Pattern Recognition 46 (2013) 725–740

Contents lists available at SciVerse ScienceDirect

Pattern Recognition

journal homepage: www.elsevier.com/locate/pr

EDCircles: A real-time circle detector with a false detection control

Cuneyt Akinlar*, Cihan Topal

Department of Computer Engineering, Anadolu University, Eskisehir 26470, Turkey

ARTICLE INFO

Article history:
Received 9 April 2012
Received in revised form
21 September 2012
Accepted 26 September 2012
Available online 3 October 2012

Keywords:
Circle detection
Ellipse detection
Real-time image processing
Helmholtz Principle
NFA

ABSTRACT

We propose a real-time, parameter-free circle detection algorithm that has high detection rates, produces accurate results and controls the number of false circle detections. The algorithm makes use of the contiguous (connected) set of edge segments produced by our parameter-free edge segment detector, the Edge Drawing Parameter Free (EDPF) algorithm; hence the name EDCircles. The proposed algorithm first computes the edge segments in a given image using EDPF, which are then converted into line segments. The detected line segments are converted into circular arcs, which are joined together using two heuristic algorithms to detect candidate circles and near-circular ellipses. The candidates are finally validated by an *a contrario* validation step due to the Helmholtz principle, which eliminates false detections leaving only valid circles and near-circular ellipses. We show through experimentation that EDCircles works real-time (10–20 ms for 640 × 480 images), has high detection rates, produces accurate results, and is very suitable for the next generation real-time vision applications including automatic inspection of manufactured products, eye pupil detection, circular traffic sign detection, etc.

1. Introduction

Detection of circular objects in digital images is an important and recurring problem in image processing [1] and computer vision [2], and has many applications especially in such automation problems as automatic inspection of manufactured products [3], aided vectorization of line drawing images [4,5], pupil and iris detection [6–8], circular traffic sign detection [9–11], and many others.

An ideal circle detection algorithm should run with a fixed set of internal parameters for all images, i.e., require no parameter tuning for different images, be very fast (real-time if possible), detect multiple small and large circles, work with synthetic, natural and noisy images, have high detection rate and good accuracy, and produce a few or no false detections. The circle detection algorithm presented in this paper satisfies all of these properties.

Traditionally, the most popular circle detection techniques are based on the famous circle Hough transform (CHT) [12–16]. These techniques first compute an edge map of the image using a traditional edge detector such as Canny [17], map the edge pixels into the three dimensional Hough circle space (x, y, r) and extract circles that contain a certain number of edge pixels. Not only CHT-based techniques are very slow and memory-demanding, but they also produce many false detections especially in the presence of noise. Additionally, these methods have many parameters that must be preset by the user, which greatly limits their use.

To overcome the limitations of the classical CHT-based methods, many variants have been proposed including probabilistic HT [18,19], randomized HT [20,21], fuzzy HT [22], etc. There are also approaches based on HT and hypothesis filtering [23–25]. All these methods try to correct different shortcomings of CHT, but are still memory-demanding and slow to be of any use in real-time applications.

Apart from the CHT-based methods, there are several randomized algorithms for circle detection. Chen et al. [26] propose a randomized circle detection (RCD) algorithm that randomly selects four pixels from the edge map of an image, uses a distance criterion to determine whether there is a possible circle in the image. They then use an evidence-collecting step to test if the candidate circle is a real-circle. RCD produces good results, but is slow. Recently, Chung et al. [27,28] have proposed efficient sampling and refinement strategies to speed up RCD and increase the accuracy of RCD's results. Although the new RCD variants named GRCD-R, GLRCD-R [28] have good detection rates and produce accurate results, they still are far from being real-time. Furthermore, all RCD-variants work on the edge map of an image computed by a traditional edge detector such as the Sobel filter or the Canny edge detector, which have many parameters that must be set by the user.

Recently, many efforts have concentrated on using genetic algorithms and evolutionary computation techniques in circle detection [29–36]. Ayala-Ramirez et al. [30] proposed a genetic algorithm (GA) for circle detection, which is capable of detecting multiple circles but fails frequently to detect small or imperfect

* Corresponding author. Tel.: +90 2223213550x6553.
E-mail addresses: cakinlar@anadolu.edu.tr (C. Akinlar), cihant@andolu.edu.tr (C. Topal).

0031-3203/$ - see front matter © 2012 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.patcog.2012.09.020

circles. Dasgupta et al. [31–33] developed a swarm intelligence technique named adaptive bacterial foraging optimization (ABFO) for circle detection. Their algorithm produces good results but is sensitive to noise. Cuevas et al. use discrete differential evolution (DDE) optimization [34], harmony search optimization (HSA) [35] and an artificial immune system optimization technique named Clonal Selection Algorithm (CSA) [36] for circle detection. Although these evolutionary computation techniques have good detection rates and accurate results, they usually require multiple runs to detect multiple circles, and are quite slow to be suitable for real-time applications. Just like RCD, these algorithms work on an edge map pre-computed by a traditional edge detection algorithm with many parameters.

Frosio et al. [37] propose a real-time circle detection algorithm based on maximum likelihood. Their method is fast and can detect partially occluded circular objects, but requires that the radius of the circles to be detected be predefined, which greatly limits its applications. Wu et al. [41] present a circle detection algorithm that runs 7 frames/s on 640×480 images. The authors claim to achieve high success rate, but there is not much experimental validation to back their claims. Zhang et al. [38] propose an ellipse detection algorithm that can be used for real-time face detection. Liu et al. [39] present an ellipse detector for noisy images and Prasad et al. [40] present an ellipse detector using the edge curvature and convexity information. While both algorithms produce good results, they are slow and not suitable for real-time applications.

Vizireanu et al. [42–44] make use of mathematical morphology for shape decomposition of an image and use the morphological shape decomposition representation of the image for recognition of different shapes and patterns in the image. While their algorithms are good for the detection of general shapes in an image, they are not suitable for real-time applications.

Desolneux et al. [60] is the first to talk about the *a contrario* circular arc detection. Recently, Patraucean et al. [45,46] propose a parameter-free ellipse detection algorithm based on the *a contrario* framework of Desolneux et al. [58]. The authors extend the line segment detector (LSD) by Grompone von Gioi et al. [63] to detect circular and elliptic arcs in a given image without requiring any parameters, while controlling the number of false detections by the Helmholtz principle [58]. They then use the proposed algorithm (named ELSD [46]) for the detection and identification of Bubble Tags [47].

In this paper, we present a real-time (10–20 ms on 640×480 images), parameter-free circle detection algorithm that has high detection rates, produces accurate results, and has an *a contrario* validation step due to the Helmholtz principle that lets it control the number of false detections. The proposed algorithm makes use of the contiguous (connected) set of edge segments produced by our parameter-free edge segment detector, the edge drawing parameter free (EDPF) [48–53]; hence the name EDCircles [54,55]. Given an input image, EDCircles first computes the edge segments of the image using EDPF. Next, the resulting edge segments are turned into line segments using our line segment detector, EDLines [56,57]. Computed lines are then converted into arcs, which are combined together using two heuristic algorithms to generate many candidate circles and near-circular ellipses. Finally, the candidates are validated by the Helmholtz principle [58–63], which eliminates false detections leaving only valid circles and near-circular ellipses.

2. The proposed algorithm: EDCircles

EDCircles follows several steps to compute the circles in a given image. The general idea is to extract line segments in an image, convert them into circular arcs and then combine these arcs to detect circles and near-circular ellipses. General outline of EDCircles algorithm is presented in Algorithm 1 and we will describe each step of EDCircles in detail in the following sections.

Algorithm 1. Steps of EDCircles algorithm.

1. Detect edge segments by EDPF and extract complete circles and ellipses.
2. Convert the remaining edge segments into line segments.
3. Detect arcs by combining line segments.
4. Join arcs to detect circle candidates.
5. Join the remaining arcs to detect near-circular ellipse candidates.
6. Validate the candidate circles/ellipses using the Helmholtz principle.
7. Output the remaining valid circles/ellipses.

2.1. Edge segment detection by edge drawing parameter free (EDPF)

Given an image, the first step of EDCircles is the detection of the edge segments in the image. To achieve this, we employ our recently proposed, real-time edge/edge segment detector, edge drawing (ED) [48–51]. Unlike traditional edge detectors, e.g., Canny [17], 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 points in the image, called the anchors, and then joins 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 [49].

ED has many parameters that must be set by the user, which requires the tuning of ED's parameters for different types of images. Ideally, one would want to have a real-time edge/edge segment detector which runs with a fixed set of internal parameters for all types of images and requires no parameter tuning. To achieve this goal, we have recently incorporated ED with the *a contrario* edge validation mechanism due to the Helmholtz principle [58–60], and obtained a real-time parameter-free edge segment detector, which we name edge drawing parameter free (EDPF) [52,53]. EDPF works by running ED with all ED's parameters at their extremes, which detects all possible edge segments in a given image with many false positives. We then validate the extracted edge segments by the Helmholtz principle, which eliminates false detections leaving only perceptually meaningful edge segments with respect to the *a contrario* approach.

Fig. 1(a) shows a 424×436 grayscale synthetic image containing a big circle obstructed by four rectangular blocks, a small ellipse obstructed by three rectangular blocks, a small circle, an ellipse and an arbitrary polygon-like object. When this image is fed into EDPF, the edge segments shown in Fig. 1(b) are produced. Each color in the edge map represents a different edge segment, each of which is a contiguous chain of pixels. For this image, EDPF outputs 15 edge segments in just 3.7 ms in a PC with 2.2 GHz Intel 2670QM CPU. 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 small circle, the ellipse and the polygonal object are in Fig. 1(b), it is also possible that an object's boundary be traced by many different edge segments. This is the case for the big circle as the circle's boundary is traced by four different edge segments, and the small obstructed ellipse, which is traced by three different edge segments. The result totally depends on the structure of the objects, the amount of obstruction and noise in the image. That is, there is no

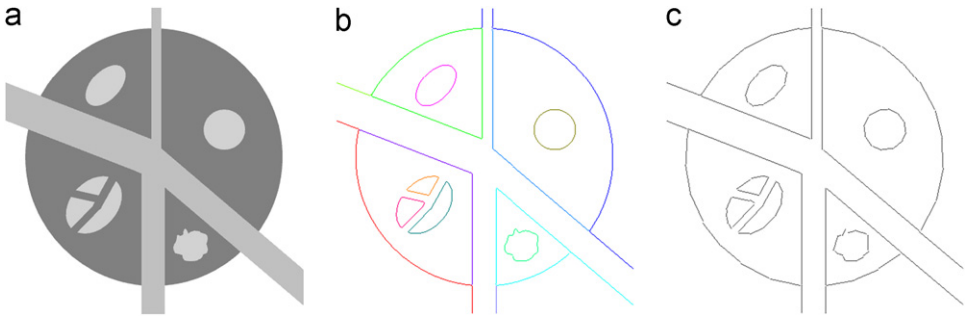


Fig. 1. (a) A sample image (424 × 436). (b) Edge segments (a contiguous chain of pixels) extracted by EDPF. Each color represents a different edge segment. EDPF outputs 15 edge segments in 3.7 milliseconds (ms). (c) Lines approximating the edge segments. A total of 98 lines are extracted. (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this article.)

way to tell beforehand how the edge segments will trace the boundaries of the objects in a given image.

Notice from Fig. 1 that in some cases, e.g., the small circle, the ellipse and the polygon, the entire boundary of an object in the image is returned as a closed curve; that is, the edge segment starts at a pixel on the boundary of an object, traces its entire boundary and ends at where it starts. In other words, the first and last pixels of the edge segment are neighbors of each other. It is highly likely that such a closed edge segment traces the boundary of a circle, an ellipse or a polygonal shape as is the case in Fig. 1. So as the first step after the detection of the edge segments, we go over all edge segments, take the closed ones and see if the closed edge segment traces the entire boundary of a circle or an ellipse.

Processing of a closed edge segment follows a very simple idea: We first fit a circle to the entire list of pixels in the edge segment using the least squares circle fit algorithm [64] and compute the root mean square error. If the circle fit error, i.e., the root mean square error, is smaller than some threshold (fixed at 1.5 pixels for the proposed algorithm), then we add the circle to the list of circle candidates. Just because the circle fit error is small does not mean that the edge segment is an actual circle; it is just a candidate yet and needs to go through circle validation by the Helmholtz principle to be returned as a real circle. Section 2.6 describes the details of circle validation.

If the circle fit fails, then we try fitting an ellipse to the pixels of the edge segment. We use the ellipse fit algorithm described in [65], which returns an ellipse equation of the form $Ax^2+Bxy+Cy^2+Dx+Ey+F=0$. If the ellipse fit error, i.e., the root mean square error, is smaller than a certain threshold (fixed at 1.5 pixels for the proposed algorithm), then we add the ellipse to the list of ellipse candidates, which also needs to go through validation by the Helmholtz principle before being returned as a real ellipse.

If the edge segment is accepted either as a circle or an ellipse candidate, it is removed from the list of edge segments and is not processed any further. Otherwise, the edge segment is used in further processing along with other non-closed edge segments.

2.2. Conversion of edge segments into line segments

After the removal of the closed edge segments, which are taken as circle or ellipse candidates, the remaining edge segments are converted into *line segments* (*lines* for short in the rest of the paper). The motivation for this step comes from the observation that any circular shape is approximated by a consecutive set of lines (as seen in Fig. 1(c)), and these lines can easily be turned into circular arcs by a simple post-processing step as described in the next section.

Conversion of an edge segment into a set of lines follows the algorithm given in our line detector, EDLines [56,57]. The idea is to start with a short line that satisfies a certain straightness criterion, and extend the line for as long as the root mean square error is smaller than a certain threshold, i.e., 1 pixel error. Refer to EDLines [56,57] for the details of line segment extraction, where we validate the lines after detection using the Helmholtz principle to eliminate invalid detections. In EDCircles, though, we do not validate the lines after detection. The reason for this decision comes from our observation that the line segment validation algorithm due to the Helmholtz principle usually eliminates many short lines, which may be valuable for the detection of small circles in an image. So, unlike EDLines, we do not eliminate any detected lines and use all detected lines for further processing and detection of arcs.

Fig. 1(c) shows the lines extracted from the image shown in Fig. 1(a). Clearly, circular objects are approximated by a set of consecutive lines. In the next section, we describe how these lines can be converted into circular arcs by processing of consecutive lines.

2.3. Circular arc detection

We define a circular arc to be a set of at least three consecutive lines that turn in the same direction. Using this definition, we detect a circular arc as follows: Given a list of lines making up an edge segment, simply walk over the lines and compute the angle between consecutive lines and the direction of turn from one line to the next. If at least three lines turn in the same direction and the angle between the lines is in-between certain thresholds, then these lines may form a circular arc.

Fig. 2 illustrates a hypothetical edge segment being approximated by 18 consecutive line segments, labeled l_1 through l_{18}. To compute the angle between two consecutive lines, we simply threat each line as a vector and compute the vector dot product. Similarly, to compute the turn of direction from one line to the next, we simply compute the vector cross product and use the sign of the result as the turn direction.

Fig. 3(a) illustrates the approximation of the blue right vertical edge segment in Fig. 1(b) by 11 consecutive line segments, labeled v1 through v11. Fig. 3(b) shows the details of the 11 lines: their lengths, the angle between consecutive lines and the direction of the turn going from one line to the next, where a '+' denotes a left turn, and '−' denotes a right turn.

Our arc detection algorithm is based on the following idea: For a set of lines to be a potential arc candidate, they all must have the same turn direction (to the left or to the right) and the angle between consecutive lines must be in-between certain thresholds. If the angle is too small, we assume that the lines are collinear so they cannot be part of an arc; if the angle is too big, we assume that the lines 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°, and the high angle threshold to 60°. These values have been obtained by

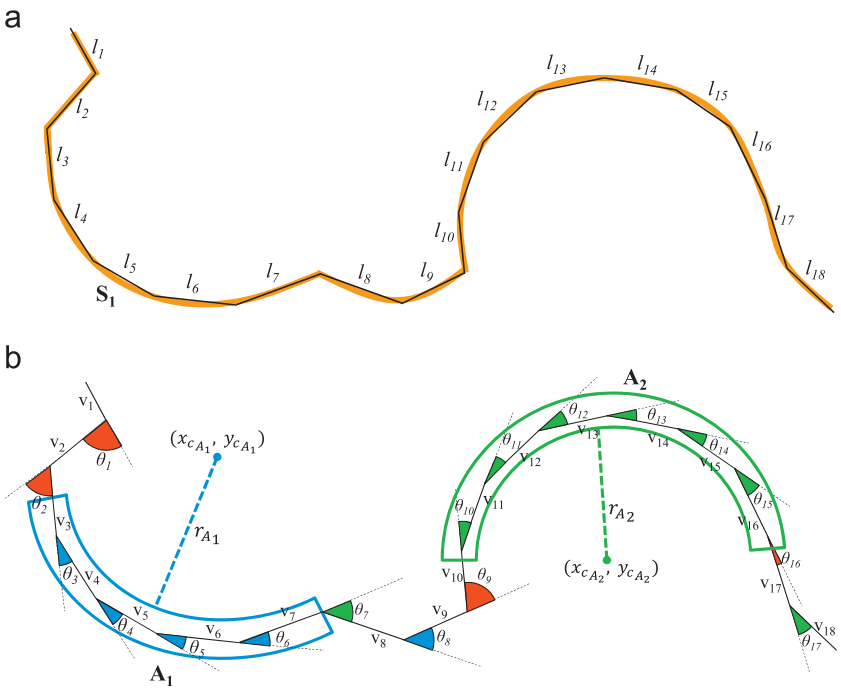
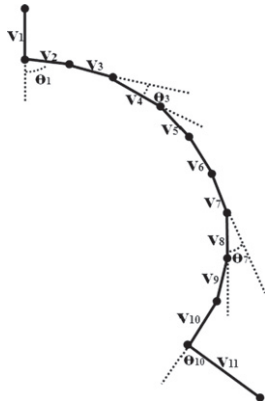


Fig. 2. (a) A hypothetical edge segment being approximated by 18 consecutive line segments labeled l_1 through l_{18}. (b) The angle θ_i between v_i and v_{i+1} are illustrated and colored with red, green or blue. If the angle is bigger than a high threshold, e.g., θ_1, θ_2 and θ_9 (colored red), or if the angle is smaller than a low threshold, e.g., θ_{16} (also colored red), then these lines cannot be part of an arc. Otherwise, if three or more consecutive lines turn to the left, e.g., lines v_3 through v_7 (angles colored blue), then these lines may form an arc. Similarly, if three or more consecutive lines turn to the right, e.g., lines v_{10} through v_{16} (angles colored green), then these lines may form an arc. (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this article.)

a



b

Line	Length	Angle	Turn Dir.
1	26	77.6	+
2	45	15.1	-
3	40	14.0	-
4	31	12.6	-
5	26	14.2	-
6	40	12.4	-
7	38	10.8	-
8	43	13.8	-
9	47	14.2	-
10	41	78.8	+
11	63	n/a	n/a

Fig. 3. (a) An illustration of the blue right vertical segment in Fig. 1(b) being approximated by 11 consecutive line segments labeled v_1 through v_{11}. The angle between line segments v_1 and v_2 (θ_1), v_3 and v_4 (θ_3), v_7 and v_8 (θ_7), and v_{10} and v_{11} (θ_{10}) are also illustrated. (b) Lines making up the blue right vertical segment in Fig. 1(b).

experimentation on a variety of images containing various circular objects.

The bottom part of Fig. 2 depicts the angles between consecutive lines of the edge segment shown at the top of Fig. 2, and the turn of direction from one line to the next. The angles smaller than the low angle threshold or bigger than the high angle threshold, e.g., θ_1, θ_2, θ_9, and θ_{16}, have been colored red; all other angles have been colored either blue or green depending on the turn of direction. Specifically, if the next line turns to the left, the angle has been colored blue, and if the next line turns to the right, then the angle has been colored green.

Having computed the angles and the turn of direction information, we simply walk over the lines of an edge segment looking for a set of at least three consecutive lines which all turn in the same direction and the turn angle from one line to the next is in-between the low and high angle thresholds. In Fig. 2, lines v_3 through v_7 satisfy our criterion and is a potential arc candidate. Similarly, lines v_{10} through v_{16} make up for another arc candidate.

Given a set of at least three lines that satisfy our arc candidate constraints, we first try fitting a circle to all pixels making up the lines using the circle fit algorithm in [64]. If the circle fit succeeds, i.e., if the root mean square error is less than 1.5 pixels, then the extracted arc is simply added to the list of arcs, and we are done. Otherwise, we start with a short arc consisting of only of three lines and extend it line-by-line by fitting a new circle [64] until the root mean square error exceeds 1.5 pixels. At this point, the detected arc is added to the list of arcs, and we continue processing the rest of the lines to detect more circular arcs. Using this algorithm, we detect two arcs in Fig. 2: Lines v_3 through v_7 form arc A1 with center $(x_{c_{A1}}, y_{c_{A1}})$ and radius r_{A1}. Similarly, lines v_{10} through v_{16} form arc A_2 with center $(x_{c_{A2}}, y_{c_{A2}})$ and radius r_{A2}. In a complex image consisting of many edge segments, we will have hundreds of arcs.

Fig. 4 shows the arcs computed from the lines of Fig. 1(c), and Table 1 gives the details of these arcs. An arc spans a part between (StartAngle, EndAngle) of the great circle specified by (Center X, Center Y, Radius). The arc is assumed to move counter-clockwise from StartAngle to EndAngle over the great circle. As an example, A_2 covers a total of 61° from 91° to 152° of the great circle with center coordinates (210.6, 211.3) and radius = 182.6.

2.4. Candidate circle detection by arc join

After the computation of the arcs, the next step is to join the arcs into circle candidates. To do this, we first sort all arcs with respect to their length in descending order, and start extending the longest arc first. The motivation for this decision comes from the observation that the longest arc is the closest to a full circle, so it must be extended and completed into a full circle before the

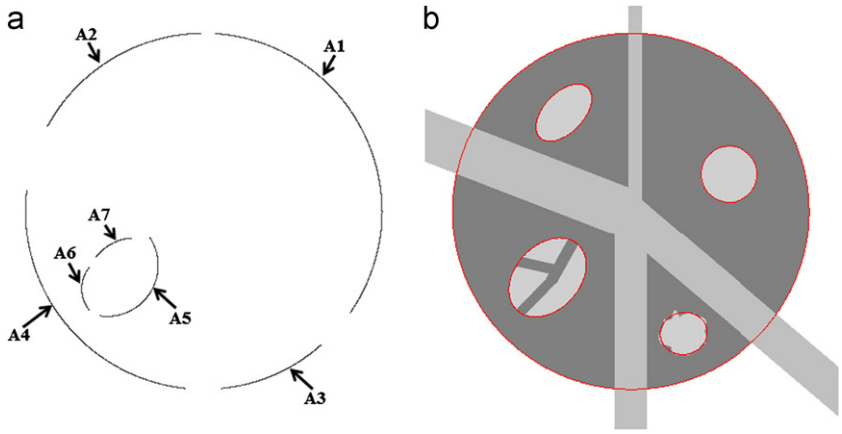


Fig. 4. (a) Arcs computed from the lines of Fig. 1(c). (b) Candidate circles and ellipses before validation (overlayed on top of the image with red color). (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this article.)

Table 1
Details of the arcs shown in Fig. 4(a). An arc spans a part between (StartAngle, EndAngle) of a great circle specified by (center X, center Y, Radius). The arc moves counter-clockwise from StartAngle to EndAngle over the circle.

Arc	Center X	Center Y	Radius	Start angle (deg.)	End angle (deg.)
A_1	210.3	211.8	182.2	325	86
A_2	210.6	211.3	182.6	91	152
A_3	212.2	215.9	178.5	275	312
A_4	210.7	211.6	183.0	173	264
A_5	111.1	267.5	52.3	275	312
A_6	120.1	291.4	34.9	141	219
A_7	139.4	288.6	49.2	94	143

other arcs would. During the extension of an arc, the idea is to look for arcs having similar radii and close centers, and collect a list of candidate arcs that may be combined with the current arc.

Given an arc A_1 to extend into a full circle, we go over all detected arcs and generate a set of candidate arcs that may be joined with A_1. We have two criterions for arc join: (1) *Radius difference constraint:* The radius difference between A_1 and the candidate arc A_2 must be within some threshold. Specifically, if A_2's radius is within 25% of A_1's radius, then A_2 is taken as a candidate for join; otherwise A_2 cannot be joined with A_1. As an example, if A_1's radius is 100, then all arcs whose radii are between 75 and 125 would be taken as candidates for arc join. (2) *Center distance constraint:* The distance between the center of A_1 and the center of the candidate arc A_2 must be within some threshold. Specifically, we require that the distance between the centers of A_1 and A_2 must not exceed 25% of A1's radius. As an example, if A_1's radius is 100, then all arcs whose centers are within 25 pixels of A_1's center would be taken as candidates for arc join assuming they also satisfy the radius difference constraint.

Fig. 5 illustrates possible scenarios during arc join for circle detection. In Fig. 5(a), we illustrate a case where all potential arc candidates satisfy the center distance constraint, but one fails the radius difference constraint. Here, A_1 is the arc to be extended with A_2, A_3 and A_4 as potential candidates for arc join. As illustrated, the centers of all arcs are very close to each other; that is, the distance of the centers of A_2, A_3 and A_4 from the center of A_1 are all within the center distance threshold r_T. As for the radius difference constraint, only A_3 and A_4 satisfy it, while A_2's radius falls out of the radius difference range. So in Fig. 5(a), only arcs A_3 and A_4 would be selected as candidates for joining with A_1.

In Fig. 5(b), we illustrate a case where all potential arc candidates satisfy the radius difference constraint, but one fails the center distance constraint. Here, A_1 is the arc to be extended with A_2, A_3 and A_4 as potential candidates for arc join. As illustrated, the radii of all arcs are very close to each other, so they all satisfy the radius difference constraint. As for the center distance constraint, only A_2 and A_4 satisfy it, while A_3's center falls out of the center distance threshold r_T. So in Fig. 5(b), only arcs A_2 and A_4 would be selected as candidates for joining with A_1.

After the computation of the candidate arcs, the next step is to combine them one-by-one with the extended arc A_1 by fitting a new circle to the pixels making up both of the arcs. Instead of trying the join in random order, we start with the arc whose either end-point is the closest to either end-point of A_1. The motivation for this decision comes from the observation that if there is more than one arc that is part of the same great circle, it is better to start the join with the arc closest to the extended arc A_1. In Fig. 5(a) for example, we would first join A_1 with A_4 and then with A_3. Similarly, in Fig. 5(b) we would first join A_1 with A_2 and then A_4.

After an arc A_1 is extended with other arcs on the same great circle, we decide at the last step whether to make the extended arc a circle candidate. Here, we take the view that if an arc spans at least 50% of the circumference of its great circle, then we make the arc a circle candidate. Otherwise, the arc is left for near-circular ellipse detection. In Fig. 5(a) for example, when A_1 is joined with A_4 and A_3, the extended arc would span more than 50% of the circumference of its great circle. So the extended arc would be made a circle candidate. In Fig. 5(c) however, when A_1, A_2 and A_3 are joined together, we observe that the extended arc does not span at least 50% of the circumference of its great circle, i.e., $\theta_1+\theta_2+\theta_3<\pi$; so the extended arc is not taken as a circle candidate. Computation of the total arc span is performed by simply looking at the ratio of the total number of pixels making up the joined arcs to the circumference of the newly fitted circle. If this ratio is greater than 50%, then the extended arc is taken as a circle candidate.

To exemplify the ideas presented above, here is how the seven arcs depicted in Fig. 4(a) and detailed in Table 1 would be processed: we first take A_1, the longest arc, as the arc to be extended, with A_2, A_3, A_4, A_5, A_6 and A_7 as the remaining arcs. Since the radii of A_2, A_3 and A_4 are within 25% of A_1's radius and their center distances are within the center distance threshold, only these three arcs would be taken as candidates for join. We next join A_1 and A_2 since A_2's end-point is closest to A_1 (refer to Fig. 4(a)). After A_1 and A_2 are joined, the extended arc would now be joined with A_3 since A_3's end-point would now be closest to the extended arc. Finally, A_4 would be joined. Since the final extended arc covers more than 50% of its great circle, it is taken as a circle candidate. Continuing similarly, the next longest remaining arc is A_5, so we try extending A_5 with A_6 and A_7 being the only remaining arcs in our list of arcs. The only candidate

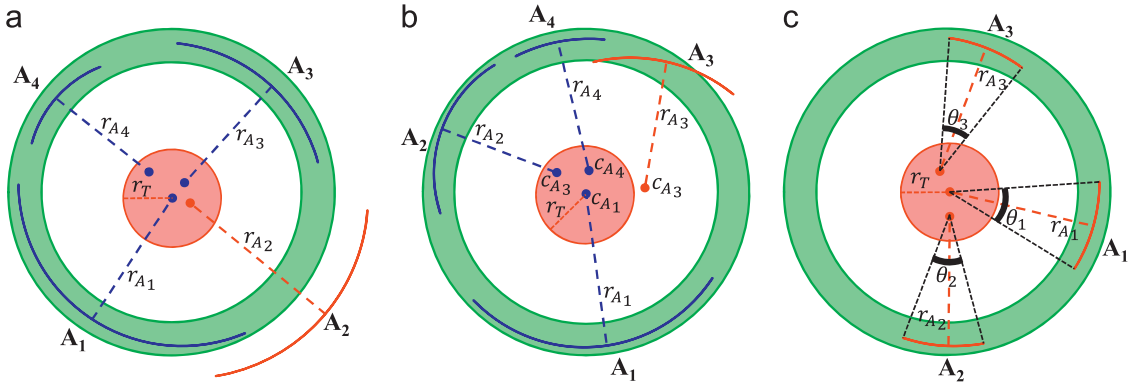


Fig. 5. Illustrations of possible scenarios during arc join for circle detection. (a) A_1 is the arc to be extended with A_2, A_3 and A_4 as potential candidates for arc join. All arcs satisfy the center distance constraint, but A_2 fails the radius difference constraint. So, only A_3 and A_4 are selected as candidates for joining with A_1. (b) A_1 is the arc to be extended with A_2, A_3 and A_4 as potential candidates for arc join. All arcs satisfy the radius difference constraint, but A_3 fails the center distance constraint. So, only A_2 and A_4 are selected as candidates for joining with A_1. (c) A_1, A_2 and A_3 all satisfy the radius difference and center distance constraints and are joined into a bigger arc. But, since the total length of the extended arc does not span at least 50% of the circumference of its great circle, i.e., $\theta_1+\theta_2+\theta_3<\pi$, it is not taken as a circle candidate.

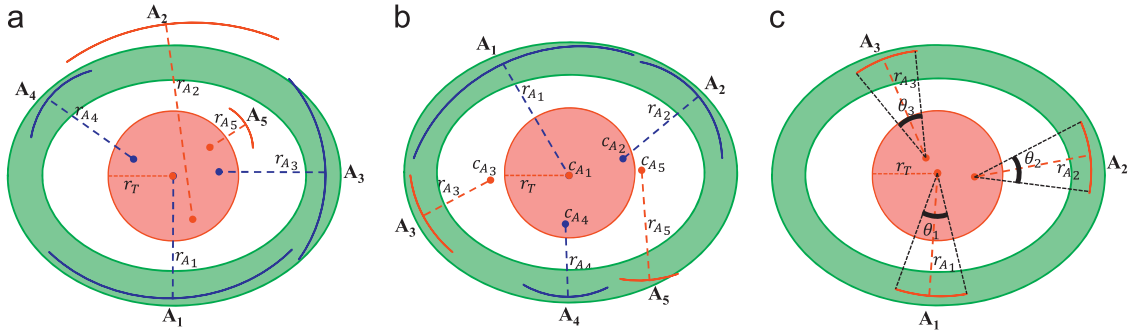


Fig. 6. Illustrations of possible scenarios during arc join for ellipse detection. (a) A_1 is the arc to be extended with A_2, A_3, A_4 and A_5 as potential candidates for arc join. All arcs satisfy the center distance constraint, but A_2 and A_5 fail the radius difference constraint. So, only A_3 and A_4 are selected as candidates for joining with A_1. (b) A_1 is the arc to be extended with A_2, A_3, A_4 and A_5 as potential candidates for arc join. All arcs satisfy the radius difference constraint, but A_3 and A_5 fail the center distance constraint. So, only A_2 and A_4 are selected as candidates for joining with A_1. (c) A_1, A_2 and A_3 all satisfy the radius difference and center distance constraints and are joined into a bigger elliptic arc. But, since the total length of the extended arc does not span at least 50% of the circumference of its great ellipse, i.e., $\theta_1+\theta_2+\theta_3<\pi$, it is not taken as an ellipse candidate.

arc that can be joined with A_5 is A_7 since A_6's radius is not within 25% of A_5's radius. When we try to join A_5 and A_7 by fitting a circle, the circle fit error turns out to be bigger than the error threshold of 1.5 pixels, so A_5 and A_7 are not joined together. Notice that A_5, A_6 and A_7 are part of a great ellipse rather than a great circle, so attempts to join them into a great circle fails. Clearly, these arcs need to be joined into an ellipse rather than a circle.

2.5. *Candidate near-circular ellipse detection by arc join*

The algorithm presented in the previous section deals with the detection of perfect circle candidates. But this algorithm would not detect near-circular ellipses. It is known that a circle appears slightly elliptical depending on the viewpoint of the camera. So it is important that the circle detection algorithm detects these imperfect circular objects in an image.

The arc join for ellipse detection is very similar to the arc join for perfect circle detection presented in Section 2.4. Given an arc A_1 to extend into an ellipse, we go over the remaining arcs and generate a set of candidate arcs that may be joined with A_1 into an ellipse. We employ the same two criterions for arc join as in Section 2.4 but with relaxed constraints: (1) *Radius difference constraint:* If A_2's radius is within 50% of A_1's radius, then A_2 is taken as a candidate for join; otherwise A_2 cannot be joined with A_1. As an example, if A_1's radius is 100, then all arcs whose radii are between 50 and 150 would be taken as candidates for arc join. (2) *Center distance constraint:* We require that the distance between the centers of A_1 and A_2 must not exceed 50% of A_1's radius. As an example, if A_1's radius is 100, then all arcs whose centers are within 50 pixels of A_1's center would be taken as candidates for arc join assuming they also satisfy the radius difference constraint.

Fig. 6 illustrates possible scenarios during arc join for ellipse detection. In Fig. 6(a), we illustrate a case where all potential arc candidates satisfy the center distance constraint, but two fail the radius difference constraint. Here, A_1 is the arc to be extended with A_2, A_3, A_4 and A_5 as potential candidates for arc join. As illustrated, the centers of all arcs are very close to each other; that is, the distance of the centers of A_2, A_3, A_4 and A_5 from the center of A_1 are all within the center distance threshold r_T. As for the radius difference constraint, only A_3 and A_4 satisfy it, while A_2's and A_5's radii fall out of the radius difference range. So in Fig. 6(a), only arcs A_3 and A_4 would be selected as candidates for joining with A_1.

In Fig. 6(b), we illustrate a case where all potential arc candidates satisfy the radius difference constraint, but two fail the center distance constraint. Here, A_1 is the arc to be extended with A_2, A_3, A_4 and A_5 as potential candidates for arc join. As illustrated, the radii of all arcs are very close to each other, so they all satisfy the radius difference constraint. As for the center distance constraint, only A_2 and A_4 satisfy it, while A_3's center and A_5's center fall out of the center distance threshold r_T. So in Fig. 6(b), only arcs A_2 and A_4 would be selected as candidates for joining with A_1.

After the computation of the candidate arcs, the next step is to combine them one-by-one with the extended arc A_1 by fitting a new ellipse to the pixels making up both of the arcs using the

ellipse fitting algorithm in [65]. If the root mean square error is smaller than 1.5 pixels, the join succeeds; otherwise, it fails. Instead of trying the join in random order, we start with the arc whose either end-point is the closest to either end-point of A_1. The motivation for this decision comes from the observation that if there is more than one arc that is part of the same great ellipse, it is better to start the join with the arc closest to the extended arc A_1. In Fig. 6(a) for example, we would first join A_1 with A_3 and then with A_4. Similarly, in Fig. 6(b) we would first join A_1 with A_2 and then A_4.

After an arc A_1 is extended with other arcs on the same great ellipse, we decide at the last step whether to make the extended arc an ellipse candidate. Here, we take the view that if the extended arc spans at least 50% of the circumference of its great ellipse, then we make the arc an ellipse candidate. Otherwise, the arc is eliminated from further processing. In Fig. 6(a) for example, when A_1 is joined with A_3 and A_4, the extended arc spans more than 50% of the circumference of its great ellipse. So the extended arc is made an ellipse candidate. In Fig. 6(c) however, when A_1, A_2 and A_3 are joined together, we observe that the extended arc does not span at least 50% of the circumference of its great ellipse, i.e., $\theta_1+\theta_2+\theta_3<\pi$. So the extended arc is not taken as an ellipse candidate. Computation of the total arc span is performed by simply looking at the ratio of the total number of pixels making up the joined arcs to the circumference of the newly fitted ellipse. If this ratio is greater than 50%, then the extended arc is taken as an ellipse candidate.

To exemplify the ideas presented above, here is how the remaining three arcs, A_5, A_6 and A_7, in Fig. 4(a) and detailed in Table 1 would be processed: since A_5 is the longest arc, we try extending A_5 with A_6 and A_7 as the candidates. Since the radii of both A_5 and A_6 are within 50% of A_5's radius and their center distances are within the center distance threshold, both arcs would be taken as candidates. We then try joining A_5 and A_6 since A_6's end-point is closest to A_5 (refer to Fig. 4(a)). Finally, A_7 would be joined. Since the final extended arc covers more than 50% of its great ellipse, it is taken as an ellipse candidate.

Fig. 4(b) shows all circle and ellipse candidates for the image in Fig. 1(a) overlayed in red color on top of the image. We have three circle and two ellipse candidates of which the small circle, the small ellipse and the small polygon-like object became candidates at the first step of the algorithm due to their edge segment tracing their entire boundary as a closed curve; the big circle became a candidate at the fourth step by joining the arcs A_1, A_2, A_3 and A_4 into a perfect circle; and the occluded ellipse became a candidate at the fifth step by joining the arcs A_5, A_6 and A_7 into an ellipse. All five candidates would now have to go through validation by the Helmholtz principle (described in the next section) before being returned as true positive detections.

2.6. Circle and ellipse validation by the Helmholtz principle

In the last two sections we described how EDCircles joins arcs into circle or ellipse candidates. Just because several arcs can be joined into circles and ellipses does not mean that all candidates are valid detections. So as a final step, EDCircles employs a circle and ellipse validation algorithm to eliminate invalid detections and return only the valid circles and ellipses.

Before moving on to circle and ellipse validation algorithm employed by EDCircles, we first need to define the Helmholtz principle, lay out the foundation for its *a contrario* validation framework, and describe how it is used to detect meaningful alignments, i.e., line segments, in a given image. We then adapt the line validation technique to circle and ellipse validation.

Helmholtz principle simply states that for a geometric structure to be perceptually meaningful, the expectation of this structure (grouping or Gestalt) by chance must be very low in a random situation [59,60]. This is an *a contrario* approach, where the objects are detected as outliers of the background model. As shown by Desolneux et al. [60], a suitable background model is one in which all pixels are independent. They show that the simplest such model is the Gaussian white noise. Stated in other words, no meaningful structure is perceptible in a Gaussian white noise image [60].

Desolneux et al. use the Helmholtz principle to find meaning alignments, i.e., line segments, in a given image without requiring any parameters [61]. Their idea is to compute the level line orientation field (which is orthogonal to the gradient orientation field) of a given image, and look for a contiguous set of pixels having similar level line orientation. Fig. 7(a) shows the level line orientation field for an image, where the aligned pixels that make up for a line segment are marked inside a rectangle. The authors define what *aligned* means as follows: Two points (or line segments) P and Q have the same direction, i.e., aligned, with precision $p=1/n$ if angle(P) and angle(Q) are within $p/\pi=\pi/n$ degrees of each other. Desolneux et al. state that "in agreement with psychophysics [68] and numerical experimentation, the realistic values of n range from 32 to 4 and it is, in general, useless to consider larger values of n" [60]. In EDCircles, we fix the value of n to 8 and thus, p, the precision or the accuracy of direction between two pixels, is equal to $p=\frac{1}{8}=0.125$ and two points are aligned (or p-aligned) if their angles are within $p\pi=\pi/8=22.5^\circ$ of each other.

The gradient magnitude and the level line angle at a pixel (x, y) are computed using a 2×2 mask as follows [61]:

$$g_x(x,y)=\frac{I(x+1,y)-I(x,y)+I(x+1,y+1)-I(x,y+1)}{2} \quad (1)$$

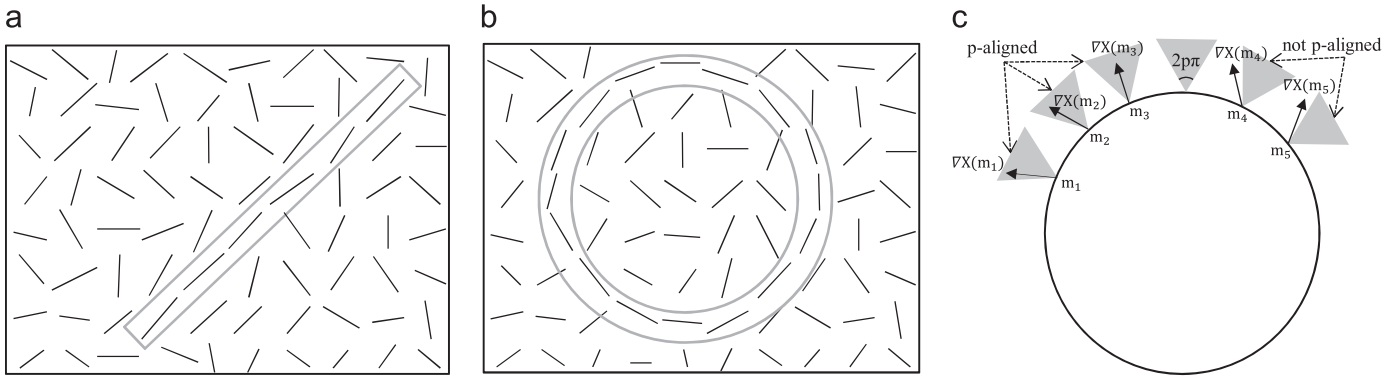


Fig. 7. (a) Level line orientation field (orthogonal to the gradient orientation) of an image. Aligned pixels (inside the rectangle) clearly make up for a line segment. (b) Another level line orientation field. The circular structure (inside the two circles) is clearly visible. (c) Illustration of several p-aligned and not p-aligned gradients. Observe that aligned gradients are perpendicular to the level-line angle and are inside the tolerance cone of $2p\pi$.

$$g_y(x,y) = \frac{I(x,y+1) - I(x,y) + I(x+1,y+1) - I(x+1,y)}{2} \quad (2)$$

$$g(x,y) = \sqrt{g_x(x,y)^2 + g_y(x,y)^2} \quad (3)$$

$$\alpha = \angle g_x g_y = \arctan\left(\frac{g_x(x,y)}{-g_y(x,y)}\right) \quad (4)$$

where $I(x,y)$ is the intensity of the input image at pixel (x, y), $g(x, y)$ is the gradient magnitude, and angle(x, y) is the level line angle. Authors in [61] state that the reason for using such a simple gradient operator is to reduce the dependency of the computed gradients, and thus, preserve pixel independence as much as possible.

To make validation by the Helmholtz principle concrete, Desolneux et al. define what is called the *number of false alarms* (*NFA*) of a line segment as follows [61]: Let L be a line segment of length n with at least k points having their directions aligned with the direction of L in an image of size $N \times N$ pixels. Define NFA of L as [62,63]

$$NFA(n,k) = N^4 \sum_{i=k}^{n} \binom{n}{i} \cdot p^i \cdot (1-p)^{n-i} \quad (5)$$

where N^4 represents the number of potential line segments in an $N \times N$ image. This is due to the fact that a line segment has two end-points, each of which can be located in any of the N^2 pixels of the image; thus, a total of $N^2 \times N^2 = N^4$ line segments. The probability p used in the computation of the binomial tail is the accuracy of the alignment between the pixel's level line angle and the line segment.

An event (a line segment in this case) is called ϵ-meaningful if its $NFA(n,k) \le \epsilon$. Desolneux et al. [59–61] advises setting ϵ to 1, which corresponds to one false detection per image. Given these definitions, a line segment is validated as follows: for a line segment of length n, compute the level line angle of each pixel along the line and count the number of aligned pixels k. Then, compute $NFA(n,k)$ and accept the line segment as valid if $NFA(n,k) \le 1$. Otherwise, reject the line segment.

Desoulneux et al.'s line segment validation framework outlined above has successfully been used by two recent line segment detectors; namely, LSD by Grompone von Gioi et al. [62,63], and EDLines by Akinlar et al. [56,57], to validate the detected line segments and eliminate false detections.

In this paper, we adapt Desolneux et al.'s line segment validation framework outlined above to circle and ellipse detection. The idea is very simple: just like a contiguous set of pixels having level line angles aligned with a line segment make up for a line segment, a contiguous set of pixels having level line angles aligned with a circle make up for a circle. Fig. 7(b) shows the level line orientation field of an image, where the aligned pixels that make up for a circle are marked inside two circles. To define alignment between a pixel and a circle, we simply adapt the alignment definition between a pixel and a line segment: A pixel P on the boundary of a circle is aligned with the circle, if P is aligned with the tangent to the circle at P. Recall from above that a point P and a line segment L is aligned with precision p, if angle(P) and angle(L) are within $p*\pi$ degrees of each other. Assuming that the tangent line of a circle at a given point P is T, we can simply use this definition for alignment between a point and a circle; that is, angle(P) and angle(T) have to be within $p*\pi$ degrees of each other if P and the circle are aligned.

Fig. 7(c) shows the gradient directions (perpendicular to the level line angles) of several points on the boundary of a circle, some of which are p-aligned with the circle, and some of which are not. The gray triangle illustrates the tolerance cone between the ideal gradient direction and the observed gradient direction. If the observed gradient direction is inside the cone, then the point is assumed to be aligned with the circle, otherwise the point is assumed to be non-aligned with the circle.

Instead of treating the circles and ellipses as two different gestalts to be validated, we formulate our problem simply as an ellipse validation problem since a circle is essentially an ellipse whose major and minor axis lengths are equal. With this in mind, we adapt the definition of the *number of false alarms* (*NFA*) of a line segment to an ellipse as follows: let E be an ellipse having a circumference of n points with at least k points having their directions aligned with E in an image of size $N \times N$ pixels. Define NFA of E as

$$NFA(n,k) = N^5 \sum_{i=k}^{n} \binom{n}{i} \cdot p^i \cdot (1-p)^{n-i} \quad (6)$$

where N^5 represents the total number of potential ellipses in an $N \times N$ image. This is due to the fact that an ellipse has 5 degrees of freedom, i.e., its center coordinates, the major axis, the minor axis, and the orientation; hence, a total of N^5 ellipses.

Given this NFA definition, we validate a circle/ellipse (circle for short) as follows: For a circle having length n, compute the level line angle of each pixel along the circle and count the number of aligned pixels k. Compute $NFA(n,k)$ and accept the circle as valid if $NFA(n,k) \le \epsilon$; otherwise the circle is rejected. It is important to point out that we perform the gradient computation in the validation step over the original non-filtered image as required by the *a contrario* framework.

The epsilon (ϵ) in the above comparison denotes the expected number of detections under the background model. In other words, if Gaussian white noise image is fed into the algorithm, we should get at most ϵ many detections. We set ϵ to 1 as advised by Desolneux et al. [59,60], which corresponds to one false detection per image.

While the observed level line angle is computed by the 2×2 mask given above, computing the ideal pixel angle at a point on the circumference of a circle/ellipse requires the computation of the tangent line at that point. If the level line angle and the tangent line angle are within $p*\pi = 0.125*180 = 22.5^\circ$ of each other, then they are aligned; otherwise they are not.

2.7. Complexity of EDCircles

Given an $N \times N$ image, the first step of EDCircles is the edge segment detection by EDPF, which is an $O(N^2)$ operation. The detected edge segments are then converted to line segments, which is a linear time operation on the number of edge pixels. Assuming we have L line segments, conversion of the line segments into arcs is again a linear time operation on the number of line segments; that is, O(L). If we now have A arcs, the rest of the algorithm first involves sorting the arcs with respect to their length, which can be done in $O(A \log A)$, and then taking the longest remaining arc from the sorted list, finding the candidate arcs in $O(A)$, and joining the candidate arcs with the target arc using the greedy heuristic. This is a quadratic operation on the number of arcs; that is, $O(A^2)$.

3. Experiments

To measure the performance of EDCircles [54,55], 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 of EDCircles. We stress 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 at http://ceng.anadolu.edu.tr/CV/EDCircles/Demo.aspx.

Fig. 8 shows the performance of EDCircles and `cvHoughCircles` on five synthetic and natural images containing circular

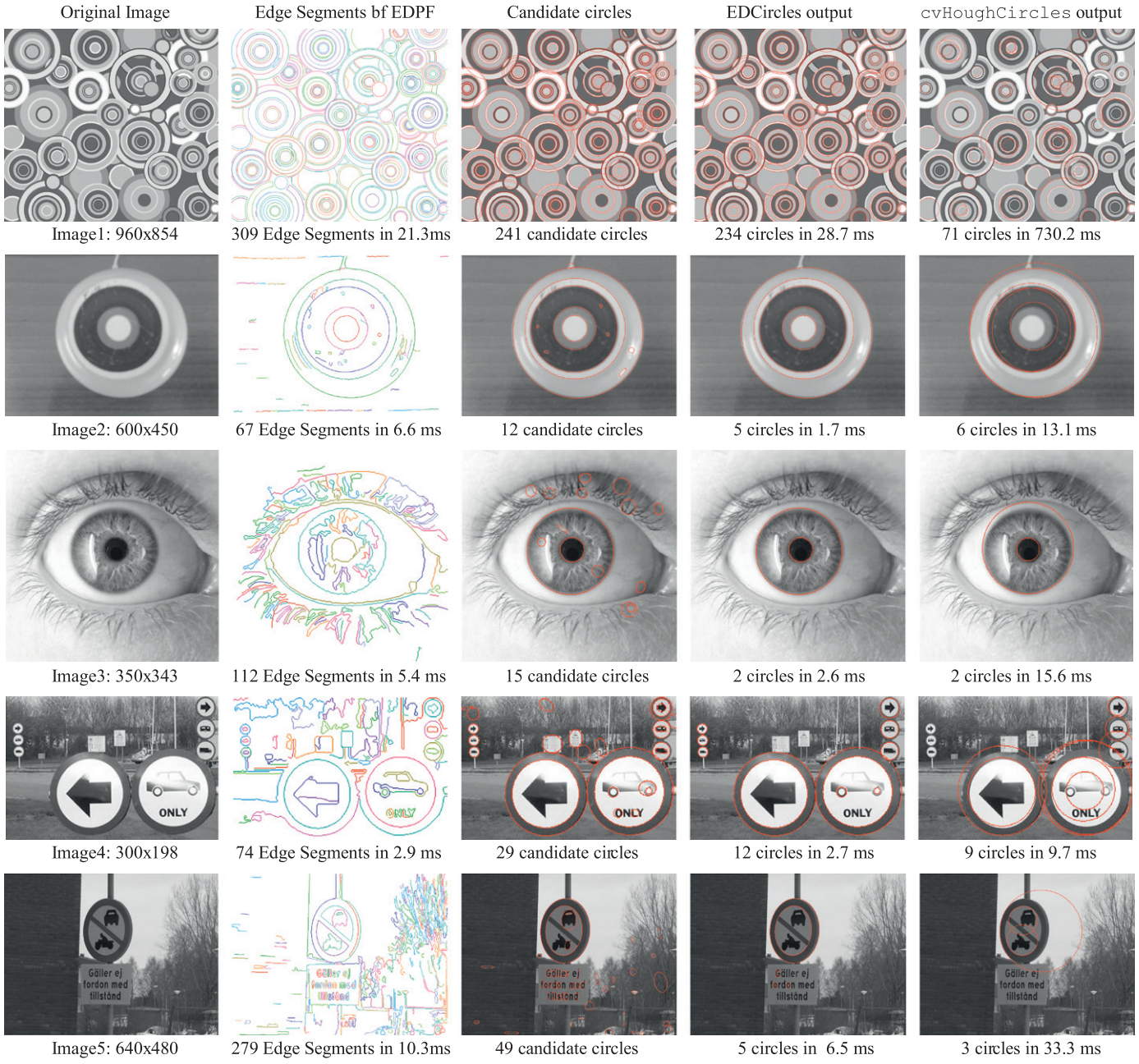


Fig. 8. Circle detection results by EDCircles (4th column), and OpenCV `cvHoughCircles` (last column). (For interpretation of the references to color in this figure caption, the reader is referred to the web version of this article.)

objects. The running times were measured in a PC with a 2.2 GHz Intel 2670QM CPU. The first column shows the original image, and the second column shows the edge segments detected by EDPF. In the third column, the candidate circles and ellipses (circles for short in the rest of this paper) are overlayed on top of the image with red color. Notice that there are many false candidate circles, especially visible on the eye and traffic sign images. The fourth column shows the final result output by EDCircles. Notice that the circle validation algorithm due to the Helmholtz principle eliminates all invalid circle candidates leaving only valid circle detections. For example, there are 29 candidate circles in Image4, 17 of which are false detections; therefore, EDCircles outputs only 12 valid circles. Notice that the detected circles are of various sizes from very small to large.

The fifth column in Fig. 8 shows the circles detected by OpenCV's Circle Hough Transform (CHT)-based circle detection algorithm, `cvHoughCircles`. The reason for choosing OpenCV [66] for comparison is because OpenCV's implementations of generic image processing and computer vision algorithms are known to be the fastest in the literature and are widely used, and because OpenCV 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 for each image. To obtain the results by `cvHoughCircles`, we first smoothed the images with a 5×5 Gaussian kernel with $\sigma=1.0$. We then tried many different parameters and present the best results in Fig. 8. Table 3 lists the parameters used to obtain `cvHoughCircles`' results shown in Fig. 8. Notice from Table 3 that `cvHoughCircles` requires a different set of parameters to obtain the best results for each image. With a single set of default parameters, `cvHoughCircles` either fails to detect many valid circles or produces a lot more false detections, which are not shown in this paper.

Table 2
Dissection of EDCircles's running time and its comparison to `cvHoughCircles`.

Image (width × height)	Edge segment detection by EDPF (ms)	Circle detection and validation (ms)	Total (ms)	`cvHough Circles` (ms)	Speedup
Image1 (960 × 854)	21.3	28.7	50.1	730.2	14.5
Image2 (600 × 450)	6.6	1.7	8.3	13.1	1.5
Image3 (350 × 343)	5.4	2.6	8.0	15.6	1.9
Image4 (300 × 198)	2.9	2.7	5.6	9.7	1.7
Image5 (640 × 480)	10.3	6.5	16.8	33.3	1.9

Table 3
`cvHoughCircles`'s parameters used for each image in Fig. 8.

Image	dp	Minimum distance between circles	Canny high threshold	Accumulator threshold
Image1	1	16	120	140
Image2	1	8	80	100
Image3	1	2	100	120
Image4	1	2	120	80
Image5	1	2	140	90

It is clear from Fig. 8 that EDCircles detects most valid circles in real-time, whereas `cvHoughCircles` not only fails in detecting many valid circles but also produces many false detections. This is more evident in Image1, where `cvHoughCircles` takes close to 1 s to execute and outputs 71 circles many of which are false detections. We stress once again that EDCircles is parameter-free, i.e., it uses the same internal parameters for all images; whereas, to get the best results with `cvHoughCircles`, the user has to supply a different set of parameters for each image by trial-and-error.

The last row, i.e., Image5, in Fig. 8 requires special attention as it shows a circular traffic sign from the Sweedish Traffic Sign database [67]. Due to the view-point of the camera, this circular sign appears as an ellipse rather than a circle. Notice from Fig. 8 that EDCircles detects this elliptic circle correctly; however, a traditional circle detector such as `cvHoughCircles` detects three different circles none of which correctly represents the traffic sign.

Table 2 dissects the running time of EDCircles for each image in Fig. 8 and compares it with the running time of `cvHoughCircles`. Clearly, EDCircles runs real-time for typical camera input sizes of 640 × 480 with much of the time spent on edge segment detection by EDPF rather than circle detection and validation. Also note that EDCircles is up to 14 times faster than `cvHoughCircles`, which is probably the fastest implementation of the classical CHT algorithm.

We next compare EDCircles to a randomized circle detection (RCD) algorithm first proposed in [26] and recently improved in [27,28], where the authors discuss different RCD variants and present the results for the variant called GRCD-R. Unfortunately, there are no publicly available implementations of this algorithm. So we take the test images from the author's original paper along with their results [28], feed the test images into EDCircles and compare the results.

Figs. 9 and 10 show the 10 test images and the results produced by GRCD-R taken from [28], and the results produced by EDCircles. Chung et al. state that the running times for GRCD-R's results were obtained in a 3 GHz Intel E8400 CPU [28]. To make a fair running time comparison between GRCD-R and EDCircles, we run EDCircles in a 2.2 GHz Intel E4500 CPU, which is also a Core 2 duo CPU similar to E8400, but has a slower clock speed and lower CPU rating according to CPU benchmarks.

It is clear from the results in Figs. 9 and 10 that EDCircles detects most valid circles except for some light reflections in 'gobang' and 'stability-ball' images and the central ball in 'gobang' image, and runs 5 times faster than GRCD-R on the average. Refer to Table 4 for a side-by-side comparison of the execution times of GRCD-R and EDCircles for the 10 images in Figs. 9 and 10. It is important to note that the execution times for GRCD-R include just the circle detection after edge detection, whereas the execution times for EDCircles include both the edge segment detection by EDPF and the following circle detection and validation. Considering that at least half of the running time of EDCircles is spent on edge segment detection and that EDCircles was run on a slower CPU, the actual performance of EDCircles compared to GRCD-R is better than reflected in Table 4.

Also notice that EDCircles detects circles of various sizes in an image, while GRCD-R fails in detecting many valid circles. This is more evident in 'logo' and 'speaker' images of Fig. 10, where GRCD-R detects only the big circles, while EDCircles detects many circles having small and large radius.

One disadvantage of EDCircles is multiple circle detections around circles having fuzzy boundaries. This is more evident in 'coin' and 'cake' images of Fig. 9, where EDCircles detects two circles around two different coins, and two circles around three different circular objects. Notice that those objects have shadows around their boundaries, which are detected by EDPF and turned into circles by EDCircles. The reason GRCD-R does not detect multiple circles around the same objects is probably because the edge maps used by GRCD-R was cleaner and did not include these shadows as edges. Since we do not know about the edge maps used in GRCD-R's circle computations, we are not in a position to make a fair comparison between GRCD-R and EDCircles in this sense.

To evaluate the accuracy of EDCircles, we run the traditional circular Hough transform (CHT) [12] on the 10 test images shown in Figs. 9 and 10, and obtain the ground truth results. To make the CHT results as accurate as possible, we set granularity of the x-coordinate, y-coordinate and the radius to one pixel precision. Table 5 shows the average differences between the parameters of each circle detected by CHT and those detected by GRCD-R and EDCircles for each of the test images. As seen from the results, EDCircles has sub-pixel accuracy, although it does not produce as accurate results as GRCD-R.

In the next experiment, we measure the performance of EDCircles in detecting near-circular elliptic objects. As we pointed out before, depending on the viewpoint of the camera, circular objects may appear slightly elliptical in the image. A good circle detector should be able to take care of such cases.

Fig. 11 shows the performance of EDCircles in detecting near-circular elliptic objects in four images. Two of the images are from a pupil detection application; the other two are from the Sweedish Traffic Sign Database [67] and contain several circular traffic signs. Notice that all circular objects, which appear slightly elliptical in the images due to the viewpoint of the camera, have been detected with no false detections. We would like to stress though that EDCircles is not a general ellipse detection algorithm and is not intended to be one. Rather, EDCircles tries to detect circular objects which look slightly elliptical due to the view-point of the camera.

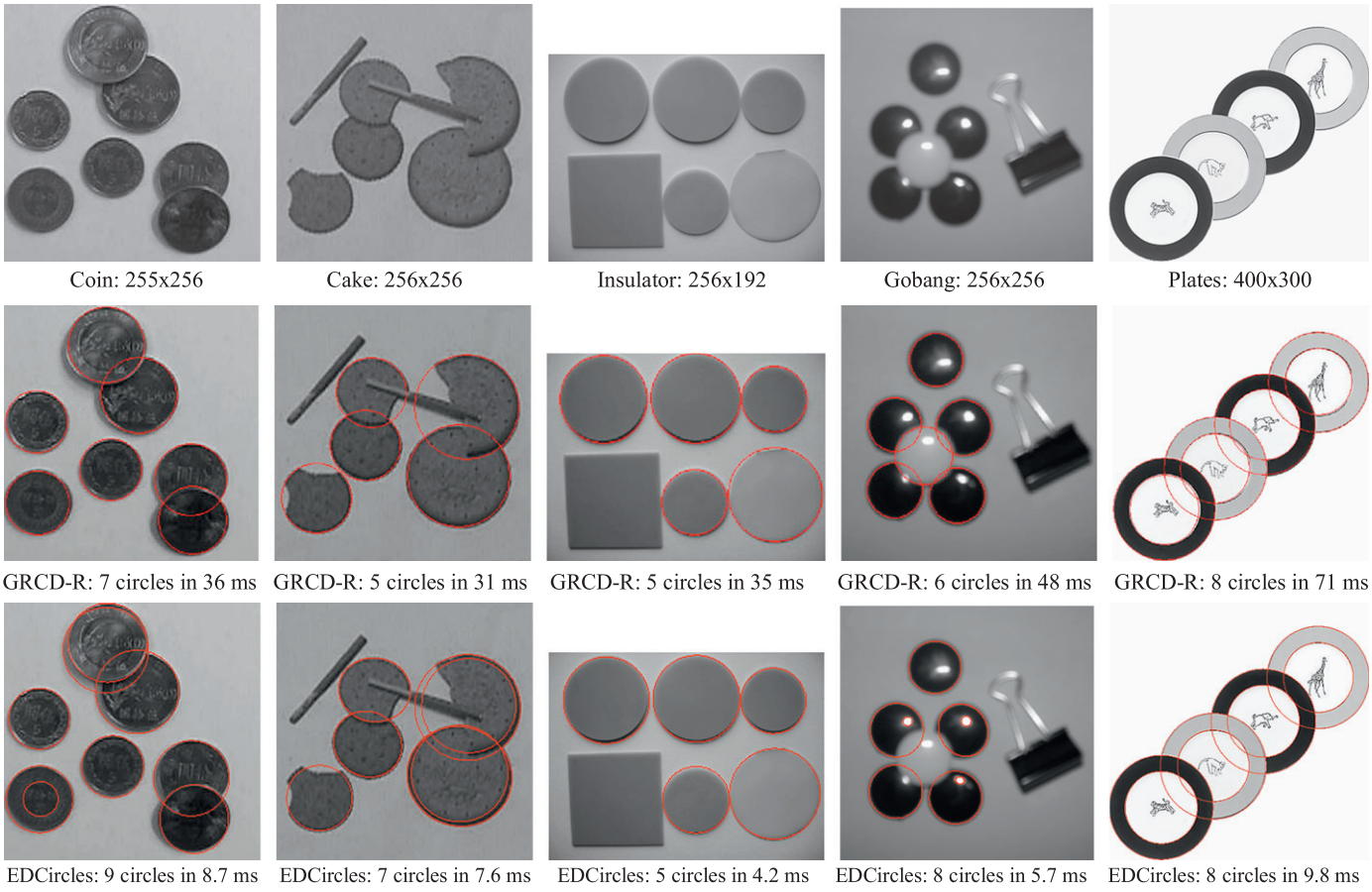


Fig. 9. (1st row) Five test images, (2nd row) circle detection results by GRCD-R, and (3rd row) circle detection results by EDCircles. The test images, and GRCD-R's circle detection results and running times were taken from the authors' original paper [28]. The running times for GRCD-R were obtained in a 3 GHz Intel E8400 CPU and include just the circle detection after edge detection, whereas the running times for EDCircles were obtained in a 2.2 GHz Intel E4500 CPU and include both the edge segment detection and the circle detection. Since E4500 is a slower CPU than E8400, EDCircles's performance compared to GRCD-R is better than reflected here.

Informally, we can say that EDCircles tries to detect ellipses where the major axis is at most twice as big as the minor axis. This is due to the arc join heuristic algorithm outlined in Section 2.5, which only joins two arcs for an ellipse if the arcs' radii differ by at most 50% from each other. If a circular object appears way too elliptic due to the viewpoint of the camera, EDCircles would not be able to detect it. Arbitrary ellipse detections shown on EDCircles Web site [55] are mostly due to the entire boundary of the ellipse being detected as a single closed edge segment.

Fig. 12 examines several failure cases for EDCircles. The first column shows a set of tightly concentric circles divided into eight quarters, which provokes EDCircles's arc join heuristics. As seen from the results, in several cases EDCircles incorrectly joins an arc from an outer circle with an arc from an inner circle resulting in circles that span half of an outer circle and half of an inner circle. The reason for this has to do with the greedy join heuristic employed by EDCirles. Recall from Section 2.4 that while EDCircles tries to extend an arc, it generates a set of candidate arcs and joins the currently extended arc with the candidate arc whose endpoint is the *closest*. If the join succeeds, the arcs are immediately joined, and the joined arcs are never separated afterwards. This is a greedy heuristic and is chosen for performance reasons. Although this heuristic results in good circle detections in many cases, it fails especially if there are tightly coupled concentric circles divided into two or more regions as in Fig. 12. A similar problem can also be observed in the natural image of 'Drain Cover', where EDCircles again joins an arc from an outer circle with an arc from the inner circle. The images in the first and second column of Fig. 12 illustrate another weak point of EDCircles; the failure in detection of small divided circles. Notice that in both images EDCircles fails in detecting the small circles at the center of the images. This has to do with EDCircle's arc generation strategy. Recall from Section 2.3 that an arc is generated with at least three lines that turn in the same direction and satisfy the angle constraints, which is generally not satisfied in small circles and the detection fails. The third column in Fig. 12 shows three blood cells with fuzzy boundaries, where detection either fails or produces non-perfect results. With objects having fuzzy boundaries, EDPF, the edge segment detector employed by EDCircles, produces ragged edge segments, which cannot be turned into arcs, and the detection fails. The last column in Fig. 12 shows a spiral and the circles detected by EDPF. As the spiral turns, EDCircles generates many arcs, which are combined together into the circles shown in the figure. We again see arcs from an inner circle being joined with arcs from an outer circle. The validation due to the Helmholtz principle is also not able to eliminate these false detections either.

Our last experiment is to measure the performance of EDCircles in noisy images. To this end, we take an image containing several small and big circles, add varying levels of Gaussian white noise to the image, and feed the images to EDCircles.

Fig. 13 demonstrates the performance of EDCircles as the amount of Gaussian white noise is increased over a sample image containing a big circle and several small circles. Notice that with increasing noise, the detection of the small circles starts to fail;

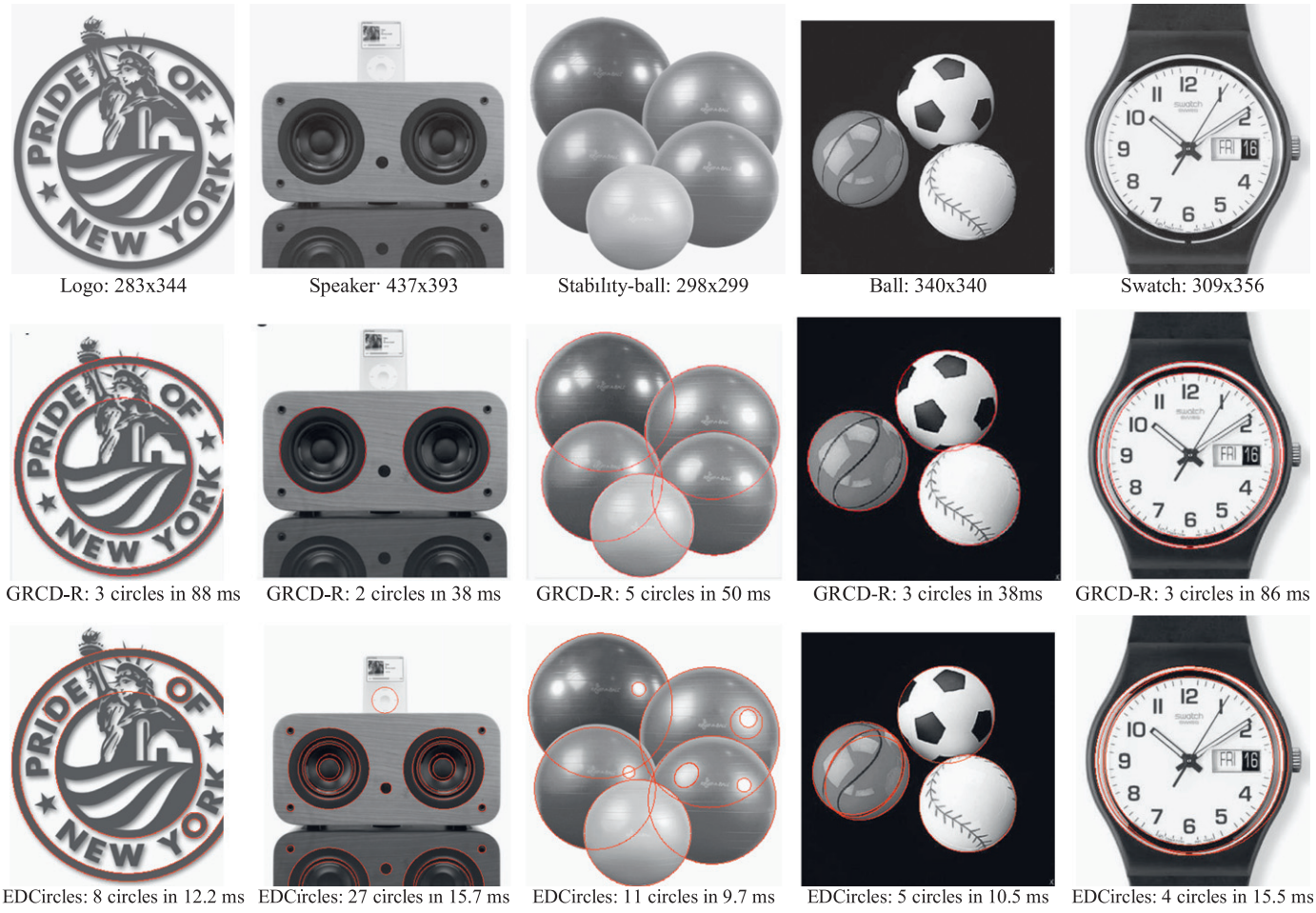


Fig. 10. (1st row) Five test images, (2nd row) circle detection results by GRCD-R, and (3rd row) circle detection results by EDCircles. The test images, and GRCD-R's circle detection results and running times were taken from the authors' original paper [28]. The running times for GRCD-R were obtained in a 3 GHz Intel E8400 CPU and include just the circle detection after edge detection, whereas the running times for EDCircles were obtained in a 2.2 GHz Intel E4500 CPU and include both the edge segment detection and the circle detection. Since E4500 is a slower CPU than E8400, EDCircles's performance compared to GRCD-R is better than reflected here.

Table 4
Execution-time performance comparison between GRCD-R and EDCircles in terms of milliseconds for the 10 images in Figs. 9 and 10. The execution times for GRCD-R include just the circle detection after edge detection, whereas the execution times for EDCircles include both the edge segment detection by EDPF and the following circle detection and validation.

Image	GRCD-R	EDCircles
Coins	36	8.7
Cake	31	7.6
Insulator	35	4.2
Gobang	48	5.7
Plates	71	9.8
Logo	88	12.2
Speaker	38	15.7
Stability-ball	50	9.7
Ball	38	10.5
Swatch	86	15.5
Average	52	9.9

Table 5
Average differences between the parameters of each circle detected by circular Hough transform (CHT) and those detected by GRCD-R and EDCircles for the 10 test images in Figs. 9 and 10.

Image	GRCD-R		EDCircles	
	$(\Delta a,\Delta b)$	Δr	$(\Delta a,\Delta b)$	Δr
Coins	(0.55, 0.34)	0.47	(0.81, 1.01)	0.66
Cake	(0.43, 0.67)	0.46	(0.83, 0.75)	1.18
Insulator	(0.31, 0.57)	0.45	(0.50, 1.13)	0.70
Gobang	(0.38, 0.43)	0.50	(0.45, 0.33)	0.25
Plates	(0.50, 0.54)	0.52	(0.43, 0.26)	0.62
Logo	(0.02, 0.40)	0.06	(0.61, 0.57)	0.49
Speaker	(0.26, 0.01)	0.46	(0.59, 0.47)	0.66
Stability-ball	(0.42, 0.65)	0.51	(0.75, 1.02)	0.84
Ball	(0.28, 0.35)	0.50	(0.66, 1.10)	0.85
Swatch	(0.41, 0.63)	0.71	(0.65, 0.64)	0.68
Average	(0.36, 0.46)	0.46	(0.63, 0.73)	0.66

until at Gaussian noise with $\sigma = 60$, only the big circle is detected. Increasing the noise further causes complete detection failure, which is not shown in the figure. The reason for the detection failure comes from the fact that as the noise is increased, the boundaries of the circles are approximated by many short edge segments instead of a few long edge segments as would be the case in less noisy images. This means that the arcs approximating the boundary of a circle would be short and cannot be joined together to make up for the circle. Notice though that there are no false detections in none of the images, which is also very important. We also observe that the running time of EDCircles increases in noisy images. The reason for this is the increased edge segment detection

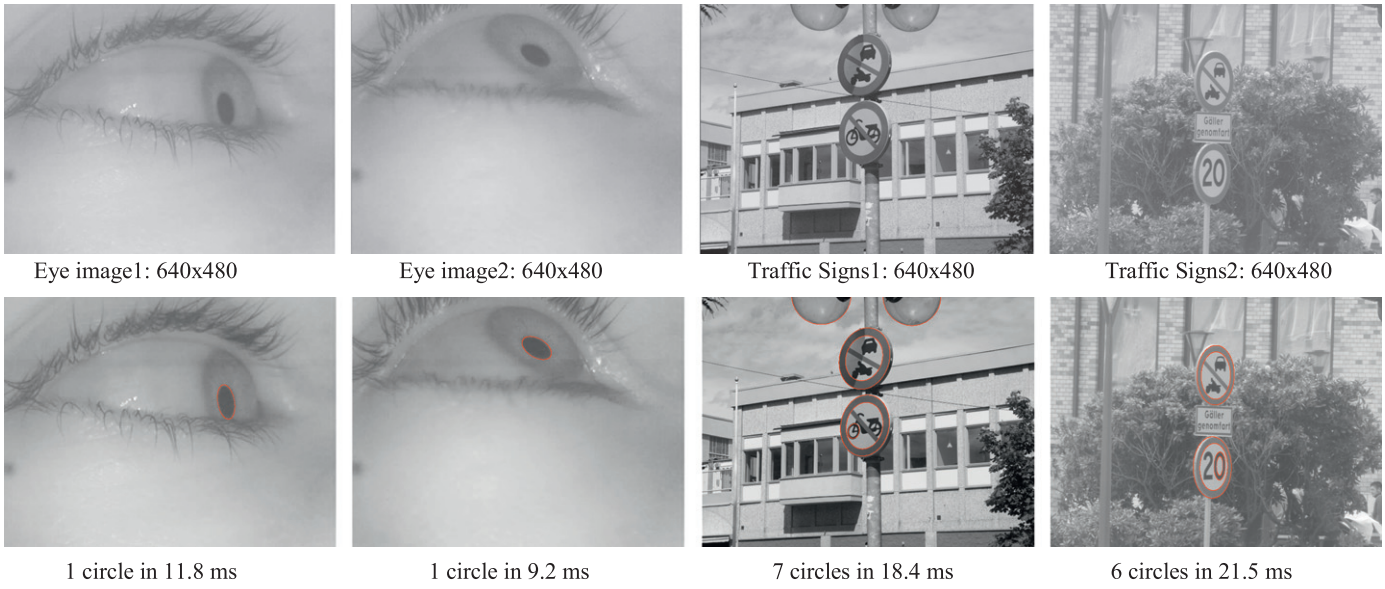


Fig. 11. The performance of EDCircles in detecting near-circular elliptic objects in four images. Notice that all objects have been detected successfully in real-time without any false detections.

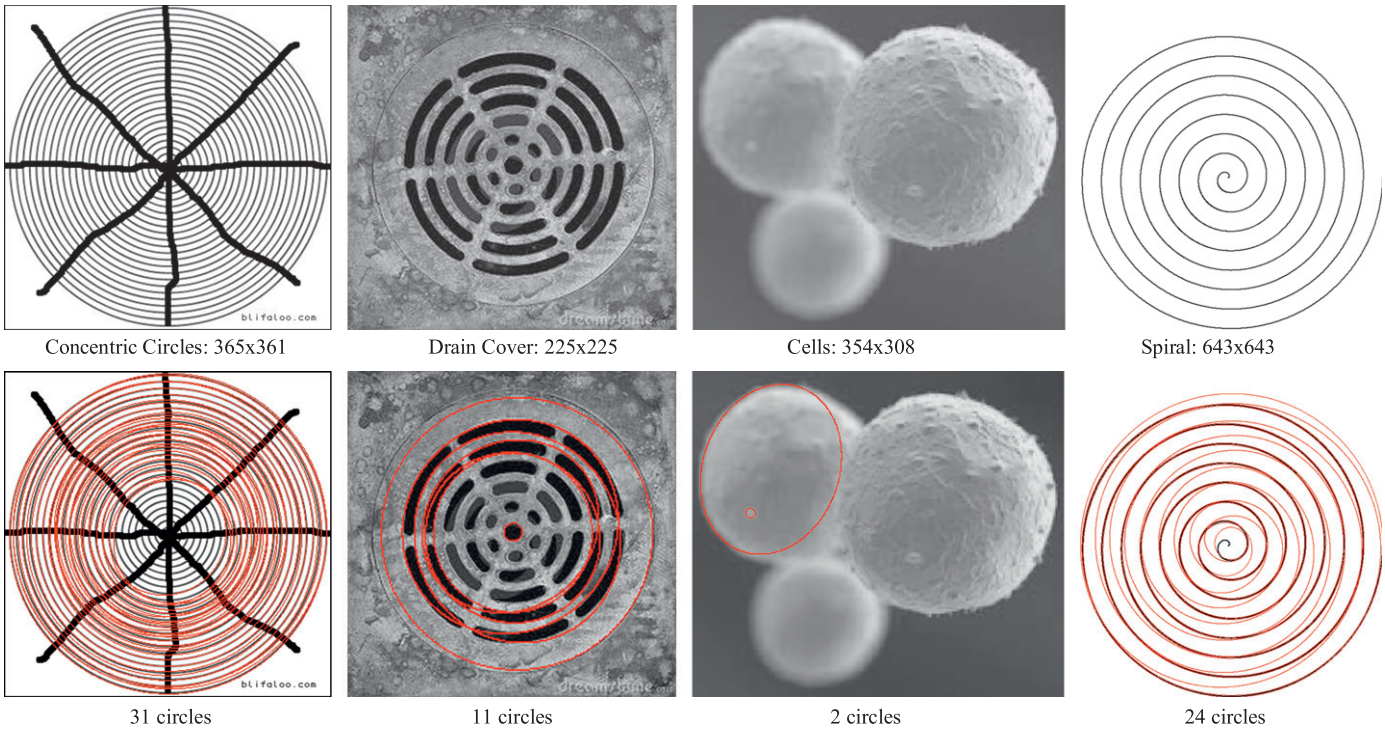


Fig. 12. Several failure cases for EDCircles.

time by EDPF. As the amount of noise increases in an image, EDPF takes more time to compute the edge segments because many pixels start becoming potential edge elements. Circle detection after edge segment detection remains pretty constant across all images.

Fig. 14 shows the results of analyzing a very noisy image (containing a Gaussian noise with $\sigma = 100$) at different scales. At full scale, no structure is detected, although they are visible to a human observer. This is because human visual system performs a multiscale analysis and is able to detect the structures inside strong noise. Running EDCircles at half and quarter resolutions, we see that several circles are now detected. We conclude that when strong noise is present, a denoising or zoom-in step would help in the detection of many valid circles by EDCircles.

4. Conclusions

EDCircles is a real-time, parameter-free circle detection algorithm that works by post-processing the edge segments detected by our parameter-free edge segment detector, edge drawing parameter free (EDPF). EDCircles first converts each edge segment, which is a contiguous chain of pixels, into a set of line

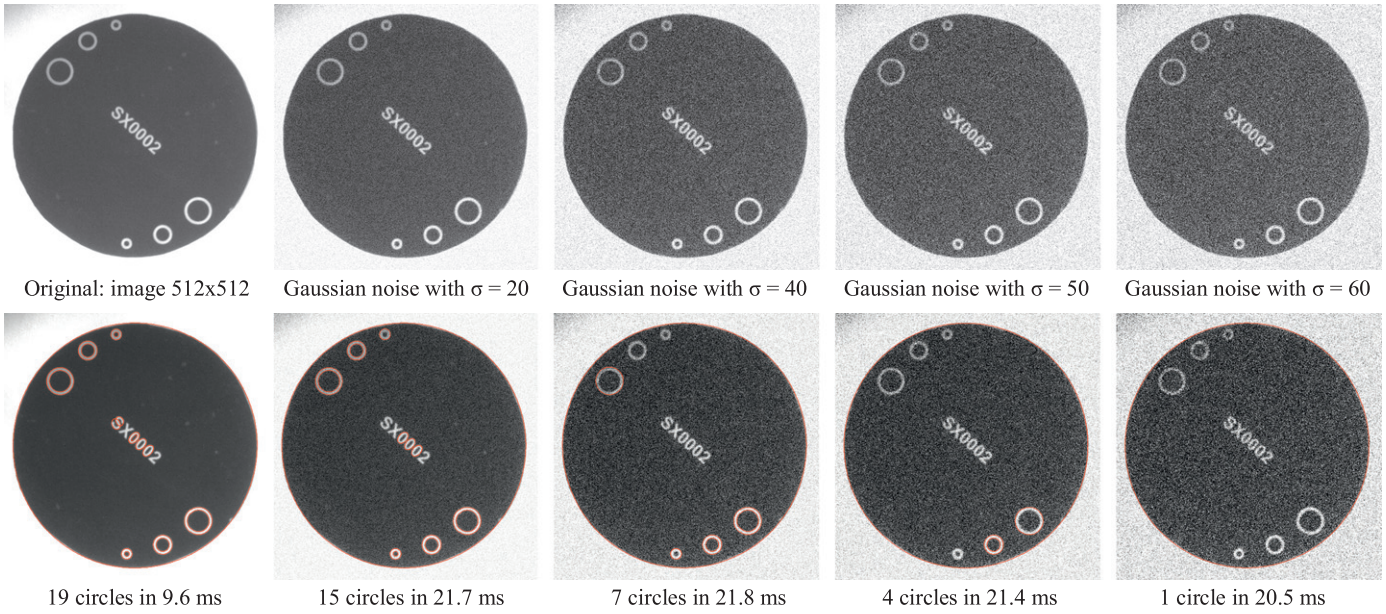


Fig. 13. The performance of EDCircles as the amount of Gaussian white noise is increased up to $\sigma = 60$. Notice that with increasing noise, the detection of the small circles starts to fail; until at noise with $\sigma = 60$, only the big circle is detected. Increasing the noise further causes complete detection failure, which is not shown in the figure.

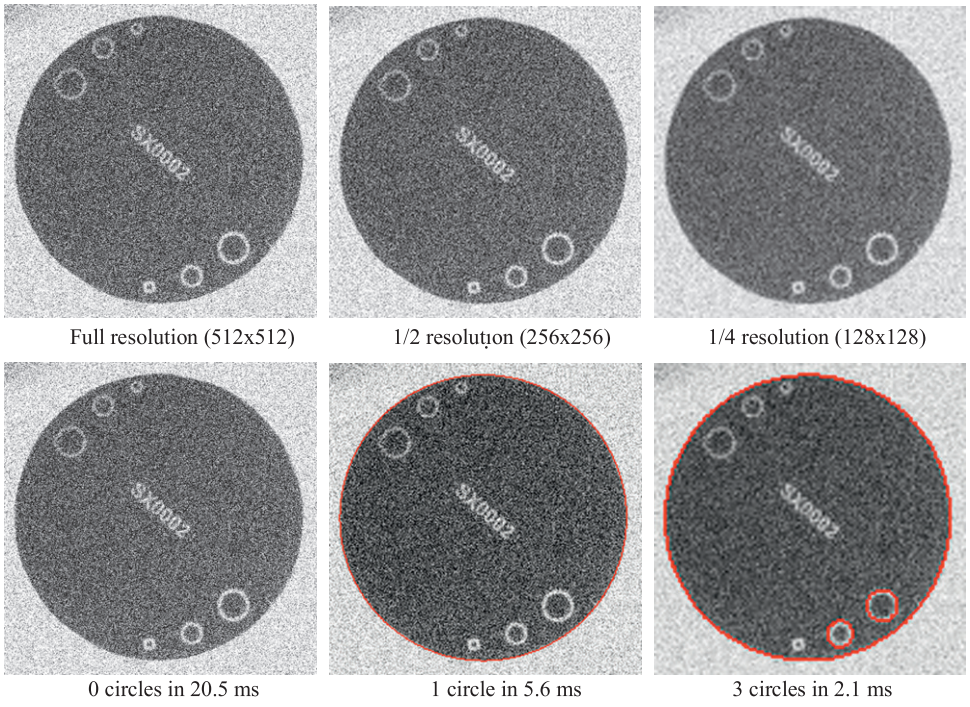


Fig. 14. The results of analyzing a very noisy image (containing a Gaussian noise with $\sigma = 100$) at different scales. At full scale, no structure is detected, but no false detections are done either. At 1/2 resolution, the big circle is detected. At 1/4 resolution, three circles are detected.

segments. The line segments are then converted into circular arcs, which are joined together using two heuristic algorithms into candidate circles and near-circular ellipses. At the final step, all candidate circles are validated using the Helmholtz principle, which eliminates false detections leaving only valid circle detections. 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) show that EDCircles have high detection rates, good accuracy, runs in blazing speed and produces only a few or no false detections. We expect EDCircles to be widely used in such real-time automation tasks as the automatic inspection of manufactured products, pupil detection, circular traffic sign detection, and similar such applications.

Acknowledgments

We are deeply indebted to anonymous reviewers for their insightful comments, which greatly helped shape this paper for the better. We also thank the Scientific and Technological Research Council of Turkey (TUBITAK) for supporting this work with the Grant no. 111E053.

References

[1] R. Gonzales, R. Woods, Digital Image Processing, Addison-Wesley, New York, 1992.
[2] D.A. Forsyth, Computer Vision: A Modern Approach, Prentice-Hall, New Jersey, 2002.
[3] L. da Fontura Costa, R. Cesar Marcondes Jr, Shape Analysis and Classification, CRC, Boca Raton, 2001.
[4] X. Hilaire, K. Tombre, Robust and accurate vectorization of line drawings, IEEE Transactions Pattern Analysis and Machine Intelligence 28 (6) (2006) 890–904.
[5] B. Lamiroy, Y. Guebbas, Robust and precise circular arc detection, Lecture Notes in Computer Science, vol. 6020, 2010, pp. 49–60.
[6] J.B. Hiley, A.H. Redekopp, R. Fazel-Rezai, A low cost human computer interface based on eye tracking, in: Proceedings of IEEE EBMS, 2006, p. 3226.
[7] Y. Ebisawa, Robust pupil detection by image difference with positional compensation, in: Proceedings of IEEE VECIMS, 2009, pp. 143–148.
[8] J.S. Agustin, H. Skovsgaard, E. Mollenbach, M. Barret, M. Tall, D.W. Hansen, J.P. Hansen, Evaluation of a low-cost open-source gaze tracker, in: Proceedings of ETRA, ACM, 2010.
[9] A. Arlicot, B. Soheilian, N. Paparoditis, Circular road sign extraction from street level images using color, shape and texture database maps, in: Proceedings of IAPRS, vol. XXXVIII, 2009, pp. 205–210.
[10] F. Larsson, M. Felsberg, Using Fourier descriptors and spatial models for traffic sign recognition, Lecture Notes in Computer Science, vol. 6688, 2011, pp. 238–249.
[11] H. Fleyeh, E. Davami, Eigen-based traffic sign recognition, IET Intelligent Transport Systems 5 (3) (2011) 190–196.
[12] P.V.C. Hough, Methods and means for recognizing complex patterns, US Patent, 3069654, 1962.
[13] J. Illingsworth, J. Kittler, A survey of the Hough transform, Computational Vision, Graphics, and Image Processing 44 (1988) 87–116.
[14] R.O. Duda, P.E. Hart, Use of the Hough transform to detect lines and curves in pictures, Communications of the ACM 15 (1972) 11–15.
[15] E.R. Davies, A modified Hough scheme for general circle location, Pattern Recognition Letters 7 (1988) 37–43.
[16] P. Kierkegaard, A method for detection of circular arcs based on the Hough transform, Machine Vision Applications 5 (1992) 249–263.
[17] J. Canny, A computational approach to edge detection, IEEE Transactions on Pattern Analysis and Machine Intelligence 8 (6) (1986) 679–698.
[18] D. Shaked, O. Yaron, N. Kiryati, Deriving stopping rules for the probabilistic Hough transform by sequential analysis, Computer Vision and Image Understanding 63 (1996) 512–526.
[19] H. Muammar, M. Nixon, Approaches to extending the Hough transform, in: Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP), vol. 3, 1989, pp. 1556–1559.
[20] L. Xu, E. Oja, P. Kultanen, A new curve detection method: randomized Hough transform, Pattern Recognition Letters 15 (5) (1990) 331–338.
[21] L. Xu, E. Oja, Randomized Hough transform (RHT): basic mechanisms, algorithms, and computational complexities, Computer Vision Graphics Image Process: Image Understanding 57 (2) (1993) 131–154.
[22] J.H. Han, L.T. Koczy, T. Poston, Fuzzy Hough transform, in: Proceedings of the International Conference on Fuzzy Systems, vol. 2, 1993, pp. 803–808.
[23] C. Kimme, D. Ballard, J. Sklansky, Finding circles by an array of accumulators, Proceedings of ACM 18 (1975) 120–122.
[24] T. Atherton, D. Kerbyson, Size invariant circle detection, Image and Vision Computing 17 (1999) 196–203.
[25] G. Schuster, A. Katsaggelos, Robust circle detection using a weighted MSE estimator, in: Proceedings of ICIP, 2004, pp. 2111–2114.
[26] T.C. Chen, K.L. Chung, An efficient randomized algorithm for detecting circles, Computer Vision and Image Understanding 83 (2001) 172–191.
[27] K. Chung, 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 190 (1) (2007) 132–149.
[28] K.L. Chung, Y.H. Huang, S.M. Shen, A.S. Krylov, D.V. Yurin, E.V. Semeikina, Efficient sampling strategy and refinement strategy for randomized circle detection, Pattern Recognition 45 (2012) 252–263.
[29] J. Yao, Fast robust genetic-algorithm based ellipse detection, in: Proceedings of ICPR, 2004, pp. 859–862.
[30] V. Ayala-Ramirez, C.H. Garcia-Capulin, A. Peres-Garcia, R.E. Sanchez-Yanez, Circle detection on images using genetic algorithms, Pattern Recognition Letters 27 (6) (2006) 652–657.
[31] S. Das, S. Dasgupta, A. Biswas, A. Abraham, Automatic circle detection on images with annealed differential evolution, in: Proceedings of IEEE International Conference on Hybrid Intelligent Systems, 2008, pp. 684–689.
[32] S. Dasgupta, A. Biswas, S. Das, A. Abraham, Automatic circle detection on digital images with an adaptive bacterial foraging algorithm, in: Proceedings of GECCO, 2008.
[33] S. Dasgupta, S. Das, A. Biswas, A. Abraham, Automatic circle detection on digital images with an adaptive bacterial foraging algorithm, Soft Computing 14 (2009) 1151–1164.
[34] E. Cuevas, D. Zaldiwar, M. Perez-Cisneros, M. Ramirez-Ortegon, Circle detection using discrete differential evolution optimization, Pattern Analysis Applications 14 (2010) 93–107.
[35] E. Cuevas, N. Ortega-Sanchez, D. Zaldiwar, M. Perez-Cisneros, Circle detection by harmony search optimization, Journal of Intelligent Robot Systems (2011).
[36] E. Cuevas, V. Osuna-Enciso, F. Wario, D. Zaldiwar, M. Perez-Cisneros, Automatic multiple circle detection based on artificial immune systems, Expert Systems with Applications 39 (2012) 713–722.
[37] I. Frosio, N.A. Borghese, Real-time accurate circle fitting with occlusions, Pattern Recognition 41 (3) (2008) 1041–1055.
[38] S.C. Zhang, Z.Q. Liu, A robust, real-time ellipse detector, Pattern Recognition 38 (2) (2005) 273–287.
[39] Z.Y. Liu, H. Qiao, Multiple ellipses detection in noisy environments: a hierarchical approach, Pattern Recognition 42 (11) (2009) 2421–2433.
[40] D.K. Prasad, M.K.H. Leung, S.Y. Cho, Edge curvature and convexity based ellipse detection method, Pattern Recognition 45 (9) (2012) 3204–3221.
[41] J. Wu, J. Li, C. Xiao, F. Tan, C. Gu, Real-time robust algorithm for circle object detection, in: Proceedings of IEEE Conference for Young Computer Scientists, 2008, pp. 1722–1727.
[42] D.N. Vizireanu, Generalizations of binary morphological shape decomposition, Journal of Electronic Imaging 16 (1) (2007) 1–6 01302.
[43] D.N. Vizireanu, Morphological shape decomposition interframe interpolation method, Journal of Electronic Imaging 17 (1) (2008) 1–5 013007.
[44] D.N. Vizireanu, S. Halunga, G. Marghescu, Morphological shape decomposition interframe interpolation method, Journal of Electronic Imaging 19 (2) (2010) 1–3 023018.
[45] V. Patraucean, P. Gurdjos, G. Morin, J. Conter, Detection de primitives lineaires et circulaires par une approche a contrario, in: Proceedings of the 13th Congres des jeunes chercheurs en vision par ordinateur (ORASIS), ⟨http://hal.inria.fr/inria-00595749/en/⟩, 2011.
[46] V. Patraucean, Detection and Identification of Elliptical Structure Arrangements in Images: Theory and Algorithms, Ph.D. Thesis, ⟨http://ethesis.inp-toulouse.fr/archive/00001847⟩, 2012.
[47] V. Patraucean, P. Gurdjos, J. Conter. Bubble tag-based system for object authentication, in: Proceedings of the 8th International Conference on Communications, 2010, pp. 133–136.
[48] C. Topal, C. Akinlar, Y. Genc, Edge drawing: a heuristic approach to robust real-time edge detection, in: The Twentieth International Conference on Pattern Recognition (ICPR), Istanbul, Turkey, August 23–26, 2010, pp. 2424–2427.
[49] O. Ozsen, C. Topal, C. Akinlar, Real-time edge segment detection with edge drawing algorithm, in: Proceedings of the International Symposium on Image and Signal Processing and Analysis (ISPA), Dubrovnik, Crotia, September 2011.
[50] C. Topal, C. Akinlar, Edge drawing: a combined real-time edge and segment detector, Journal of Visual Communication and Image Representation 23 (6) (2012) 862–872.
[51] Edge drawing web site. ⟨http://ceng.anadolu.edu.tr/CV/EdgeDrawing⟩.
[52] C. Akinlar, C. Topal, EDPF: a parameter-free edge segment detector with a false detection control, International Journal of Pattern Recognition and Artificial Intelligence 23 (6) (2012) 862–872.
[53] EDPF web site. ⟨http://ceng.anadolu.edu.tr/CV/EDPF⟩.
[54] C. Akinlar, C. Topal, EDCircles: real-time circle detection by edge drawing (ED), in: Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP), Kyoto, Japan, 2012.
[55] EDCircles web site. ⟨http://ceng.anadolu.edu.tr/CV/EDCircles⟩.
[56] C. Akinlar, C. Topal, EDLines: a real-time line segment detector with a false detection control, Pattern Recognition Letters 32 (13) (2011) 1633–1642.
[57] EDLines web site. ⟨http://ceng.anadolu.edu.tr/CV/EDLines⟩.
[58] A. Desolneux, L. Moisan, J.M. Morel, Edge detection by Helmholtz principle, Journal of Mathematical Imaging and Vision 14 (May (3)) (2001) 271–284.
[59] A. Desolneux, L. Moisan, J.M. Morel, Seeing, Thinking and Knowing, Gestalt Theory and Computer Vision, Kluwer Academic Publishers, 2004, pp. 71–101.
[60] A. Desolneux, L. Moisan, J.M. Morel, From Gestalt Theory to Image Analysis: A Probabilistic Approach, Springer, 2008.
[61] A. Desolneux, L. Moisan, J.M. Morel, Meaningful alignments, International Journal of Computer Vision 40 (1) (2000) 7–23.
[62] R. Grompone von Gioi, J. Jakubowicz, J.M. Morel, G. Randall, On straight line segment detection, Journal of Mathematics Imaging and Vision 32 (November (3)) (2008) 313–347.
[63] R. Grompone von Gioi, J. Jakubowicz, J.M. Morel, G. Randall, LSD: a fast line segment detector with a false detection control, IEEE Transactions on Pattern Analysis and Machine Intelligence 32 (April (4)) (2010) 722–732.
[64] R. Bullock, Least squares circle fit, ⟨www.dtcenter.org/met/users/docs/write_ups/circle_fit.pdf⟩, 2006.
[65] A. Fitzgibbon, M. Pilu, R.B. Fisher, Direct least square fitting of ellipses, IEEE Transactions Pattern Analysis and Machine Intelligence 21 (May (5)) (1999).
[66] OpenCV 2.1, ⟨http://opencv.willowgarage.com⟩.
[67] Sweedish Traffic Signs web site. ⟨http://users.du.se/hfl/traffic_signs/⟩.
[68] Pyscophysics, ⟨http://en.wikipedia.org/wiki/Psychophysics⟩.

Cuneyt Akinlar received his B.Sc. degree in Computer Engineering from Bilkent University in 1994; M.Sc. and Ph.D. degrees in Computer Science from the University of Maryland, College Park in 1997 and 2001, respectively. He is currently an Assistant Professor in Anadolu University, Computer Engineering Department (AU-CENG). Before joining AU-CENG, he worked at Panasonic Technologies, Princeton, NJ, and Siemens Corporate Research, Princeton, NJ as student intern and research scientist between 1999 and 2003. His current research interests include real-time image processing and computer vision algorithms, high-performance computing, storage systems and computer networks.

Cihan Topal received the B.Sc. degree in Electrical Engineering and M.S. degree in Computer Engineering, both from Anadolu University, in 2005 and 2008 respectively. He worked as a student intern at Siemens Corporate Research, Princeton, New Jersey. He is currently working at Anadolu University Computer Engineering Department, Turkey, towards his Ph.D. degree. His research interests are in the area of image processing and computer vision applications, and pattern recognition.