An Algorithm for Extracting Lines and Circles with High-Speed and Accuracy

John W. Gates, Miki Haseyama, Hideo Kitajima

School of Engineering, Hokkaido University
N-13, W-8, Kita-ku, Sapporo 060-0826, Japan
john@media.eng.hokudai.ac.jp

Abstract

This paper presents a fast and accurate algorithm for extracting lines and circular arcs from digital images. Conventional line and circle extraction algorithms first extract the edge pixels and then connect these edge pixels into lines or circles. In contrast the proposed algorithm extracts the lines and circular arcs directly from the input image by tracing the perimeter of an intensity region and then using simple linear regression to compute the equations of the lines and arcs. This proposed method allows the algorithm to use the intensity information in the image to correlate the edge pixels, thus simplifying the computation of the lines and arcs and increasing the robustness in the presence of noise. Experiments were performed on a variety of synthetic and natural images (256 × 256 gray-scale) to demonstrate that the algorithm can extract lines with high accuracy, while realizing an average computation time of 0.0058 seconds using a 1.4 GHz Athlon processor.

1 Introduction

The extraction of lines and circles from digital images is one of the fundamental problems in the field of digital image processing and pattern recognition [1], [2]. Even though this problem has been studied for many years and many different algorithms have been proposed, there is still a need for reliable, high-speed line and circle extraction algorithms for real-time, real-world applications. Another problem is that most of the research has concentrated on the extraction of lines as this is much simpler than the extraction of circles. The reason is that lines have only two degrees of freedom whereas circles contain three degrees of freedom and are therefore much more complex. Furthermore, there are no algorithms that can extract both lines and circles simultaneously.

Some of the well-known line extraction algorithms are the Hough transform [1], [2], [3] and its derivatives [4], [5], [6], [7], chain coding [8], polygonal approximation [9], and other algorithms [1], [2], [10], [11]. The Hough transform, which is one of the best-known line extraction algorithms, uses a histogram in order to compensate for the lack of correlation between the extracted edge pixels. This is very time consuming as every edge pixel influences every other edge pixel. Many papers have been written on improving the speed of the Hough transform. However, even with these improvements, many speed and accuracy problems still remain.

There are also several different approaches for the extraction of circles from digital images [12], [13], [14], [15], [16]. However, most of these algorithms are very slow and cannot be easily used in real-time applications. Most of the current algorithms extend the Hough transform line detection algorithm to the detection of circles. The Hough transform detects lines by transforming the input image into a two-dimensional histogram which corresponds to the two degrees of freedom present in the equation of a straight line. Thus when this principle is extended to the detection of circles, a three-dimensional histogram is required. This extra dimension greatly increases the computation time of the algorithm. To overcome this limitation, various approaches have been suggested, such as using the gradient information to improve the detection of the center of the circle. However, even these faster algorithms require a significant amount of time to extract circles from simple images.

The fundamental difference between the proposed algorithm and these other algorithms is that the proposed algorithm processes the original intensity image directly, whereas the other algorithms use at least one pre-processing step to locate the edge pixels. This pre-processing step is used to reduce the effects of blurred edge transitions and other sources of noise, which are common in natural images. However there are three serious problems with pre-processing that reduce the effectiveness of the subsequent line or circle extraction algorithms. The first problem is that the edge pixels are

usually correlated in the intensity image in that they form the boundary between two relatively uniform intensity regions, and this correlation is removed by pre-processing. The second problem is that pre-processing consumes a significant amount of processing time, especially if the image is complex and extraneous edge pixels need to be removed. The third problem is that the edge pixel extraction is usually performed by a high-pass spatial filter and is therefore sensitive to high-frequency noise.

Another fundamental difference between the proposed algorithm and the traditional algorithms is that the proposed algorithm extracts both lines and circular arcs simultaneously. This is cannot be done by the traditional methods as the histograms for extracting lines are computed differently than the histograms used for extracting circles. However, images from real-world applications usually contain both lines and arcs, thus it is advantageous to be able to extract both types of features at the same time.

In this paper an accurate, high-speed extraction algorithm that simultaneously extracts both lines and circular arcs directly from the intensity image is presented. By using the correlation provided by the intensity information in the image, simple linear regression can be used to extract the lines and circular arcs, thus greatly reducing the computation cost of the extraction process. The proposed algorithm has a variety of practical applications due to its high computation speed. Some of these applications are driver assistance, autonomous robots, high-speed character recognition, pattern recognition, face recognition, aerial image classification, security systems, and automatic quality control.

In the following section the proposed algorithm is explained. The basic principle of the algorithm is to combine the edge pixel extraction and the feature extraction into a single step. To do this any edge operator may be used to locate and follow the boundary of an intensity region. Then the features of the lines and circular arcs that form the boundary are calculated using simple linear regression. In the third section experiments are performed using a variety of synthetic and natural images to verify the speed and accuracy of the proposed algorithm. Finally the paper is concluded in the fourth section.

2 The Algorithm

2.1 The Definition of a Line and a Circle

Almost all of the conventional line and circle extraction algorithms first generate a binary edge image by pre-processing the original intensity image with an edge extraction operator. Then the features of the line or circle are calculated using the binary edge image. As a result of this pre-processing the definition of a line becomes, "a collection of edge pixels that satisfy the equation of a straight line, $y = m \cdot x + b$." Where x and y are the coordinates of the edge pixel, m is the slope and b the intercept of the line. In a similar manner, the definition of a circle becomes, "a collection of edge pixels that satisfy the equation, $(x - X_0)^2 + (y - Y_0)^2 = R^2$." Where R is the radius of the circle, and X_0 and Y_0 are the coordinates of the origin of the circle. However an alternative definition of a line is, "the straight line segments that form the border between two regions of different intensity within an image," and the corresponding definition of a circle is, "the circular arc segments that form the border between two regions of different intensity within an image." These new definitions for lines and circles lead directly to a new extraction algorithm with the following steps:

(1) Locate a region of uniform intensity.
(2) Trace the perimeter of the region and record the coordinates of the pixels.
(3) Form lines or circular arcs from this set of correlated pixel coordinates.
(4) Repeat steps 1 through 3 for the remaining regions of uniform intensity.

Step (1) is similar to the pre-processing used by the conventional algorithms and step (3) is similar to the conventional extraction algorithms. The difference is that the conventional algorithms perform these steps individually in series, while the proposed algorithm performs them in parallel. Steps (1), (2), and (3) in the proposed algorithm correspond to the scanning, perimeter tracing, and feature extraction modes, respectively, and are described below.

2.2 The Scanning Mode

The scanning mode begins at the bottom, left corner of the image, (0, 0), and scans from left to right and bottom to top looking for a significant transition in image intensity that would indicate the presence of a new intensity region. Once such an intensity transition has been detected the scanning mode will call the perimeter tracing mode to extract the pixels that correspond to the border of the intensity region. However, if this pixel has already been determined to belong to a line or circular arc the scanning mode will advance to the next pixel without calling the perimeter tracing mode. This prevents the same line or circle from being extracted multiple times.

Any of the conventional edge operators may be used in this mode to detect an intensity transition. In this paper the following equations were used to determine if the pixel at (x, y) is the start of a new intensity region:

$$\left| I_{x,y} - I_{x+1,y} \right| > T \quad (1)$$

$$\left| I_{x,y} - I_{x,y+1} \right| > T \quad (2)$$

Where $I_{x,y}$ is the intensity of the pixel at (x, y) and T is the threshold. This threshold level was determined by experimentation and set at a value of 40 for 8-bit grayscale

images and was not changed for all the experiments. If either equation (1) or (2) is true then a new intensity region has been located and the perimeter tracing mode described in the next section is used to trace the perimeter of the intensity region.

2.3 The Perimeter Tracing Mode

When the value of the scanning mode exceeds a certain threshold the border of an intensity region has been located and the algorithm begins the perimeter tracing mode. The tracing mode assumes that a pixel with an intensity within the range of the starting pixel plus or minus the threshold is "inside" of the intensity region and if the intensity is outside this range then the pixel is assumed to be "outside" of the intensity region.

To follow the perimeter of the intensity region the tracing mode must be able to move to each of the eight neighboring pixels, thus eight separate tracing functions are required. In each of the tracing functions at least one of the four cardinal pixels must be outside of the intensity region. From this outside pixel the tracing function will search for the nearest pixel inside the region by using the tracing functions shown in Fig. 1.

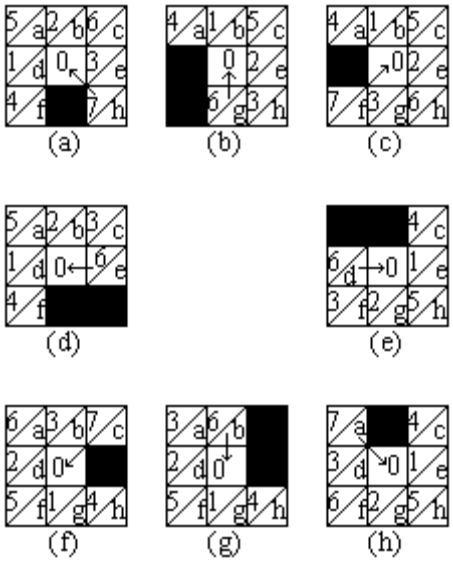


Figure 1: *Perimeter tracing functions. In this figure, the numbers show the searching order, the letters show the tracing function called, the black squares show the pixels that must be outside of the intensity region and the arrows show the direction the tracing mode has moved since the previous iteration.*

To understand Fig. 1, the tracing function shown in Fig. 1(a) will be described in detail. In this figure, the tracing mode has just moved from the bottom right pixel to the center pixel as shown by the arrow. Since the bottom center pixel is known to be outside of the intensity region the tracing function searches the cardinal pixels in a clockwise manner from this pixel, looking for a pixel that is inside the region. The reason the cardinal pixels are given preference over the diagonal pixels is that they represent definite boundaries whereas the diagonal pixels may be noise. If one of the cardinal pixels is inside of the intensity region, its counter-clockwise diagonal pixel will also be tested, for example if the middle left pixel is found to be inside the region then the bottom left pixel will also be tested. Once the intensity transition has been located the tracing function will call the function labeled in the box. For example if the bottom left pixel is determined to be the boundary pixel then tracing function (f) will be called. Then the pixel will be marked as belonging to a line or a circle and its coordinates will be recorded in a list for the line feature extraction mode. It should be noted that the perimeter tracing mode is allowed to backtrack on itself so that it can escape from possible traps. It can be seen that these functions form a recursive algorithm that can move counterclockwise around the intensity region. The tracing mode ends when it arrives back at the starting point or at the edge of the image. Once a terminal condition occurs the perimeter tracing mode will end and the feature extraction mode will begin.

2.4 The Feature Extraction Mode

The feature extraction mode begins at the start of the list of pixel coordinates extracted by the perimeter tracing mode and uses simple linear regression [17] to compute the equations of the line segments and circular arcs. This mode uses the mean-square error (*MSE*) from the actual pixel to the calculated line or circular arc to control which pixels belong to the curve. First, the feature extraction mode assumes the pixels belong to a line segment, however if the *MSE* exceeds the *MSE* threshold then the mode will try to fit the pixels to a circular arc.

To calculate the *MSE* of the best-fit line for the first N pixels in the list the following equations are used:

$$S_{xx} = \sum_{i=1}^{N} x_i^2 - \frac{1}{N}\left(\sum_{i=1}^{N} x_i\right)^2 \quad (3)$$

$$S_{yy} = \sum_{i=1}^{N} y_i^2 - \frac{1}{N}\left(\sum_{i=1}^{N} y_i\right)^2 \quad (4)$$

$$S_{xy} = \sum_{i=1}^{N} x_i y_i - \frac{1}{N}\sum_{i=1}^{N} x_i \sum_{i=1}^{N} y_i \quad (5)$$

If $|S_{xx}| > |S_{yy}|$ then the slope, m, and the intercept, b, of the best-fit line, $y = m \cdot x + b$, and the error sum of squares (*SSE*) are given by Equations (6) to (9), otherwise the complementary equations which replace S_{xx} with S_{yy} and x_i with y_i are used.

$$m = \frac{S_{xy}}{S_{xx}} \quad (6)$$

$$b = \frac{1}{N}\left(\sum_{i=1}^{N} y_i - m\sum_{i=1}^{N} x_i\right) \quad (7)$$

$$SSE = S_{yy} - m \cdot S_{xy} \tag{8}$$

$$MSE = \frac{SSE}{N} \tag{9}$$

If the *MSE* for the line segment is greater than the *MSE* threshold, the least squares circle fitting algorithm given by the following equations is used to calculate the best-fit circle parameters.

$$S_{x^2x} = \sum_{i=1}^{N} x_i^3 - \frac{1}{N}\sum_{i=1}^{N} x_i^2 \sum_{i=1}^{N} x_i \tag{10}$$

$$S_{x^2y} = \sum_{i=1}^{N} x_i^2 y_i - \frac{1}{N}\sum_{i=1}^{N} x_i^2 \sum_{i=1}^{N} y_i \tag{11}$$

$$S_{xy^2} = \sum_{i=1}^{N} x_i y_i^2 - \frac{1}{N}\sum_{i=1}^{N} x_i \sum_{i=1}^{N} y_i^2 \tag{12}$$

$$S_{y^2y} = \sum_{i=1}^{N} y_i^3 - \frac{1}{N}\sum_{i=1}^{N} y_i^2 \sum_{i=1}^{N} y_i \tag{13}$$

$$Y_0 = \frac{S_{x^2y} + S_{y^2y} - \frac{S_{xy}}{S_{xx}}\left(S_{x^2x} + S_{xy^2}\right)}{2\left(S_{yy} - \frac{\left(S_{xy}\right)^2}{S_{xx}}\right)} \tag{14}$$

$$X_0 = \frac{S_{x^2x} + S_{xy^2} - 2Y_0 S_{xy}}{2S_{xx}} \tag{15}$$

$$R = \sqrt{\frac{\sum_{i=1}^{N} x_i^2 + \sum_{i=1}^{N} y_i^2 - 2X_0\sum_{i=1}^{N} x_i - 2Y_0\sum_{i=1}^{N} y_i}{N} + X_0^2 + Y_0^2} \tag{16}$$

Since a circle is a quadratic equation the *MSE* must be calculated by the following equation:

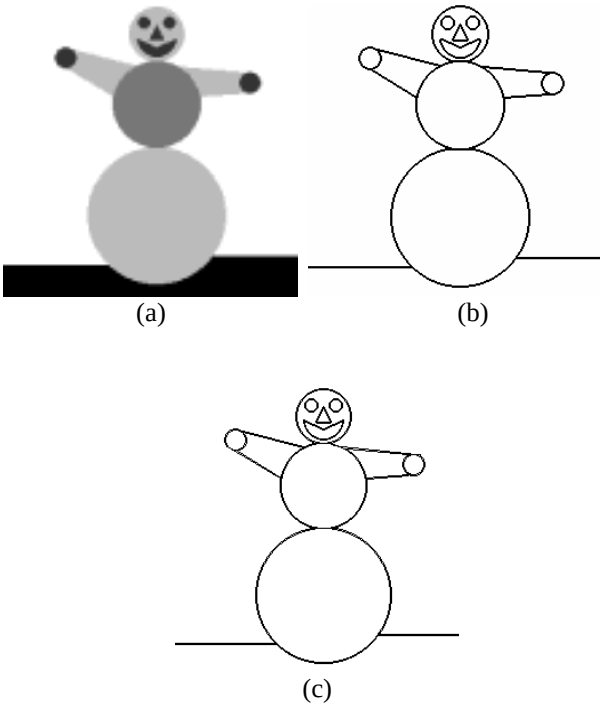
$$MSE = \frac{1}{N}\sum_{i=1}^{N}\left(R - \sqrt{\left(x_i - X_0\right)^2 + \left(y_i - Y_0\right)^2}\right)^2 \tag{17}$$

If the *MSE* is greater than a pre-determined threshold for the given line segment or circular arc then the feature extraction mode will start a new extraction from this pixel. This procedure is continued until all the pixels in the list have been processed, and then the feature extraction mode will return to the scanning mode. The extracted line segments and circular arcs are stored in a list and can be used directly or further analyzed as required by the application.

3 Experiments

In order to demonstrate the high speed and accuracy of the proposed extraction algorithm, two experiments were performed. In the first experiment a synthetic image of a snowman was used to test the robustness of the algorithm. In the second experiment several synthetic and real world image were used to demonstrate the speed of the proposed algorithm. Both experiments were performed on a 1.4 GHz Athlon processor, and all test images were 256 × 256 grayscale images.

3.1 Robustness Experiment



(a) (b) (c)

Figure 2: *Noiseless snowman experiment. (a) original snowman image; (b) true edge pixels; (c) extracted line segments and circular arcs.*

To test the robustness of the line segment and circular arc extraction, the synthetic image of a snowman shown in Fig. 2(a) was used. Since the algorithm extracts lines and circles on both sides of the intensity transition, the algorithm will extract two lines or circles for each side of the intensity region. Therefore, to test the accuracy of the extraction algorithm the true edge pixels shown in Fig. 2(b) were used. The results of the extraction algorithm are shown in Fig. 2(c) and are summarized in Table 1, column 1, by recording the number of lines, the number of circles, the strict accuracy and the ±1 accuracy. The strict accuracy is the percentage of pixels that are exactly the same as those shown in Fig. 2(b) and includes both types of errors due to missing pixels and falsely classified pixels. Therefore it is possible to have a negative accuracy if there are more false pixels than correctly classified pixels. The ±1 accuracy is a more lenient accuracy in that it allows a pixel to fall within a radius of one pixel from the exact pixel. From the results shown in Table 1, column 1, it can be seen that the algorithm extracts the lines and circular arcs with very high accuracy.

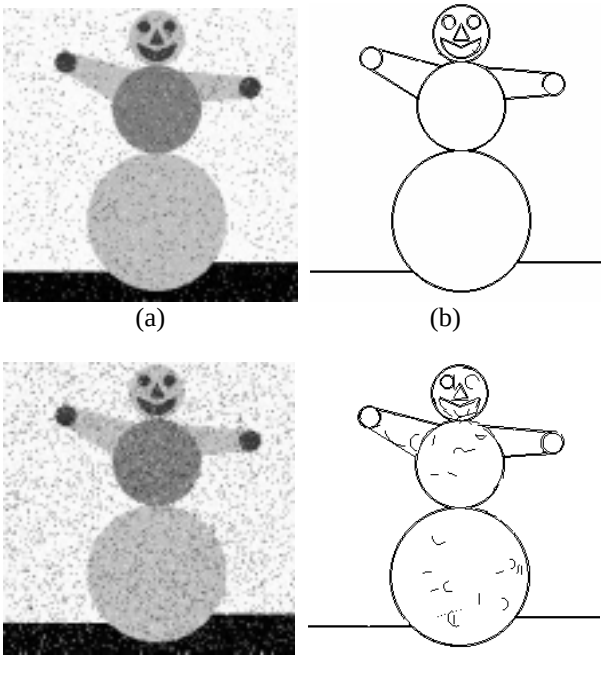


Figure 3: *Corrupted snowman experiment. (a) test image mildly corrupted with Gaussian noise (mean = 0, variance = 100) and with salt and pepper noise (probability = 5%); (b) extraction results; (c) test image heavily corrupted with Gaussian noise (mean = 0, variance = 200) and with salt and pepper noise (probability = 10%); (d) extraction results.*

Table 1: Accuracy Results.

	Fig. 2(a)	Fig. 3(a)	Fig. 3(c)
Lines	23	28	57
Circles	24	25	36
Strict Accuracy (%)	91.37	80.78	52.8
±1 Accuracy (%)	99.96	99.73	88.83

To show that the algorithm is robust in the presence of noise the original image shown in Fig. 2(a) was corrupted with additive Gaussian noise (mean = 0, variance = 100, 200) and additive salt and pepper noise (probability = 5, 10%) [18]. The first image shown in Fig. 3(a) was mildly corrupted with Gaussian noise (mean = 0, variance = 100) and with salt and pepper noise (probability = 5%) and the second image was heavily corrupted with Gaussian noise (mean = 0, variance = 200) and with salt and pepper noise (probability = 10%). The resulting images, extracted with the same parameters as was used in the noiseless case, are shown in Figs. 3(b) and (c), respectively. The strict accuracy and the ±1 accuracy were calculated by using the true edge pixels shown in Fig. 2(b). From these images and the results shown in Table 1, columns 2 and 3, it can be seen that the proposed algorithm retains high accuracy with low levels of noise and is still robust in extracting lines and circular arcs from highly corrupted images.

3.2 Speed Experiment

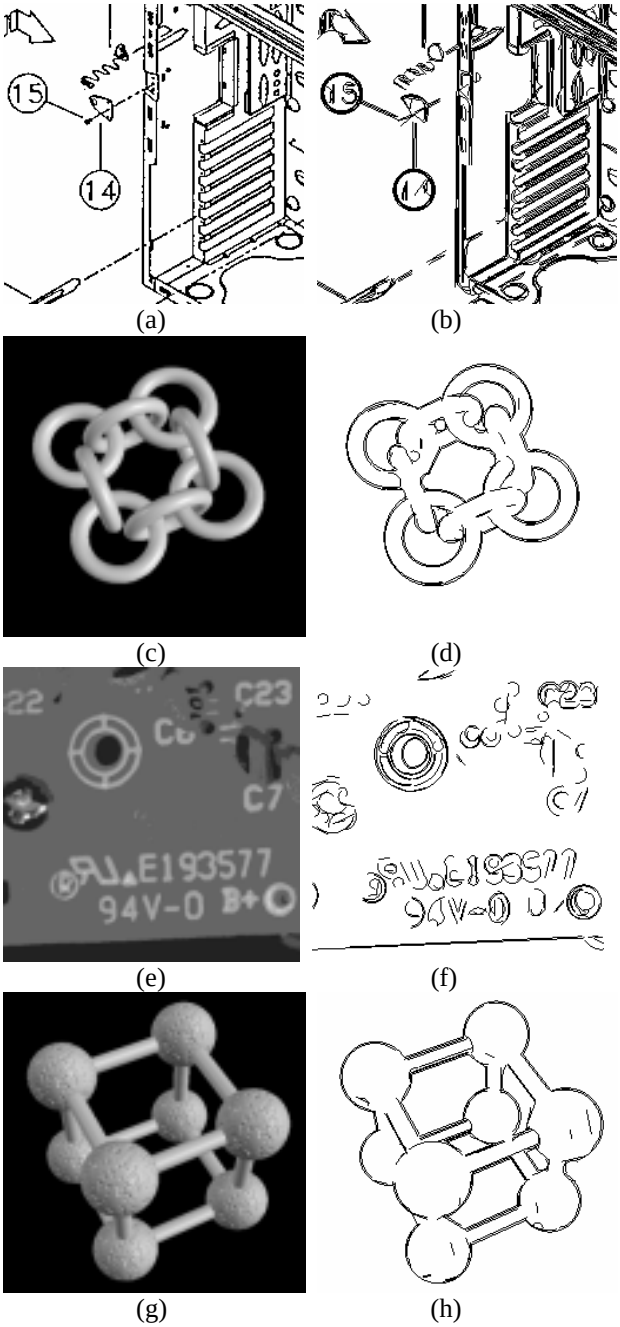


Figure 4: *Computational speed experiment. (a) blueprint; (b) results; (c) chain image; (d) results; (e) circuit board; (f) results; (g) structure image; (h) results.*

To demonstrate the speed of the proposed algorithm several real-world and synthetic images were tested. These images are shown in Fig. 4 and consist of a blueprint, a chain, a circuit board, and a structure. The parameters of

the algorithm were the same as those used in the previous experiment. All experiments were performed on a 1.4 GHz Athlon processor, and all test images were 256 × 256 grayscale images. The computation times for the images shown in Figs. 2, 3, and 4 are recorded in Table 2, column 3. From Table 2 it can be seen that the proposed algorithm has an average processing time of 0.0058 seconds for 256 × 256 grayscale images, thus verifying the speed of the algorithm.

Table 2: Computation Time.

	Lines	Circles	Time (s)
Fig. 2(a)	23	24	0.00358
Fig. 3(a)	28	25	0.00543
Fig. 3(c)	57	36	0.00796
Fig. 4(a)	413	185	0.00849
Fig. 4(c)	63	64	0.00470
Fig. 4(e)	91	146	0.00543
Fig. 4(g)	96	43	0.00471
Average	110	75	0.0058

3.3 Standard Hough Transform Comparison

In order to compare the speed and accuracy of the proposed algorithm with the conventional approaches, the standard Hough transform for lines was modified for circles (SHTC). Thus the two dimensional histogram used for lines was expanded to a three dimensional histogram. The three dimensions are the *x* and *y* coordinates of the origin and range from 0 to the size of the input image, and the radius which can range from 3 pixels to twice the size of the image. To convert the input image into the edge image needed for the SHTC, the Sobel operator was used, but this computation cost is not recorded in the results. The same experiments presented in Sections 3.1 and 3.2 were repeated using the SHTC and the results are shown in Fig. 5 and are summarized in Table 3. In Table 3, Columns 2 and 3, there are negative accuracies. This results from having more false circles than true circles, as can be seen in Fig. 5(d). As can be seen from the results the SHTC is only designed to extract the circles and the standard Hough transform for lines would have to be used on the unclassified edge pixels to extract the lines thus increasing the extraction time.

Table 3: SHTC Experimental Results.

	Strict Acc. (%)	±1 Acc.(%)	Time (s)
Fig. 2(a)	41.55	87.37	57.42
Fig. 3(a)	21.60	57.54	2980.48
Fig. 3(c)	-79.77	-42.10	4987.21
Fig. 4(a)	N.A.	N.A.	413.62
Fig. 4(c)	N.A.	N.A.	136.63
Fig. 4(e)	N.A.	N.A.	225.77
Fig. 4(g)	N.A.	N.A.	202.21
Average	-5.54	34.27	1286.19

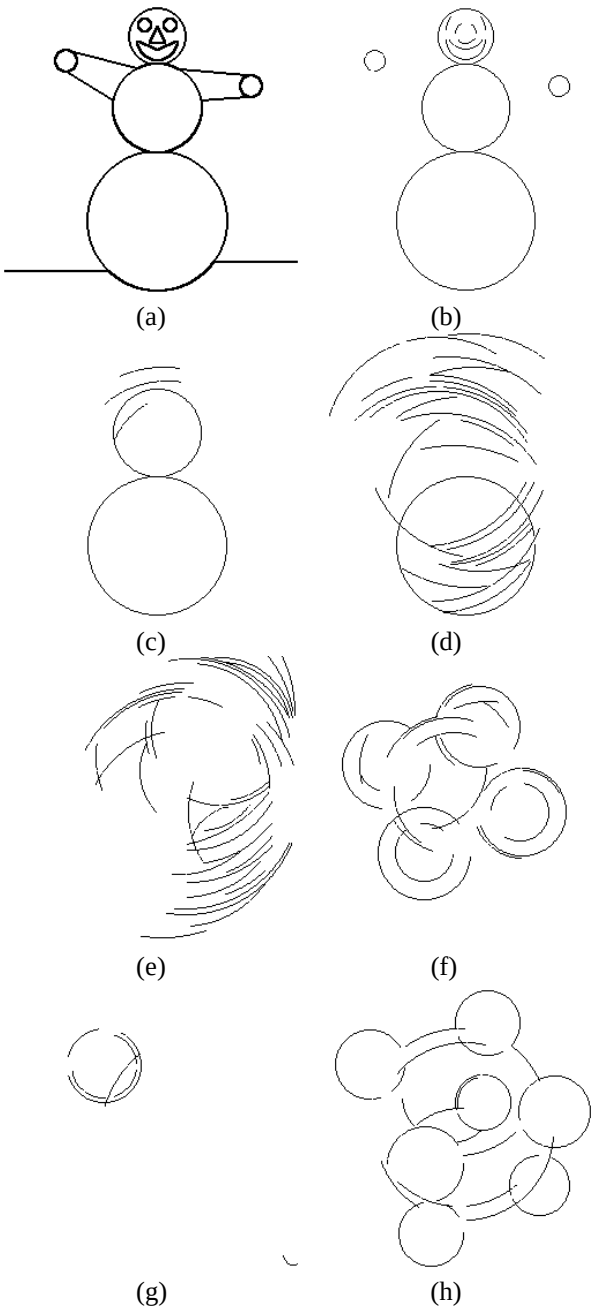


Figure 5: *SHTC experiments. (a) results of the Sobel operator for Fig. 2(a); (b) extraction results for Fig. 2(a); (c) extraction results for Fig. 3(a); (d) extraction results for Fig. 3(c); (e) extraction results for Fig. 4(a); (f) extraction results for Fig. 4(c); (g) extraction results for Fig. 4(e); (h) extraction results for Fig. 4(g).*

If the results shown in Table 3 are compared with those shown in Tables 1 and 2, it can be seen that the proposed algorithm is more that 200,000 times faster than the SHTC while realizing a nine fold reduction in error.

3.4 Modified SHTC Comparison

The SHTC is one of the most accurate of the traditional approaches to circle extraction, however it is also one of the slowest approaches due to its three dimensional histogram. In order to improve the speed of circle extraction several modified approaches have been proposed that reduce the size of the histogram to two dimensions by exploiting the geometrical properties of circles. In order to compare the speed of the proposed algorithm with these modified SHTC algorithms, the voting time required for two of these modified methods was used. These two methods are the Davies method [19] and the Tsuji method [20]. The time required for voting only was calculated for the same 256 × 256 grayscale images and the same 1.4 GHz. Athlon processor was used as in the previous experiments. Only the voting time was calculated and not the entire circle extraction process, therefore no accuracy calculations are possible. The voting computation time for both of these methods is recorded in Table 4.

Table 4: Modified SHTC Voting Time

	Davies Method Voting Time (s)	Tsuji Method Voting Time (s)
Fig. 2(a)	0.19	172.42
Fig. 3(a)	12.52	1509.07
Fig. 3(c)	24.17	2626.45
Fig. 4(a)	18.53	1682.45
Fig. 4(c)	2.53	502.81
Fig. 4(e)	2.84	859.45
Fig. 4(g)	5.34	674.11
Average	9.44	1146.68

By contrasting the results shown in Table 4 with those shown in Table 2 it can be seen that the proposed algorithm is more than 1600 times faster than the voting time of the Davies method and 197000 times faster than the voting time of the Tsuji method. These results demonstrate the efficiency of the proposed method.

4 Conclusion

This paper has presented an algorithm that can extract lines and circles simultaneously from the original intensity image. Experiments have shown that the algorithm is robust in the presence of noise and can extract lines and circles with an average ±1 accuracy of more than 90%, even from heavily corrupted images. The algorithm is very fast with an average processing time of 0.0058 seconds using a 1.4 GHz Athlon processor and 256 × 256 grayscale images.

Acknowledgements

This work was made possible by support from the Japan Society for the Promotion of Science.

References

[1] R.C. Gonzalez and P. Wintz, *Digital Image Processing, second ed*. Addison-Wesley, Reading Massachusetts, 1987.

[2] D.H. Ballard and C.M. Brown, *Computer Vision*. Prentice-Hall, Englewood Cliffs, New Jersey, 1982.

[3] D.H. Ballard, "Generalizing the Hough Transform to Detect Arbitrary Shapes," *Pattern Recognition*, vol. 13, no. 2, pp. 111-122, Mar. 1981.

[4] H. Li, M.A. Lavin and R.J. LeMaster, "Fast Hough Transform: A Hierarchical Approach," *Computer Vision, Graphics, and Image Processing*, vol. 36, no. 2, pp. 139-161, Nov. 1986.

[5] O. Chutatape and L. Guo, "A Modified Hough Transform for Line Detection and its Performance," *Pattern Recognition*, vol. 32, no. 2, pp. 181-182, Feb. 1999.

[6] Y. Zhang and R. Webber, "A Windowing Approach to Detecting Line Segments Using Hough Transform," *Pattern Recognition*, vol. 29, no. 2, pp. 255-266, Feb. 1996.

[7] L. Xu and E. Oja, "Randomized Hough Transform (RHT) – Basic Mechanisms, Algorithms, and Computational Complexities," *CVGIP – Image Understanding*, vol. 57, no. 2, pp. 131-154, Mar. 1993.

[8] J. Yuan and C.Y. Suen, "An Optimal O(n) Algorithm for Identifying Line Segments from a Sequence of Chain Codes," *Pattern Recognition*, vol. 28, no. 5, pp. 635-646, May 1995.

[9] A. Pikaz and I. Dinstein, "Optimal Polygonal Approximation of Digital Curves," *Pattern Recognition*, vol. 28, no. 3, pp. 373-379, Mar. 1995.

[10] J.W. Lee and I.S. Kweon, "Extraction of Line Features in a Noisy Image," *Pattern Recognition*, vol. 30, no. 10, pp. 1651-1660, Oct. 1997.

[11] J.W. Gates, M. Haseyama and H. Kitajima, "A Real-time Line Extraction Method," *IEEE International Symposium on Circuits and Systems'99*, vol. IV, pp. 68-71, May 1999.

[12] N. Guil and E.L. Zapata, "Lower Order Circle and Ellipse Hough Transform," *Pattern Recognition*, vol. 30, no. 10, pp. 1729-1744, 1997.

[13] R.S. Conker, "A Dual Plane Variation of the Hough Transform for Detecting Non-concentric Circles of Different Radii," *Computer Vision*

Graphics and Image Processing, vol. 43, pp. 115-132, 1988.

[14] W.C.Y. Lam and S.Y. Yuen, "Efficient Technique for Circle Detection Using Hypothesis Filtering and Hough Transform," *IEE Proceedings. Vision, Image, and Signal Processing*, vol. 143, no. 5, pp. 292-300, 1996.

[15] R. Chan and W.C. Siu, "New Parallel Hough Transform for Circles," *IEE Proceedings. Vision, Image, and Signal Processing*, vol. 138, no. 5, pp. 335-344, 1991.

[16] E. Kim, M. Haseyama and H. Kitajima, "A New Fast and Robust Circle Extraction Algorithm," *International Conference on Vision Interface'02*, pp. 439-446, May 2002.

[17] E.R. Dougherty, *Probability and Statistics for the Engineering, Computing, and Physical Sciences*. Prentice Hall, Englewood Cliffs, New Jersey, 1990.

[18] H.R. Myler and A.R. Weeks, *The Pocket Handbook of Image Processing Algorithms in C*, Prentice Hall, Englewood Cliffs, New Jersey, 1993.

[19] E.R. Davies, "A Modified Hough Scheme for General Circle Location," *Pattern Recognition Letters*, vol. 7, no. 1, pp. 37-47, 1988.

[20] S. Tsuji and F. Matsumoto, "Detection of Ellipses by a Modified Hough Transformation," IEEE Transactions on Computers, vol. C-27, no. 8, pp. 777-781, 1979.