https://doi.org/10.2352/ISSN.2470-1173.2018.16.COLOR-418
© 2018, Society for Imaging Science and Technology

Recent advances in detection and healing of streaks caused by dust in a sheetfed scanner

Daulet Kenzhebalin [a]***, Ni Yan*** [a]***, Peter Bauer*** [b]***, Jerry Wagner*** [b]***, Jan Allebach*** [a]
[a]***School of Electrical and Computer Engineering, Purdue University; West Lafayette, IN***
[b]***HP Inc.; Boise, ID***

Abstract

Sheetfed scanners are widely used for scanning stacks of loose pages at high speed. The scanhead in the sheet-fed scanners is stationary and the page is fed with an automatic document feeder. When dust particles get stuck onto the scanner glass, they reflect the incident light and cause vertical streaks in the scanned images. These artifacts are known as dust streaks. We have developed a method for detecting dust streaks and the results were published in our previous paper [1]. In this study, we have refined our features for dust detection and added features for detecting tables. In addition, we looked into two methods for healing defective images: an exemplar based method and a diffusion based method. We applied these methods to remove dust streaks, punch holes, and torn corners from scanned images.

1. Introduction

Typically, document scanning is done in one of two modes. In the first mode, the user puts the page to be scanned face down on the glass platen and the scanhead moves and scans the page. This is called flatbed scanning. In the second mode, the paper is fed by a paper advance mechanism over the stationary scanhead. This is called sheet-fed scanning. This mode is very convenient for scanning stacks of pages, since the paper advance mechanism feeds each paper sequentially without human interaction. However, if a dust particle sticks to the glass over the scanhead, then it causes vertical streaks in the scanned images by reflecting the incident light of the scanhead. An example of an image with dust streak obtained using sheet-fed scanner is shown in Fig. 1.

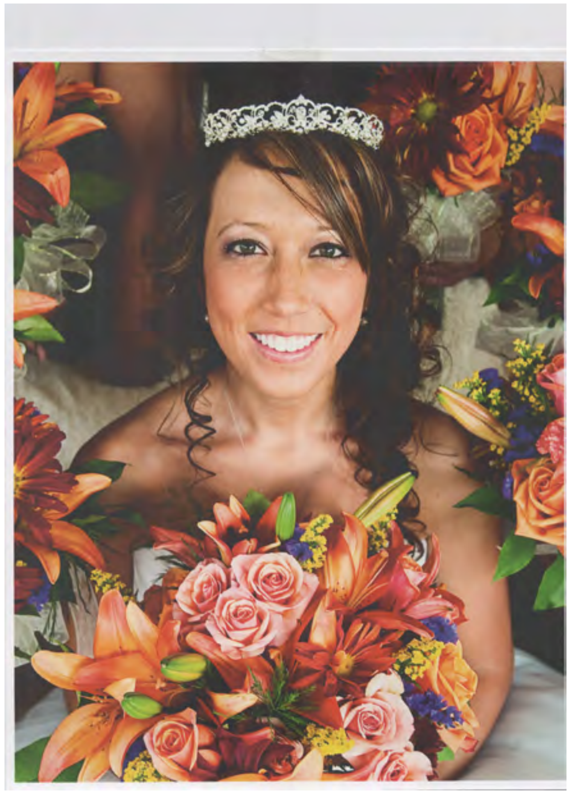


Figure 1. *Scan of an image with dust streak.*

This work builds on recent image quality work focused on printer and scanner products that was conducted in our laboratory, and which addressed assessment of page non-uniformity [2]-[7], fine-pitching banding [8]-[12], ghosting [13], local defects [14],[15], fading [16],[17], scanner MTF [18], and scanner motion quality [19].

Rosario et al. proposed an algorithm for detecting streaks in printed images [20]. However, they assume that the scanned image does not have content in it. In our case, we are designing an algorithm to work on any image.

When only a small part of the scan is available, some of the dust streaks are very hard to distinguish from the content streaks. We are designing an algorithm that processes an image in small regions for computational reasons and currently cannot exactly identify if the streak is caused by dust or if it is a part of content. Therefore, the dust detection might make errors in prediction. We categorize these errors into two types: misses and false alarms. A miss occurs when the detection algorithm does not find a real streak caused by dust. A false alarm occurs when the detection algorithm finds a line, which was not caused by dust. The aim is to have a small number of false alarms while detecting most of the real dust streaks.

After detecting dust streaks, we also want to heal the image. Image healing is the process of replacing defective pixels in the image with synthetic pixels so that the defect is no longer visible. Image healing can be done by one of two approaches: a diffusion based method or an exemplar based method. In the diffusion based method the synthetic pixels are computed using partial differential equations(PDE) [21]-[27]. In the exemplar based method the synthetic pixels are taken from the image [28]-[34].

In this paper[1], we propose a solution to finding vertical streaks in the images that were caused by dust in the scanner and healing the image to remove the defects such as dust streaks, punch holes, and torn corners.

[1]Research supported by HP Inc.

Section 2 will cover proposed procedure for detecting dust streaks, Section 3 will cover healing methods, Section 4 will cover text protection for healing, Section 5 will cover time optimization for inpainting algorithm, Section 6 will cover results, and Section 7 will cover conclusion.

2. Dust detection procedure

2.1. Preprocessing

We start with an image in gamma corrected RGB color space. We perform gamma uncorrection to get linear RGB color space. Then, we convert the image to NIQ color space. NIQ is an opponent color space, where N is the luminance channel, I and Q are chroma channels. We descreen the image to smooth it and remove halftones. The dust causes only vertical streaks. Hence, in order to preserve these dust streaks, we use a vertical average filter of size 9 for descreening. Figure 2a shows a zoom of the original image and Fig. 2b shows a descreened zoom. Here is the equation for descreening:

$$g(x,y,c) = \sum_{k=y-4}^{y+4} \frac{f(x,k,c)}{9}, \tag{1}$$

where $f(x,y,c)$ is an input image in NIQ color space, and $g(x,y,c)$ is a descreened image. Here c denotes the color channel $N = 1$, $I = 2$, or $Q = 3$.

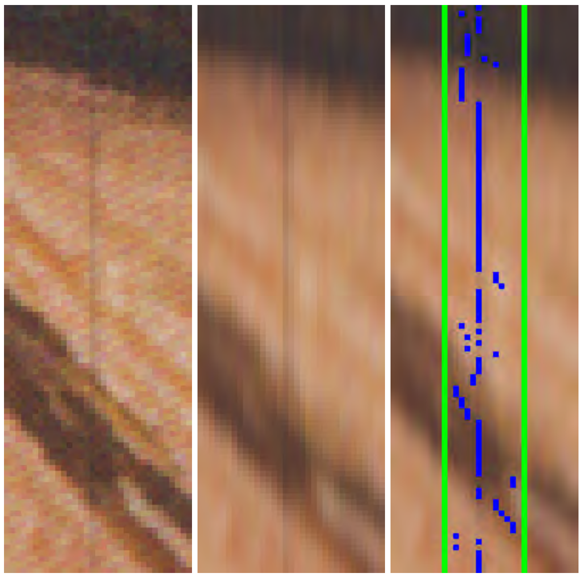

(a) Original image (b) Descreened image (c) Descreened image with annotations

Figure 2. Crop of the image shown in Fig. 1.

Next, we compute $\Delta E'$ using Eqs. 2 and 3. We are using only the luminance channel to compute $\Delta E'$ because the dust streaks mainly affect the luminance channel.

$$baseline(x,y) = \sum_{k=x-5}^{x+5} \frac{g(k,y,1)}{11}, \tag{2}$$

$$\Delta E'(x,y) = g(x,y,1) - baseline(x,y), \tag{3}$$

where $g(x,y,1)$ is luminance value of the descreened image at pixel (x,y).

2.2. Features

After computing $\Delta E'$, we split the image into vertical columnstrips of width $w = 13$ pixels and an overlap of $o = 6$ pixels.

For each columnstrip row we find peaks and valleys. We choose the index at which the peak or valley has the highest absolute value in this columnstrip row, and we call it the peak location. A peak is a data sample that is either larger than its two neighboring samples or larger than the left neighboring sample and equal to the right neighboring sample. A valley is a data sample that is either smaller than its two neighboring samples or smaller than the left neighboring sample and equal to the other. The leftmost and rightmost pixels of the columnstrip cannot be peak locations because they don't have two neighbors. In Fig. 3 the indices 2 and 7 are valleys, whereas the indices 6 and 9 are peaks. The peak location is 6.

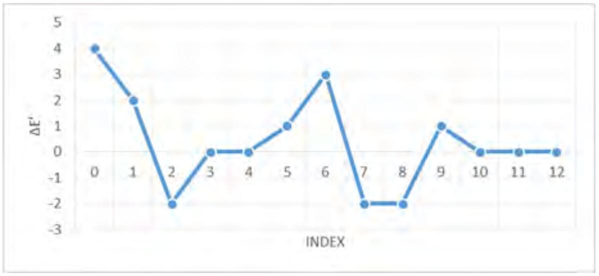


Figure 3. Example of a columnstrip row.

Figure 2c shows a crop of a descreened image with annotations for one columnstrip. The green lines are the edges of the columnstrip. The blue marks are the peak locations in each row. In the areas where the background is smooth, the peak locations form a vertical line due to presence of dust streak. In the area where the dust streak goes through hair, the blue dots are not vertically aligned due to the influence of the content.

In addition to peak location, we will define the peak edge. The peak edges show the width of the peak. We define the peak edge as the index closest to the peak index, where the value drops below 25% of the peak value. In Fig. 3 the indices 4 and 7 are left peak edge and right peak edge indices, respectively.

In some cases, the peak location is not vertically aligned even though there is no content interfering with the streak. This happens when the dust streak is wider than 1 pixel and has multiple columns with similar $\Delta E'$ values. We want the peak location to be vertically aligned across multiple rows for detecting dust streaks. Therefore, we adjusted the peak location based on two conditions and called it the augmented peak location (APL). The first condition is that the difference between the index of the current row center and the index of the APL of the preceding row is less than 2. The second condition is that the APL of the preceding row is between the left and right peak edges of the current row. If both of the conditions are satisfied, then we assign the APL value of the preceding row to the APL of the current row.

2.2.1. Feature 1. Minimum locally summed augmented peak location derivative magnitude (MLSAPLDM)

MLSAPLDM is the primary feature that we will use to find vertical streaks. We will compute the augmented peak location derivative magnitude according to Eq. 4. Then we will sum the augmented peak location derivative magnitude for a window of size 20 according to Eq. 5. Then we will find the minimum of two adjacent windows and use that value as the MLSAPLDM according to Eq. 6. A low MLSAPLDM value implies that there may be a vertical streak in the image.

$$apldm(x,y) = |apl(x,y) - apl(x,y+1)|, \tag{4}$$

$$lsapldm(x,y) = \sum_{k=y}^{y+19} apldm(x,k), \tag{5}$$

$$f_1(x,y) = min\big(lsapldm(x,y), lsapldm(x,y-20)\big), \tag{6}$$

where $apl(x,y)$ is the augmented peak location in columnstrip x and row y, $apldm$ is the augmented peak location derivative magnitude, $lsapldm$ is the locally summed augmented peak location derivative magnitude, and f_1 is the minimum locally summed augmented peak location derivative magnitude.

2.2.2. Feature 2. Peaking factor (PF)

The PF computes the strength of the peak. We use the true area under the curve. PF is computed according to Eq. 7.

$$f_2(x,y) = \sum_{k=lpe(x,y)+1}^{rpe(x,y)-1} |\Delta E'(k,y)|, \tag{7}$$

where $lpe(x,y)$ and $rpe(x,y)$ are left peak edge and right peak edges in columnstrip x and row y, respectively.

2.2.3. Feature 3. Side difference (SD)

The edges of the content such as edges of pictures will also have a low MLSAPLDM value. However, these are not dust streaks. To remove these false alarms, we use the SD. The SD is a color difference between the two sides of the peak location. We take the average of 3 pixels following the peak edge on both sides according to Eqs. 8-9. We call the average value of the left side and the average value of the right side $color_l$ and $color_r$, respectively. The SD is computed using Eq. 10:

$$color_l(x,y,c) = \frac{1}{3} \sum_{k=lpe(x,y)-2}^{lpe(x,y)} g(k,y,c), \tag{8}$$

$$color_r(x,y,c) = \frac{1}{3} \sum_{k=rpe(x,y)-2}^{rpe(x,y)} g(k,y,c), \tag{9}$$

$$f_3(x,y) = \sqrt{\sum_{c=0}^{2} (color_l(x,y,c) - color_r(x,y,c))^2}, \tag{10}$$

where x is the columnstrip index, y is the row index, c is the channel index, $g(x,y,c)$ is the descreened image, lpe is the left peak edge, and $color_l$ and $color_r$ are the average colors of the left and right sides of the peak.

2.2.4. Feature 4. Columnstrip derivative (CD)

Table lines cause false alarms since they look like vertical streaks. We developed this feature to capture most of the table lines. The Columnstrip derivative is computed for each columnstrip row according to Eq. 11. The CD computes the luminance difference between two columnstrip rows that are two rows apart, averaged over 13 consecutive columns. Note that this feature is computed on the original image before descreening.

$$f_4(x,y) = \frac{1}{13} \sum_{k=xc}^{xc+13} |f(k,y,0) - f(k,y+2,0)|, \tag{11}$$

where $f(k,y,0)$ is luminance value in column y, row k, and xc is the leftmost column of columnstrip x.

2.3. Thresholding and obtaining defective mask

After computing the features defined in Sections 2.2.1-2.2.4, we can obtain an initial defective mask $mask_1$. We mark a columnstrip row as nondefective if it doesn't satisfy a condition for a defective pixel and mark it as defective if it does satisfy such a condition. Equation 12 describes the initial mask where 1 is defective and 0 is nondefective. We obtain another mask for tables $mask_{table}$ using the same procedure but with different thresholds. Since most table lines are stronger than dust streaks the threshold for the PF will be higher for table lines. In addition, the table lines can have a skew depending on the page alignment, and the MLSAPLDM threshold is less conservative, that is higher, for table lines mask.

$$mask_1(x,y) = \begin{cases} 1 & \text{if } f_1(x,y) < T_1 \text{ and } f_2(x,y) > T_{2min} \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

After obtaining two masks for table lines and dust streaks, we will use table line detection algorithm to refine the table lines mask. This is described in Section 2.4. Then, we will mark table lines and adjacent columnstrips as nondefective on the dust streaks mask.

We will use PF and SD to remove the remaining false alarms. Equation 13 describes how we obtain $mask_2(x,y)$.

$$mask_2(x,y) = \begin{cases} 1 & \text{if } mask_1(x,y) = 1, f_3(x,y) < T_3, \\ & f_2(x,y) < T_{2max} \\ 0 & \text{otherwise} \end{cases} \tag{13}$$

2.4. Table line detection

Vertical lines of tables cause false alarms. We need to detect them and mark as not a dust streak. Most tables have a rectangle box around them. That means that vertical table lines start and end with horizontal line intersections. We already have a procedure for detecting vertical lines, so we use the feature CD for detecting horizontal lines. The algorithm for detecting table lines is provided in Algorithm 1.

Examples of table detection are shown in Figs. 4 and 5. Magenta means that there is a table line, and red means that there is dust streak based on our algorithm result. From the image in Fig. 4b one can see that table detection has identified almost all of the

Algorithm 1 Table line detection.

```
Find all rows which have at least 14 consecutive columnstrips
where $f_4(x,y) > 70$
for each columnstrip $x$ do
    if columnstrip $x$ has two or more horizontal lines then
        if vertical line was detected in more than half of rows between two consecutive horizontal lines ($y_1$ and $y_2$) and less than 9 of 20 rows above $y_1$ then
            $y \leftarrow y_1$
            while $y \neq y_2$ do
                $mask_{table}(x,y) \leftarrow 1$
                $y \leftarrow y+1$
            end while
        else
            $y \leftarrow y_1$
            while $y \neq y_2$ do
                $mask_{table}(x,y) \leftarrow 0$
                $y \leftarrow y+1$
            end while
        end if
    else
        $y \leftarrow 0$
        while $y \neq height$ do
            $mask_{table}(x,y) \leftarrow 0$
            $y \leftarrow y+1$
        end while
    end if
end for
```

table lines and a dust streak correctly. The small segment that was not detected is due to the fact the the vertical line is skewed, and was not detected in that area. However, in some cases the table line detection algorithm doesn't find all table lines. An example where our algorithm doesn't find all table lines is shown in Fig. 5b. This image has a table with a background of various colors. The table lines are all black and the contrast between background and table line varies a lot. Varying contrast is a reason for not detected table lines. Based on these and other results we have seen, the table line detection finds most table lines. But it doesn't work when the table line has a low contrast to background.

2.5. Postprocessing

Our procedure needs a cleanup routine to connect defective segments that are close and to remove defective segments if they are too short. We call a run of consecutive defective rows in a columnstrip a defective segment. Connecting two defective segments means that the rows between two segments will be marked as defective. Removing a defective segment means that the segment will be marked as nondefective. Postprocessing consists of two steps. The first step removes very short streaks, and connects streaks if the gap between them is very small. In the second step, we search for regions of 250 pixels in the columnstrip that have at least 150 defective rows. We chose the number 150 based on our scanner resolution of 300 dpi, and the requirement for streaks to be no less than 0.5 in. in length, which is 150 pixels.

In the first step, we will connect small segments if the gap between them is smaller than 5 pixels. After connecting all small segments that are close together, we will check their length, and remove segments that are shorter than 40 pixels.

In the second step, we will look at a sliding window of 250 pixels with a step size of 50 pixels. In each window we will compute the following statistics: first defective row (y_1), last defective row (y_2), longest gap ($maxgap$), and total number of defective rows (m). We will create a new mask to store the final results of the postprocessing. We will mark the region from y_1 to y_2 as defective if $(y_2 - y_1) > 150$, $maxgap < 50$, and $m > 120$. If any of these conditions are not satisfied, then we don't mark anything.

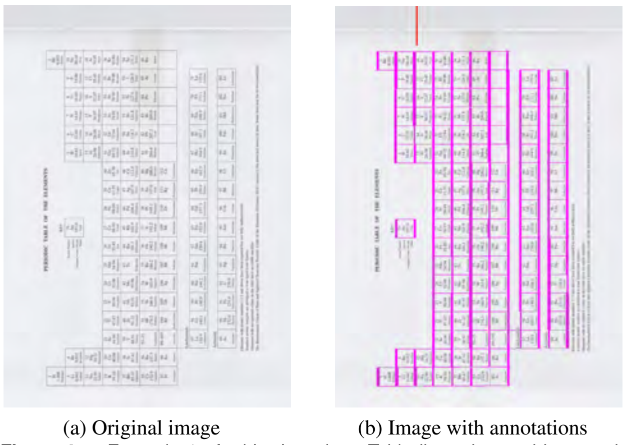

(a) Original image (b) Image with annotations

Figure 4. *Example 1 of table detection. Table lines detected by our algorithm are shown in magenta. Dust streaks detected by our algorithm are shown in red.*

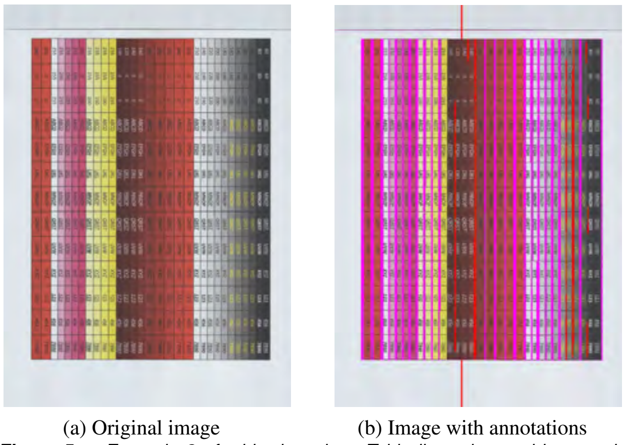

(a) Original image (b) Image with annotations

Figure 5. *Example 2 of table detection. Table lines detected by our algorithm are shown in magenta. Dust streaks detected by our algorithm are shown in red.*

2.6. Finding exact locations of dust streaks

The defective mask that we have found does not contain information about where in the columnstrip the dust streak is located. Healing the entire columnstrip does not look good, because only a couple of pixels in the columnstrip have been distorted by the dust streak. We can use the left peak edge ($lpe(x,y)$) and the right peak edge ($rpe(x,y)$) to find the actual defective pixels. However, sometimes the columnstrip row might contain other content, which has a higher $|\Delta E'|$ value; and $lpe(x,y)$

and $rpe(x,y)$ will not accurately indicate the location of the dust streak. So, we need to refine the results.

To refine the defective pixels, we will find columns in the columnstrip that have been affected by dust. Dust particles generally don't move. So the defective pixels should stay in the same columns. For each defective columnstrip row, we will mark the pixels between $lpe(x,y)$ and $rpe(x,y)$ as defective pixels. We will only keep the columns that belong to at least half of the defective rows of the columnstrip. All other columns will be marked as nondefective.

3. Healing methods

Image healing can be done done by a diffusion based method or an exemplar based method. We will apply these methods to heal dust streaks. We will also apply the exemplar based method to heal torn corners and punch holes, which are common problems for scanned documents.

The advantage of the diffusion based healing method is low computation. Each row is healed separately by replacing the defective region with synthetic pixels. The synthetic pixels will be interpolated from the surrounding 2-4 pixels around the defective region depending on the PDE being used. The disadvantage of this approach is that if the region being healed is large, it will look blurry.

The advantage of the exemplar based method is that it does not cause blurriness. A major disadvantage is the computation time. Since the synthetic region is a patch from the image, a search algorithm is needed. An exhaustive search for the best patch requires a lot of computation.

3.1 Healing using exemplar based method

For our exemplar based method, we followed the work by A. Criminisi et al., which presented a novel algorithm for removing large objects from images in a visually plausible way [30]. Here are the terminologies that we will use in this Section: target area (Ω), source area (Φ), fill-front ($\nabla\Omega$) and patch (ψ) [30]. The target area (Ω) is the area that we need to synthesize. The source area (Φ) is the rest of the image, the known part. The fill-front ($\nabla\Omega$) is the intersection between Ω and Φ. A patch (ψ) is a computation unit, which is a square with side w. Given a fixed w, a patch is determined by its center pixel p. A target patch lays partially or completely in Ω. A source patch always lays completely in Φ, i.e. none of its pixels are missing. A graphical illustration of these terminologies is given in Fig. 6.

The goal of their work was to search for the best patch that is fully in the source area to fill the unknown pixels of a patch that is in the target area. The similarity is measured by the function given in Eq. 15.

$$I(\mathbf{i}) = \begin{cases} 1 & \text{if } \mathbf{i} \in \Phi \\ 0 & \text{if } \mathbf{i} \in \Omega, \end{cases} \tag{14}$$

where $\mathbf{i}$ is a pixel coordinate.

$$\varepsilon = \sum_{\mathbf{i} \in \psi_{\mathbf{p_1}}} I(\psi_{\mathbf{p_1}}(\mathbf{i})) \times (\psi_{\mathbf{p_1}}(\mathbf{i}) - \psi_{\mathbf{p_2}}(\mathbf{i}))^2, \tag{15}$$

where $\psi_{\mathbf{p_1}}$ and $\psi_{\mathbf{p_2}}$ are patches with centers at $\mathbf{p_1}$ and $\mathbf{p_2}$ respectively. $I(\mathbf{i})$ is the indicator function defined in Eq. 14.

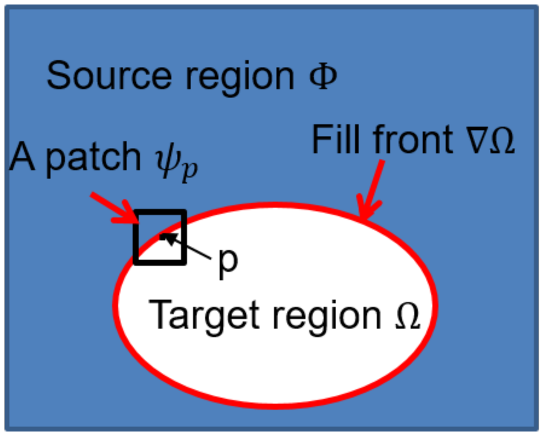


Figure 6. *Graphical illustration of terminologies.*

The key to proper healing is the patch priority $P(\psi_{\mathbf{p}})$. The patches with higher priority are healed first [30]. This feature pushes the patches with more information, or with stronger structure, to be filled earlier. We used the priority queue to control the order of filling. Each defective patch is partially replaced by its nearest neighbor patch from the source area based on the similarity measure defined by Eq. 15.

3.2 Healing using cubic interpolation

Cubic interpolation is one of the diffusion based methods. Image healing using cubic interpolation heals each row separately. The dust streaks cause vertical streaks of varying width from 1 to 5 pixels. For each row with defective pixels we will use four nearest neighbor pixels of the defective region: two pixels on the left side and two pixels on the right side. Using these pixel values, we can find the coefficients of a polynomial. We used Catmull-Rom spline interpolation [36]. The coefficient equations are given in Eqs. 16-20:

$$f(x) = ax^3 + bx^2 + cx + d, \tag{16}$$

$$a = -\frac{1}{2}q_0 + \frac{3}{2}q_1 - \frac{3}{2}q_2 + \frac{1}{2}q_3, \tag{17}$$

$$b = q_0 - \frac{5}{2}q_1 + 2q_2 - \frac{1}{2}q_3, \tag{18}$$

$$c = -\frac{1}{2}q_0 + \frac{1}{2}q_2, \tag{19}$$

$$d = q_1, \tag{20}$$

where q_o and q_1 are pixels to the left of the region, and q_2 and q_3 are pixels to the right of the region.

4. Text protection

Healing text areas is undesirable because the text can be altered and it is very hard to decide if the stroke was created by the dust streak, or if the stroke was there as a part of a character. Therefore, we don't want to alter the text, and we will not heal the regions where there is text. Only small font size text can be altered by healing because dust streaks are very narrow. For this reason, we have developed a procedure to protect text by removing the text areas with small font size from detection.

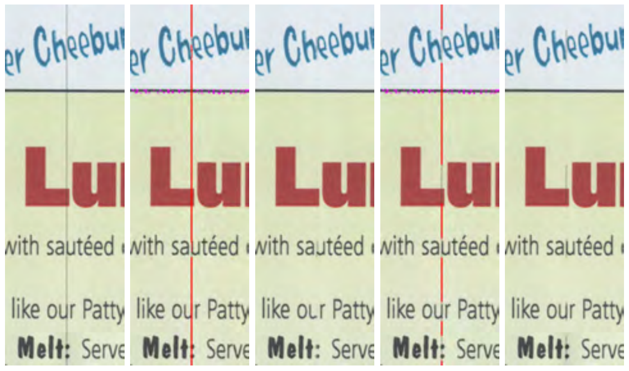
Since the purpose of this project is hardware application, we decided to use the data that we have already computed for text detection. The data that we will use is $\Delta E'$, which is the difference between the average baseline luminance and the luminance at an individual pixel. The character height that could be destroyed by healing is up to 25 pixels. The largest width of these characters is approximately 30 pixels. Based on that, we decided to compute a sum of $|\Delta E'|$ for 5 nonoverlapping columnstrips of width 13 pixels to cover at least 2 characters. This is captured in Eq. 21.

$$DeltaESum(x,y) = \sum_{k=xl-26}^{xl+38} |\Delta E'(k,y)|, \tag{21}$$

where x is a columnstrip index, y is a row index, and xl is the index of the leftmost column of columnstrip x.

We will then decide whether the row contains the text or not based on this feature. If the $DeltaESum(x,y)$ is higher than 450, then we decide that this row has a text, and we will not heal this row.

Figure 7 shows a comparison of two healed results using the cubic interpolation method. Figure 7c shows an example of an image that was healed without using text protection. As a result, the text was damaged. The letter "u" in "sauteed" and "our" was severely altered. Figure 7e shows an example of an image that was healed with text protection. The letters have been protected and not altered. In addition to protecting both letters "u", the streak that is between "L" and "U" was also protected. This is undesirable in this example. But in general, it is better to be safe in the areas close to text.



(a) (b) (c) (d) (e)

Figure 7. *Comparison of healing using cubic interpolation with and without text protection: (a) original image; (b) image with defective mask without text protection; (c) healed image without text protection; (d) image with defective mask with text protection; (e) healed image with text protection.*

5. Decreasing time complexity for exemplar based healing method (inpainting)

For healing using the inpainting method, an exhaustive search has a high computational cost. It requires looking for a best patch among all source patches. Many computer vision, and machine learning applications involve searching similar vectors in very large databases [38]-[40]. It is usually referred as a nearest neighbor (NN) problem.

Many algorithms for NN problems suffer from high dimensionality of the search vector [41]-[43]. Our approach is an extension of KD trees [25], which belong to the partition tree techniques. The partition tree techniques divide the search space into subspaces. The KD tree algorithm uses one of the k dimensions as a hyperplane at each tree node to divide the space. To deal with high dimensionality and decrease time complexity, we decided to sacrifice the precision of the result.

The KD tree is a generalization of the binary tree. At the root node, the data is split into two groups along dimension i, where $i \in [0, k-1]$. Data points with smaller values than the root are put into the left subtree, the remaining data points are put into the right subtree. The split is usually made at the median of the data in dimension i. The i is called a discriminator. The discriminator is carefully chosen at each node. First, we calculate the variance along all k dimensions among the data points that belong to this node. The dimension with the highest variance is chosen as the discriminator. Then we do the same thing for the left child node and the right child node.

The database of vectors is constructed from source region. By using a sliding window with the same size as the patch size, we can move this window one pixel at a time and get all the source vectors. Using these vectors we will then construct a KD tree.

When a query is made for a vector x, we compare the value of the query at dimension i, which corresponds to the root node. If the query is smaller, then we continue the search in the left subtree, otherwise we continue the search in the right subtree.

The immediate problem that we need to address is the incomplete target vectors. That is, we cannot use the missing pixels in the target vector that we want to heal. Therefore, we will have to continue the search in both subtrees when the discriminator value is missing in the target vector. This problem can be easily solved for the dust streaks. Since dust streaks are narrow and long, the dimensions where the target patches are missing values will not be used as discriminators. We cannot apply this to torn corners and punch holes since the target vectors have missing pixels at different dimensions.

We traverse the tree until we find a leaf node. The leaf node is not always the nearest neighbor to the query vector. Then we need to backtrack the search path and compare the distances. Also, at each node we need to check if we need to traverse the other subtree.

To save time, we will limit the amount of backtracking and sacrifice the accuracy of finding the true nearest neighbor. To compensate for accuracy, we will decrease the probability of failure by constructing multiple KD trees. Assuming the probability of failure of finding the true nearest neighbor is P_f, and it is independent and identically distributed among all the trees, then our failure rate for m trees is P_f^m.

We will use randomness to build different trees. For the computation of variance for all dimensions, we will use only a small subset of data, for example 100 data points. These points are chosen uniformly at random. Next, instead of choosing the top variance as the discriminator, we will choose uniformly one of the top m discriminators. The probability of getting the same discriminator at two nodes is less than $m^{(-2)}$. For $m = 5$ the probability of getting same three nodes at the top of two trees is less than 25^{-3}. This is a satisfactorily small number, and hence we will use 5 in our application.

The multiple KD trees pipeline is summarized in Fig. 8.

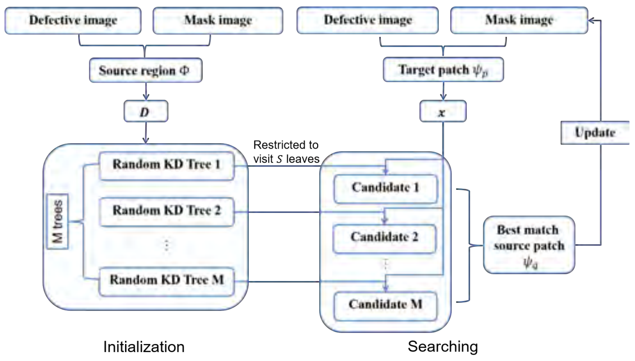


Figure 8. Pipeline for optimizing image healing using inpainting.

6. Results

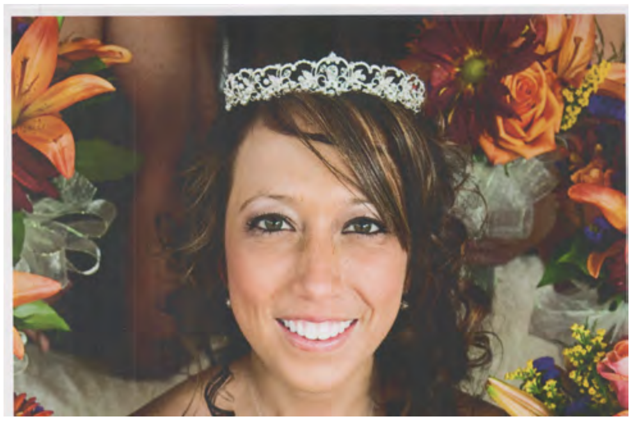
6.1. Results of dust detection procedure

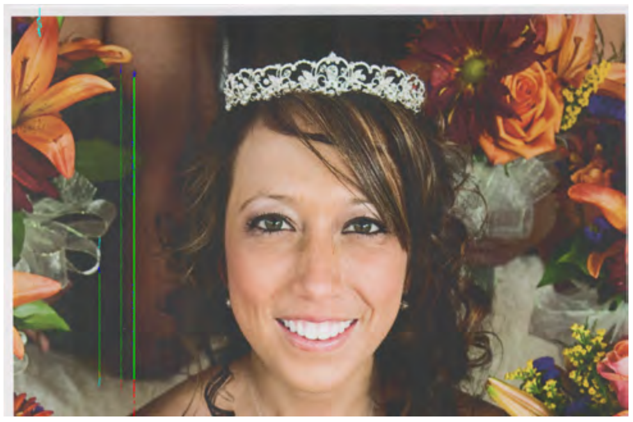
We used the same training procedure as reported in [1]. This time, however, we focused more on decreasing the false alarm rate. As a result, we increased the allowed miss rate to 30%. We achieved false alarm rate of 0.03%. That is, 22,179 false alarm columnstrip rows and 28,418 missed columnstrip rows.

Examples of the detection and healing algorithms are provided in Figs. 9-11. In these examples, we are showing the results of healing using cubic interpolation since it is a preferred method of healing for dust streaks due to its low computation time. Figure 9a shows an original image, Fig. 9b shows an algorithm result superimposed on the original image, and Fig. 9c shows a healed image. The detection result is color coded to show ground truth and algorithm results on the same image. The color codes are summarized in Table 1. In Fig. 9, the three longest dust streaks were found by the algorithm. Small parts of the streaks were missed due to noise in the image. In Fig. 10, all of the dust streak was detected and healed. In Fig. 11, the dust streak on the far left side of the image was detected correctly and healed, but there were also table lines detected as dust streaks and healed incorrectly. In this example, the table lines have a low contrast to the background and are hard to distinguish from dust streaks. As a result of this wrong detection, the table lines are removed from the image in Fig. 11c.

Table 1: Color code for annotations based on ground truth and dust detection algorithm.

	Annotated in ground truth with high score	**Annotated in ground truth with low score**	**Not annotated**
Detected by the algorithm	green	yellow	red
Not detected by the algorithm	blue	cyan	no annotation


(a) Original image


(b) Image with dust detection result superimposed

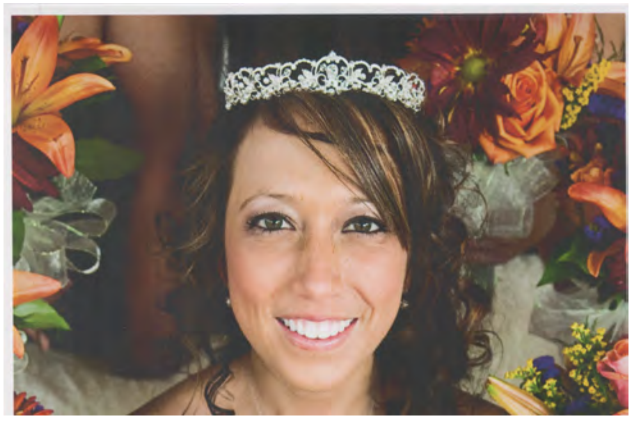
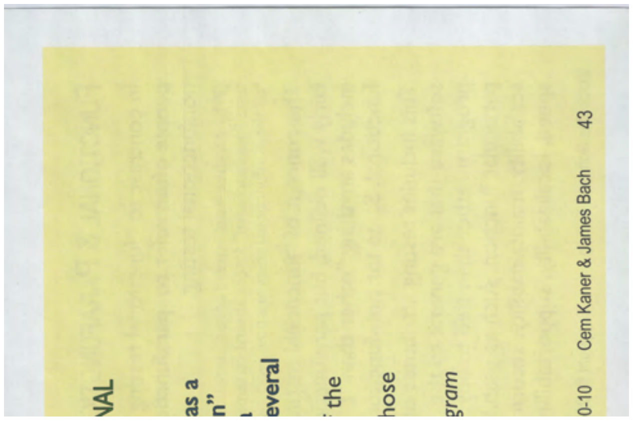
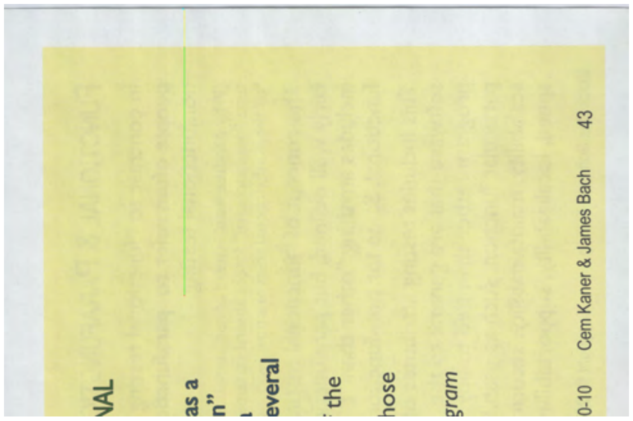

(c) Healed image using cubic interpolation

Figure 9. Example 1 of algorithm detection and healing. It is recommended that the reader zoom into these images in order to see the dust streaks, the detection algorithm results, and the healed results.

6.2 Applications of exemplar based healing for scanned documents

We used the method described in Section 3.1 to heal dust streaks, punch holes, and torn corners. In order to have ideal healing, the patch size is a very important factor. It has to match up with the document texton size [37]. Examples of the effect of


(a) Original image


(b) Image with dust detection result superimposed

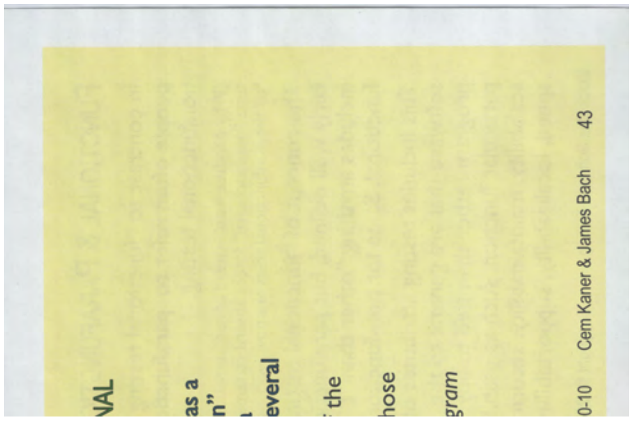
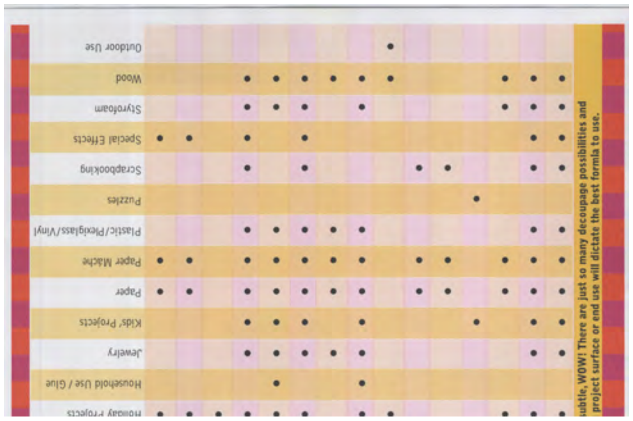
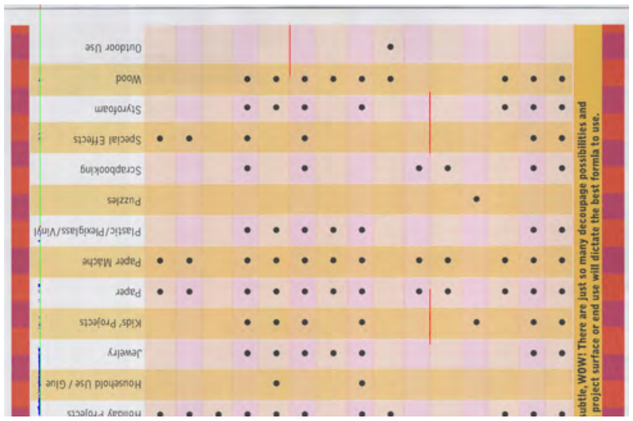

(c) Healed image using cubic interpolation

Figure 10. *Example 2 of algorithm detection and healing. It is recommended that the reader zoom into these images in order to see the dust streaks, the detection algorithm results, and the healed results.*


(a) Original image


(b) Image with dust detection result superimposed

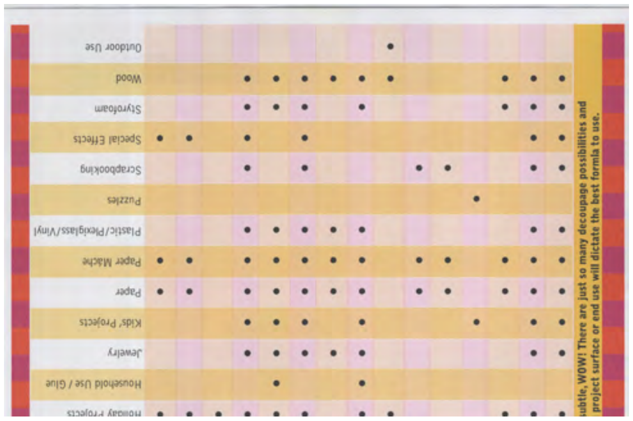

(c) Healed image using cubic interpolation

Figure 11. *Example 3 of algorithm detection and healing. It is recommended that the reader zoom into these images in order to see the dust streaks, the detection algorithm results, and the healed results.*

different patch sizes on healing are provided in Figs. 12 and 13. For a smooth background, the tolerance to changes in patch size is much larger as the texture blends easily. However, if the line goes across a high frequency content area, either a too small or a too big size patch can cause visible defects. The medium sized patches usually work best for most of dust streaks. In Fig. 12, other than patch sizes 3 and 5, the rest of the patch sizes yield a good healing result. The most difficult case occurs when the dust line goes across a text area. In Fig. 13, letters of different font sizes are affected by a dust streak. Bigger letters such as the letter ”O” at the top of the image show more tolerance to changes in patch size than the numeral ”8” at the bottom. The overall best result appears to be achieved with a patch size 11.

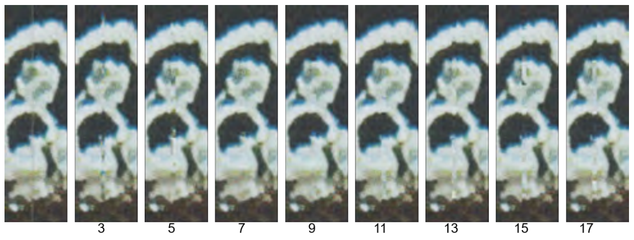


Figure 12. Example 1 of different patch sizes used to heal a dust streak. Leftmost is a zoom-in from the original image, the rest are healing with patch sizes 3, 5, 7, 9, 11, 13, 15, respectively. The original image is scanned at 300 dpi, and these images are of size 141 ×41 pixels.

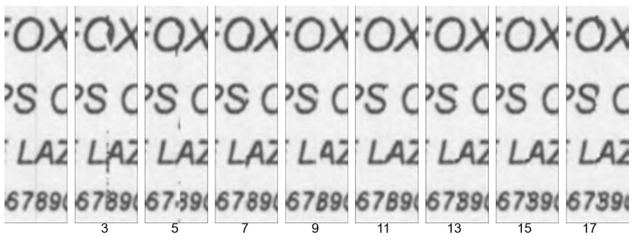


Figure 13. Example 2 of different patch sizes used to heal dust streak. Leftmost is a zoom-in from the original image, the rest are healing with patch sizes 3, 5, 7, 9, 11, 13, 15, respectively. The original image is scanned at 300 dpi, and these images are of size 141×41 pixels.

Of course, when the missing area is much bigger, such as that in a punch hole, or a torn corner, we have to do much more guessing. If there is text content, we have to take the risk and create something meaningful using the background. In these cases, we care more about the visual continuity than the correctness, since we don't have information of the latter. Examples of healing punch holes are given in Figs. 14 and 15. Examples of healing torn corners are given in Figs. 16 and 17.

The punch holes in Fig. 14 are healed well using patches of sizes 13, 17, 19, 21, 23, 25, and 27. Using patches with a smaller size yields visible artifacts. Torn corners are even harder to heal since the missing part is on the edge of the image. On all of the healed images of the torn corner, the background is replicated very nicely and matches the off-white background of the image.

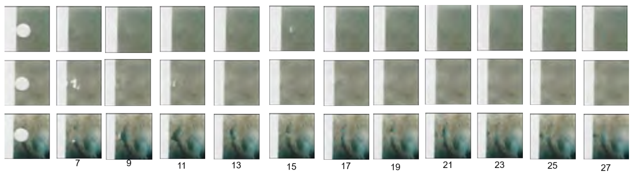


Figure 14. Example 1 of different patch sizes used to heal punch holes. Leftmost is a zoom-in from the original image, the rest have been healed with patch sizes 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, respectively. Each zoom-in is 271×271 pixels, the radius of the punch hole is about 40 pixels.

This method yields good results with the correct patch size for dust streak healing. It can replicate the background for torn corners and punch holes. However, it doesn't work as well for cases with content cropped out due to a torn corner or punch hole. The main disadvantage is that it has a high computational cost.

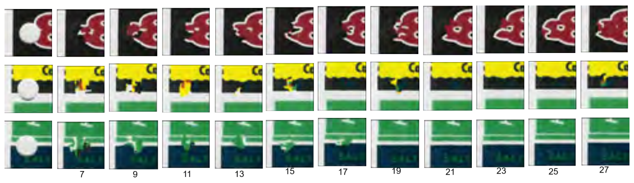


Figure 15. Example 2 of different patch sizes used to heal punch holes. Leftmost is a zoom-in from the original image, the rest have been healed with patch sizes 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, respectively. Each zoom-in is 191×191 pixels, the radius of the punch hole is about 40 pixels.

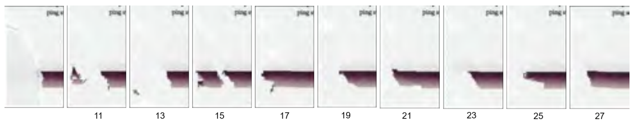


Figure 16. Example 1 of different patch sizes used to heal a torn corner. Leftmost is a zoom-in from the original image, the rest have been healed with patch sizes 11, 13, 15, 17, 19, 21, 23, 25, 27, respectively. Each zoom-in is 600×350 pixels.

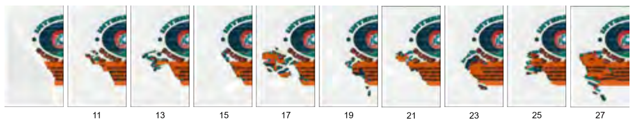


Figure 17. Example 2 of different patch sizes used to heal a torn corner. Leftmost is a zoom-in from the original image, the rest have been healed with patch sizes 11, 13, 15, 17, 19, 21, 23, 25, 27, respectively. Each zoom-in is 600×350 pixels.

6.3 Results of applying cubic interpolation to healing dust streaks

Examples of using cubic interpolation are shown in Figs. 18-20. The examples show that narrow streaks are healed very well. Since most of the dust streaks are 1-3 pixels wide, this method works well for healing dust streaks. However, for wider streaks this method is not as good. An example of wide streak healing is given in Fig. 20. Figure 20a shows an original image with a dust streak; Fig. 20b shows the image with defective mask; and Fig. 20c shows the healed image with the new synthetic pixels in place of defective pixels. When healing wide streaks some of the defective pixels are being used for interpolation because the defective mask does not cover all of defective pixels. Increasing the width of the defective mask will help in this case, but it might interfere with content in other cases and cause significant changes to the image.

6.3 Results of using multiple KD trees with exemplar based image healing

Since the data in missing pixels such as punch holes is missing, there is no real ground truth. We will use result of exhaustive search image healing as the ground truth and compare the result of using multiple KD trees to that. The error metric we used is

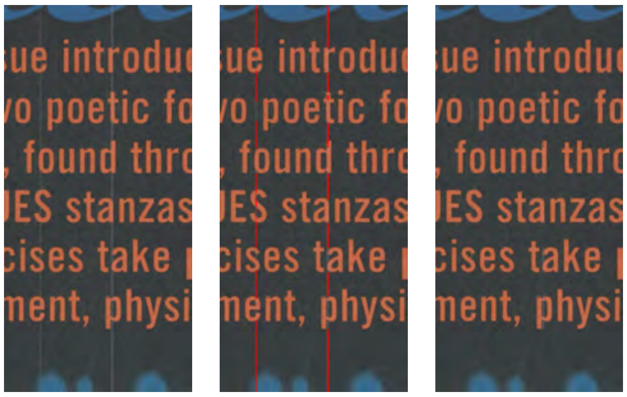

(a) Original image (b) Image with defective mask (c) Healed image

Figure 18. *Example 1 of healed image.*

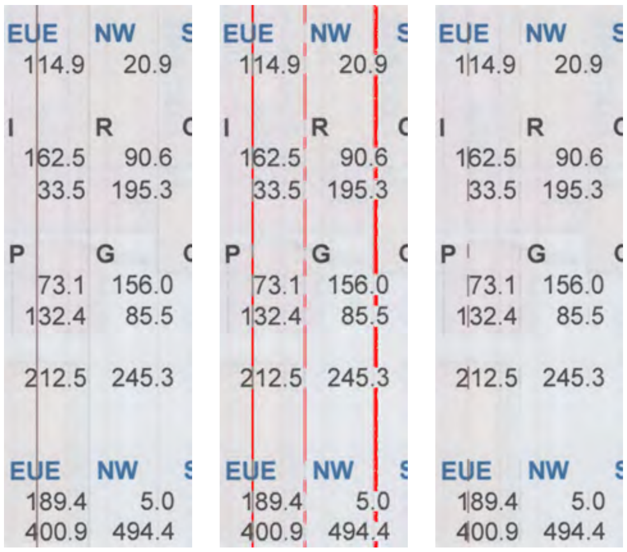

(a) Original image (b) Image with defective mask (c) Healed image

Figure 19. *Example 2 of healed image.*

$\overline{\Delta E}$:

$$\overline{\Delta E} = \frac{1}{N}\sum_{i}(1 - I(i)) \sqrt{(r_e(i)-r_o(i))^2 + (g_e(i)-g_o(i))^2 + (b_e(i)-b_o(i))^2}, \qquad (22)$$

where $I(i)$ is an indicator function for the pixels in the target region, $r_e(i)$ is a pixel value of the red channel in the exhaustive search result image, $r_o(i)$ is a pixel value of the red channel in the optimized search result image, and $N = \sum I(i)$ is the number of missing pixels.

Considering the randomness in the algorithm, for the same parameters, the output can vary. Thus for the data collection, we run the algorithm 10 times for each parameter set. The results that are presented here are the averages. We will look at how the limit on number of leaf nodes visited and the number of trees built impact the results.

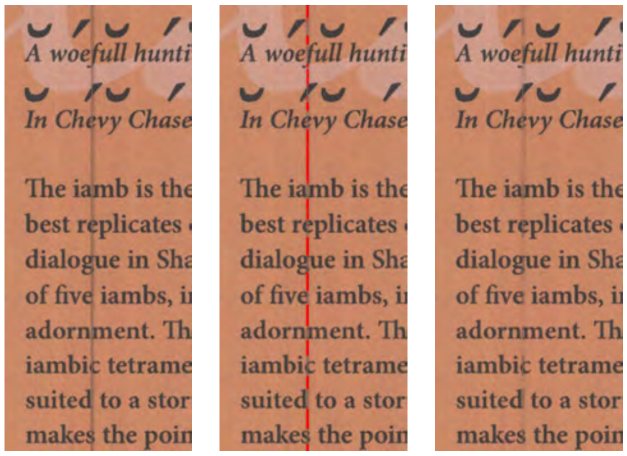

(a) Original image (b) Image with defective mask (c) Healed image

Figure 20. *Example 3 of healed image when the streak is wide.*

First, we compare how the number of trees we use affects the performance of the algorithm. Figures 21 and 22 show quality and time performance with three settings. In each setting, each tree can visit only a fixed number of leaves, while the number of trees linearly increases. As we can see, the mean $\overline{\Delta E}$ drops significantly when more than 2 trees are used. But if the number of leaves is not too small, it does not decrease significantly as the number of trees used increases further. Increasing the number of leaves from 50 to 400 significantly decreases the error for each number of trees used. However, further increasing the number of leaves beyond 400 does not significantly further decrease the error. Looking at time performance, we see that the exhaustive search takes a lot more time than the multiple KD trees approach.

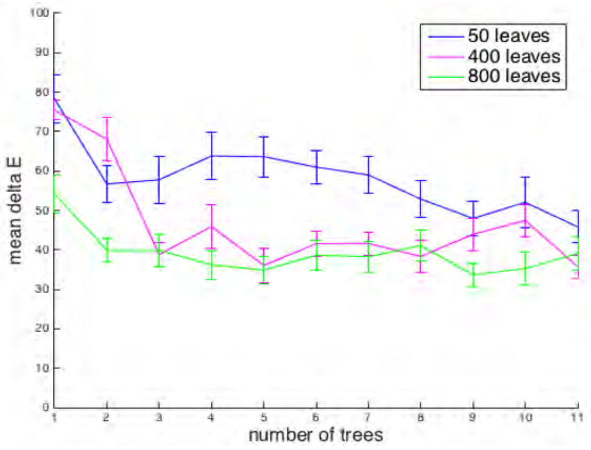


Figure 21. *Impact on* $\overline{\Delta E}$ *of number of trees built.*

Figures 23 and 24 show the effect of the number of leaf nodes each tree is allowed to visit. Increasing the number of leaves visited generally gives an incremental increase in the quality of healing for a setting with 1 tree, and almost no improvement for the other two settings. This is because using 6 or more trees already gives an increase in quality; and the limit on number of leaf nodes is compensated by the higher number of trees used. The computa-

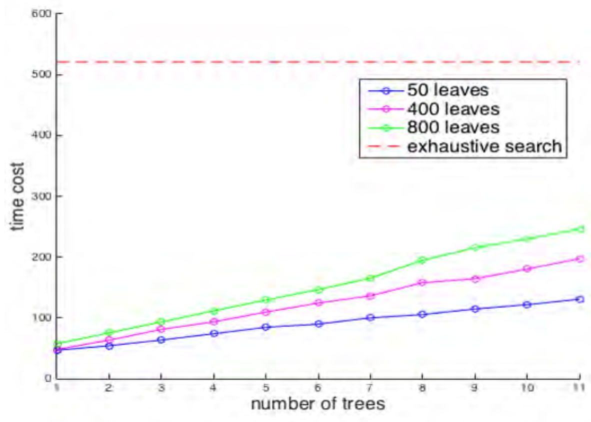


Figure 22. *Impact on time performance of number of trees built.*

tional time goes up linearly as we increase the limit on the number of leaf nodes visited.

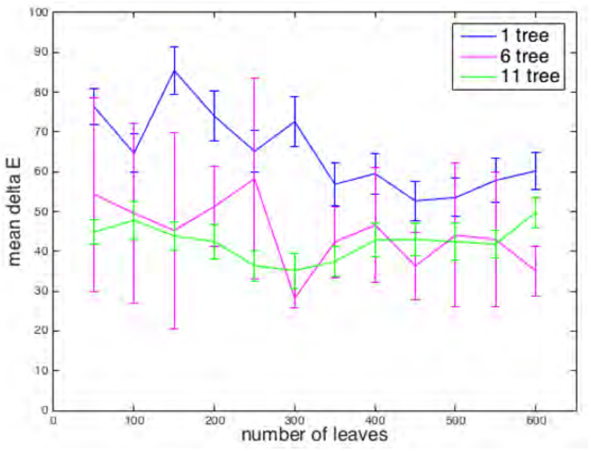


Figure 23. *Impact on $\overline{\Delta E}$ of the limit on number leaf nodes visited.*

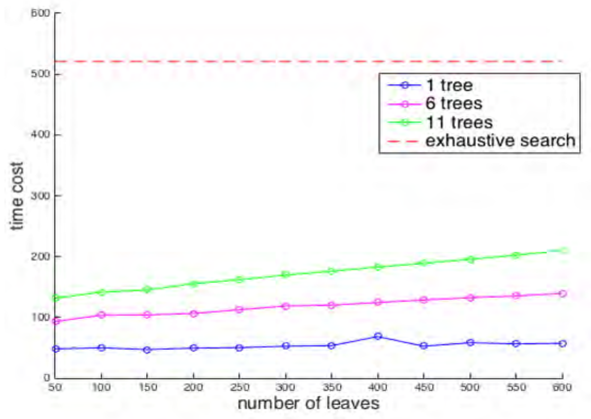
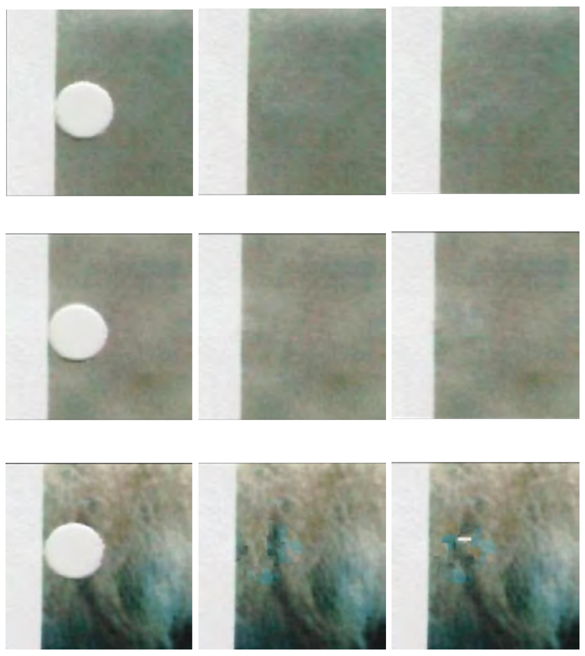


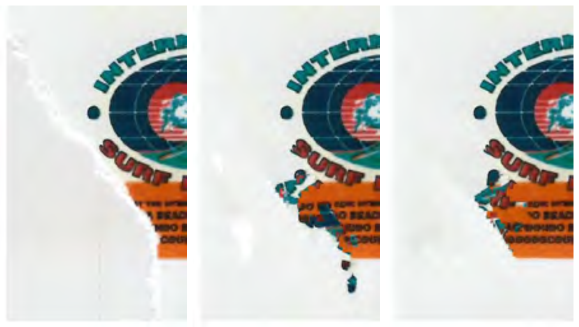
Figure 24. *Impact on time performance of the limit on number leaf nodes visited.*

The images comparing the results obtained using exhaustive search and using multiple KD trees are given in Figs. 25 and 26. The results show that using multiple KD trees provides reasonably good results compared to exhaustive search healing.



(a) (b) (c)

Figure 25. *Comparison of healing punch hole with exhaustive search and with multiple KD trees. Only crops of punch holes are shown: a) original image, b) healed image with exhaustive search and patch size 21, c) healed image with multiple KD trees, patch size 21, 4 trees, 400 leaf node visits limit.*



(a) (b) (c)

Figure 26. *Comparison of healing torn corner with exhaustive search and with multiple KD trees. Only crops of torn corner are shown: a) original image, b) healed image with exhaustive search and patch size 19, c) healed image with multiple KD trees, patch size 19, 4 trees, 400 leaf node visits limit.*

5. Conclusion

In conclusion, we continued our work on dust detection by adding more features, such as table line detection and text protection, and refining our existing features. We investigated two different methods for healing dust streaks. The results show that the cubic interpolation method performs well for narrow and well-defined dust streaks, which is the common case for dust streaks.

The exemplar based method also performs well, but at a much higher computational cost. The exemplar based method was used to heal punch holes and torn corners. It was found that patch size plays a crucial role for getting best results. We also proposed a method for finding the nearest neighbor in exemplar based healing using multiple KD trees. The results show that using multiple KD trees and limiting the number of leaf nodes visited in each tree provides similar healing results as using exhaustive search and saves time. In the future, the table line detection could be improved to include a wider range of tables with varying background contrast.

References

[1] D. Kenzhebalin, X. Liu, N. Yan, P. Bauer, J. Wagner, and J. P. Allebach, ”Detection of streaks caused by dust in the sheetfed scanners,” *Image Quality and System Performance XIV*, (Part of IS&T Electronic Imaging 2017), R. Jenkin, and E. Jin, Eds., Burlingame, CA, 29 January-2 February 2017.

[2] X. Jing, S. Astling, R. Jessome, E. Maggard, T. Nelson, M. Q. Shaw, and J. P. Allebach, ”A General Approach for Assessment of Print Quality,” *Image Quality and System Performance X*, (Part of IS&T and SPIE Electronic Imaging 2013), Vol. 8653, P. D. Burns and S. Triantaphillidou, Eds. San Francisco, CA, 3-7 February 2013.

[3] X. Liu, G. Overall, T. Riggs, R. Silveston-Keith, J. Whitney, G. T. C. Chiu, and J. P. Allebach, ”Wavelet-Based Figure of Merit for Macrouniformity,” *Image Quality and System Performance X*, (Part of IS&T and SPIE Electronic Imaging 2013), Vol. 8653, P. D. Burns and S. Triantaphillidou, Eds. San Francisco, CA, 3-7 February 2013.

[4] W. Wang, G. Overall, T. Riggs, R. Silveston-Keith, J. Whitney, G. T. C. Chiu, and J. P. Allebach, ”Figure of Merit for Macrouniformity Based on Image Quality Ruler Evaluation and Machine Learning Framework,” *Image Quality and System Performance X*, (Part of IS&T and SPIE Electronic Imaging 2013), Vol. 8653, P. D. Burns and S. Triantaphillidou, Eds. San Francisco, CA, 3-7 February 2013.

[5] M. Q. Nguyen, S. Astling, R. Jessome, E. Maggard, T. Nelson, M. Q. Shaw, and J. P. Allebach, ”Perceptual Metrics and Visualization Tools for Evaluation of Page Uniformity,” *Image Quality and System Performance XI*, (Part of IS&T and SPIE Electronic Imaging 2014), Vol. 9016, S. Triantaphillidou and M.-C. Larabi, Eds. San Francisco, CA, 3-5 February 2014.

[6] M. Q. Nguyen and J. P. Allebach, ”Controlling Misses and False Alarms in a Machine Learning Framework,” *Image Quality and System Performance XII*, (Part of IS&T and SPIE Electronic Imaging 2015), Vol. 9396, M.-C. Larabi and S. Triantaphillidou, Eds. San Francisco, CA, 8- 12 February 2015.

[7] W. Wang, Y. Guo, and J. P. Allebach, ”Image Quality Evaluation Using Image Quality Ruler and Graphical Model,” *IEEE International Conference on Image Processing 2015*, Quebec City, Canada, 27-30 September 2015.

[8] S. Hu, H. Nachlieli, D. Shaked, S. Shiffman, and J. P. Allebach, ”Color-Dependent Banding Characterization and Simulation on Natural Document Images,” *Color Imaging XVII: Displaying, Processing, Hardcopy, and Applications*, (Part of IS&T and SPIE Electronic Imaging 2012), Vol. 8292, R. Eschbach, G. Marcu, and A. Rizzi, Eds., San Francisco, CA, 23-26 January 2012.

[9] X. Jing, H. Nachlieli, D. Shaked, S. Shiffman, and J. P. Allebach, ”Masking Mediated Print Defect Visibility Predictor,” *Image Quality and System Performance IX*, (Part of IS&T and SPIE Electronic Imaging 2012) Vol. 8293, F. Gaykema and P. D. Burns, Eds, San Francisco, CA, 23-26 January 2012.

[10] J. Zhang, H. Nachlieli, D. Shaked, S. Shiffman, and J. P. Allebach, ”Psychophysical Evaluation of Banding Visibility in the Presence of Print Content,” *Image Quality and System Performance IX*, (Part of IS&T and SPIE Electronic Imaging 2012), Vol. 8293, F. Gaykema and P. D. Burns, Eds, San Francisco, CA, 23-26 January 2012.

[11] J. Zhang, S. Astling, R. Jessome, E. Maggard, T. Nelson, M. Q. Shaw, and J. P. Allebach, ”Assessment of Presence of Isolated Periodic and Aperiodic Bands in Laser Electrophotographic Printer Output,” *Image Quality and System Performance X*, (Part of IS&T and SPIE Electronic Imaging 2013), Vol. 8653, P. D. Burns and S. Triantaphillidou, Eds. San Francisco, CA, 3-7 February 2013.

[12] J. Zhang and J. P. Allebach, ”Estimation of Repetitive Interval of Periodic Bands in Laser Electrophotographic Printer Output,” *Image Quality and System Performance XII*, (Part of IS&T and SPIE Electronic Imaging 2015), Vol. 9396, M.-C. Larabi and S. Triantaphillidou, Eds. San Francisco, CA, 8-12 February 2015.

[13] X. Jing, S. Astling, R. Jessome, E. Maggard, T. Nelson, M. Shaw, and J. P. Allebach, ”Electrophotographic Ghosting Detection and Evaluation,” *NIP-31 IS&T 2015 Conference on Digital Fabrication and Digital Printing*, Portland, OR, 27 September-1 October 2015.

[14] J. Wang, T. Nelson, R. Jessome, S. Astling, E. Maggard, M. Shaw, and J. Allebach, ”Local Defect Detection and Print Quality Assessment,” *International Congress of Imaging Science (ICIS)*, Tel Aviv, Israel, 12-14 May 2014.

[15] J. Wang, T. Nelson, R. Jessome, S. Astling, E. Maggard, M. Q. Shaw, and J. P. Allebach, ”Local Defect Detection and Print Quality Assessment,” *Image Quality and System Performance XIII* (Part of IS&T Electronic Imaging 2016), R. Jenkin and M.-C. Larabi, Eds. San Francisco, CA, 14-18 February 2016.

[16] N. Yan, E. Maggard, R. Fothergill, R. J. Jessome, and J. P. Allebach, ”Autonomous Detection of ISO Fade Point with Color Laser Printers,” *Image Quality and System Performance XII*, (Part of IS&T and SPIE Electronic Imaging 2015), Vol. 9396, M.-C. Larabi and S. Triantaphillidou, Eds. San Francisco, CA, 8-12 February 2015.

[17] Y. Ju, E. Maggard, R. J. Jessome, and J. P. Allebach, ”Autonomous Detection of Text Fade Point with Color Laser Printers,” *Image Quality and System Performance XII*, (Part of IS&T and SPIE Electronic Imaging 2015), Vol. 9396, M.-C. Larabi and S. Triantaphillidou, Eds. San Francisco, CA, 8-12 February 2015.

[18] W. Wang, P. Bauer, J. K. Wagner, and J. P. Allebach, ”MFP Scanner Diagnostics Using Self-Printed Target to Measure the Modulation Transfer Function,” *Image Quality and System Performance XI*, (Part of IS&T and SPIE Electronic Imaging 2014), Vol. 9016, S. Triantaphillidou and M.- C. Larabi, Eds. San Francisco, CA, 3-5 February 2014.

[19] M. Kim, J. P. Allebach, P. Bauer, and J. K. Wagner, ”MFP Scanner Motion Characterization Using Self-Printed Target,” *Image Quality and System Performance XII*, (Part of IS&T and SPIE Electronic Imaging 2015), Vol. 9396, M.-C. Larabi and S. Triantaphillidou, Eds. San Francisco, CA, 8-12 February 2015.

[20] H. S. Rosario, E. Saber, W. Wu, and K. Chandu, ”Streak Detection in Mottled and Noisy Images,” *Journal of Electronic Imaging 2007*, Vol. 16, 1 October 2007.

[21] C. Guillemot and O. Le Meur, ”Image Inpainting: Overview and Recent Advances,” *IEEE Signal Processing Magazine 2014*, Vol. 31, pp. 127-144, 2014.

[22] J. Weickert, ”Theoretical Foundations of Anisotropic Diffusion in Image Processing,”, in *Theoretical Foundations of Computer Vision*,

Springer, pp. 221-236, 2000.

[23] M. Bertalmio, G. Sapiro, V. Caselles, and C. Ballester, ”Image Inpainting,” in *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, ACM Press/Addison-Wesley Publishing Co., pp. 417-424, 2000.

[24] M. Bertalmio, A. L. Bertozzi, and G. Sapiro, ”Navier-Stokes, Fluid Dynamics, and Image and Video Inpainting,” *IEEE Computer Society Conference on Computer Vision and Pattern Recognition* , Kauai, HI, 8-14 December, 2001.

[25] A. Telea, ”An Image Inpainting Technique Based on the Fast Marching Method,” *Journal of graphics tools*, Vol. 9, pp. 23-34, 2004.

[26] D. Tschumperl, ”Fast Anisotropic Smoothing of Multi-Valued Images Using Curvature Preserving PDE’s,” *International Journal of Computer Vision*, Vol. 68, pp. 65-82, 2006.

[27] A. Levin, A. Zomet, and Y. Weiss, ”Learning How to Inpaint From Global Image Statistics.” *International Conference on Computer Vision*, pp. 305-312, 2003.

[28] L.-Y. Wei and M. Levoy, ”Fast Texture Synthesis Using Tree-Structured Vector Quantization,”, *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, ACMPress/Addison-Wesley Publishing Co., pp. 479-488, 2000.

[29] A. Bugeau, M. Bertalmio, V. Caselles, and G. Sapiro, ”A Comprehensive Framework for Image Inpainting,” *IEEE Transactions on Image Processing*, Vol. 19, no. 10, pp. 2634-2645, 2010.

[30] A. Criminisi, P. Perez, and K. Toyama, ”Object Removal by Exemplar-Based Inpainting,” *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, Vol. 2, 2003.

[31] Z. Xu and J. Sun, ”Image Inpainting by Patch Propagation Using Patch Sparsity,” *IEEE Transactions on Image Processing*, Vol. 19, no. 5, pp. 1153-1165, 2010.

[32] M. Elad, J.-L. Starck, P. Querre, and D. L. Donoho, ”Simultaneous Cartoon and Texture Image Inpainting Using Morphological Component Analysis (MCA),” *Applied and Computational Harmonic Analysis*, Vol. 19, no. 3, pp. 340-358, 2005.

[33] O. LeMeur, J. Gautier, and C. Guillemot, ”Examplar-Based Inpainting Based on Local Geometry,”, *IEEE International Conference on Image Processing 2011*, pp. 3401-3404, Brussels, Belgium, 11-14 September 2011.

[34] I. Ram, M. Elad, and I. Cohen, ”Image Processing Using Smooth Ordering of Its Patches,” *IEEE Transactions on Image Processing*, Vol. 22, no. 7, pp. 2764-2774, 2013.

[35] S. Di Zenzo, ”A Note on the Gradient of a Multi-Image,” *Computer Vision, Graphics, and Image Processing*, Vol. 33, no. 1, pp. 116-125, 1986.

[36] E. Catmull and R. Rom, ”A Class of Local Interpolating Splines,” *Computer Aided Geometric Design* , R. E. Barnhill and R. F. Riesenfeld, Eds. New York: Academic, pp. 317-326, 1974.

[37] B. Julesz, ”Textons, the Elements of Texture Perception, and Their Interactions,” *Nature*, Vol. 290, no. 5802, pp. 91-97, 1981.

[38] D. G. Lowe, ”Distinctive Image Features from Scale-Invariant Keypoints,” *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91-110, 2004.

[39] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman ,”Object Retrieval with Large Vocabularies and Fast Spatial Matching,” *IEEE Computer Society Conference on Computer Vision and Pattern Recognition* , 2007.

[40] J. Sivic, A. Zisserman et al., ”Video Google: a Text Retrieval Approach to Object Matching in Videos.”, *International Conference on Computer Vision*, vol. 2, no. 1470, pp. 1470-1477, 2003.

[41] M. Muja and D. G. Lowe, ”Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration,”, *VISSAPP 2009*, pp. 331-340, 2009.

[42] A. Beygelzimer, S. Kakade, and J. Langford, ”Cover Trees for Nearest Neighbor,” *Proceedings of the 23rd International Conference on Machine Learning* pp. 97-104, 2006.

[43] B. Leibe, K. Mikolajczyk, and B. Schiele, ”Efficient Clustering and Matching for Object Class Recognition.” *BMVC 2006*, pp. 789-798, 2006.

Author Biography

Daulet Kenzhebalin received his BS in computer engineering from Purdue University (2015). Currently he is pursuing PhD in electrical engineering at Purdue University. His primary area of research has been image processing.

Ni Yan received her PhD degree in Electrical and Computer Engineering from Purdue University, West Lafayette IN, USA (2017), and her BE in Electrical and Computer Engineering from Beijing University of Posts and Telecommunications, Beijing, China (2012). Her research has mainly focused on image quality assessment, and machine learning. Now she is working as a research scientist in Datavisor Inc, Mountain View CA, USA.

Jerry Wagner received his BS and MS in Electrical Engineering from Pennsylvania State University (1978 and 1980) and his MS in Computer Science from Rochester Institute of Technology (1992). He has worked in the Research Division at Eastman Kodak Company, and he has been employed in product development by HP Inc. for the last 17 years. His work has focused on image processing for scanners and multi-function peripherals.

Peter Bauer received his Diplom-Ingenieur (FH) in Computer Science from the University of Applied Sciences in Rosenheim Germany. He has worked in Hewlett Packard Research Laboratories in Bristol for 4 years. For the last 20 years he has worked for HP Inc. in product development focused on image processing for embedded systems.

Jan P. Allebach is Hewlett-Packard Distinguished Professor of Electrical and Computer Engineering at Purdue University. Allebach is a Fellow of the IEEE, the National Academy of Inventors, the Society for Imaging Science and Technology (IS&T), and SPIE. He was named Electronic Imaging Scientist of the Year by IS&T and SPIE, and was named Honorary Member of IS&T, the highest award that IS&T bestows. He has received the IEEE Daniel E. Noble Award and the IS&T/OSA Edwin Land Medal, and is a member of the National Academy of Engineering.