https://doi.org/10.2352/ISSN.2470-1173.2019.14.COLOR-087

Automatic Detection of Scanned Page Orientation*

Zhenhua Hu[1], Peter Bauer[2], Todd Harris[2], Jan Allebach[1]
[1]School of Electrical and Computer Engineering, Purdue University, West Lafayette, Indiana, 47907
[2]HP Inc., Boise, Idaho, 83714

Abstract

With the increasing demand to scan text documents and old books, having a scanner that could automatically detect the orientations of the scanned pages would be greatly beneficial. This paper proposes a fast method to detect orientations based on a support vector machine (SVM), using features developed for each connected component on the scanned page. Results show that the algorithm can achieve an accuracy of 99.2% in orientation detection and 98.2% in script detection for pages scanned at 200 dpi.

Introduction

Recently, with the trend of the paperless office, lots of documents are being scanned every day. Also, the need to preserve old ancient books or other materials requires a lot of scanning. Moreover, correct orientation is very important before further processing such as optical character recognition (OCR). When scanning, the page could be in one of the 4 orientations: 0, 90, 180, and 270 degrees. Sometimes it is difficult or even impossible to know the orientations of all pages while scanning. Manually aligning them would be tedious and very time-consuming. As a result, it would be great if the scanner could automatically detect the orientations and align them accordingly during the scanning process. Since the page could be in portrait or landscape mode, the scanner should be able to work in both cases. Further, since this demand is worldwide, it would be best if the scanner could detect the orientation of pages in various language scripts, such as Roman, Chinese, Devanagari, Japanese, and Korean. Furthermore, for some documents like annual reports which contain a lot of numbers, we also need the scanner to detect their orientations correctly.

To accurately detect one page's orientation, we developed an algorithm to find features of text characters, and to predict its script and orientation using support vector machine (SVM). The algorithm starts by turning the scanned image into a grayscale image, and then partitioned it into 16 sub-images with equal width and height. Then, Otsu's method [1] is applied to each sub-image to get a binary image. After getting the binary image, connected components labeling and analysis are applied to the whole page to get connected components of text characters and reject non-text connected components. Once all text connected components are identified, text features are calculated for each of them, and their normalization is the page's feature vector. Finally, the feature vector is then fed into a support vector machine (SVM) to do training and prediction.

*Research supported by HP Inc., Boise ID 83714

13 scripts are taken into consideration in this work, namely, Chinese, Devanagari, Japanese, Korean, Numeral, English, French, German, Greek, Italian, Portuguese, Russian, and Spanish. Based on their similarities and differences, a script hierarchy is developed to better decide the script of a page via SVM. To reduce the number of script classes, we consider English, French, German, Greek, Italian, Portuguese, Russian, and Spanish as one script: Roman, since they look very similar to each other. After the script of a page is identified, its orientation will be identified afterwards.

Several methods have been proposed to detect orientations of document pages. Guo et al. [2] used three vertical component runs to get a 96-dimensional feature vector, which was then fed into an SVM to determine the Up/Down orientations of text pages. van Beusekom et al. [3] proposed a model to find the text line, the number of ascenders and descenders to determine one page's orientation. They tested their model on Batin script and achieved an overall accuracy of 98.8% on all orientations. They also tested the method on Japanese script but only got an accuracy of 85.7%. Roy et al. [4] proposed an algorithm to detect one page's up/down orientation based on text-asymmetry ratios computed from strip-based projection files. The algorithm also needs to find the ascenders and descenders based on text lines. But for some pages written in traditional Chinese, where text characters are written vertically, these text line based methods might not work well. And also for some Asian scripts like Chinese and Japanese, there are no obvious ascenders or descenders. Fan et al. [5] invented a method that utilized the OCR method to find characters of 'i' or 'T' to determine the pages orientation. Ghosh et al. [6] proposed a method to identify the script and orientation of 11 official scripts in India. Their feature vector is composed of a reservoir area, a white hole area, and horizontal and vertical white-black transitions. They also designed a complex hierarchy to detect scripts. Their method achieved fairly high accuracies; but it is specially designed for Indian languages. Rashid et al. [7] proposed to use a convolutional neural network (CNN) to identify scripts. They achieved above 95% recognition accuracy at the connected component level on datasets of Greek-Latin, Arabic-Latin, and Antiqua-Fraktur documents. Their CNN contains 2 convolutional layers with 4 and 8 feature maps, followed by 2 sub-sampling layers. For the 3 script pairs, they achieved an accuracy of 98.40%, 95.61%, and 96.61%, respectively. But they did not try detecting the orientation.

Lu et al. [8] proposed an algorithm to detect document images' up/down orientation through document vectorization. They also used the same feature vector to detect page script. The idea is to find a feature vector vertically on a text character, and then convert it into a vector. Orientation and script detection of a document image are determined based on distances between the de-

tected document vector and the pre-constructed vector templates. The algorithm was tested on Chinese, Roman, Arabic, and Chinese scripts and achieved good results. Based on their work, Jain et al. [9, 10] developed an algorithm that can detect the 4 orientations mentioned above and multiple scripts using an artificial neural network. In their algorithm, they constructed features from the horizontal direction as well as the vertical direction. They also introduced foreground pixel densities as part of the feature vector. Their method achieved fairly good accuracies on Chinese, Japanese, and Korean, but performed rather badly on Devanagari, Numeral and some Roman scripts. Our work is mainly based on Jain's algorithm. While keeping their features, we also introduce features from the side background areas in a text character's bounding box. Moreover, we develop new methods to rule out non-text connected components that might affect detection accuracy.

Methodology

Overall Algorithm

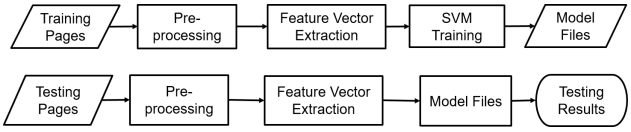


Figure 1. *Overall algorithm workflow.*

The algorithm workflow is shown in Fig. 1. Document pages are first scanned into color images, and then transformed into grayscale. Then Otsu's binarization is applied to get binary images. After connected components labeling, we apply methods to remove non-text connected components. Finally, feature vectors are extracted and fed into SVM to do training and prediction. After training, we save all the parameters into a self-defined header file. In prediction, the algorithm determines an image's script first, and then the orientation based on these header files.

Image Binarization

The image is binarized based on Otsu's method [1]. We choose this method because it is fast and effective regarding document images. The algorithm assumes that an image contains two classes of pixels (foreground pixels and background pixels) that follow a bi-modal histogram. It computes the histogram and probabilities of each intensity level, steps through all possible thresholds from the lowest to highest gray level, and calculates each threshold's intra-class variance. The optimum threshold separating the two classes is the one with minimal intra-class variance. In this case, because the sum of pairwise squared distances is constant, their inter-class variance is maximal.

In order to get better local details, the image is equally separated into 16 (4×4) sub-images and Otsu's method is applied to each sub-image independently. The foreground and background areas are determined based on the assumption that the background area always constitutes a larger portion than the foreground area.

Fig. 2 shows an example binarization result. From the two enlarged areas on the right, we can see that it has a good performance on text areas. But non-text areas can result in a lot of non-text connected components, which will have an unpredictable effect on the image's feature vector. So we need to get rid of them in the connected component analysis part.

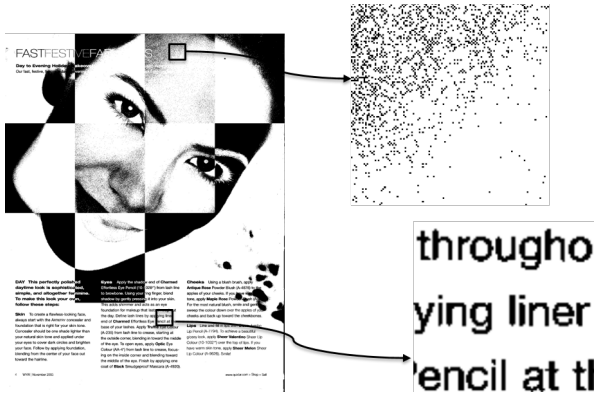


Figure 2. *Examples of binarization result.*

Connected Components Analysis

Since our feature vector is extracted from text characters, in this part we'll remove non-text connected components and keep text connected components.

A document page may contain pictures, tables, or other non-text areas [11]. Normally, some of them will stay as foreground pixels after Otsu's algorithm. In this work, several constraints are introduced to remove non-text areas, based on their characteristics that are distinct from texts.

Size Limit

Since sizes of text characters are usually in a small range, we can utilize it to remove very large or small connected components by setting limits on their width and height. Explicitly, the size limits are (all parameters are chosen empirically):

(a) Lower limits. Among width and height, one must be larger than *0.01* × resolution and the other must be larger than *0.03* × resolution. We set two thresholds because some text connected components have different lengths and widths.

(b) Upper limits. We set hard and dynamic limits on the maximum size of connected components. For hard limits, one connected components width should not be larger than *0.45555* × image width. Similarly, its height should not be larger than *0.45555* × image height. For dynamic limits, a connected component's size must not exceed *2.775* × average of the connected components' (width + height).

Transition Limit

Normally, a text character in any script should contain both foreground and background pixels. Usually, the number of foreground-background transitions is below a certain threshold. So we can employ this to rule out non-text connected components. This could also be used to eliminate text connected components which have multiple characters bonded together when the scanning resolution is low or the page itself has low quality.

For a connected component, imagine there is a line passing through it vertically or horizontally. We call it a transition if the line encounters a foreground pixel from background pixels.

In this work, we count the number of transitions through the center of a connected component both horizontally and vertically. Empirically, the limit is set to be 8 in both directions. And a

connected component is to be removed if the transition number in either direction exceeds the limit.

Besides, in document images where text characters are written on a large picture, like a magazine cover, the picture is often rendered with halftone patterns. As a result, when we apply Otsu's method, we can often find connected components with a halftone pattern; and these connected components often meet the size limits, as shown in Fig. 3. Although applying a low-pass filter to the whole image might be the easiest to remove them, this can also blur the edges [12], making some text characters bond together when the scanning resolution is low. So a new method is required to remove these connected components.

Figure 3. *Examples of connected components rendered with a halftone pattern. We can see a lot of interleaved pixels in these connected components after binarization, and they cannot be identified by the size limits.*

Considering the fact that in connected components rendered with a halftone pattern, foreground and background pixels are always interleaved, the notion of horizontal or vertical transition density is advocated. The way to calculate horizontal transition density is: first horizontally count the total number transitions along all rows in the bounding box, then divide this number by the bounding box height. The vertical transition density is calculated similarly, except that the transitions are counted along the vertical direction, and the denominator is the bounding box width.

Empirically, the thresholds are set to both be *1.5* in both directions. Any connected components with either transition density larger than *1.5* will be removed.

Also, considering the fact that some halftone-rendered connected components have only one pixel at each bounding box edge, while text characters have multiple foreground pixels on at least one edge, we can also eliminate these connected components.

Aspect Ratio

Since text characters' aspect ratios are within a certain range, we set the ratio to be within the range of $\frac{1}{6}$ to *6* to rule out non-text connected components.

Connected Component Analysis Result

Fig. 4 shows the connected component analysis result for Fig. 2. We can see that connected component analysis removes most of the non-text connected components while keeping most of the text characters; and these would be enough to do feature vector extraction.

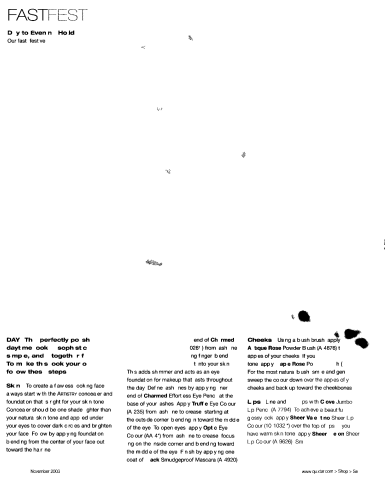


Figure 4. *Example of connected components analysis result.*

Feature Vector Extraction

After text connected components are found, our next step is to extract a feature vector that represents the document image. The feature vector is formed by concatenating by 4 vectors: the vertical document vector (VDV), the horizontal document vector (HDV), the zonal document vector (ZDV), and the profile document vector (PDV). These four vectors are computed by normalizing 4 features of every text connected component: the vertical component run (VCR), the horizontal component run (HCR), the zonal document run (ZDR), and the profile component run (PCR), respectively [10]. In this section, we'll talk about how to get these features.

Vertical Document Vector

The method to calculate vertical component run (VCR) was first brought up by Lu et al. [8] to detect scripts like Arabic, Chinese, Korean and Roman. The idea is to vertically separate the bounding box into three equal areas, called the top zone, the middle zone, and the bottom zone. Then, assume that there is a virtual scan line passing through the center of a connected component. We define a transition as the passing of this line from a background pixel to a foreground pixel. We record the number of transitions and their distributions, as shown in Fig. 5.

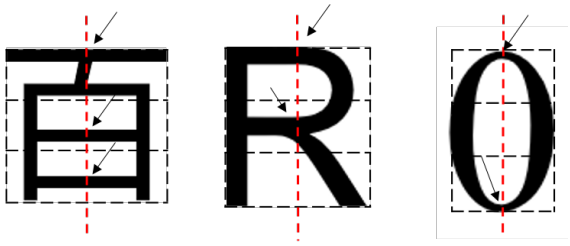


Figure 5. *Examples of vertical transitions on connected components. The bounding boxes are separated into three equal zones vertically. The vertical dashed line in the middle is the imaginary line through the connected component's center. And the arrows in the picture show the locations of transitions.*

Using the method in [8], for the three examples mentioned in Fig. 5, the first Chinese character's VCR is calculated

as [00100000 10000000 01000000 00100000]. For the number 0, its VCR is [01000000 10000000 01000000 00000000]. And the character R's VCR is [01000000 10000000 00000000 01000000].

Normally, a document contains a large number of text characters. Each of them has its own VCR. Next we calculate the sum of them as a feature vector for the whole page. The feature vector is called vertical document vector (VDV), and it can also be normalized in the way mentioned in [8] to reduce the document length effect.

Horizontal Document Vector

Based on the method of VCR and VDV, the idea of getting the horizontal component run (HCR) and horizontal document vector (HDV) are introduced to detect document pages at 90 or 270 degrees orientation. In HCR and HDV, a connected component's bounding box is separated horizontally with a horizontal line passing through its center. And the we record the horizontal transition distributions, as shown in fig. 6.

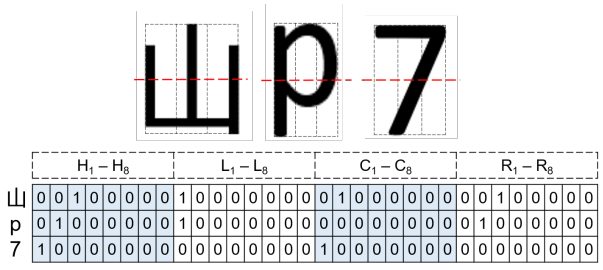


Figure 6. *Examples of vertical transitions on connected components. The bounding boxes are separated horizontally into three equal zones. The horizontal dashed line in the middle is the imaginary line through the connected component's center. The table at the bottom contains the HCRs of the characters.*

Zonal Density Vector

We also introduced the zonal density vector (ZDV) as part of the feature vector based on [9, 10]. The ZDV is obtained through the zonal density run (ZDR), which reflects the distribution of foreground pixel densities. The steps to get ZDR are:

(1) Divide a connected component's bounding box area equally into *9 (3 × 3)* small areas.

(2) For each small area, calculate its foreground pixel density (fd) follows this equation:

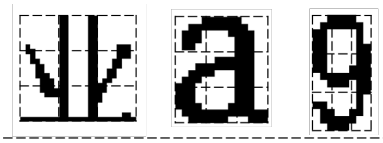
$$fd = \frac{numForePixel}{numAreaPixel} \times 100 \tag{1}$$

where *numForePixel* refers to the number of foreground pixels in a certain area, and *numAreaPixel* is the total number of pixels in the area. Here we multiply by *100* to put the value in range [0, 100]. The small area is iterated first row wise, and then column wise. Their *fds* are denoted as $fd_1, fd_2, ..., fd_9$.

(3) We concatenate all values calculated in (2), to get a ZDR with 9 elements.

Fig. 7 shows ZDR examples for a Chinese character, the letter 'a' and the number '9'.

The ZDV is formed by summing the ZDRs obtained from all connected components in the document. Since each document has different number of text characters, we also need to normalize the ZDV using following equation:



	fd_1	fd_2	fd_3	fd_4	fd_5	fd_6	fd_7	fd_8	fd_9
Ш	2	49	6	27	49	27	26	56	24
a	46	46	44	36	42	56	70	40	66
9	62	37	48	54	37	84	41	39	35

Figure 7. *Examples of ZDRs of connected components. Each of the character's bounding box in the top is separated equally into 3 parts, and their pixel densities are listed in the table below.*

$$ZDV_{norm} = \sum_{i=1}^{N} ZDR_i / N \tag{2}$$

Here, N is the total number of connected components in the document, and ZDR_i is the ZDR of the i-th connected component.

Notice that ZDV_{norm} is also in range $[0, 100]$.

Profile Component Run

So far the features found for a single connected component are mainly focused on the foreground pixels in its bounding box. The background areas surrounding the foreground pixels, however, also contain important information about the character [13]. And this information is very important for a relatively simple-structured script like Numeral. As a result, the profile component run (PCR) of a connected component and the profile document vector (PDV) of a document are utilized to better detect the document's script and orientation.

The PCR is obtained from the four side profiles, which are are the four background areas in a connected component's bounding box that surround the foreground pixels. These are the leftward, upward, rightward, and downward background areas. The steps to get the PCR from a connected component are:

(1) Identify the connected component's bounding box.

(2) On the left edge, identify 5 points on each edge of the bounding box, namely, the 2 vertices, the middle point, and two points whose distance from the nearest vertex is *1/6* of the left edge length.

(3) For each point found in (2), count the number of consecutive background pixels before hitting a foreground pixel along the line which is perpendicular to the left edge.

(4) For each number obtained in (3), multiply it by 100, and then divide it by the bounding box's width, to remove the effect of the connected component's bounding box size. Denote them as $Pl_1, Pl_2, ..., Pl_5$.

(5) Repeat steps (2) - (4) for the other 3 edge. Note that the denominator for the right edge is also the bounding box's width, but for the top and bottom edges the denominator is the bounding box's height. So we get $Pr_1, Pr_2, ..., Pr_5$ for the right edge, $Pt_1, Pt_2, ..., Pt_5$ for the top edge, and $Pb_1, Pb_2, ..., Pb_5$ for the bottom edge, as shown in Fig. 8.

(6) Combine all values obtained in (4) and (5), we will get a PCR for the connected component, which is

$$PCR = [Pl_1, ..., Pl_5, Pr_1, ..., Pr_5, Pt_1, ..., Pt_5, Pb_1, ..., Pb_5] \quad (3)$$

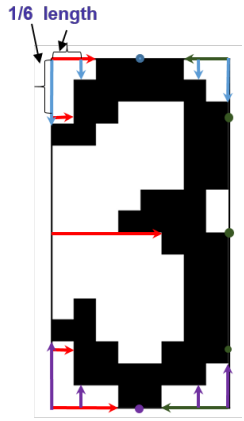


Figure 8. *Examples of PCR of a connected component. The arrow lines in the figure shows the distance from bounding edges to foreground pixels in the the 4 side profiles.*

The PDV is achieved by summing all PCRs of the document. In order to reduce the effect of document length, we also need to normalize it using the following equation:

$$PDV_{norm} = \sum_{i=1}^{N} PCR_i / N \quad (4)$$

Here, N is the total number of connected components in the document, and PCR_i is the PCR of the ith connected component.

Notice that PDV_{norm} is also in range $[0, 100]$.

Concatenating all the features from the previous sections, we have a *93*-dimensional feature vector containing VDV, HDV, ZDV and PDV that represents one single document page. Then the feature vector will be used in a support vector machine for training and prediction.

Script Detection Hierarchy

We use support vector machine (SVM) [14] to detect the script before detecting the orientation. So a script detection hierarchy is built (shown in Fig. 9). It is based on the similarity of different scripts. First, we treat Numeral and Roman as a group, and tell them apart from other scripts, which include Devanagari, Korean, Chinese, and Japanese. Then a model is developed to distinguish Numeral from Roman. Since Devanagari is a lot different from Korean, Chinese, and Japanese, we distinguish it first. Then Korean is identified from Chinese and Japanese. Finally, Chinese and Japanese are distinguished from each other.

Experiment Result

We collect our own data for training or testing. The scripts and their corresponding number of scanned images collected are shown in Table 1.

In this algorithm, Roman script contains English, French, German, Greek, Italian, Portuguese, Russian, and Spanish. Each of them has a similar number of pages.

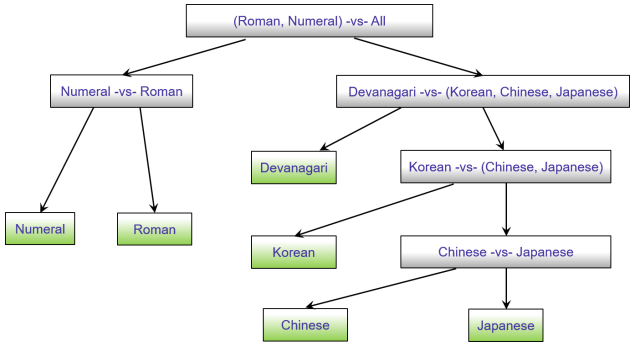


Figure 9. *Script Detection Hierarchy.*

Table 1: All Scripts and the Number of Pages

Script	Page#
Chinese	455
Devanagari	360
Japanese	378
Korean	438
Roman	2,476
Numeral	638
Overall	4,745

Note that pages we collected are all in 0 degree, and by simple rotations we can have images in 90, 180, and 270 degrees. So the total number of images in our data set is $4,745 \times 4 = 18,980$.

Test Results

We use 3-fold cross-validation to test the accuracy of our algorithm. Namely, we separate the whole data set into 3 equal-numbered folders, and use 2 of them for training, and one for testing. And we'll switch folders used for training and testing. The accuracy is computed as the average of three testing accuracies. Images here are all in *200* dpi.

Table 2 shows the cross-validation result of script and orientation detection for all scripts. From this table, we can see that for script detection, the overall accuracy for all scripts and 4 orientations is 98.2%. The best result is 99.1% with Japanese, while the lowest accuracy is 96.2% with Numeral. For orientation detection, the overall accuracy for all scripts of all 4 orientations is 99.2%. The best orientation accuracy is 99.8% with Japanese script, while the lowest accuracy is 98.3% with the numeral script.

Table 2: Orientation and Script Detection Accuracies for All Scripts

Script	Orientation Accuracy	Script Accuracy
Chinese	99.0%	96.8%
Devanagari	99.4%	99.1%
Japanese	99.8%	98.2%
Korean	99.2%	98.3%
Roman	99.4%	98.7%
Numeral	98.3%	96.2%
Overall	99.2%	98.2%

The Roman script actually is comprised of 8 scripts: English, French, German, Greek, Italian, Portuguese, Russian, and Spanish. Its detection result is got by summing up detection results of all of these scripts. Table 3 shows detailed detection results for these 8 scripts. From it we can see that, although Greek and Russian look quite different from other scripts, they do get fairly good detection results, with both achieving 99.5% in orientation accuracy, and 99.1% and 99.2% in script accuracy, respectively. The lowest orientation accuracy happens with German, at 98.5%, and the lowest script accuracy happens with Portuguese at 98.3%. Spanish has the best orientation and script detection accuracies, with the values being 100% and 99.9%, respectively.

Orientation and Script Detection Accuracies for Roman Scripts

Script	Orientation Accuracy	Script Accuracy
English	99.5%	98.9%
French	99.6%	99.8%
German	98.5%	98.9%
Greek	99.5%	99.1%
Italian	99.4%	99.4%
Portuguese	99.1%	98.3%
Russian	99.5%	99.2%
Spanish	100%	99.9%
Overall	99.4%	98.7%

Conclusion

Determining the orientation and script of scanned pages has important applications for scanned document processing. In this paper we proposed an algorithm to automatically detect the orientations of scanned pages. We described a series of preprocessing steps to identify the connected components on the page that contain text characters.We then extracted a set of features from these connected components that are fed to a hierarchy of SVM classifiers to determine script and orientation. By training and testing our new algorithm on a set of 18,980 scanned pages, we have demonstrated very good accuracy.

References

[1] Nobuyuki Otsu, A threshold selection method from gray-level histograms, IEEE Trans. Syst., Man, Cybern., vol. 9, pg. 62-66, (1979).

[2] Jun Guo, Xiaoping Liu, Chen Youguang Chen, etc. A revised feature extraction method for detecting text page up/down orientation, in Proc. 2011 Int. Conf. Appl. Superconductivity Electromagnetic Devices (ASEMD 2011), Sydney, NSW, Australia, pp. 105-108. (2011).

[3] Joost van Beusekom, Faisal Shafait, and Thomas M Breuel, Combined orientation and skew detection using geometric text-line modeling, Int. J. Document Anal. Recognition (IJDAR), vol. 13, pg. 79-92. (2010).

[4] Vandana Roy, Kadagattur Gopinatha Srinidhi, Yifeng Wu, eic, System and method for document orientation detection, Apr. 29 2014, U.S. Patent 8712188.

[5] Zhigang Fan, Michael R Campanelli, and Dennis Venable, Page orientation detection based on selective character recognition, Jun. 12 2012, US Patent 8200043.

[6] Shamita Ghosh, Bidyut B Chaudhuri, Composite script identification and orientation detection for indian text images, Proc. 2011 11th Int. Conf. Document Anal. Recognition (ICDAR 2011), Beijing, China, pg. 294-298. (2011).

[7] Sheikh Faisal Rashid, Faisal Shafait, and Thomas M Breuel, Discriminative learning for script recognition, Proc. 2010 IEEE Int. Conf. Image Process., Hong Kong, China, pg. 2145-2148. (2010).

[8] Shijian Lu, Chew Lim Tan, Script and language identification in noisy and degraded document images, IEEE Trans. Pattern Anal. Mach. Intell., vol. 30, pg. 14-24. (2008).

[9] Chirag Jain, Srinidhi Kadagattur, and Yifeng Wu, System and method for script and orientation detection of images, Dec. 2 2014, U.S. Patent 8903175.

[10] Chirag Jain, Chanaveeragouda Virupaxgouda Goudar, Kadagattur Gopinatha Srinidhi, etc, System and method for script and orientation detection of images using artificial neural networks, Nov. 18, 2014, U.S. Patent 8891822.

[11] Cheng Lu, Jan Allebach, Jerry Wagner, Brandi Pitta, David Larson, Yandong Guo, Online image classification under monotonic decision boundary constraint, in Proc. Color Imaging XX: Displaying, Process., Hardcopy, Appl. Int. Society Opt. Photonics, Vol. 9395, pg. 93950c.(2015).

[12] Ping Wah Wong, Inverse halftoning and kernel estimation for error diffusion, IEEE Trans. Image Process., vol. 4, pg. 486-498.(1995).

[13] Min-Chul Jung, Yong-Chul Shin, Sargur N Srihari, Mach. printed character segmentation method using side profiles, in Proc. 1999 IEEE Int. Conf. Syst., Man, Cybern., Tokyo, Japan, pg. 863-867. (1999).

[14] Chih-Chung Chang, Chih-Jen Lin, LIBSVM: A library for support vector machines, ACM Trans. Intelligent Syst. Technol., vol. 2, pg. 27:1-27:27, 2011, software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm.

Author Biography

Zhenhua Hu received his BS in biomedical engineering from the Shandong University (2011) and his MS in biomedical engineering from Zhejiang University (2014). Currently, he is a Ph.D. candidate at Purdue University. His work has focused on image processing, machine learning, image quality and data compression.

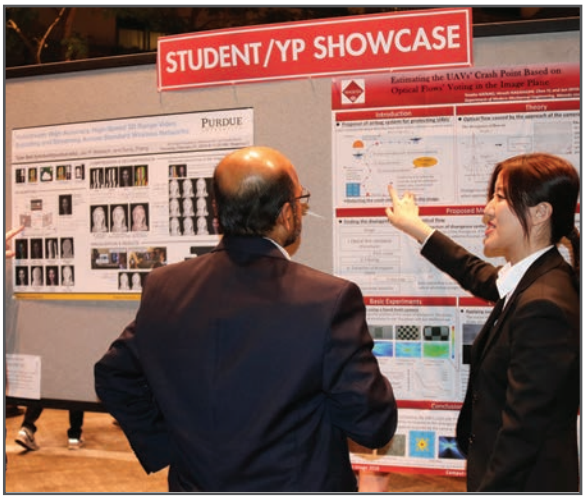
JOIN US AT THE NEXT EI!

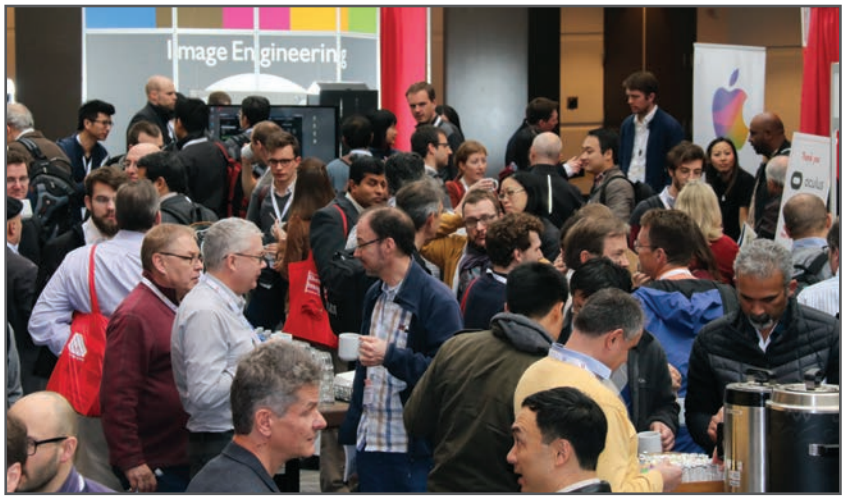
IS&T International Symposium on

Electronic Imaging

SCIENCE AND TECHNOLOGY

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





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

www.electronicimaging.org

IS&T
imaging.org