EDCIRCLES: REAL-TIME CIRCLE DETECTION BY EDGE DRAWING (ED)

Cuneyt Akinlar *Cihan Topal*

Computer Engineering Department, Anadolu University, Eskisehir, TURKEY
{cakinlar, cihant}@anadolu.edu.tr

ABSTRACT

We propose a real-time, parameter-free circle detection algorithm that produces accurate results with very few false positives. The algorithm makes use of the contiguous (connected) set of edge segments produced by our novel edge segment detector, the Edge Drawing (ED) algorithm; hence the name EDCircles. The proposed algorithm first fits line segments to ED's edge segments and then processes these line segments to detect circles in a given image. We show through experimentation that EDCircles works real-time, produces good results, and is very suitable for the next generation real-time automation applications including automatic inspection of manufactured products, human eye iris and pupil detection, circular traffic sign detection, etc.

Index Terms— Real-Time Circle Detection, Edge Drawing Algorithm, EDLines, Eye Pupil Detection, Circular Traffic Sign Detection

1. INTRODUCTION

Detection of circular objects in digital images is an important and recurring problem in image processing and computer vision, and has many applications especially in such automation problems as automatic inspection of manufactured products, pupil and iris detection [1], circular traffic sign detection [2], and many others.

Traditionally, the most popular circle detection techniques are based on the famous Circle Hough Transform (HT) [4-6]. These techniques first compute an edge map of the image using an edge detector, e.g., Canny [3], map the edge pixels into the three dimensional Hough Circle space and extract circles that contain a certain number of edge pixels. Not only HT-based techniques are very slow and space-demanding, but they also produce many false detection. Additionally, these methods have many parameters that must be preset by the user, which limit their use.

To overcome the limitations of the classical HT methods, many variants have been proposed including probabilistic HT [7], randomized HT [8], fuzzy HT [9] etc. All these methods try to correct different shortcomings of HT, but each have shortcomings of their own, and are still slow to be of any use in real-time applications.

Recently, some genetic algorithm-based circle detection methods have been proposed [10, 11]. These techniques have mixed results and high running times, and therefore, are not suitable for real-time automation applications.

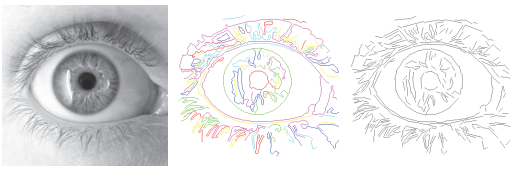
Frosio et al. [12] propose a real-time circle detection algorithm based on maximum likelihood. Their method is fast and can detect partially occluded circular objects, but it requires that the radius of the circles to be detected be predefined, which greatly limits its applications. Wu et al. [13] present a circle detection algorithm that runs 7 frames/sec on 640x480 images. The authors claim to achieve high success rate, but there is not much experimental validation to back their claims.

In this paper, we present a real-time, parameter-free circle detection algorithm that produces accurate results with very few false positives. Our algorithm makes use of the contiguous (connected) set of edge segments produced by our novel edge segment detector, the Edge Drawing (ED) algorithm [14-16]; hence the name EDCircles. Our algorithm first fits line segments to ED's edge segments [17-19], then processes the line segments to extract circular arcs, and finally, combines these arcs to detect complete circles.

The rest of the paper is organized as follows: Section 2 gives an overview of ED and line extraction. Section 3 describes the details of the EDCircles algorithm. Section 4 presents experimental results and section 5 concludes the paper.

2. EDGE DETECTION BY EDGE DRAWING (ED) AND LINE SEGMENT DETECTION BY EDLINES

Edge Drawing (ED) [14-16] is our recently-proposed, real-time edge/edge segment detector. Unlike traditional edge detectors, e.g., Canny [3], which work by identifying a set of potential edge pixels in an image and eliminating non-edge pixels through operations such as non-maximal suppression, hysteresis thresholding, erosion etc., ED follows a proactive approach and works by first identifying a set of gradient extrema points in the image, called the *anchors*, and then links these anchors using a smart routing procedure; that is, ED literally draws edges in an image. ED outputs not only a binary edge map similar to those output by traditional edge detectors, but it also outputs the result as a set of edge segments each of which is a contiguous (connected) pixel chain [23-25].



(a) (b) (c)

Figure 1. (a) An eye image (350x343), (b) Edge segments extracted by ED. Each color represents a different edge segment. ED outputs 107 edge segments in 8.22 ms. (c) Line segments extracted from the edge segments.

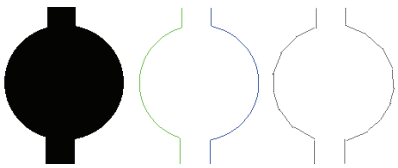
Fig. 1 shows a 350x343 grayscale eye image and the edge segments output by ED. Each color in the edge map represents a different edge segment. For this image, ED outputs 107 edge segments in 8.22 milliseconds. Notice the high quality nature of the edge map with all details clearly visible.

Each edge segment traces the boundary of one or more objects in the figure. While the boundary of an object may be traced by a single edge segment, as the pupil and iris segments are in Fig. 1(b), it is also possible that an object's boundary be traced by many different edge segments. The result totally depends on the structure of the objects, and the amount of obstruction and noise in the image.

Given the set of edge segments output by ED, it is easy to fit lines and extract line segments in the image. In [17-19] we present a real-time, parameter-free line segment detector called EDLines, which takes ED's edge segments as input and extracts line segments by simply fitting a line to a set of contiguous pixels satisfying a straightness criterion. The line segments are then validated using the Helmholtz principle [20], which guarantees that only a few false detections are made. Fig. 1(c) shows the line segments extracted from ED's edge segments by EDLines. Notice that all circular boundaries, e.g., pupil and iris, are approximated by a contiguous set of line segments. In the next section we show how circle detection can be made by a post-processing of the extracted line segments.

3. EDCIRCLES: A REAL-TIME CIRCLE DETECTOR

To detect the circles in a given image, our idea is to process the line segments detected by EDLines. Since the boundary of a circular object is approximated by a chain of short line segments (refer to Fig. 1), our idea is to look at consecutive line segments over the same edge segment, and see if they form a circular arc or a complete circle.

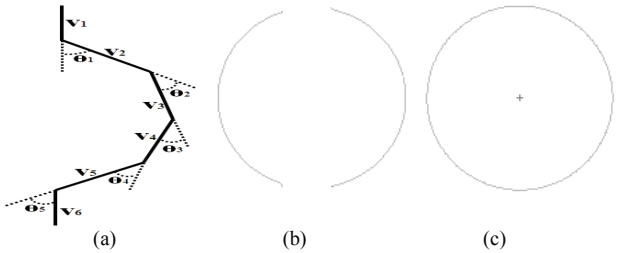


(a) (b) (c)

Figure 2. (a) A sample circle obstructed by a vertical rectangle, (b) Two edge segments extracted by ED, (c) Line segments extracted from the edge segments.

Fig. 2 shows an example circular object obstructed by a vertical rectangle, where the circle boundary is traced by two edge segments: One edge segment (green) starts from the top left corner and moves all the way down, and the other edge segment (blue) starts from the top right corner and moves all the way down. Fig. 2(c) shows the line segments approximating the edge segments.

To detect a circular arc formed by a consecutive set of line segments on the same edge segment, we simply walk over the edge segments and compute the angle between each consecutive line segment. If the angle is between certain thresholds, then the two line segments may be part of a circular arc. If not, we skip the first line segment, and continue with the next pair.



(a) (b) (c)

Figure 3. (a) An illustration of the right vertical segment in Fig. 2 being approximated by 6 consecutive lines v1 through v6, and the angle between consecutive lines, (b) Two circular arcs detected from the two edge segments in Fig. 2, (c) One circle detected by combining the two arcs shown in (b).

Fig. 3 shows an illustration of the right vertical segment in Fig. 2 being approximated by 6 consecutive lines, v1 through v6, and the angle between consecutive lines. Although the real segment is approximated by 10 lines as detailed in Table 1, we assume 6 lines for the purposes of illustration. To process these line segments, we take consecutive line segments and compute the angle between two lines depicted as Θ_1 through Θ_5 in the figure. This can easily be done by considering each line to be a vector and taking the dot product of the two vectors: $v1.v2 = |v1||v2|.\cos\Theta$. We also compute the direction of the turn going from the first line to the next, which is computed by taking the cross product of the two vectors.

Table1: Lines making up the right vertical segment in Fig. 2

Line	Length	Angle	Turn Dir.
1	29	67	-
2	25	17	+
3	14	17	+
4	20	18	+
5	19	18	+
6	22	22	+
7	24	23	+
8	21	21	+
9	22	68	+
10	24	-	-

Table 1 shows the 10 lines making up the right vertical segment in Fig. 2, their lengths, the angle between the two

consecutive line segments and the direction of the turn going from one line to the next. After computing this table, we simply go over the line segments, and try to find a set of consecutive line segments which may form an arc. Obviously, the line segments making up an arc must have the same turn direction (all – or all +) and the angle between consecutive lines must be in-between certain thresholds. If the angle is too small, we assume that the line segments are collinear, so they cannot be part of an arc; if the angle is too big, we assume that the line segments are part of a strictly turning object such as a square, a rectangle, etc. For the purposes of our current implementation, we fix the low angle threshold to 6 degrees, and the high angle threshold to 60 degrees. These values have been obtained by experimentation on a variety of images containing various circular objects.

Processing the line segments in Table 1, our arc detection algorithm finds the line segments 2-9 to be a potential arc. We then fit a circle to these line segment pixels using the least square circle fit algorithm [21]. If the mean square error is less than a certain threshold (2 pixels), then we take these pixels to be an arc. Using this algorithm, we detect the two arcs shown in Fig. 3(b), one for each segment.

The final step of the algorithm is to go over the detected arcs and try to extract circles by combining the arcs. To do this, we collect arcs having similar radius and center, and combine them by fitting a new circle. If the final mean square error is less than the error threshold (2 pixels), and the arc pixels make up at least half, i.e., 50%, of the circle's circumference, then the arcs are combined into a circle. Using this algorithm, we combine the two arcs in Fig. 3(b) and obtain the circle in Fig. 3(c).

4. EXPERIMENTAL RESULTS

To measure the performance of EDCircles [22], we take both synthetic and natural images containing circular objects and feed them into our algorithm. We then show the detected circles and the running time. To compare the performance of EDCircles to a fast circle detection algorithm from the literature, we choose OpenCV's circle detection function, `cvHoughCircles`. The reason for choosing OpenCV is due to its comparable speed to EDCircles and because it is open source so that anyone can repeat the same results. We note that `cvHoughCircles` has many parameters (similar to many other circle detection algorithms in the literature) that must be pre-supplied by the user. To obtain the results by `cvHoughCircles`, we tried many different parameters and present the best results. With a single set of default parameters `cvHoughCircles` either fails to detect many valid circles or produces many false positives. Note, however, that EDCircles is parameter-free in the sense that it has one set of internal parameters used for all images presented in this paper and on the EDCircles demo Web site [22].

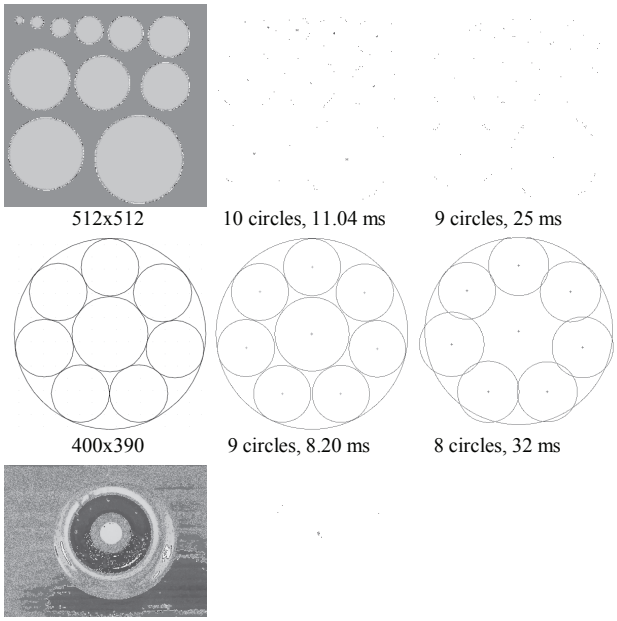


Figure 4: Cirle detection results by EDCircles (middle), OpenCV `cvHoughCircles` (right).

Fig. 4 shows the performance of EDCircles and `cvHoughCircles` on three synthetic and natural images. The running times were measured in a PC with a 2.2 GHz E4500 CPU and 2GB RAM. It is clear from the figure that EDCircles detects all valid circles (except very small ones) in real-time, whereas `cvHoughCircles` fails in detecting many valid circles.

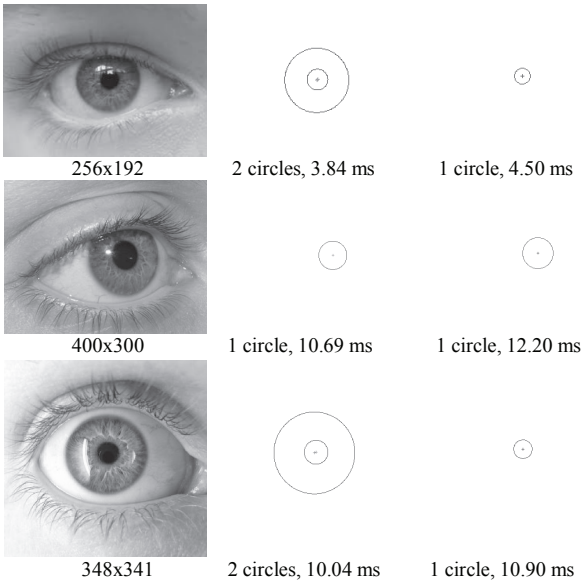


Figure 5: Eye pupil or iris detection results by EDCircles (middle), OpenCV `cvHoughCircles` (right)

Fig. 5 shows the performance of EDCircles and `cvHoughCircles` on three eye images. Detection of the eye pupil and iris is a very important problem for eye tracking automation applications and must be performed in

real-time [1]. The results in Fig. 5 show that EDCircles is able to detect the pupil or the iris successfully in all images. `cvHoughCircles` also detects the pupil in all images, but the parameters must be chosen carefully as we did here, for otherwise it produces many false detections.

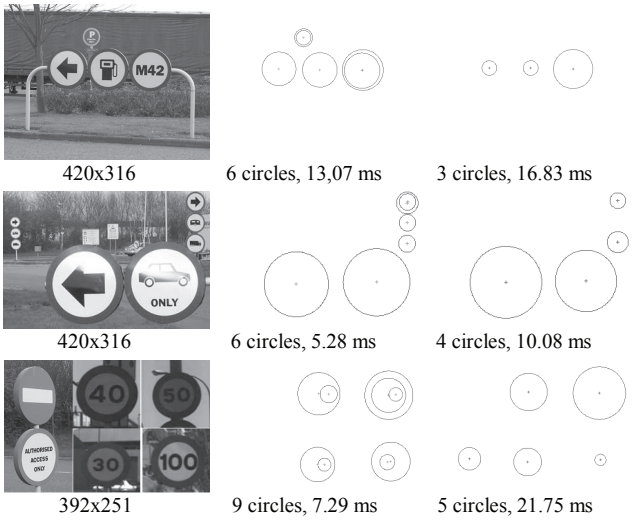


Figure 6: Circular traffic sign detection results by EDCircles (middle), OpenCV `cvHoughCircles` (right)

Fig. 6 shows the performance of EDCircles and OpenCV `cvHoughCircles` on circular traffic sign detection problem, which is another important automation problem requiring real-time performance [2]. It is clear from the results that EDCircles is able to detect almost all circular traffic signs except very small ones (having radius less than 10 pixels) or those which appear as an ellipse rather than a circle in the image. Clearly, depending on the camera position, a circle appears as an ellipse on the image plane, which the current version of EDCircles cannot handle. Our goal is to extend EDCircles to detect such elliptic objects in the next version of the code.

As for the running time performance of EDCircles, it is clear from the results presented in Figs. 4, 5 and 6 that EDCircles runs in blazing real-time speed and is up to 4 times faster than OpenCV's `cvHoughCircles`.

5. CONCLUSIONS

EDCircles is real-time, parameter-free circle detection algorithm that works by analyzing the line segments extracted by our real-time line segment detector EDLines. Experiments shown in this paper (and others that the reader may wish to perform online at http://ceng.anadolu.edu.tr/CV/EDCircles/Demo.aspx [22]) show that EDCircles produces good results with very few false positives, and it achieves this in blazing speed. The problem with the current version of EDCircles is that it fails in detection of small-sized circles (having radius less than 10 pixels) and those which look elliptical rather than circular. As future work, our goal is to extend EDCircles to handle these more difficult cases.

6. ACKNOWLEDGEMENTS

This work is partially supported by The Scientific and Technological Research Council of Turkey (TUBITAK) grant 111E053.

7. REFERENCES

[1] Y. Ebisawa, "Robust pupil detection by image difference with positional compensation", In Proc. of IEEE VECIMS, pp.143-148, 2009.

[2] A. Arlicot, B. Soheilian, and N. Paparoditis, "Circular Road Sign Extraction from Street Level Images Using Color, Shape and Texture Database Maps," Proc. IAPRS, vol. XXXVIII, pp. 205-210, 2009.

[3] J. Canny, "A computational approach to edge detection," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 8, no. 6, pp. 679-698, 1986.

[4] P.V.C. Hough, "Methods and means for recognizing complex patterns," US Patent, 3069654, 1962.

[5] J. Illingsworth, and J. Kittler, "A survey of the Hough transform," Computational Vision, Graphics, and Image Processing, vol. 44, pp. 87-116, 1988.

[6] E.R. Davies, "A Modified Hough Scheme for General Circle Location," Pattern Recognition Letters, vol. 7, pp. 37-43, 1988.

[7] D. Shaked, O. Yaron, and N. Kiryati, "Deriving stopping rules for the probabilistic Hough Transform by sequential analysis," Computer Vision and Image Understanding, vol. 63, pp. 512-526, 1996.

[8] K. Chung, and Y. Huang, "Speed up the computation of randomized algorithms for detecting lines, circles, and ellipses using novel tuning and LUT-based voting platform," Applied Mathematics and Computation, vol. 190, no. 1, pp. 132-149, 2007.

[9] J.H. Han, L.T. Koczy, and T. Poston, "Fuzzy Hough Transform," Proc. Int. Conf. on Fuzzy Systems, vol.2,pp. 803-808, 1993.

[10] J. Yao, "Fast Robust Genetic-algorithm based Ellipse Detection," Proc. ICPR, pp. 859-862, 2004.

[11] V. Ayala-Ramirez, C.H. Garcia-Capulin, A. Peres-Garcia, and R.E. Sanchez-Yanez, "Circle detection on images using genetic algorithms," Pattern Recognition, vol. 27, no. 6, pp. 652-657, 2005.

[12] I. Frosio, N.A. Borghese, "Real Time Accurate Circle Fitting," Pattern Recognition, vol. 14, no. 3, pp. 1041-1055, 2008.

[13] J. Wu, J. Li, C. Xiao, F. Tan, and C. Gu, "Real-time Robust Algorithm for Circle Object Detection," Proc. IEEE Conf. for Young Computer Scientists, pp. 1722-1727, 2008.

[14] C. Topal, C. Akinlar, and Y. Genc, "Edge Drawing: A Heuristic Approach to Robust Real-Time Edge Detection," The twentieth Int'l Conf. on Pattern Recognition (ICPR), Istanbul, Turkey, August 23-26 2010.

[15] C. Topal, O. Ozsen, and C. Akinlar, "Real-time Edge Segment Detection with Edge Drawing Algorithm," Proc. ISPA, Crotia, 2011.

[16] Edge Drawing Web Site. http://ceng.anadolu.edu.tr/CV/EdgeDrawing

[17] C. Akinlar, and C. Topal, "EDLines: Realtime Line Segment Detection by Edge Drawing (ED)," IEEE Int'l Conf. on Image Processing (ICIP), Brussels, Belgium, Sep. 2011.

[18] C. Akinlar, and C. Topal, "EDLines: A Real-time Line Segment Detector with a False Detection Control," Pattern Recognition Letters, vol. 32, no. 13, pp. 1633-1642, Oct. 2011.

[19] EDLines Web Site. http://ceng.anadolu.edu.tr/CV/EDLines

[20] A. Desolneux, L. Moisan, and J.M. Morel, From Gestalt Theory to Image Analysis: A Prob. Approach, Springer, 2008.

[21] R. Bullock, "Least Squares Circle Fit," www.dtcenter.org/met/users/docs/write_ups/circle_fit.pdf, 2006.

[22] EDCircles Web Site. http://ceng.anadolu.edu.tr/CV/EDCircles