Removing Salt-and-Pepper Noise from Binary Images of Engineering Drawings

Hasan S. M. Al-Khaffaf[1], Abdullah Z. Talib, Rosalina Abdul Salam
School of Computer Sciences, Universiti Sains Malaysia, 11800 USM Penang, Malaysia
hasan, azht, rosalina@cs.usm.my

Abstract

Removing noise in engineering drawing images is important before applying image analysis processes. Noise should be removed while keeping the fine detail of the image intact. A noise removal algorithm that can remove noise while retaining fine graphical elements is presented in this paper. The algorithm studies the neighborhood of thin lines before choosing to remove or retain it. Real scanned images from GREC'03 and GREC'05 arc segmentation contests corrupted by 15% uniform salt/pepper noise are used in this experiment. Objective distortion measurements including PSNR and MSE show that our algorithm gives better quality images compared with other methods.

1. Introduction

Noise may appear in engineering drawing images during data acquisition such as scanning [1]. The noise should be removed prior to performing image analysis processes. The difficulty in removing salt/pepper noise from binary image is due to the fact that image data as well as the noise share the same small set of values (either 0 or 1) which complicates the process of detecting and removing the noise. This is different from grey images where salt/pepper noise could be distinguished as pixels having big difference in grey level values compared with their neighborhood. Many algorithms have been developed to remove salt/pepper noise in document images with different performance in removing noise and retaining fine details of the image. Most methods can easily remove isolated pixels while leaving some noise attached to graphical elements. Other methods may remove attached noise with less ability in retaining thin graphical elements. In this paper many algorithms will be studied. Then a procedure that enhances removal of salt/pepper noise is proposed. This procedure is then incorporated with a third party method to produce an algorithm that removes noise and retains thin graphical elements. The following definitions are assumed throughout this paper. Foreground pixels are black (or 1) while background pixels are white (or 0).

Median filter is a well known method that can remove salt/pepper noise from images [2]. The removal of noise is performed by replacing the value of window center by the median of center neighborhood. Its disadvantage is the distortion of corners and thin lines in the image.

Center Weighted Median (CWM) is a superior enhancement to Median filter [3]. The center is given more weight compared to the surrounding neighbors. This filter can retain fine details of the image.

Morphological operators are another mean to process images using set theory [4]. It can be used to remove salt/pepper noise by a combination of dilation/erosion operators. The disadvantage of this method is that graphical elements may suffer a merge with close ones and thin lines will be removed.

Another filter that can remove noise from document images is the kFill filter [5, 6]. Since it is related to the proposed work, its description is shown in the next section (Sect. 2). Its disadvantage is the shortening of one-pixel-wide graphical objects from their end points.

A single iteration filter based on kFill is proposed in [4]. The kFill algorithm is redesigned to work in one iteration. Its disadvantage is that the algorithm only shortens one-pixel-wide blobs connected to graphical elements without removing it while it also shortens one-pixel-wide graphical elements.

In a recent study [1], an algorithm based on activity detection is proposed to remove salt/pepper noise from document images that contain graphics, characters, and dithered areas. One disadvantage of this algorithm is that noise speckles adjacent to graphical elements could

[1]This work is supported by a Science Fund grant from the Ministry of Science, Technology, and Innovation (MOSTI), Malaysia under project number 01-01-05-SF0147. The first author is also supported by USM Fellowship.

not be removed since it is considered part of graphical element connected components. The other drawback is the many thresholds (five) that need to be preset.

2. kFill algorithm

kFill is a filter that utilizes two sub iterations in order to remove salt and pepper noise [5, 6]. A $k * k$ window is moved in raster scan over the whole images. In the rest of algorithm description, salt removal sub iteration is assumed. For pepper noise removal one needs to use white instead of black in the following paragraphs. Three variables (n, c ,r) are calculated from pixels surrounding the center, where n represents the number of black pixels in the window, c stands for the number of connected components of black pixels in the window, and r is the number of black pixels in window corners. The center is excluded in the calculation of the three variable values. The center is filled with black if the conditions in the following equation are met:

$$(c = 1) \wedge [(n > 3k-4) \vee [(n = 3k-4) \wedge (r = 2)]] \quad (1)$$

Comparing c variable with 1 ensures that the connectivity of neighboring pixels in not changed. The n variable is used to prevent filling the core when the number of neighboring pixels are little. When there are moderate surrounding pixels, protecting sharp corners in the drawing is guaranteed by the r variable.

3. Proposed algorithm

The proposed algorithm is based on kFill [5, 6]. Before describing the proposed algorithm, the following definitions are set: An end point is a pixel with only one 8-connected neighbor pixel similar to its value. A branch point is a pixel which has three or more neighbors where all pixels have the same value. A $3 * 3$ window is used in cleaning and tracking operations. The pixel at the center of the window is called the core.

The algorithm has two sub iterations for black and white fill. In black fill sub iteration the window slides in raster scan manner over the image. Three variables are calculated from the pixels surrounding the core (n, c, r). These variables are calculated as in kFill method. If the conditions in Eq. 1 are met then one of the following equations will hold

$$(c = 1) \wedge [(n = 6) \vee (n = 8) \vee [(n = 5) \wedge (r = 2)]] \quad (2)$$

$$(n = 7) \quad (3)$$

The two equations are directly derived from Eq. 1 with k value sets to 3. The formulation of the two equations is similar to that of kFill algorithm. However, one condition is excluded from Eq. 1 to form a separate equation (Eq. 3). This condition is related to the processing of end points of expected thin lines. While Eq. 2 include all the other conditions where the core will be filled. If conditions in Eq. 2 are met then the core will be changed to black. Otherwise if the condition in Eq. 3 is met (i.e. an end point is detected) an additional procedure (named TrackAndMayDel and to be discussed in section 4) is called to verify whether this thin line is part of a noisy part of the image or not. Note that in the original algorithm the core is filled if conditions in Eq. 1 are met while in our algorithm the core is considered candidate for filling if conditions in Eq. 1 are met then it is up to Eq. 2 and Eq. 3 to decide the correct action. Figure 1 shows the flowchart of the proposed algorithm. The proposed algorithm is more compute-intensive than kFill, if only one iteration is considered. In case of multiple iterations, the former method may perform with less iterations compared to the latter. One reason is that our method has the ability to stop removing end points of thin lines, while kFill lacks such feature.

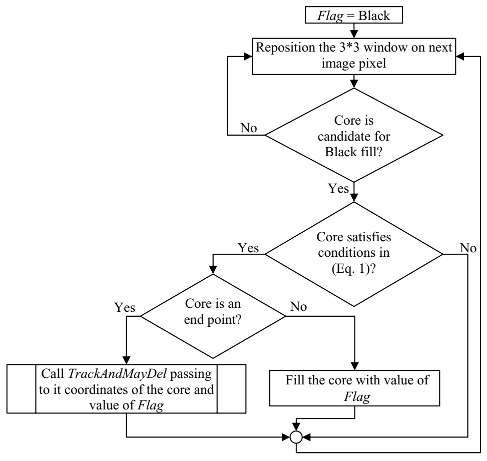

Flag = Black
Reposition the 3*3 window on next image pixel
Core is candidate for Black fill?
No
Yes
Core satisfies conditions in (Eq. 1)?
Yes
No
Core is an end point?
Yes
No
Call *TrackAndMayDel* passing to it coordinates of the core and value of *Flag*
Fill the core with value of *Flag*

Figure 1: **Flowchart of proposed algorithm (black fill sub iteration).**

4. Proposed TrackAndMayDel procedure (TAMD)

To overcome some of the limitations in noise removal algorithms, a procedure is proposed to help better choices before changing pixels values. The idea is to perform further investigation before changing of end points. The proposed procedure has the following properties:

1. It works during black- and white-fill sub iterations.

2. Its work is based on tracking and studying a small number of connected pixels around the pixel to be filled. After which a decision will be made to either retain or change tracked pixels.

3. The proposed procedure could be integrated with noise removal algorithms logic or performed as a post-processing operation.

The method starts performing when an end point is detected. The purpose of this procedure is to check whether the already detected end point is part of thin line or part of small spurious branch connected to a graphical element. The $TAMD$ procedure gives the proposed algorithm the ability to check pixels outside the $3 * 3$ window. This widens the area of the image viewed by the algorithms without the need to use a larger window size (which is costly in term of processing time). A flowchart of the procedure is shown in Figure 2. When there are two actions inside the process symbol in Figure 2, the first one ends with '.' to separate the two actions.

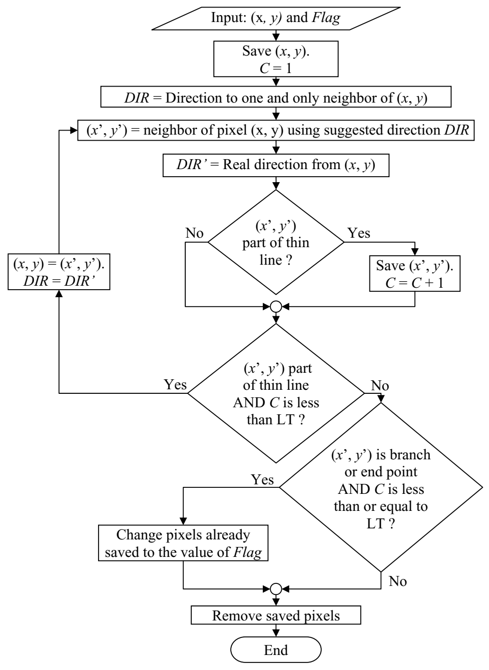

Input: (x, y) and Flag
Save (x, y). C = 1
DIR = Direction to one and only neighbor of (x, y)
(x', y') = neighbor of pixel (x, y) using suggested direction DIR
DIR' = Real direction from (x, y)
(x', y') part of thin line ?
No
Yes
Save (x', y'). C = C + 1
(x, y) = (x', y'). DIR = DIR'
(x', y') part of thin line AND C is less than LT ?
Yes
No
(x', y') is branch or end point AND C is less than or equal to LT ?
Yes
No
Change pixels already saved to the value of Flag
Remove saved pixels
End

Figure 2: **TAMD flowchart**

The procedure consists of two parts. The first is a tracking part in which the branch is tracked starting from its end point and the tracked pixels are recorded. Tracking will stop when either an end point or branch point is reached; or the number of tracked points exceeds a predefined threshold (LT). Second, the procedure will remove the already tracked branch if it is shorter than or equal to LT; and it is either connected to thick element or it is a separate object. Tracking is performed by $GetNeighborPixel$ procedure (The 4th step in Fig. 2). The functionality of this procedure is described by means of an example. Suppose tracking starts at pixel (x, y). Finding the neighbor of pixel (x, y) involves searching its neighborhood. We follow a trend that makes it easier and faster to get the neighbor by searching in a suggested direction (DIR) first, then in the directions that make little deviation with it. For the very first pixel (x, y), there is no suggested direction, so the search starts in its neighbors one by one till it finds the only neighbor, lets call it $(x`, y`)$. The real direction from (x, y) to $(x`, y`)$ is considered as the suggested direction to be used in tracking from $(x`, y`)$ to its neighbor. The operation continued until stop conditions are met.

5. Experimental results and discussion

The proposed algorithm as well as six other algorithms are implemented in C++ using Visual Studio 7.0. The experiment is performed on a PC running Windows XP operating system. Our test data are eight real scanned images used in arc segmentation contests held during Graphics Recognition Workshop 2003 and 2005 [7]. The images are then corrupted by 5%, 10%, and 15% uniform salt/pepper noise. Figure 3 shows clean and noisy images. Figure 4 show image 1.tif (one of many images used in our study) cleaned by different algorithms. Due to space constraint, only partial parts of the images corrupted by 15% noise are shown in Figures 3 and 4.

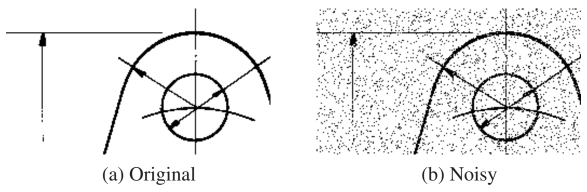
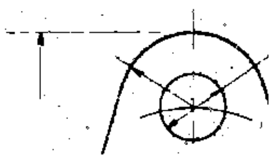
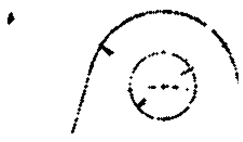


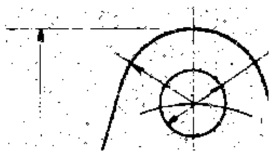
Figure 3: **Original and noisy images.**

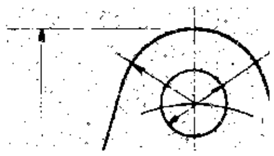
The algorithms studied have parameters (e.g. window size). To keep the size of the experiment within a manageable size, the test in this paper is limited to $3 * 3$ window size for median, CWM, kFill and Enhanced kFill. The weight for CWM is set to 5 since it will preserve one pixel wide straight lines as proved by [3]. The default values (chosen for strong cleaning) suggested by the authors are used in Activity Detector. Based on an empirical test on many images corrupted by salt/pepper noise a value of 8 is used for LT in our proposed method.

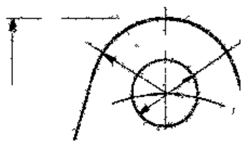
Table 1 shows the performance of the proposed method with other algorithms. Our proposed algorithm shows higher PSNR values compared to the other methods. From Figure 4 it is easy to notice that the proposed algorithm can remove noise (separate or attached to graphical elements) while retaining thin lines at the same time. Other algorithms may be good at removing noise, but not retaining thin lines or vice versa.

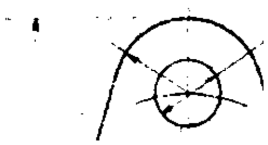

(a) Proposed


(b) Opening-Closing


(c) kFill


(d) Enhanced kFill


(e) Activity Detector


(f) Median

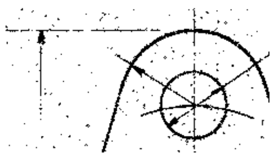

(g) CWM 5

Figure 4: **Image 1.tif cleaned by different methods**

References

[1] P.Y. Simard and H.S. Malvar, "An efficient binary image activity detector based on connected components", Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing, pp.229–32, 2004.

[2] M. Sonka, V. Hlavac, and R. Boyle, *Image Processing, Analysis, and Machine Vision*, Second ed: PWS, 1999.

Table 1: Performance comparison among different algorithms

Image	Criteria	OpnClos	kFill	ours	EnhkFill	ActDetec	Median	CWM5
1.tif	PSNR	16.1	18.99	20.85	19.08	19.47	18.56	18.08
	MSE	0.0246	0.0126	0.0082	0.0123	0.0113	0.0139	0.0156
2.tif	PSNR	16.04	19.21	20.96	19.29	20.02	19.47	18.43
	MSE	0.0249	0.012	0.008	0.0118	0.01	0.0113	0.0144
5.tif	PSNR	17.6	19.13	21.41	19.15	19.57	20.33	18.24
	MSE	0.0174	0.0122	0.0072	0.0122	0.011	0.0093	0.015
6.tif	PSNR	21.4	20.35	23.76	20.38	22.73	23.73	19.25
	MSE	0.0072	0.0092	0.0042	0.0092	0.0053	0.0042	0.0119
7.tif	PSNR	17.04	19.16	21.33	19.19	19.53	20.41	18.09
	MSE	0.0197	0.0121	0.0074	0.0121	0.0111	0.0091	0.0155
8.tif	PSNR	19.55	20.12	22.88	20.2	21.67	22.38	19.17
	MSE	0.0111	0.0097	0.0052	0.0095	0.0068	0.0058	0.0121
9.tif	PSNR	18.31	19.96	22.67	19.96	21.23	21.31	18.97
	MSE	0.0147	0.0101	0.0054	0.0101	0.0075	0.0074	0.0127
10.tif	PSNR	17.03	19.12	21.15	19.19	19.23	19.95	18.04
	MSE	0.0198	0.0122	0.0077	0.0121	0.0119	0.0101	0.0157

[3] S.J. Ko and Y.H. Lee, "Center weighted median filters and their applications to image enhancement", *Circuits and Systems, IEEE Transactions on*, 38(9): pp. 984-993, 1991.

[4] K. Chinnasarn, Y. Rangsanseri, and P. Thitimajshima, "Removing salt-and-pepper noise in text/graphics images", The 1998 IEEE Asia-Pacific Conference on Circuits and Systems, Chiangmai, pp.459–462, 1998.

[5] L. O'Gorman, "Image and document processing techniques for the rightpages electronic library system", Proc. 11th IAPR International Conference on Pattern Recognition. Conference B: Pattern Recognition Methodology and Systems, The Hague, pp.260–263, 1992.

[6] G.A. Story, L. O'Gorman, D. Fox, L.L. Schaper, and H.V. Jagadish, "The rightpages image-based electronic library for alerting and browsing", Computer, vol.25, no.9, pp.17–26, 1992.

[7] L. Wenyin, Real scanned images used in Arc Segmentation Contests, 2008. `http://www.cs.cityu.edu.hk/~liuwy/ArcContest/`