Pattern Recognition Letters 32 (2011) 1633–1642

Contents lists available at ScienceDirect

Pattern Recognition Letters

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

EDLines: A real-time line segment detector with a false detection control

Cuneyt Akinlar [*], Cihan Topal

Computer Engineering Department, Anadolu University, Eskisehir, Turkey

ARTICLE INFO

Article history:
Received 8 February 2011
Available online 7 June 2011
Communicated by A. Fernandez-Caballero

Keywords:
Real-time line segment detection
Edge Drawing algorithm
LSD
Helmholtz principle
NFA

ABSTRACT

We propose a linear time line segment detector that gives accurate results, requires no parameter tuning, and runs up to 11 times faster than the fastest known line segment detector in the literature; namely, the line segment detector (LSD) by Grompone von Gioi et al. The proposed algorithm makes use of the clean, contiguous (connected) chain of edge pixels produced by our novel edge detector, the *Edge Drawing* (*ED*) algorithm; hence the name *EDLines*. The detector includes a line validation step due to the Helmholtz principle, which lets it control the number of false detections. With its accurate results and blazing speed, *EDLines* will be very suitable for the next generation real-time computer vision and image processing applications.

1. Introduction

Line segment detection is an important and recurring problem in image processing and computer vision, and has many applications. Not only line segments give a high-level description of the objects in an image, they are also helpful in such applications as image compression (Franti et al., 1998), crack detection in materials (Mahadevan and Casasent, 2001), stereo-matching (Ji and Zhang, 1988), robot-navigation (Kahn et al., 1987, 1990), roadbed detection (Tupin et al., 1998), processing of sports videos and satellite pictures (Yu et al., 2005), and many others (Zheng et al., 2005; Zhu et al., 2001).

An ideal line segment detection algorithm would process any image regardless of its origin, orientation or size, and produce robust and accurate line segments, i.e., no false detections, in a very short amount of time, preferably in real-time. Furthermore, such an algorithm would need no parameter tuning at all, running with a single set of default parameters for all types of images. The line segment detection algorithm outlined in this paper, named *EDLines* (Akinlar and Topal, 2011), satisfies all of these requirements.

Traditional line segment detection algorithms start by computing an edge map, typically by the famous Canny edge detector (Canny, 1986). The next step is then to apply the Hough transform (Hough, 1962; Illinworth and Kittler, 1988; Kalviainen et al., 1996), and finally extract all lines that contain a certain number of edge points. Lines are broken into segments using gap and length thresholds (Duda and Hart, 1972). These methods are generally slow, and they usually combine non-contiguous line segments together producing a lot of false detections.

There are many variants of the Hough transform (HT), e.g., Randomized Hough transform (Xu et al., 1990), Progressive Probabilistic Hough transform (Kiryati et al., 1991; Matas et al., 2000; Galambos et al., 2001), Elliptical Gaussian kernel-based Hough transform (Fernandes and Oliveira, 2008; KHT software package), and many others (Princen et al., 1990; Gorman and Clowes, 1976; Lam et al., 1994; Kang et al., 2007; Chao et al., 2009), each trying to remedy different shortcomings of the standard Hough transform. Among these techniques, (Fernandes and Oliveira, 2008; KHT software package) stands out by presenting an efficient voting scheme for the HT procedure. Given a binary edge map, KHT achieves real-time performance in software while being robust to the detection of spurious lines. KHT also has a default set of parameters that can be used for all edge maps. However, all Hough transform based techniques require a binary edge map as input, and they usually generate infinitely long lines – rather than line segments – in (angle, radius) representation, which must then be broken down to line segments. A clean input edge map is very critical for these techniques, but the parameters to be used for edge map generation is not automatic and have to be determined by the user. Fig. 1 shows the line segments produced by the OpenCV's Hough transform method, i.e., `cvHoughLines2` function. To obtain this result, different thresholds have been set by trial-and-error during Canny edge detection and line extraction, and the best result have been given. Although some of the long line segments detected on the house are accurate, there are many irrelevant line segment detections, especially around anisotropic structures such as the bushes, grass, and the trees. The Hough transform result produced by KHT software package, although much faster than OpenCV's `cvHoughLines2`, is not given in the paper as KHT produces infinitely long lines rather than line

* Corresponding author. Tel.: +90 222 321 3550x6553; fax: +90 222 323 9501.
E-mail addresses: cakinlar@anadolu.edu.tr (C. Akinlar), cihant@anadolu.edu.tr (C. Topal).

0167-8655/$ - see front matter © 2011 Elsevier B.V. All rights reserved.
doi:10.1016/j.patrec.2011.06.001

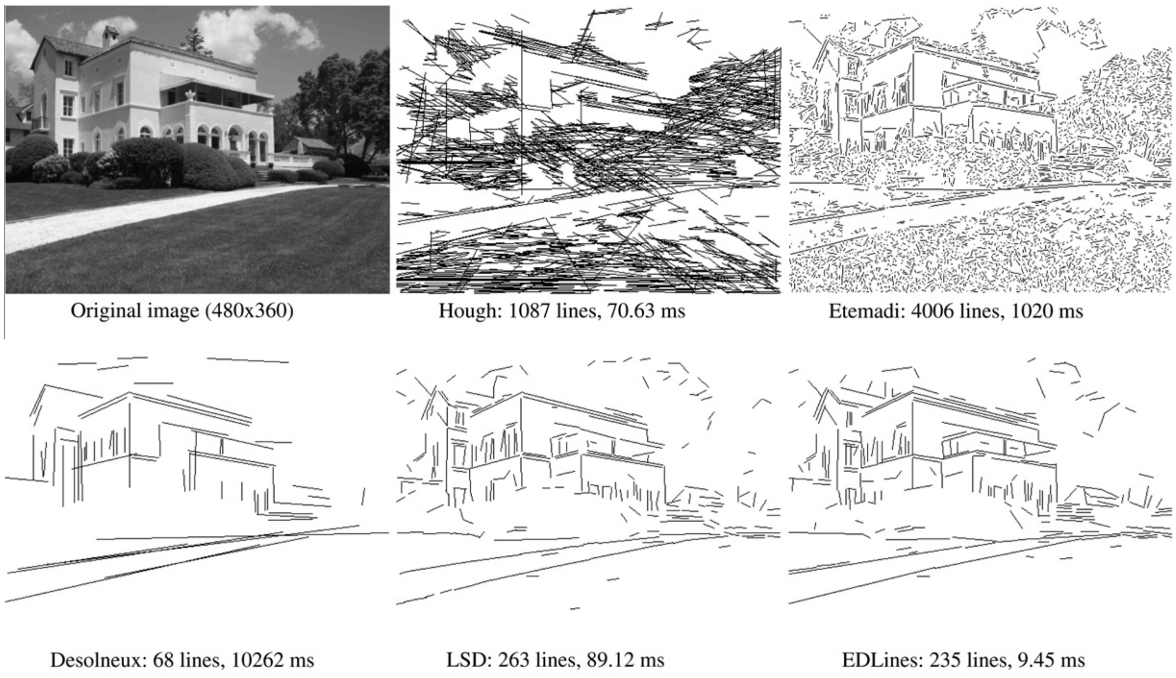
Original image (480x360)
Hough: 1087 lines, 70.63 ms
Etemadi: 4006 lines, 1020 ms
Desolneux: 68 lines, 10262 ms
LSD: 263 lines, 89.12 ms
EDLines: 235 lines, 9.45 ms

Fig. 1. Line segment detection results from various detectors. The processing times have been obtained in a PC with 2.2 GHz Intel E4500 CPU and 2 GB RAM.

segments, which must be broken down to line segments by a post-processing step. For the record, KHT software package takes only 16.5 ms (5 ms for Canny edge map generation plus 11.5 ms for line extraction) for the house image in Fig. 1, and outputs 360 lines.

Instead of using the Hough transform, Chan and Yip (1996) and Etemadi (1992) propose different techniques to extract line segments after the computation of the edge map. Specifically, Etemadi's method (Etemadi, 1992) is based on the idea of generating chains of pixels from a given edge map, and then extracting line segments and arcs by walking over these chains. Although the line segment and arc extraction process is parameterless, edge detection still requires parameters that are left to the user. This method produces well-localized line segments and arcs, but it generates too many segments with a lot of false positives especially in images with noisy backgrounds or those containing complex structures such as trees, grass, cloudy sky, or similar anisotropic structures. Fig. 1 shows Etemadi's result with only line segments displayed.

Alternative to using the binary edge map for line segment detection, there are also techniques that work by just making use of the pixels' gradient orientations (Burns et al., 1986; Beveridge et al., 1996; Desolneux et al., 2000, 2008; Grompone von Gioi et al., 2008a,b, 2010). The first such proposal is due to Burns et al. (1986) and Beveridge et al. (1996), which computes the level line orientation at each pixel and then generates line segments by combining pixels having the same orientation. Starting at a pixel, they look at the level line angles of the neighboring pixels, and combine them if they have the same orientation. Although this work laid the ground for some of the recent line detectors, it produces results similar Etemadi's method generating too many line segments with a lot of false positives.

Desolneux et al. (2000, 2008) used Burn's idea of utilizing the pixel orientation, and proposed a parameterless line detector that controls the number of false positives. Their idea is to count the number of aligned pixels in a certain orientation and accept the set of pixels as a line segment if the observed structure is perceptually meaningful. This is called the Helmholtz principle from Gestalt theory (Desolneux et al., 2008), and is used as the line validation method. Although this method produces no false positives as promised, many of the valid short line segments are missing (see Desolneux et al.'s result in Fig. 1). The method is also very global in nature generating very long lines which in fact should be broken down into several smaller line segments (observe that the side of the windows on two different floors have been combined). Furthermore, the algorithm is very computationally intensive, thus very slow (it takes more than 10 s on the given image).

By extending Burns's work for line segment generation and combining it with Desolneux's line validation method due to the Helmholtz principle (Desolneux et al., 2008), Grompone von Gioi et al. (2008a,b, 2010) has recently proposed a parameterless line detection algorithm, called the line segment detector (LSD), that produces accurate line segments, and also controls the number of false detections. Although LSD produces good results for most types of images (see LSD's result in Fig. 1), it fails especially in images where the background contains some white noise, and its running time is still prohibitive; which makes it unsuitable for the next-generation real-time applications.

In this paper, we propose a fast, parameterless line segment detector, named *EDLines* (Akinlar and Topal, 2011), that produces robust and accurate results, and runs up to 11 times faster than the fastest known line segment detector; namely, the LSD by Grompone von Gioi et al. (2008a,b, 2010). Our detector also includes a line validation step due to the Helmholtz principle (Desolneux et al., 2008), which lets it control the number of false detections. Looking at the result obtained by EDLines, we see that it is very similar to that of LSD with all major line segments detected, and has very few false positives. Furthermore, EDLines runs real-time at a blazing speed of 9.45 ms, about 10 times faster than LSD for the given image.

EDLines is comprised of three steps: (1) Given a grayscale image, we first run our fast, novel edge detector, the *Edge Drawing* (*ED*) algorithm (Topal et al., 2010; Topal and Akinlar, submitted for publication; Edge Drawing Web Site), which produces a set of clean, contiguous chains of pixels, which we call *edge segments*. Edge segments intuitively correspond to object boundaries. (2) Next, we extract line segments from the generated pixel chains by means of a straightness criterion, i.e., by the Least Squares Line Fitting Method. (3) Finally, a line validation step due to the Helmholtz principle (Desolneux et al., 2008; Grompone von Gioi et al., 2008a) is used to eliminate false line segment detections.

The rest of the paper is organized as follows: In Section 2, we briefly describe our edge detector, the Edge Drawing (ED) algorithm (Topal et al., 2010; Topal and Akinlar, submitted for publication; Edge Drawing Web Site). Section 3 describes how we fit lines to pixel chains produced by ED. Section 4 describes our line validation step. Section 5 discusses the internal default parameters of the algorithm, which are fixed and are never changed for different images. Section 6 compares the performance of EDLines to others in the literature through experimentation, and Section 7 concludes the paper.

2. Edge detection by Edge Drawing

Edge Drawing (ED) is our recently-proposed, novel, fast edge detection algorithm (Topal et al., 2010; Topal and Akinlar, submitted for publication; Edge Drawing Web Site). What makes ED stand out from the existing edge detectors, e.g., Canny (1986), is the following: While the other edge detectors give out a binary edge image as output, where the detected edge pixels are usually independent, disjoint, discontinuous entities; ED produces a set of edge segments, which are clean, contiguous, i.e., connected, chains of edge pixels. Thus, while the outputs of other edge detectors require further processing to generate potential object boundaries, which may not even be possible or result in inaccuracies; ED not only produces perfectly connected object boundaries by default, but it also achieves this in blazing speed compared to other edge detector (Topal et al., 2010).

Given a grayscale image, ED performs edge detection in four steps:

(1) The image is first passed through a filter, e.g., Gauss, to suppress noise and smooth out the image. We use a 5×5 Gaussian kernel with $\sigma = 1$ by default.
(2) The next step is to compute the gradient magnitude and direction at each pixel of the smoothed image. Any of the known gradient operators, e.g., Prewitt, Sobel, Scharr, etc., can be used at this step.
(3) In the third step, we compute a set of pixels, called the *anchors*, which are pixels with a very high probability of being edge elements (*edgels*). The anchors correspond to pixels where the gradient operator produces maximal values, i.e., the peaks of the gradient map.
(4) Finally, we connect the anchors computed in the third step by drawing edges between them; hence the name *Edge Drawing* (*ED*). The whole process is similar to children's boundary completion puzzles, where a child is given a dotted boundary of an object, and s/he is asked to complete the boundary by connecting the dots. Starting from an anchor (dot), ED makes use of the neighboring pixels' gradient magnitudes and directions, and walks to the next anchor by going over the gradient maximas. If you visualize the gradient map as a mountain in 3D, this is very much like walking over the mountain top from one peak to the other.

Fig. 2 shows ED in action on a 128×128 pixels grayscale image. Fig. 2(b) shows the gradient map, where white pixels correspond to high gradient values (mountain tops). Fig. 2(c) shows an example set of anchors, which correspond to peaks of the gradient map, and clearly depict the boundaries of the rectangles in the image. The final edge map, shown in Fig. 2(d), is obtained by linking the anchors (dots) (refer to Topal et al. (2010) for a detailed description of how ED links the anchors to obtain the final edge map). As mentioned before, ED not only produces a binary edge map similar to other edge detectors, but it also produces a set of edge segments, which are connected chains of pixels corresponding to object boundaries. In the given example, ED generates four edge segments, one for the boundary of each rectangle. Given these edge segments, all that remains for line segment extraction is to go over these pixel chains, and fit lines to the pixels. Next section describes how this is done.

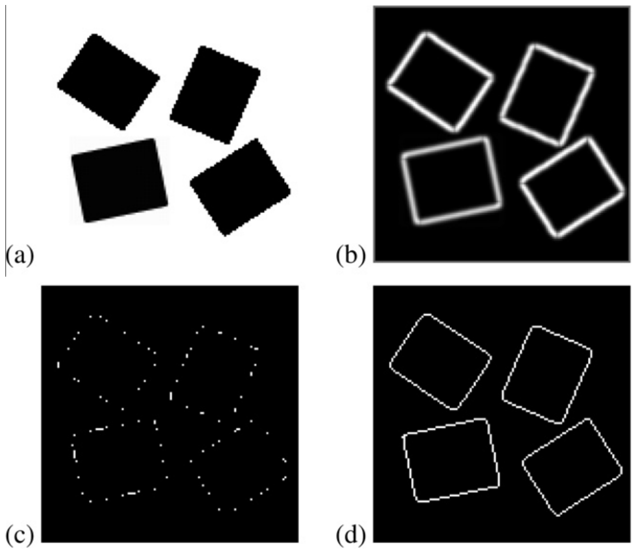

Fig. 2. (a) A grayscale image containing four rectangles, (b) gradient map, (c) anchor points, (d) final edge map.

3. Line segment extraction

Given an edge segment comprised of a contiguous chain of edge pixels, the goal of this step is to split this chain into one or more straight line segments. The basic idea is to walk over the pixels in sequence, and fit lines to the pixels using the Least Squares Line Fitting Method until the error exceeds a certain threshold, e.g., 1 pixel error. When the error exceeds this threshold, we generate a new line segment. The algorithm then recursively processes the remaining pixels of the chain until all pixels are processed.

Table 1 shows the algorithm used to extract line segments from a chain of pixels. The idea is to generate a minimal length initial line segment by the least squares line fitting method (the first while loop), and then extend this line segment by adding more pixels to it (the second while loop). The value of the minimum line length depends on the line validation parameters, and is explained in Section 5.4. After the detection of a minimal length initial line segment, we simply walk over the remaining pixels of the chain and compute the distance of each pixel to the currently fitted line. We add pixels to the current line for as long as the pixels are within a certain distance from the current line, e.g., 1 pixel error. Intuitively, we would continue adding pixels to the current line segment until we turn a corner and the direction of the line changes. At that point, we output the current line segment. The remaining pixels of the chain are then processed recursively to extract further line segments.

4. Line validation

Similar to Desolneux et al. (2000) and LSD (Grompone von Gioi et al., 2008b, 2010), our line validation method is based on the Helmholtz principle, which basically states that for a structure to

Table 1
Algorithm to extract line segments from a pixel chain.

```
LineFit(Pixel *pixelChain, int noPixels){
double lineFitError = INFINITY; // current line fit error
LineEquation lineEquation; // y = ax + b OR x = ay + b

while (noPixels > MIN_LINE_LENGTH){
  LeastSquaresLineFit(pixelChain, MIN_LINE_LENGTH, &lineEquation,
   &lineFitError);
  if (lineFitError <= 1.0) break; // OK. An initial line segment detected
  pixelChain ++; // Skip the first pixel & try with the remaining pixels
  noPixels–; // One less pixel
} // end-while

if (lineFitError > 1.0) return; // no initial line segment. Done.

// An initial line segment detected. Try to extend this line segment
int lineLen = MIN_LINE_LENGTH;
while (lineLen < noPixels){
  double d = ComputePointDistance2Line(lineEquation,
   pixelChain[lineLen]);
  if (d > 1.0) break;
  lineLen++;
} //end-while

// End of the current line segment. Compute the final line equation & output it.
LeastSquaresLineFit(pixelChain, lineLen, &lineEquation);
Output "lineEquation"

// Extract line segments from the remaining pixels
LineFit(pixelChain + lineLen, noPixels-lineLen);
} //end-LineFit
```

be perceptually meaningful, the expectation of this structure (grouping or Gestalt) by chance must be very low (Desolneux et al., 2008). This is an "*a contrario*" approach, where the objects are detected as outliers of the background model. As shown by Desolneux et al. (2000), a suitable background model is one in which all pixels (thus the gradient angles) are independent. They show that the simplest such model is the Gaussian white noise. To make validation by the Helmholtz principle concrete, Desolneux defines what is called the "*Number of False Alarms* (*NFA*)" of a line segment as follows (Desolneux et al., 2000): Let A be a segment of length "*n*" with at least "*k*" points having their directions aligned with the direction of A in an image of size $N \times N$ pixels. Define NFA of *A* as:

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

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, and each end point 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 line direction, and its value will be discussed in Section 5.

An event (a line segment in this case) is called ε-meaningful if its $NFA(n,k) \leqslant \varepsilon$. Desolneux et al. (2000) advises setting ε to 1, which corresponds to one false detection per image. Given these definitions, we validate our line segments as follows: For a line segment of length "*n*", we compute the gradient angle of each pixel along the line segment and count the number of *aligned* pixels "*k*". We then compute $NFA(n, k)$, and accept the line segment as valid if $NFA(n,k) \leqslant 1$. Otherwise the line 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. We would also like to note that line validation is an optional last step in EDLines, and can be omitted if deemed unnecessary. We observed that line validation rejects only short line segments. Long lines are big deviations from the background, and are never rejected. Therefore, an application that would make use of only long lines, e.g., stereo matching, can skip the line validation via the Helmholtz principle altogether, and use a line validation method based on a principle that could be tested faster, e.g., pick lines longer than a certain threshold. This would further speed up EDLines.

5. Internal parameters

There are several internal parameters used by EDLines. Below, we describe these parameters and explain how and why they are set the way they are.

5.1. Gradient magnitude and direction computation

Recall from Section 2 that the first step in our Edge Drawing (ED) algorithm is the computation of the image gradient. To compute the gradient magnitude and the level line angle at pixel (x, y), we follow Burns et al. (1986) and Desolneux et al. (2008), and LSD (Grompone von Gioi et al., 2010; LSD), and use a 2×2 mask as follows:

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

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

$$g(x,y) = \sqrt{g_x(x,y)^2 + g_y(x,y)^2}, \quad angle(x,y) = arctan\left(\frac{gx(x,y)}{-gy(x,y)}\right)$$

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 Grompone von Gioi et al. (2010) 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; because, pixel independence is required by the "*a contrario*" validation due to Helmholtz principle.

5.2. Line validation parameters

Recall from Section 4 that our line validation is based on the length of the line segment and the number of aligned pixels on the line. To apply this definition, we need to define what "*aligned*" means. We borrow that definition from Desolneux et al. (2000): Two points (or line segments) *P* and *Q* have the same direction, i.e., *aligned*, with precision $1/n$ if $angle(P)$ and $angle(Q)$ are within π/n degrees of each other. Desolneux also states that "in agreement with Pyschophysics and numerical experimentation, '*n*' should not exceed 8". Burns et al. (1986) and LSD (Grompone von Gioi et al., 2010) have also arrived at the same conclusion. Therefore, we assume that two points (or a point and a line segment) are *aligned* if their angles are within $\pi/8 = 22.5°$ of each other. This corresponds to 8 different angle orientations and a precision "$p = 1/8 = 0.125$". This is the probability used in the NFA computation.

5.3. Gradient threshold, anchor threshold and scan interval

After the gradient computation in ED, pixels with gradient values less than a certain threshold "ρ" are ignored during anchors' computation and linking. This lets us eliminate pixels which definitely may not contain edge elements (edgels), and therefore no line segments. Following the discussion in Grompone von Gioi et al. (2010), we set "ρ" so as to leave out points where angle error is larger than the angle tolerance. With maximum quantization error between two consecutive pixels being equal to 2 (occurs when adjacent pixels get errors of +1 and -1), and an angle tolerance of

22.5° (refer to Section 5.2), we compute the gradient threshold by the following formula given in Grompone von Gioi et al. (2010):

$$\rho = \frac{2}{\sin(22.5)} = 5.22$$

Going over the gradient map and eliminating pixels with gradient values less than "ρ", we get what we call the "*edge areas*" of the image. These are pixels where an edgel may be located. Furthermore, none of the eliminated pixels may contain an edgel.

ED has two more parameters: *Anchor Threshold* and *Scan Interval* (called *Detail Ratio* in Topal et al. (2010)). As explained in Section 2, after the computation of the thresholded gradient cluster, called the *edge areas* (Fig. 2(b)), ED computes a set of anchors (Fig. 2(c)), which are linked using a smart routing procedure to obtain the final edge segments (Fig. 2(d)) (Topal et al., 2010). Currently, ED performs anchor computation by non-maximal suppression with a threshold, called an *Anchor Threshold*. While classical non-maximal suppression considers a pixel to be an edgel candidate if the gradient value at the pixel is bigger than or equal to both of the pixel's neighbors in the gradient direction, ED considers a pixel to be an anchor if its gradient value is bigger than both of its neighbors in the gradient direction by an amount of *Anchor Threshold*. The reason for having a threshold for anchor computation is to choose quality anchors, as that would result in high-quality edge segments. There is no theoretical basis for setting an optimal value for *Anchor Threshold*; the choice is purely empirical. Therefore, after many experiments on many different types of images in EDLines, we concluded that an *Anchor Threshold* of 3 results in good quality edge segments, and thus, good quality line segments. ED's other parameter, *Scan Interval*, is used to select anchors at different row/column intervals. The goal of this parameter is to adjust the number of anchors, and thus, the amount of detail in the final edge map. With a *Scan Interval* of "*k*", ED scans every *k*th row/column for anchors. Thus, a *Scan Interval* of 1 means, ED scans every row/column, i.e., every pixel; a *Scan Interval* of 4 means, ED scans every 4th row/column. As the number of anchors decrease, so are the details in the final edge map and vice versa. Since we want to detect all line segments in a given image, we want ED to produce an edge map with all details in the image; therefore, we set *Scan Interval* to 1 in EDLines.

5.4. Line fit parameters

There are two parameters associated with line fitting to a chain of pixels: (1) Minimum line length and (2) maximum mean square line fit error.

Recall from Section 4 that a line segment of length "*n*" with "*k*" aligned pixels is valid if $NFA(n,k) \leqslant 1$. For the minimum length line to be valid, "*k*" must be equal to "*n*". So, we get $NFA(n,n) = N^4 * P^n \leqslant 1$. Extracting the minimum line length "*n*" from this formula, we get: $n \geqslant \frac{-4log(N)}{log(p)}$, where $p = 0.125$. As an example, for a 512 × 512 image, the minimum line length turns out to be 12 pixels. Minimum line length is computed using this formula at the beginning of the line fitting step, and we do not detect line segments shorter than the shortest valid line.

Maximum mean square line fit error is the other parameter used by the line fitting step of EDLines (refer to the algorithm in Table 1). As we extend a line segment, we continuously compute the distance of a pixel to the currently fitted line. If the error exceeds a certain threshold, we output the current line segment, and start a new one. In our algorithm, we fix this parameter as 1 pixel error based on many experiments on different sets of images.

6. Experiments

We start the experiments by evaluating Edge Drawing (ED)'s edge maps and their effects on the final extracted line segments.

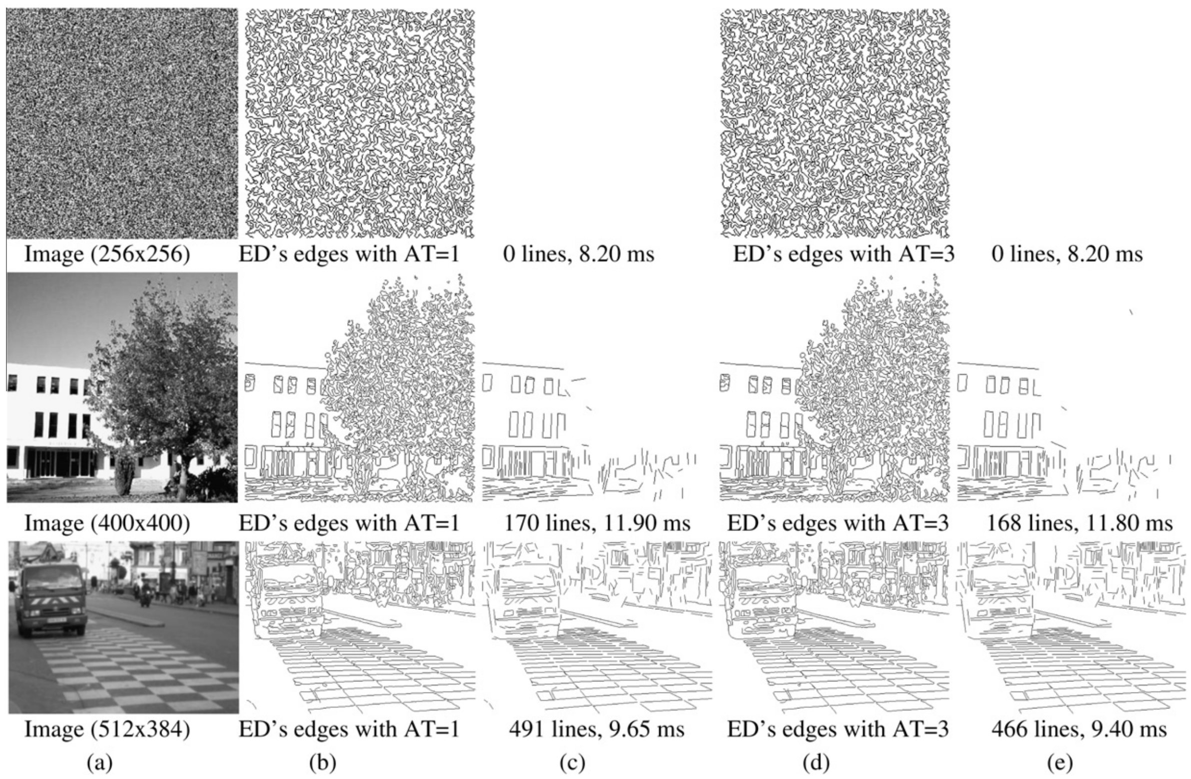

Fig. 3. (a) Original images, (b) and (c) ED's edge maps with *Anchor Threshold* = 1, and the line segments extracted from these edge maps, (d) and (e) ED's edge maps with *Anchor Threshold* = 3, and the line segments extracted from these edge maps. The processing times have been obtained in a PC with 2.2 GHz Intel E4500 CPU and 2 GB RAM.

In all EDLines experiments, images were first smoothed by a 5×5 Gaussian kernel with $\sigma = 1$, and EDLines were run with other fixed internal parameters described in Section 5. Recall from Section 5.3 that ED's *Gradient Threshold* is theoretically computed, but ED's *Anchor Threshold* is an empirical parameter that needs to be determined by experimentation. Fig. 3 shows three images, ED's edge maps with two different *Anchor Thresholds*, 1 and 3 respectively, and the line segments extracted from these edge maps. The first row shows a Gaussian white noise image. According to the Helmholtz principle, no line segments must be detected in this image. We see from the edge maps that ED produces a lot of garbage-like edge segments, which leads to many short line segments (not shown in the figure); but all extracted line segments are eliminated during the validation step leaving zero line segment detections as should be the case. The second row shows a house with a tree. Again, ED's edge maps contain many garbage-like formations around the tree, which are almost completely eliminated during the validation step. The third row shows a truck going over a checkered road. EDLines seems to detect all meaningful line segments with both edge maps. Also notice the real-time behavior of EDLines with all images. We observe from both the second and third images that more line segments are detected with an *Anchor Threshold* of 1. This is expected since a smaller *Anchor Threshold* would lead to more anchors and more detailed edge maps, but the edge maps would only differ in very minute details. Both edge maps would lead to the same long line segments; the difference would only be in the very short line segments. Choosing a larger value for *Anchor Threshold* would further reduce the details in ED's edge maps, but this has the risk of missing some important details especially on boundaries having smooth transitions. If no anchors are detected on such boundaries, no edge segments would be detected; thus no line segments would be detected. With all these considerations, we fixed ED's *Anchor Threshold* to be 3 based on experiments on many different sets of images, and will use this value in the rest of the experiments.

We now compare the performance of EDLines (EDLines Web Site) to other line extraction algorithms in the literature. Fig. 4 shows the line segment extraction results from various detectors for a sample challenging image. The processing times have been obtained in a PC with 2.2 GHz Intel E4500 CPU and 2 GB RAM.

To obtain the Hough transform result, we first generated a binary edge map using `cvCanny` function in OpenCV by setting the low threshold to 40 and the high threshold to 100. We then fed the resulting edge map into `cvHoughLines2` function with Hough method set to `CV_HOUGH_PROBABILISTIC`, a line accumulator threshold of 40, and a minimum line length threshold of 10. The threshold values for edge detection and line fitting have been determined by trial-and-error to obtain the perceptually best outcome. Looking at the result, we see that there are some accurately detected line segments, but there are also many irrelevant line segment detections, especially around anisotropic structures such as the carpet, hair, and pasta. We should also note that OpenCV's implementation of the probabilistic Hough transform is surprisingly fast.

To obtain Etemadi's result, we used the same edge map obtained by `cvCanny`, and fed this to Etemadi's original line and arc extraction algorithm implemented in ORT-2.3 software package. Here, only detected line segments are displayed. Looking at the result, we see that Etemadi's method produces too many line segments with a lot of false positives especially around anisotropic parts of the image (similar to the Hough transform), and requires quite a long time to execute.

To obtain Desolneux et al.'s result, we used the `falign_mdl` utility included in the freely distributed image processing framework MegaWave2). Since this method is parameterless, we ran it with the default parameters. Looking at the result, we see that Desolneux et al.'s method has no false positives as promised. But it misses on some of the valid short line segments, and it is very global in nature generating very long lines which in fact should be broken down into several smaller line segments. Moreover, this

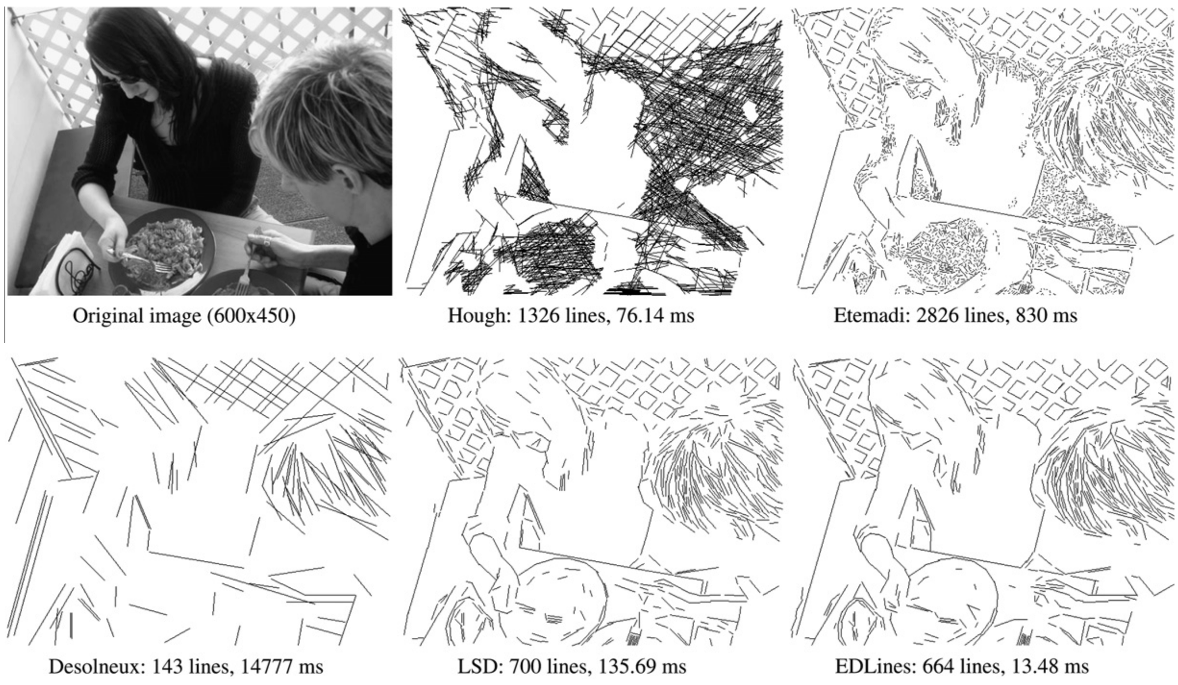
Original image (600x450)
Hough: 1326 lines, 76.14 ms
Etemadi: 2826 lines, 830 ms
Desolneux: 143 lines, 14777 ms
LSD: 700 lines, 135.69 ms
EDLines: 664 lines, 13.48 ms

Fig. 4. Line segment detection results from various detectors. The processing times have been obtained in a PC with 2.2 GHz Intel E4500 CPU and 2 GB RAM. Hough and Etemadi have many false detections. Desolneux has no false detections, but is too global in nature producing very long lines, which in fact should be broken down into several line segments. LSD produces good results with very few false positives, but its running time is still prohibitive especially for real-time applications. EDLines produce very similar results to LSD, but runs 10 times faster for this particular image.

method is very computationally intensive taking more than 14 s on the sample image.

To obtain the result produced by LSD, we downloaded the freely distributed LSD package from IPOL (LSD). Since LSD is parameterless, we ran it with the default parameters. We have to note that by default LSD scales the image width and height by 0.8; that is, it reduces the original image to 64% of its size before processing. The results for LSD given in this paper are based on this 0.8 scaling parameter. Without any scaling, LSD takes much longer to execute, and produces many more (especially short) line segments. Looking at the result, we see that LSD produces good results with very few false positives, but its running time is still prohibitive especially for real-time applications. Furthermore, LSD starts failing in images containing some white noise as noted in Grompone von Gioi et al. (2010) and as we show below.

At the end of Fig. 4, the result produced by EDLines is shown. Looking at the result, we see that it is very similar to that of LSD with all major line segments detected, and has very few false positives. Unlike LSD however, EDLines runs real-time at a blazing speed of 13.48 ms, about 10 times faster than LSD for the given sample image.

Fig. 5 shows the results produced by EDLines for several natural images. Images on the left column have geometrical structures, e.g., squares, rectangles and other polygons; and images on the right column have non-geometrical structures. See that, EDLines extracts the structure of the image in each case with very few false positives. Curved objects such as the human head or the bicycle wheel have been approximated by the concatenation of a few line segments.

Table 2 shows the dissection of the running times of EDLines, LSD (Grompone von Gioi et al., 2010) and KHT (Fernandes and Oliveira, 2008) on seven images given in Figs. 1 and 5 in increasing order of image size. In the case of KHT, the total time includes only the time taken by KHT given a binary edge map; it does not include the time taken to compute the binary edge map, nor does it include the time it would take to break down KHT's infinitely long lines in (angle, radius) representation to line segments by a post-processing step. We see that EDLines and LSD produce very similar results in each case, but EDLines runs at least 7, up to 11 times faster than LSD. We also see that about 65% of the total running time of EDLines is spent on the generation of the edge segment chains. Line fitting takes about 25%, and line validation takes the remaining 10% of the total running time. As we noted before, line validation is an optional last step in EDLines, and can be omitted if deemed unnecessary. For example, in a stereo-matching application where long lines would typically be used, a simple line validation scheme based on line length can be used. Since long lines are never rejected during the validation step, such a simple validation scheme would work. This would further speed up EDLines.

Fig. 6 demonstrates the effects of white noise in line segment detection by different algorithms. For this experiment, we have taken the house image in Fig. 1, added some Gaussian white noise, and fed it to various detectors. The three rows in Fig. 6 correspond to analysis at full, half, and quarter resolutions, respectively. At full resolution, Hough and Etemadi produce useless results, while LSD fails to detect the general structure of the image. At half and quarter resolutions, things start to get better for Hough, Etemadi and LSD. EDLines however, extracts the general structure of the image

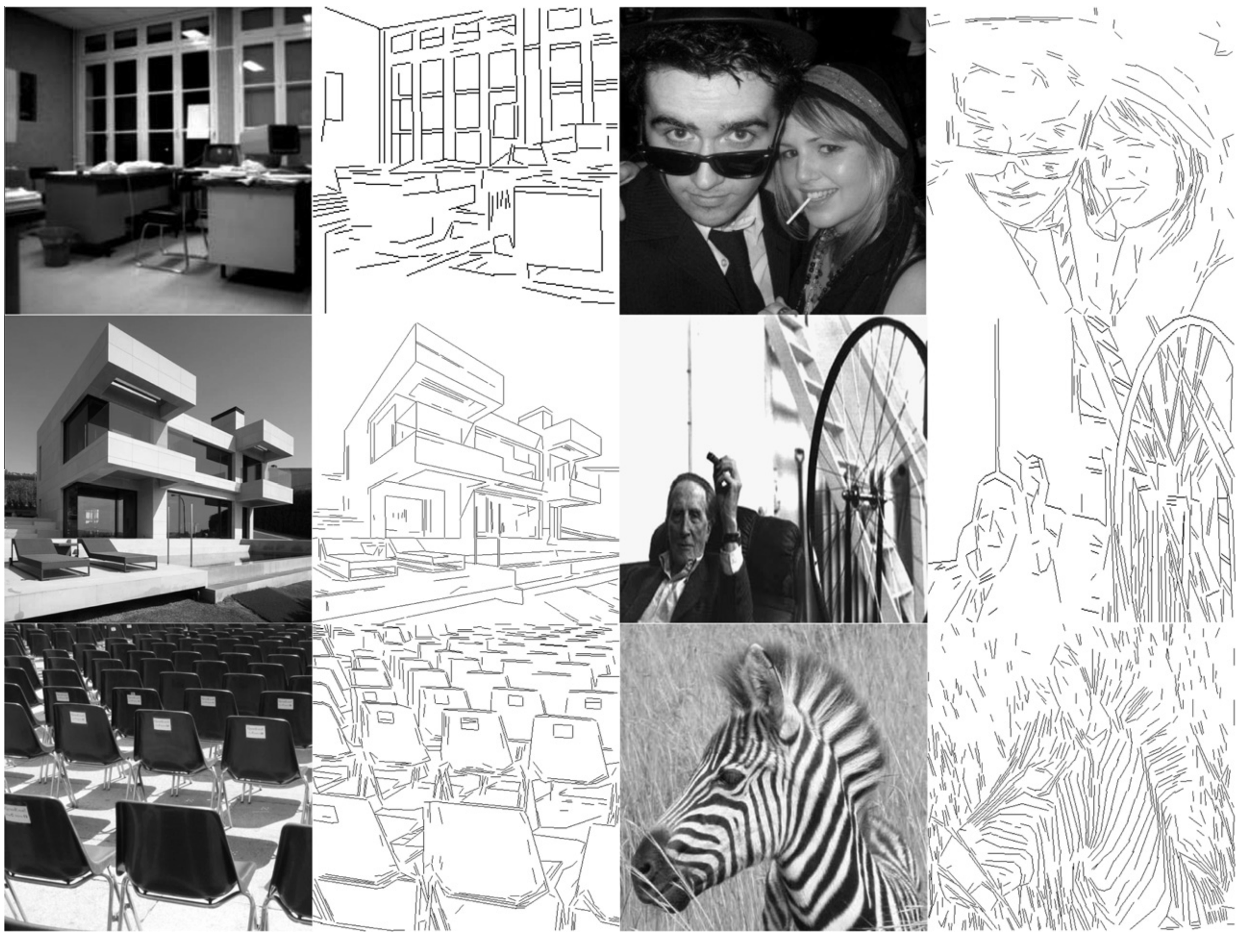
Fig. 5. Results produced by EDLines for several natural images. In each case, the structure of the image has been extracted with very few false positives.

Table 2
Dissection of the running times of EDLines, LSD and KHT on seven images given in Figs. 1 and 5 in increasing order of image size. In the case of KHT, the total time includes only the time taken by KHT given a binary edge map; it does not include the time taken to compute the binary edge map, nor does it include the time it would take to break down KHT's infinitely long lines in (angle, radius) representation to line segments by a post-processing step.

Image information		EDLines					LSD (Grompone von Gioi et al., 2010)		Speedup	KHT (Fernandes and Oliveira, 2008)	
Description	Image size (pixels)	Number of Lines	Edge Detection (ms)	Line Fit (ms)	Line Validation (ms)	Total Running Time (ms)	Scale = 0.8			Number of lines	Time (ms)
							Number of lines	Time (ms)			
Office	256 × 256	210	2.13	0.69	0.64	3.46	237	30.59	8.8	322	17.20
Man and bicycle	480 × 317	367	5.24	1.80	0.91	7.95	401	66.40	8.3	823	40.20
Zebra	500 × 466	770	10.01	3.93	1.84	15.78	802	181.66	11.5	1057	50.90
Boy and girl	480 × 512	364	8.25	2.14	1.13	11.52	396	122.10	10.6	1142	47.50
Chairs	512 × 512	654	8.14	2.75	1.45	12.34	716	104.54	8.5	1189	71.20
House (Fig. 4)	600 × 600	346	9.90	1.84	1.32	13.06	418	92.02	7	168	7.20
House (Fig. 1)	1200 × 900	794	46.16	13.34	5.18	64.68	931	489.16	7.6	1203	52.10
House (Fig. 1)	3072 × 2304	3994	280	104	37	421	3803	3867	9.1	333	150

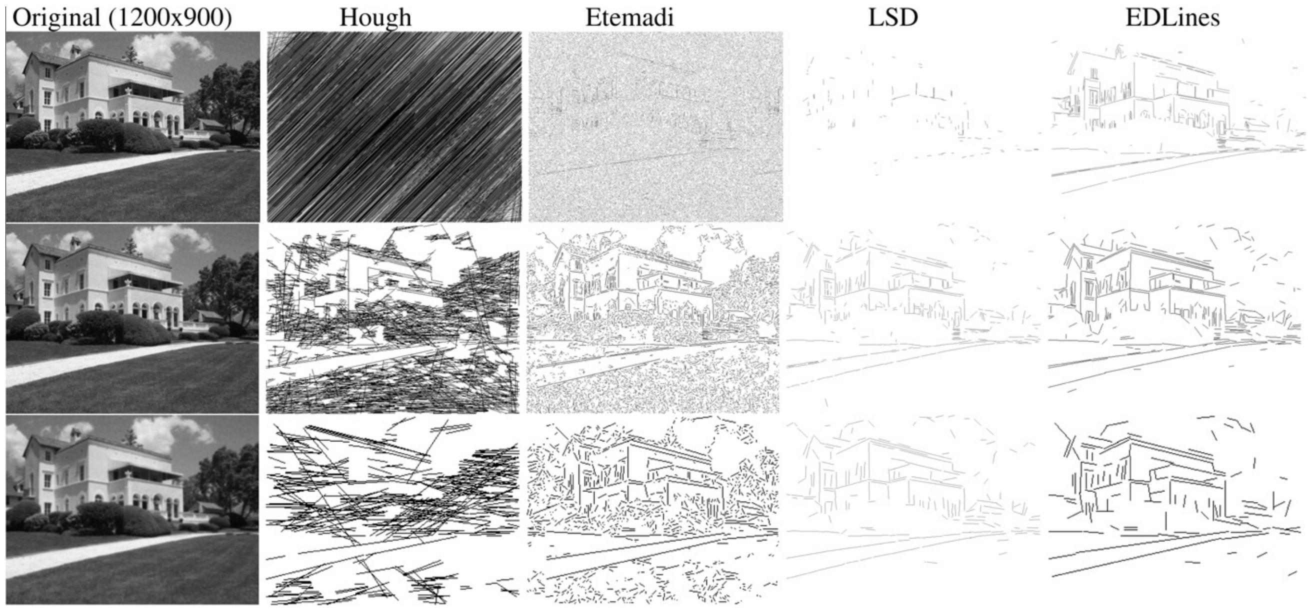

Fig. 6. The effects of white noise in edge segment detection by different algorithms. The three rows correspond to analysis at full, half, and quarter resolutions, respectively. At full resolution, Hough and Etemadi produce useless results, and LSD fails to detect the general structure of the image. At half and quarter resolutions, they start producing sort of meaningful results. EDLines however, extracts the general structure of the image at all resolutions with only a few false positives.

at all resolutions with only a few false positives. The reason LSD fails at full resolution is that the presence of noise prevents the growth of line support regions, and thus only small line segments are formed. These small segments are then eliminated during the validation step, and therefore, only a few detections are made (Grompone von Gioi et al., 2010). EDLines, however, is able to detect the general structure of the image at all resolutions.

To better analyze the effects of noise on EDLines, we performed another experiment with three noisy synthetic images. The results are shown in Fig. 7. The image at the first row of Fig. 7 has a vertical strip with a low level Gaussian white noise added. The presence of noise prevents LSD from detecting the entire vertical line at full resolution; rather only certain parts of the line are detected. This is expected since the presence of noise affects the line-support region growing algorithm in LSD, so only short line support regions can be formed. Noise also increases the NFA values of these support regions, so many of them are eliminated during validation leaving only a few short line segments, if at all (Grompone von Gioi et al., 2010). The solution proposed by LSD is to perform a multi-scale analysis and process the image at a coarser scale. Analysis of the same image at half resolution shows that LSD is now able to detect the entire vertical line. As for EDLines, we see that EDLines is able to detect the entire line at both full and half resolutions. This is due to the fact that the edge segment detection by ED uses an initial Gaussian smoothing step, which apparently is enough to handle the available noise. Also notice that neither LSD nor EDLines produce any false detections.

The second row in Fig. 7 shows the same vertical strip but with a high level Gaussian white noise added. Notice that with high level noise, LSD fails to detect any line segments at full resolution. But at half resolution, LSD produces reasonable results. Although the entire vertical strip is detected as several line segments, most of the vertical line seems to be detected. EDLines also fails to detect the entire vertical strip as one line segment at full resolution. This is due to the following: When the noise is too strong to be handled by ED's default 5 × 5 Gaussian smoothing kernel with $\sigma = 1$, pixels along the vertical strip would have aberrant gradient magnitudes and directions. Since ED works by joining anchors, which makes

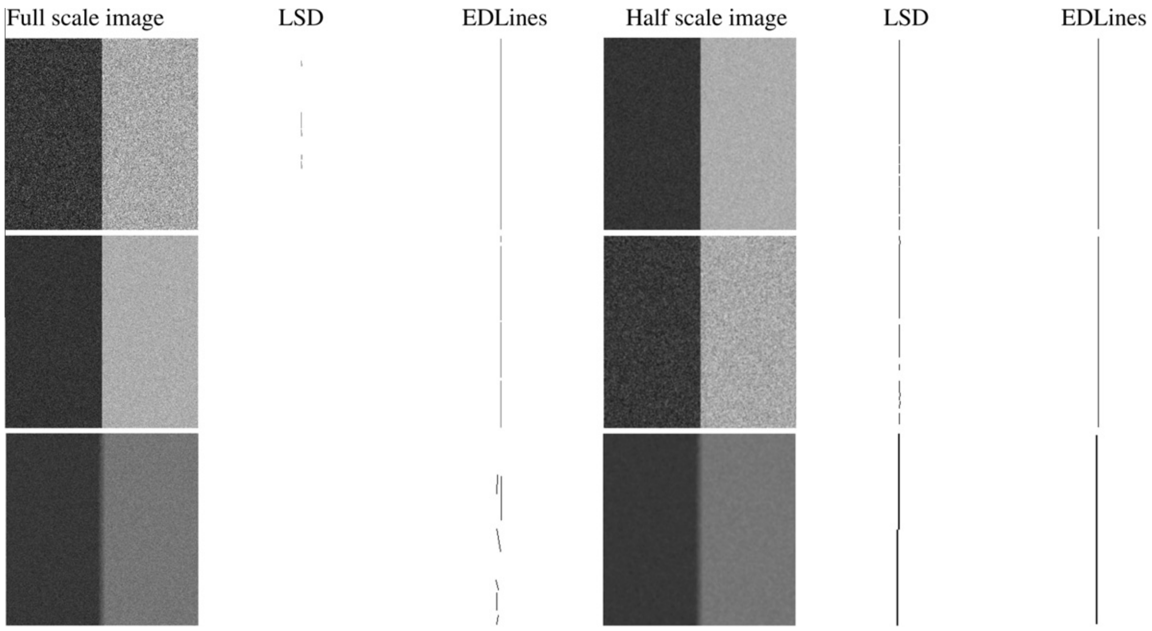

Fig. 7. Results produced by EDLines and LSD for several noisy synthetic images. As the amount of noise gets bigger, detection at full-scale images become difficult. At half-scale however, both algorithms produce good results.

use of gradient values and directions (Topal et al., 2010), noisy pixels on top of the vertical strip will divert ED sideways towards the noisy areas. This would make ED to detect the vertical strip not as a single edge segment, but rather as several disjoint edge segments. Thus, the vertical strip will be detected as several short line segments, some of which may even get eliminated during validation. This is exactly what we observe in the given example: EDLines detects the vertical strip as 4 line segments. But notice that at half resolution, EDLines is able to detect the entire vertical strip as one line segment. This is because producing the half resolution image by Gaussian sampling works as a denoising step, and is enough to take care of the added noise. This is why both LSD and EDLines were able to detect the vertical line easily. Also notice that neither LSD nor EDLines produce any false detections.

Finally, the third row in Fig. 7 shows a blurry vertical strip with some Gaussian white noise added. LSD again fails to detect any line segments at full resolution, but is able to find the entire vertical strip as one line segment at half resolution. EDLines also detects the entire vertical strip as one line segment at half resolution, but produces some questionable results at full resolution; although none of EDLines's detections can be classified as false detections. The reason for EDLines's observed behavior at full resolution can be attributed to ED's edge segment detection strategy: Around the blurry vertical strip, the gradient magnitude and direction values change very rapidly. This means that during anchor linking, ED is not able to walk along a straight line (as it would with a non-blurry vertical strip), but rather the path taken during anchor linking consists of several crooked edge segments. When line segments are extracted from these crooked edge segments, we get the result shown in Fig. 7. We again point out that EDLines's detections at full resolution cannot be classified as false detections since they all fall over the blurry vertical strip.

We conclude from the above experiments that ED's default denoising filter, i.e., 5×5 Gaussian kernel with $\sigma = 1$, is not able to handle noise of all types or power. This is clearly expected; but it is our experience that for many types of natural images, this filter lets ED produce high-quality edge segments from which EDLines can extract high-quality line segments. In the presence of high power noise, EDLines can always make use of multi-scale analysis similar to LSD (Grompone von Gioi et al., 2010) and produce good results as shown in Figs. 6 and 7.

Table 3 shows the effects of *Gaussian smoothing* and *maximum mean square line fit error* on the performance of EDLines. Each slot in the table shows the total number of edge segments detected for

Table 3
The performance of EDLines by Gaussian smoothing kernels of sizes 3×3 and 5×5 with $\sigma = 1$, and maximum mean square line fit errors of 0.8, 0.9, 1.0, 1.1 and 1.2 pixels. The values in the table are the number of line segments detected in each case.

Image information		Gauss kernel size (3 × 3)					Gauss kernel size (5 × 5)				
Description	Image size (pixels)	Maximum mean square error (pixels)					Maximum mean square error (pixels)				
		0.8	0.9	1.0	1.1	1.2	0.8	0.9	1.0	1.1	1.2
Office	256 × 256	260	259	254	253	249	218	217	210	206	203
Man and bicycle	480 × 317	394	387	378	374	369	384	373	367	365	363
Zebra	500 × 466	894	880	855	837	824	815	794	770	766	755
Boy and girl	480 × 512	384	384	377	370	370	366	364	364	362	360
Chairs	512 × 512	732	725	713	708	697	670	664	654	648	631
House (Fig. 4)	600 × 600	448	445	437	437	436	358	356	346	346	345
House (Fig. 1)	1200 × 900	987	977	964	963	962	824	818	794	790	787
House (Fig. 1)	3072 × 2304	4391	4362	4353	4344	4342	4041	4029	3994	3973	3962

the given set of parameters. Recall from Sections 2 and 5.4 that by default EDLines uses a 5 × 5 Gaussian smoothing kernel with $\sigma = 1$, and a maximum mean square line fit error of 1 pixel. One would expect *more Gaussian smoothing* and *bigger line fit errors* to result in less line segments. This is clearly observed from the results in the table. Also observe that the variations on the default parameters do not produce high variations on the obtained results, which is desirable. The variations on the running time performance of EDLines is negligible, and is not presented in the table.

7. Conclusion

EDLines is a real-time line segment detector with a false detection control due to the Helmholtz principle. It is based on the edge segment chains produced by our novel edge detector, *Edge Drawing* (*ED*). Experiments shown in this paper (and others that the reader may wish to perform online (EDLines Web Site)) show that EDLines produces accurate, well-localized line segments with only a few false positives, and it achieves this in blazing speed. Since EDLines works by analyzing a contiguous chain of pixels, it can be extended to detect other geometric structures such as arcs, circles, polygons, etc., which is left as future work.

References

Akinlar, C., Topal, C., 2011. EDLines: real-time line segment detection by edge drawing (ED), in: Proceedings of the International Conference on Image Processing (ICIP), September 2011, Brussels.

Beveridge, J.R., Graves, C., Lesher, C., 1996. Some Lessons Learned from Coding the Burns Line Extraction Algorithm in the Darpa Image Understanding Environment. Technical Report CS-96-125. Comp. Science Dept., Colorado State University.

Burns, J.B., Hanson, A.R., Riseman, E.M., 1986. Extracting straight lines. IEEE Trans. Pattern Anal. Machine Intell. 8 (4), 425–455.

Canny, J., 1986. A computational approach to edge detection. IEEE Trans. Pattern Anal. Machine Intell. 8 (4), 679–698.

Chan, T.S., Yip, R.K., 1996. Line detection algorithm, in: IEEE Internat. Conf. on Pattern Recognition, 1996, pp. 126–130.

Chao, L., Zhong, W., Lin, L., 2009. An improved ht algorithm on straight line detection based on freeman chain code, in: IEEE Internat. Congress on Image and Signal Processing, 2009, pp. 1–4.

Desolneux, A., Moisan, L., Morel, J.M., 2000. Meaningful Alignments. Internat. J. Comput. Vision 40 (1), 7–23.

Desolneux, A., Moisan, L., Morel, J.M., 2008. From Gestalt Theory to Image Analysis: A Probabilistic Approach. Springer.

Duda, R.O., Hart, P.E., 1972. Use of Hough transformation to detect lines and curves in pictures. Comm. ACM 15, 11–15.

Edge Drawing Web Site. <http://ceng.anadolu.edu.tr/cv/EdgeDrawing>.

EDLines Web Site. <http://ceng.anadolu.edu.tr/cv/EDLines>.

Etemadi, A., 1992. Robust segmentation of edge data, in: Internat. Conf. on Image Processing and its Applications, pp. 311–314.

Fernandes, Leandro A.F., Oliveira, Manuel M., 2008. Real-time line detection through an improved Hough transform voting scheme. Pattern Recogn. 41 (1), 299–314.

Franti, P., Ageenko, E.I., Kalviainen, H., Kukkonen, S., 1998. Compression of line drawing images using Hough transform for exploiting global dependencies, in: Internat. Conf. on Information Sciences, 1998.

Galambos, C., Kittler, J., Matas, J., 2001. Gradient based progressive probabilistic Hough transform. IEEE Proc. Vision Image Signal Process. 148 (3), 158–165.

Gorman, F.O., Clowes, M.B., 1976. Finding pictures edges through collinearity of feature points. IEEE Trans. Comput. C-25 (4), 241–266.

Grompone von Gioi, R., Jakubowicz, J., Morel, J.M., Randall, G., 2008a. On straight line segment detection. J. Math. Imag. Vision 32 (3), 313–347.

Grompone von Gioi, R., Jakubowicz, J., Morel, J.M., Randall, G., 2008. LSD: A Line Segment, Technical report, Centre de Mathematiques et de leurs Applications (CMLA), Ecole Normale Superieure de Cachan (ENS-CACHAN).

Grompone von Gioi, R., Jakubowicz, J., Morel, J.M., Randall, G., 2010. LSD: a fast line segment detector with a false detection control. IEEE Trans. Pattern Anal. Machine Intell. 32 (4), 722–732.

Hough, P., 1962. Method and Means for Recognizing Complex Patterns. US Patent 306954.

Illinworth, J., Kittler, K., 1988. A survey of the Hough transform. Comput. Vision Graphics Image Process., 87–107.

Ji, C.X., Zhang, Z.P., 1988. Stereo match based on linear feature, in: Internat. Conf. on Pattern Recognition, pp. 875–878.

Kahn, P., Kitchen, L., Rieseman, E.M., 1987. Real-Time Feature Extraction: A Fast Line Finder for Vision-Guided Robot Navigation. Technical Report 87-57, COINS.

Kahn, P., Kitchen, L., Rieseman, E.M., 1990. A fast line finder for vision-guided robot navigation. IEEE Trans. Pattern Anal. Machine Intell. 12 (11), 1098–1102.

Kalviainen, H., Hirvonen, P., Oja, E., 1996. Houghtool – a software package for the use of the Hough transform. Pattern Recognition Lett. 17 (8), 889–897.

Kang, W.J., Ding, X., Cui, J., 2007. Fast straight-line extraction algorithm based on improved Hough transform. Opto-Electron. Eng., 105–108.

KHT software package. <http://sourceforge.net/projects/khtsandbox>.

Kiryati, N., Eldar, Y., Bruchstein, A.M., 1991. Probabilistic Hough transform. Pattern Recogn. 24, 303–316.

Lam, L.T.S., Lam, W.C.Y., Leung, D.N.K., 1994. A knowledge-based boundary convergence algorithm for line detection. Pattern Recognition Lett. 15 (4), 383–392.

Least Squares Line Fitting. <http://en.wikipedia.org/wiki/Least_Squares>.

LSD: a Line Segment Detector Web Site. <http://www.ipol.im/pub/algo/gjmr_line_segment_detector>.

Mahadevan, S., Casasent, D.P., 2001. Detection of triple junction parameters in microscope images, in: Proc. SPIE, pp. 204–214.

Matas, J., Galambos, C., Kittler, J., 2000. Robust detection of lines using the progressive probabilistic Hough transform. Comput. Vis. Image Underst. 78 (1), 119–137.

MegaWave2 software package. <http://megawave.cmla.ens-cachan.fr>.

OpenCV library. <http://opencv.willowgarage.com>.

ORT-2.3 software package. <http://hpux.connect.org.uk/hppd/hpux/Physics/ORT-2.3>.

Princen, J., Illingworth, J., Kittler, J., 1990. A hierarchical approach to line extraction based on the Hough transform. Comput. Vision Graphics Image Process. 52, 57–77.

psychophysics. <http://en.wikipedia.org/wiki/Psychophysics>.

Topal, C., Akinlar, C., submitted for publication. Edge drawing: a real-time edge segment detector with a false detection control.

Topal, C., Akinlar, C., Genc, Y., 2010. Edge drawing: a heuristic approach to robust real-time edge detection, in: Proc. ICPR, pp. 2424–2427.

Tupin, F., Maitre, H., Mangin, J.F., Nicholas, J.M., Pechersky, E., 1998. Detection of linear features in SAR images: application to the road network extraction. IEEE Trans. Geosci. Remote Sens. 36 (2), 434–453.

Xu, L., Oja, E., Kultanen, P., 1990. A new curve detection method: randomized Hough transform (RHT). Pattern Recognition Lett. 11 (5), 331–338.

Yu, X., Lai, H., Liu, X., 2005. A gridding Hough transform for detecting the straight lines in sports videos. IEEE ICME 3, 45–48.

Zheng, Y., Li, H., Doerman, D., 2005. A parallel-line detection algorithm based on HMM decoding. IEEE Trans. Pattern Anal. Machine Intell. 27 (5).

Zhu, Y., Carrager, B., Kriegman, D., Milligan, R., Potter, C., 2001. Automated identification of filaments in cryo-electron microscopy images. J. Struct. Biol. 135, 302–312.