXHIBITS • DEMONSTRATION SESSION • PLENARY TALKS •
• INTERACTIVE PAPER SESSION • SPECIAL EVENTS • TECHNICAL SESSIONS •

www.electronicimaging.org

IS&T
imaging.org