Pattern Recognition 127 (2022) 108619

Contents lists available at ScienceDirect

Pattern Recognition

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

ELSED: Enhanced line SEgment drawing

Iago Suárez[a,b], José M. Buenaposada[c,*], Luis Baumela[b]

[a] The Graffter, Campus Montegancedo s/n. Centro de Empresas, Pozuelo de Alarcón, 28223, Spain
[b] Departamento de Inteligencia Artificial. Universidad Politécnica de Madrid, Campus Montegancedo s/n, Boadilla del Monte, 28660, Spain
[c] ETSII. Universidad Rey Juan Carlos, C/ Tulipán, s/n, Móstoles, 28933,Spain

A R T I C L E I N F O

Article history:
Received 6 August 2021
Revised 17 December 2021
Accepted 28 February 2022
Available online 2 March 2022

Keywords:
Image edge detection
Efficient line segment detection
Line segment detection evaluation

A B S T R A C T

Detecting local features, such as corners, segments or blobs, is the first step in the pipeline of many Computer Vision applications. Its speed is crucial for real-time applications. In this paper we present ELSED, the fastest line segment detector in the literature. The key for its efficiency is a local segment growing algorithm that connects gradient-aligned pixels in presence of small discontinuities. The proposed algorithm not only runs in devices with very low end hardware, but may also be parametrized to foster the detection of short or longer segments, depending on the task at hand. We also introduce new metrics to evaluate the accuracy and repeatability of segment detectors. In our experiments with different public benchmarks we prove that our method accounts the highest repeatability and it is the most efficient in the literature.[1] In the experiments we quantify the accuracy traded for such gain.

1. Introduction

Detecting segments and full lines in digital images is a recurrent problem in Computer Vision (CV). Line segments play an important role in understanding the geometric content of a scene as they are a compressed and meaningful representation of the image content. Moreover, segments are still present in low-textured settings where the classical methods based on corners or blobs usually fail. Segment detection has been employed in a large number of CV tasks such as 3D reconstruction [1,2], SLAM [3,4], Visual Odometry [5], 3D camera orientation via Vanishing Points Detection [6,7], cable detection in air-crafts [8], or road detection in Synthetic Aperture Radar images [9].

Nowadays CV algorithms are ubiquitous and they are expected to run on resource-limited devices [10]. To this end, low-level algorithms such as the local feature detectors must be very efficient. Traditional global line detection approaches based on the Hough transform lack efficiency. Thus, various local methods emerged addressing the issue of efficiency. LSD [11] was one of the first approaches to achieve excellent results with a local approach. Edge drawing methods further improve the efficiency [12–14]. In a first step, they work by connecting edge pixels following the direction perpendicular to the gradient. In a second step, they fit the desired curve, a line in the simplest case, to these edges.

The method presented in this paper improves on the drawing methods by fitting a line segment to the connected edge pixels and using its direction to guide the drawing process. Fusing the drawing and line segment fitting in a single step saves time and improves the overall quality of the detected segments. In addition, our proposal allows to jump over gradient discontinuities and detect full lines or just detect the individual linear segments without jumping. This is important because line segments are features that, at the gradient level, can be easily broken by occlusions, shadows, glitches, etc. In this way, the user can define the type of segments that best suits the application. For example, we may choose to detect large segments if the goal is to do Vanishing Points estimation or short ones for reconstruction and matching.

In this paper we present an efficient method for line segment detection termed Enhanced Line SEgment Drawing (ELSED). In our experiments we compare the accuracy and efficiency of ELSED with that of the most relevant detectors in the literature. As shown in Fig. 1, ELSED is not only the most efficient (note the logarithmic scale in the speed dimension) but also the most accurate in line segment detection and more repeatable among the fastest in the literature, as we show in the experimental section. It improves the efficiency of present methods in resource-limited devices, opening the door to new CV applications running on any type of hardware.

* Corresponding author.
E-mail address: josemiguel.buenaposada@urjc.es (J.M. Buenaposada).
[1] Source code: https://github.com/iago-suarez/ELSED

https://doi.org/10.1016/j.patcog.2022.108619
0031-3203/© 2022 The Author(s). Published by Elsevier Ltd. This is an open access article under the CC BY-NC-ND license (http://creativecommons.org/licenses/by-nc-nd/4.0/)

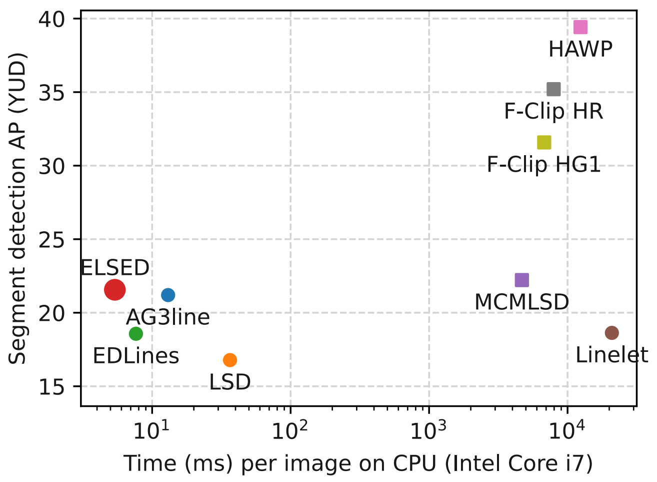


Fig. 1. Average Precision (AP) vs. execution time (ms) curve in the line segment detection problem. Local features based methods are displayed with circular markers and global ones with square markers.

2. Previous work

In this section we review the segment detection literature. To this end we organize it in three broad groups: full line detectors with global methods [15–17], those that use local properties to greedily detect line segments [11,13,14,18,19] and the deep line segment detectors [20–26].

2.1. Global information-based approaches

Global methods are able to detect full lines in the image with enough edge pixels support in spite of discontinuities. These methods start with an edge detection, for example Canny [27], and then they apply a Hough Transform-like [28] voting scheme. There are well-known issues with these methods: the omission of some weak edges, the generation of false positives in regions with high edge density (e.g. tree leaves) or the large amount of memory required to store the accumulators.

Progressive Probabilistic Hough Transforms (PPHT) [15,29] solves the efficiency problem changing the entire image voting scheme by a random sampling scheme. More recent works [16] address the quantization problem and also allow an efficient execution of the method. In [30] the direction, length, and width of a line segment is extracted in a closed-form that uses a fitted quadratic polynomial curve. To generate better segments, MCMLSD [17] uses the Elder & Zucker edge detector [31] that propagates edge uncertainty to the Hough histogram accumulator. Last, they split the detected full lines in line segments using a Markov Chain Model and a standard dynamic programming algorithm. The main drawback of this method is the efficiency. It takes 3.7 s to process a 640 $\times$ 480 image.

2.2. Local methods

Local methods overcome some of the drawbacks of global approaches by departing from strong gradient pixels and greedily add neighboring pixels using the gradient information. LSD [11] groups and validates image regions with a significant gradient magnitude and a similar gradient orientation in $O(N)$, being N the number of image pixels. Unlike the non-maximal suppression (NMS) used in the Canny edge detector, LSD uses a region growing process to select interesting pixels. Then, each region is validated based on the expected Number of False Alarms (NFA), computed with an A-Contrario statistical model [32]. LSD is efficient and it is able to deal with areas with a high density of edge pixels (e.g. trees).

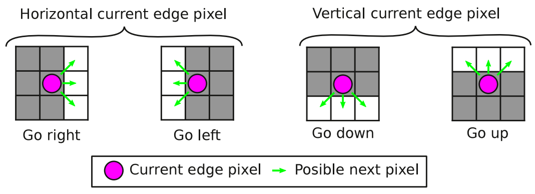
EDLines [13] is also an efficient algorithm ($O(N)$) based on local gradient orientations. It performs segment detection in two steps: 1) edge detection and 2) line segment detection using a local approach. The edge detection step is performed with the Edge Drawing (ED) algorithm [12]. ED applies the first three Canny steps: Gaussian filtering, gradient estimation, NMS and tries to connect the anchor pixels (local maxima in gradient magnitude) with a greedy procedure. In a second step EDLines performs line fitting. OTLines [18] uses an orientation transformation to improve EDLines and avoid segment detections on circular structures. AG3line [14] instead of drawing over all pixels only finds aligned anchors. They also implement a continuous validation strategy to decide whether the segment has reached its endpoint and a jumping scheme to overcome gradient discontinuities. A key difference between AG3line and ELSED is that we use every pixel in the image as part of the drawing process, which generates a chain of continuous pixels, while AG3line uses only the detected anchors. This makes the method fast but unstable, needing to validate each step and thus being slower than ELSED.

The approach of Cho et al. [19] is based on *linelets* detection, i.e. chunks of horizontally or vertically connected pixels that result from line digitization. The linelet detection is $O(N^2)$ time complexity and thus it takes 16.7 s per image. Adjacent linelets are further grouped into line segments using a probabilistic model with $O(L^2)$ complexity, where L is the number of detected linelets. Last, they validate using a mixture of experts model learnt from the gradient magnitudes, orientation and from the line length of the segments in a labeled data set. They also propose a quantitative evaluation that we improve in our experiments (Section 4).

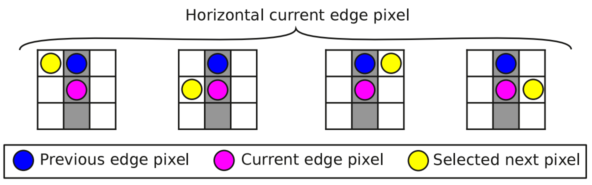
2.3. Deep line segment detectors

A closely related problem to line segments detection is wireframe parsing. It consists of predicting the scene's salient straight lines and their junctions. The ShanghaiTech Wireframe data set [20] contains over 5000 hand-labelled images that allowed different methods to train obtaining competitive results. AFM [21] uses an attraction field map that is next squeezed to obtain line segments. L-CNN [22] proposes an End-to-end model that uses a stacked hourglass backbone to obtain a junction proposal heatmap that is extensively sampled to obtain the output segments. HAWP [23] improves the L-CNN sampling step by reparametrizing the line segments in a holistic 4-D attraction field map from which segments can be obtained faster. HT-HAWP [23] Adds some Deep Hough layers to the HAWP model improving its capabilities to capture lines and slightly improving the performance in some benchmarks. F-Clip [24] proposes a simple yet effective approach to cast line segment detection as an object detection problem that can be solved with a fully convolutional one-stage method. LERT [25] detects segments using transformers that replace the junction heatmap and segment proposals to directly predict the segment endpoints. SOLD^2 [26] proposes a self-supervised way towards line detection and description that optimizes the repeatability of the detected segments.

Despite the good results of these deep methods, their computational requirements are still far away from the classical methods based on gradient. This fact makes them non-viable for limited devices where there is no GPU or rather its battery consumption is prohibitive. For this reason we introduce our drawing method that has been carefully designed to avoid floating-point operation and minimize its complexity, being CPU friendly and achieving the fastest execution times on the state of the art.



(a) ED next pixel selection.



(b) Corner-shaped edges arising from ED.

Fig. 2. ED greedy segment growing. In (a) it only takes into account the current edge pixel. In the case of (b) the walk is coming from the blue pixel and finds a horizontal edge pixel (pink one). Thus, it will start a walk to the left and another to the right that could give edge chains with the displayed configurations (blue-pink-yellow pixel sequence). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

3. Enhanced line SEgment drawing (ELSED)

In this section we introduce our line segment detection method and explain the different steps in our approach.

3.1. Enhanced edge drawing algorithm (EED)

The EED entails the following high-level steps: 1) Gaussian smoothing to suppress noise; 2) Gradient magnitude and orientation computation; 3) Extraction of anchor pixels, local maxima in the gradient magnitude; 4) Connect the anchors using the enhanced routing algorithm

For the noise reduction step we use a convolution with a 5×5 Gaussian kernel and $\sigma = 1$, for Gradient magnitude and orientation computation we first compute the horizontal, G_x, and vertical, G_y, gradients by applying the Sobel operator and then we use the $L1$ norm, $G = |G_x| + |G_y|$. We also define a gradient threshold and set $G = 0$ for those pixels below it and quantize the gradient orientation, O, into two possible values: vertical edge, $|G_x| \geq |G_y|$, or horizontal edge, $|G_x| < |G_y|$. The other two steps are explained in the next subsections.

3.1.1. Extraction of anchor pixels

The anchors are pixels where the drawing process begins. We scan image pixels with $G > 0$ and test if it is a local maxima in the gradient magnitude, G, along the quantized direction of the gradient, O. If the pixel orientation $O(x, y)$ corresponds to a vertical edge, it is an anchor if $G[x, y] - G[x-1, y] \geq T_{anchor}$ and $G[x, y] - G[x+1, y] \geq T_{anchor}$. The same applies for horizontal edge in vertical direction. To increase the processing speed, the number of anchors can be limited by increasing the value of T_{anchor} and also by scanning pixels every $SI = 2$ columns and rows.

3.1.2. Connecting the anchors by an enhanced routing algorithm

ED is faster than LSD's region growing because, from an initial anchor, it only walks along a chain of edge pixels, evaluating 3 neighbours at each step (see Fig. 2a) and selecting as next step the one with biggest gradient magnitude. The evaluations are critical for the speed of the algorithm since they are done for each reachable edge pixel walking from an anchor point.

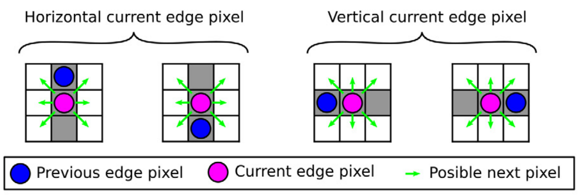
In our EED procedure, we perform the edge drawing and line fitting at the same time. This will enable us to save computations by reducing the number of checked pixels. During drawing, we consider the previous and the current pixel. We explore the same pixels as in ED during the walking processes (see "Go right", "Go left", "Go up" and "Go down" in Fig. 3b) as long as the edge orientation does not change from the previous pixel to the current one. When the previous pixel is in a vertical edge, and the current pixel is in a horizontal one, ED has 6 candidate pixels to be added to the current line segment (see Fig. 3a). This may generate situations where the algorithm would draw a corner breaking the line assumption (Fig. 2b). The same happens when the previous pixel is in a horizontal edge and the current is on a vertical one.

Here we introduce a different approach to treat these diagonal pixels while following a line. We add the assumption that the edge chain should form a line. Then, considering Bresenham's line drawing scheme, the number of checked pixels in this situation changes from 6 with ED (see Fig. 3a) to only 2 with EED (see the four "diagonal" cases in Fig. 3b), and remains 3 for the other possible previous pixels (non-diagonal cases). This has two advantages: 1) it is faster than the original ED routing algorithm as it explores fewer pixels and, 2) it avoids non-meaningful cases for finding line segments.

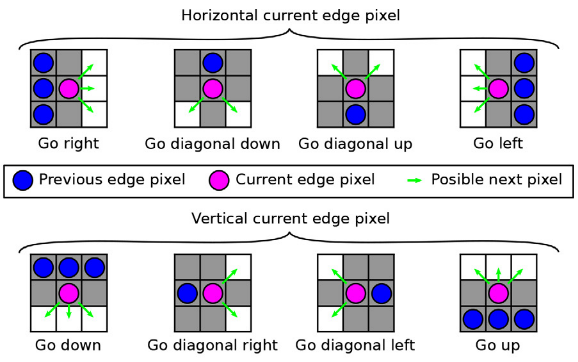
The second important idea is a also consequence of trying to find aligned edge pixels. Whereas ED changes the walking process direction when a change of edge orientation is detected (See Fig. 4a), EED tries to continue in the same direction following a line. However, any change of edge orientation is not forgotten and it is pushed into the stack of discontinuities, $\mathcal{D}_{stack}$, for later processing (see Algorithm 1). EED tries to fit a line to the current chain of pixels, if more than $T_{minLength}$ pixels have been chained and the squared error of alignment of the pixels is lower than $T_{LineFitErr}$. The last parameter for the segment search is the $T_{PxToSegDist}$ that is the maximum distance in pixels from which we consider whether

Algorithm 1 Enhanced Edge Drawing Algorithm.

```
 1: procedure EED($a$, $d_0$)
 2: Input: Anchor pixel: $a$. Anchor gradient direction: $d_0$
 3: Output: S: List of Segments, P: List of edge pixels
 4:     $P \leftarrow \{a\}$ ; $S \leftarrow \emptyset$
 5:     $\mathcal{D}_{stack} \leftarrow \emptyset$ ; $\mathcal{D}_{stack}$.push([$a$, $d_0$])
 6:     while $\mathcal{D}_{stack} \neq \emptyset$ do
 7:         segmentFound ← false
 8:         nOutliers ← 0
 9:         $c, d \leftarrow \mathcal{D}_{stack}$.pop()                ▷ Current pixel and direction
10:         $p \leftarrow$ previousPixel($c$, $d$)              ▷ Find previous pixel
11:         while $G[c] \neq 0 \wedge$ nOutliers $\leq T_{outliers}$ do
12:             $c, p \leftarrow$ drawNextPx($c$, $p$)
13:             $P \leftarrow P \cup \{c\}$
14:             if segmentFound then
15:                 s, nOuliers ← addPxToSegment(s, $c$, nOuliers)
16:             else
17:                 s, segmentFound ← fitNewSegment($P$)          ▷ Can return ∅
18:                 S ← S ∪ {s}
19:             end if
20:         end while
21:         if $G[c] \neq 0$ then
22:             $\mathcal{D}_{stack}$.push([$c$, lastDir])            ▷ Edge orientation change
23:             if canContinueForward(s) then
24:                 $p_f, d_f \leftarrow$ s.forwardPxAndDir($c$, $d$)
25:                 $\mathcal{D}_{stack}$.push([$p_f$, $d_f$])
26:             end if
27:             if canContinueBackward(s) then
28:                 $p_b, d_b \leftarrow$ s.backwardPxAndDir($c$, $d$)
29:                 $\mathcal{D}_{stack}$.push([$p_b$, $d_b$])
30:             end if
31:         end if
32:     end while
33: end procedure
```

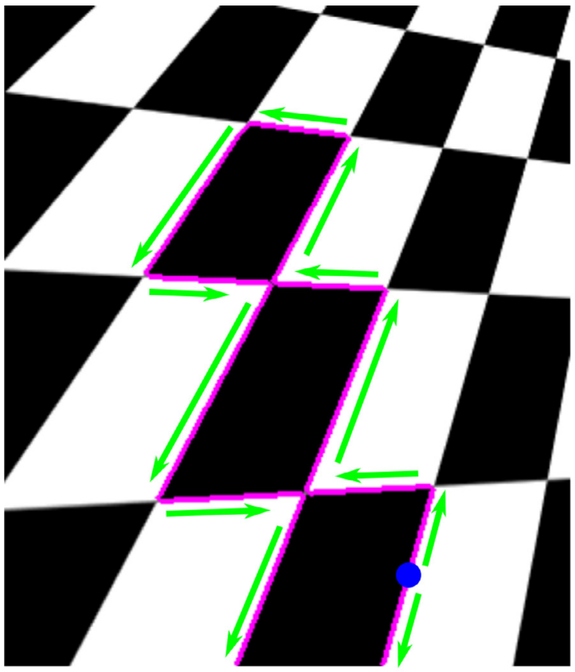


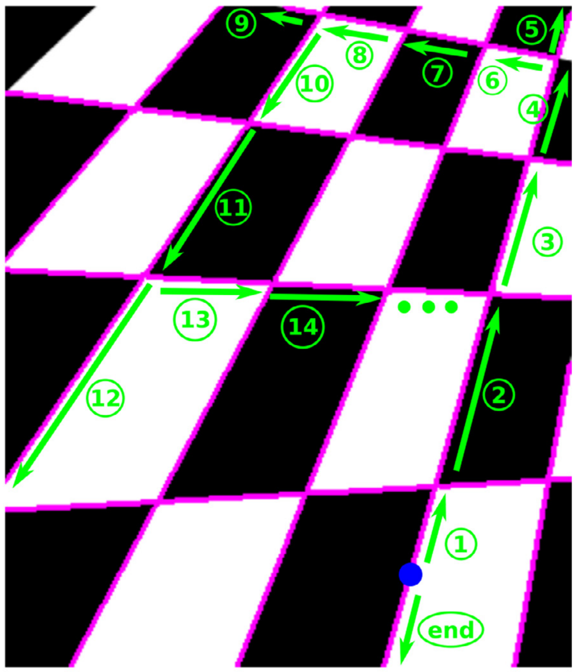
(a) Edge Drawing (ED).



(b) Enhanced Edge Drawing (EED).

Fig. 3. Drawing diagonal edges: When the previous pixel was in a different orientation (vertical vs horizontal) than the current one, original ED has 6 candidates pixels vs EED that has 3 or 2. More than one previous edge pixel is displayed when we have the same candidates for any of the previous pixel options.


(a) Edge Drawing [12]



(b) ELSED

Fig. 4. Results of the drawing process for ED and EED (purple pixels) from one single anchor (blue point). Green arrows and numbers establish which segments are visited first. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

a pixel fits or not in the current segment. This is done internally in the function *addPxToSegment* in line 15 of Algorithm 1). The process stops when no more pixels can be chained (i.e. at the limits of the image, with only already visited pixels or weak edge pixels as candidates) or a line edge discontinuity is detected (e.g. the gap between two aligned windows, a tree branch occluding part of a building, etc.). In case a discontinuity was detected we execute, in order, the following actions:

1. If we were walking along a line segment (i.e. we have fitted a line), try to extend it going on the line direction (the walking process stacked using *canContinueForward* and *forwardPxAndDir* functions in Algorithm 1, lines 23 to 26)
2. If were walking along a line segment and we cannot continue forward, try to extend the line segment backwards (the walking process stacked using *canContinueBackward* and *backwardPxAndDir* functions in Algorithm 1), lines 27 to 30)
3. Continue in the gradient direction, that is changing in the discontinuity (the walking process stacked in Algorithm 1, line 22).

This sequence of ordered actions guarantees that if a line segment is detected, all its pixels will be detected together.

We show an illustrative example in Fig. 4b, where EED starts two walking processes, one upward and another downward. Unlike ED (see Fig. 4a), EED detects the discontinuities in the edge orientation of the chessboard corners and continues walking in the current line direction. Each discontinuity is stored in $\mathcal{D}_{stack}$ for later processing. After pixels in segments (1) to (5) are chained, the next edge orientation to process is extracted from the top of $\mathcal{D}_{stack}$. The drawing process keeps drawing from it, linking the pixels of segments (6) to (9). The routing algorithm ends when there are no more edge orientation changes in $\mathcal{D}_{stack}$. Then, the next anchor point is processed by the routing algorithm. Detecting more segments from a single anchor is an important feature of our method that increases the detection recall with respect to EDLines as we will show in experiment 4.1.

3.2. The line segment discontinuities

A line segment can be interrupted by several edge discontinuities. These are regions of the image where the gradient orientation changes or the gradient magnitude goes to zero. Whether we aim to detect full lines or just line segments, the discontinuities of different lengths (i.e. the number of pixels where there is no edge or the edge direction is not aligned with the line segment) should be correctly skipped in a drawing process in order to detect the line segments correctly.

Algorithm 1 naturally deals with this phenomena. Once a discontinuity is detected, our aim is to jump over it and continue drawing in the line segment direction if possible. *A priori*, the discontinuity length is unknown and our algorithm test different length candidates. As we are using a 5×5 Gaussian smoothing kernel, any 1 pixel discontinuity will have effect in at least a neighbourhood of size 5 pixels, therefore we set the minimum discontinuity length to 5 pixels. In the functions *canContinueFordward* (Algorithm 1, line 23) and *canContinueBackward* (Algorithm 1, line 27), different jump lengths J are checked (in the default parameters of the algorithm we use $J \in [5, 7, 9]$). The drawing process will continue after the discontinuity if the ordered conjunction of the following conditions is true:

1. The segment is longer than the number of pixels, J, we want to jump.
2. The pixel, a, that is aligned with the segment and J pixels away from current pixel, c, is inside the image and has $G[a] > 0$ (i.e. is not a weak edge pixel).
3. Starting from a, EED is able to draw at least J pixels following the edge direction. We call this set of J pixels the extension pixels.
4. The extension pixels are well aligned with the line segment. To check this we calculate the auto-correlation matrix of the image gradients, $\mathtt{M}$, in a small neighbourhood (we take one pixel on each side of the extension pixels) and then, we assert that:

$$\frac{\lambda_1}{\lambda_2} \geq T_{EigenExt} \tag{1}$$

where λ_1 and λ_2, $(\lambda_1 > \lambda_2)$, are the eigenvalues of the $\mathtt{M}$, and

$$\angle(\mathbf{v}_1, \mathbf{n}) \leq T_{AngleExt} \tag{2}$$

$\angle(\cdot, \cdot)$ is the angular distance, $\mathbf{v}_1$ is the first eigenvector of $\mathtt{M}$ and $\mathbf{n}$ is a vector normal to the segment.

In Fig. 5 we show different steps of the discontinuity management algorithm in a synthetic image. In Fig. 5d we show the detection process starting from the left side of the image, we fit the edge pixels (blue) to a horizontal line (green). When the edge orientation changes from horizontal to vertical, we detect the last edge pixels as outliers (orange) and thus our method detects that we are in a discontinuity. In this case, the red pixel is the last one detected when $|E| > T_{ol}$, where T_{ol} the maximum number of outliers.

Next Fig. 5e shows the check done to decide whether we should continue drawing straight or the segment has finished. Purple pixels are the pixels in the discontinuity and are skipped using the Bresenham's algorithm drawing in the current line segment direction. Red pixels are drawn following the edge direction (in green) starting from the first pixel after the discontinuity, a. We also mark in red the neighbor pixels used to validate the region using the eigenvalues of $\mathtt{M}$ if the extension pixels in function *canContinueForward*. Pixels in this synthetic example do not have a uniform gradient direction, thus, the process discards all extensions tested with lengths $J \in [5, 7, 9]$. In the figure we show the last one, $J = 9$. Consequently, the algorithm closes the first segment and continues drawing in the dominant gradient direction upwards. When enough pixels are gathered, we fit another segment (Fig. 5f). When we reach the top of the image, the segment is extended in the backward direction downwards. When this happens, the mechanism to manage discontinuities is activated again as Fig. 5g shows, but this time the region meets all the criteria defined and thus the jump is executed. Figure 5h and i show respectively the fitted edges and the segments.

One of the limitations of local segment detection approaches is the generation of many small segments produced by gradient discontinuities. The process described above helps us alleviate this.

3.3. Validation of the generated segment candidates

After EED, we have several line segments detections, many of them potentially wrong (see red segments in Fig. 6). They occur mainly in regions with a high density of edge pixels. To validate a line we use the segment pixels' gradient orientation, comparing its direction with the one normal to the segment, i.e. the angular error. This validation can be performed efficiently, without damaging the overall performance.

For a good validation we discard the pixels lying in a discontinuity and those near the endpoints, because they usually have a different gradient orientation even in correct detections. Despite this, the gradient orientation error in a true segment is a noisy signal. In Fig. 7 we can see the probability distribution function (PDF) of orientation errors for pixels on true positive segment detections (TP) (blue), false positive segments (FP) (orange) and false positives segments detected in a random noise intensity image (green). It is clear that the pixels on TP segments have less angular error than those on FP ones. However, there is a significant overlap between FP and TP distributions.

Therefore, we use a validation criteria robust to noise. We validate a segment if at least 50% of its pixels have an angular error lower than a threshold, T_{valid}. To separate positive detections (i.e. valid ones) from negative ones, we learn a threshold $T_{valid} = 0.15$ radians which keeps a high recall discarding few true detections.

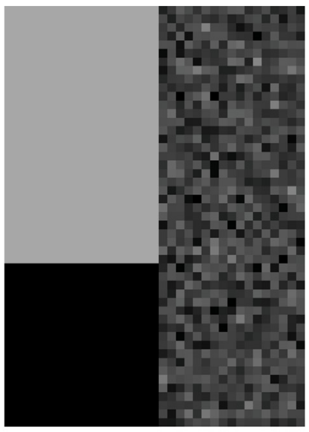
3.4. Parameter selection

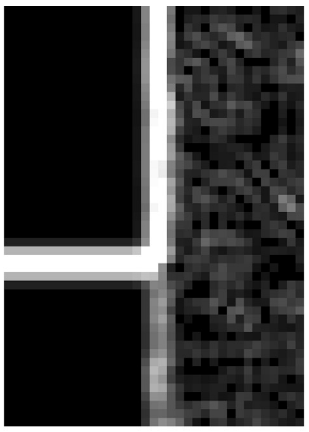
Most ELSED parameters have been set empirically and do not need to be changed by the user. We use a Gaussian smoothing filter ($\sigma = 1$, kernel size $= 5 \times 5$), a gradient threshold $T_{grad} = 30$, anchor threshold $T_{anchor} = 8$ and $SI = 2$, that defines the scan interval of anchor every SI row/column. For the line segment fitting: $T_{ol} = 3$, $T_{minLength} = 15$, $T_{LineFitErr} = 0.2$, $T_{PxToSegDist} = 1.5$ and for validation $T_{EigenExt} = 10$, $T_{AngleExt} = 10^{\circ}$, $T_{valid} = 0.15$ radians.

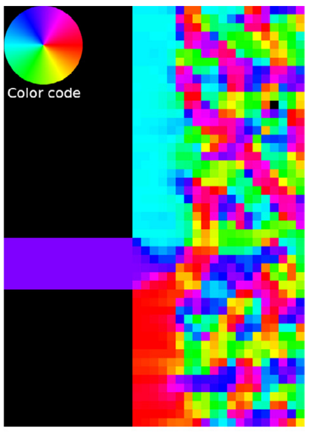
However, other parameters may be tuned by the user to define the type of segments to be detected; this is the case for the list of jump lengths that will be tested in the discontinuity management. Since this is directly related to the size of the Gaussian smoothing kernel in the first step (5×5 in our implementation) and the size of the gradient convolution kernel (3×3 in our case), we define a set of default values, $(5, 7, 9)$, that in the experiments of Section 4.4 provide good results.

4. Experiments

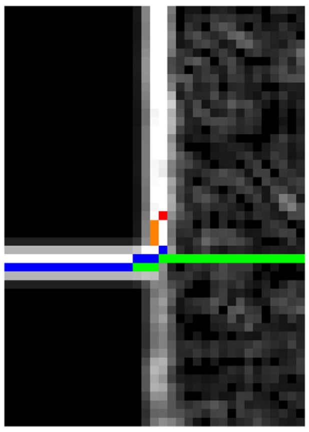
In this section we introduce a methodology to evaluate the accuracy and repeatability of segment detectors and follow it to compare our detector with the best in the literature. We also present an ablation study to analyze how each component of our algorithm contributes to the final result.

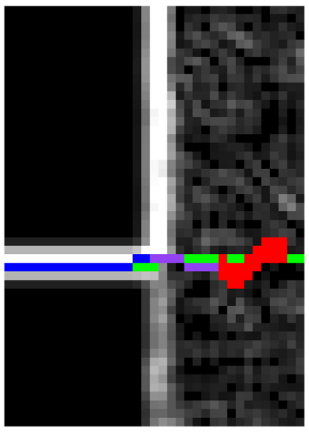

(a) Original Image

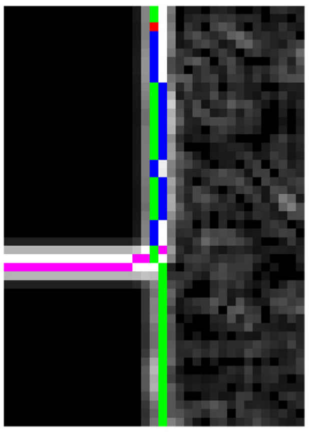

(b) Blurred Image gradient

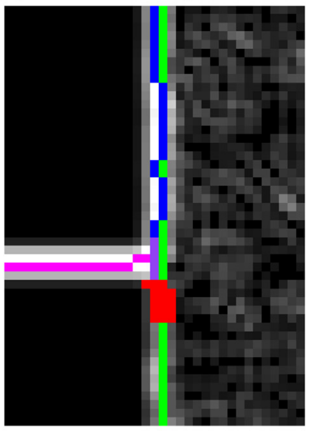


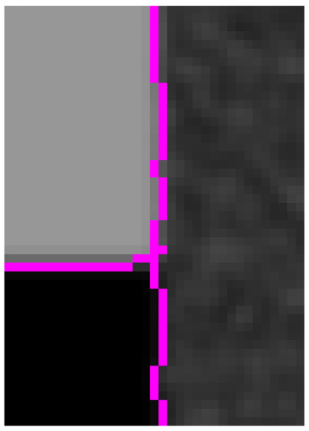
(c) Gradient orientation


(d) Discontinuity detection


(e) 1st discontinuity check


(f) Drawing 2nd segment


(g) 2nd discontinuity check


(h) Detected edges

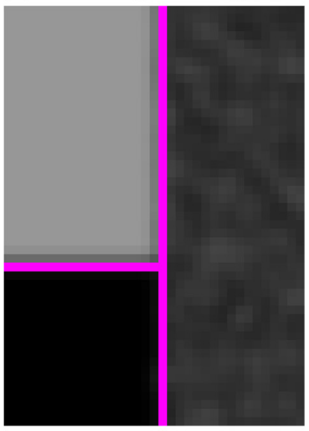

(i) Detected Segments

Fig. 5. Example of ELSED discontinuity management algorithm. Gradient orientations in (c) coded with red (right direction), purple (up), cyan (left) and light green (bottom)

We perform our evaluation in two dimensions, accuracy and efficiency. To this end we have grouped the algorithms in two sets, following the two clusters in Fig. 1. In the first set we find algorithms that run efficiently in CPU (LSD, EDLine, AG3line and ELSED). In the second set those that require more than one second to process an image (MCMLSD, Linelet, HAWP, SOLD^2, F-Clip). Although HAWP, SOLD^2 and F-Clip are DL methods and should be run in GPU, we run them also on CPU to show the different computational requirements of each approach. It is also important to note, that most low-power devices like smartphones, drones or IoT devices are usually not prepared to run the GPU for long periods of time. In our experiments we compare the accuracy and efficiency of ELSED with the approaches in each group.

4.1. Segment detection evaluation

We evaluate the segment detection in the York Urban Data set (YUD) [33] that contains indoor and outdoor man-made scenes where some salient segments have been manually labeled. The data set was extensively re-labeled later in Linelet [19], which contains all segments in the scene, but also some inconsistencies.

We propose a new evaluation framework that combines the benefits of previous evaluation protocols [17,19,22], namely, it is

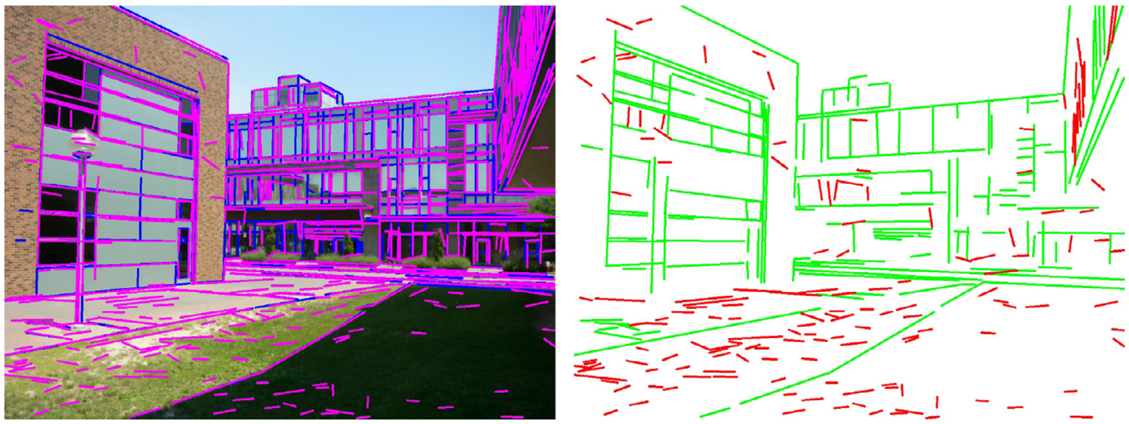


Fig. 6. Positive (green) and negative (red) segments detected by ELSED. We obtain these labels comparing the ground truth segments of the YorkUrban-LineSegment data set (blue) with unvalidated ELSED detections (magenta). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

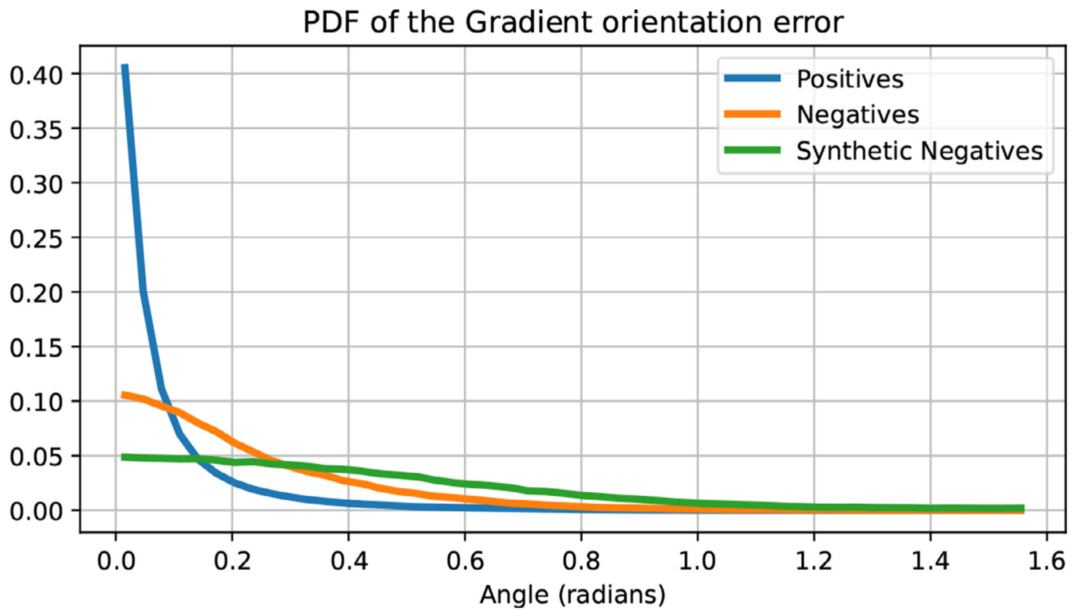


Fig. 7. PDF of the gradient orientation error in correct segments (blue), negative segments (orange) and segments detected in a random intensity image (green). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

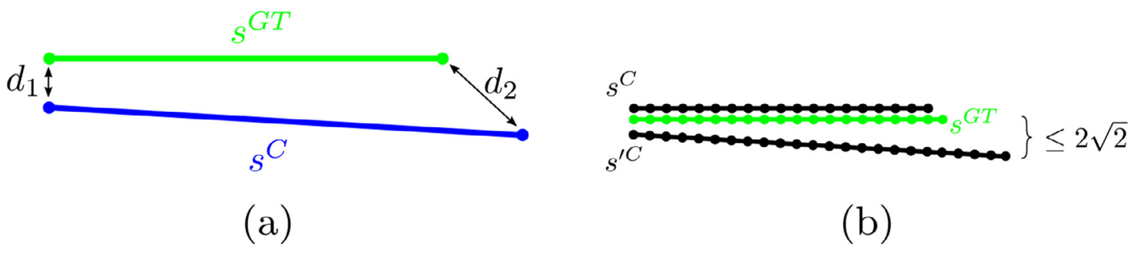


Fig. 8. Comparison of (a) Structural distance used by ELSED to fill the cost matrix $\mathbf{A}$ and (b) point-sample based distance used by MCMLSD [17]. In (b) the detected segment s'^C is matched to the ground truth (GT) segment s^{GT} while the best possible match is s^C.

fast to compute and is a fair and stable metric for line segment detection. We ensure a good 1-to-1 match between detected and ground truth (GT) segments by using the Hungarian algorithm to find the optimal bipartite match. The assignation problem is defined with a cost (or matching) matrix $\mathbf{A}$, that is filled using the structural distance (see Fig. 8a). This metric is a good trade off between perpendicular distance, misalignment and overlap. It is also faster to compute than the matched number of sampled points. To speed up the Hungarian algorithm and ensure a meaningful matching, we require matched segments to have an intersection over union bigger than $\lambda_{overlap} = 0.1$, to have an angular distance smaller than $\lambda_{ang} = 15^\circ$ and a perpendicular distance smaller than $\lambda_{dist} = 2\sqrt{2}$. The pairs of segments that do not meet these criteria, have infinite cost in the corresponding entry of $\mathbf{A}$, avoiding their matching. Let $\mathbf{x}$ be the set of detected segments and $\mathbf{y}$ the set of ground truth segments. With the 1-to-1 assignations $\mathbf{A}^*$ between them we define:

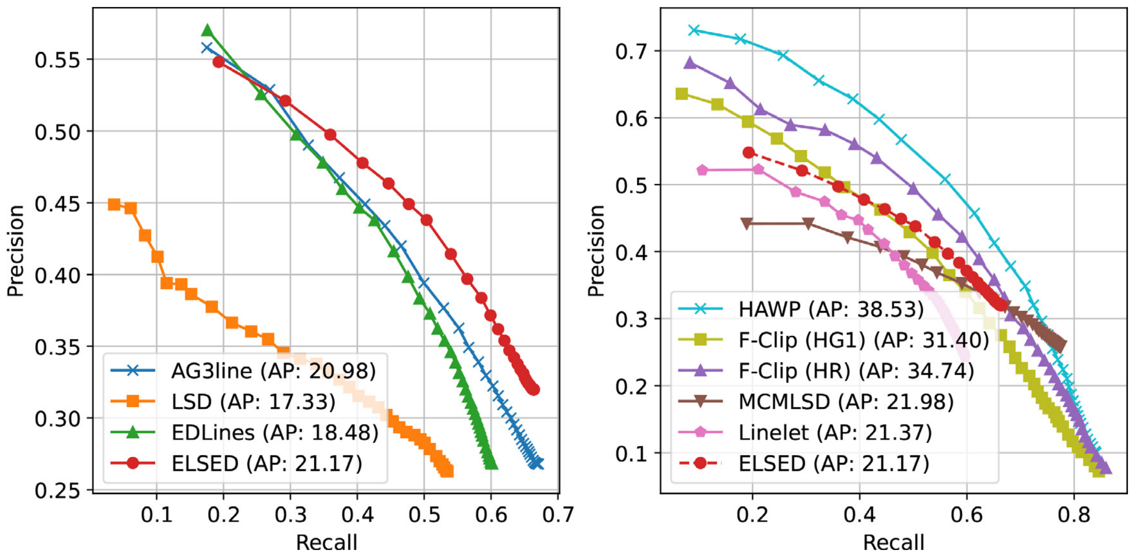


Fig. 9. Detected segments' Precision-Recall with our evaluation framework in YUD with original labels.

- Precision: Length of the matched intersection measured over the detected segment $\mathbf{x}_i$, divided by the length of the detected segments, $P = \frac{\sum_{i,j \in \mathbf{A}^*} \mathbf{x}_i \cap_{\mathbf{x}_i} \mathbf{y}_j}{\sum_i |\mathbf{x}_i|}$.
- Recall: Length of the matched intersection measured over the ground truth segment $\mathbf{y}_j$, divided by the length of the ground truth segments, $R = \frac{\sum_{i,j \in \mathbf{A}^*} \mathbf{y}_j \cap_{\mathbf{y}_j} \mathbf{x}_i}{\sum_j |\mathbf{y}_j|}$.
- Intersection over Union: Length of the matched intersection measured over the ground truth segments, divided by length of the matched union measured over the ground truth segments, $\text{IoU} = \frac{\sum_{i,j \in \mathbf{A}^*} \mathbf{y}_j \cap_{\mathbf{y}_j} \mathbf{x}_i}{\sum_{i,j \in \mathbf{A}^*} \mathbf{y}_j \cup_{\mathbf{y}_j} \mathbf{x}_i}$.

In Fig. 9 we show the Precision-Recall of our algorithm and the state-of-the-art detectors. This curve is computed w.r.t. the original YUD data set annotations and we sort the segments produced by each method using the score provided by their original code. For ELSED, the score is the percentage of pixels that have an angular error lower than T_{valid}. The curve of ELSED is better than those of LSD, EDLines and AG3line, the methods able to run on CPU efficiently (see experiment 4.5). We obtain a better precision, reaching similar or higher recall. MCMLSD gets better recall but at the cost of very bad precision. This was expected since it is the only method based on the HT and detects all full lines with enough support, even the hallucinated ones over highly textured regions. ELSED is able to improve AG3line because it has a better drawing scheme tailored to the line segment detection problem. ELSED is on par with the most efficient DL approaches, F-Clip (HG1), in the range of recall where ELSED works. Deeper networks such as HAWP that needs a GPU and are far away from real-time in CPU obtain, as expected, results with better recall and precision.

In Table 1 columns 2 to 5 show the highest recall values in the P-R curve for each method. They correspond to the case where we require the detector to find as many segments as possible. The last two columns show metrics over all the P-R points of the curve. If we look at the results with the original YUD labels, ELSED has the best precision (0.3198), F-score (0.4148) and IoU (0.7111) among the efficient detectors on CPU. It also obtains the best results for the overall metrics along all the Precision-Recall points (Average Precision, AP). AP is the usual classification metric, that is biased towards detectors with a wider recall range. Thus, we also show the AP bounded to the recall interval where the curve of each method is defined (bAP). ELSED has a bAP comparable with F-Clip (HR) and better than F-Clip (HG1). This means that, although ELSED detects fewer segments (lower recall range than F-Clip) it has a precision similar to the top DL-based methods in the detected segments.

We also present the results for the "YorkUrban-LineSegment annotation" [19]. In the new annotation, the definition of segments changes to short and broken lines from the global lines in the original YUD labels. In this case, we also show the results of ELSED without jumps (*ELSED-NJ*). ELSED-NJ obtains competitive results, being the best in terms of precision and IoU in this data set. This shows the nice property of ELSED, which allows to adapt the segment definition to the application by changing the jump length over discontinuities.

With these experiments we can conclude that, although ELSED has been designed with the objective of reducing the execution time, it is also a competitive algorithm in terms of segment detection accuracy. The reason is that the EED process is adapted to the segment detection problem and jumps over discontinuities to produce outputs that match the segment length in the annotations.

4.2. *Wireframe parsing*

This experiment presents a comparative evaluation in a different task: Wireframe Parsing. Figure 10a shows the evaluation results in the ShanghaiTech Wireframe dataset [34] and YUD, with the evaluation protocol described in L-CNN [22]. Unlike the protocol presented in Section 4.1, here all segments have the same importance and the GT segments are matched to detections by nearest neighbour.

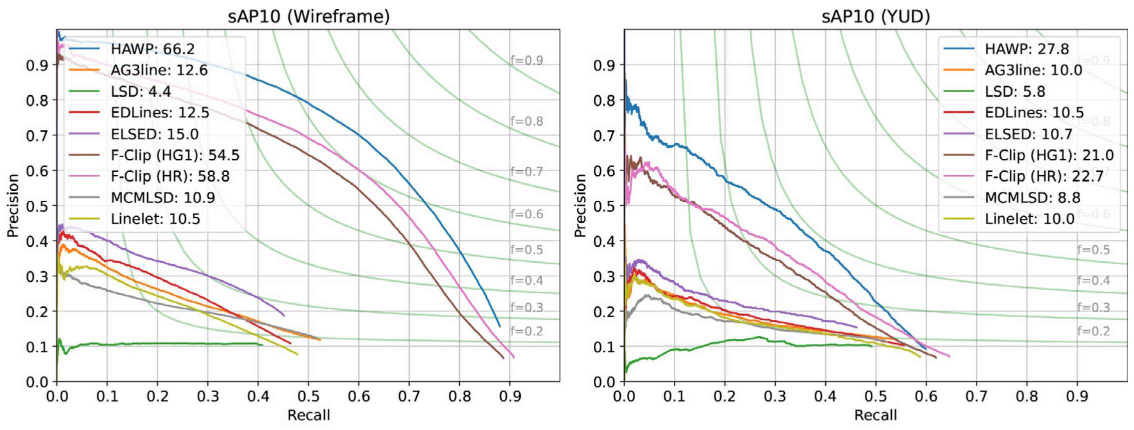
As expected, computationally-expensive methods (HAWP and F-Clip), trained for this task, achieve top-performing results. However, when the same evaluation protocol is applied in a similar dataset, YUD (Fig. 10b), their precision roughly halves. Partly because YUD's annotations are less dense, but also as result of data acquisition and labeling biases. By contrast, general-purpose line segment detectors generalize better. ELSED obtains the best results among them. This is because EED is able to retrieve more segments from each anchor and because the jump strategies properly manage gradient noise.

4.3. *Repeatability*

Regardless the type of segments aimed for the detection, a desirable property is the robustness against changes in viewpoint,

Table 1
Results with our evaluation protocol (top half: efficient methods, bottom half: slow methods). We use **bold** for best results in each experiment. The columns show for each method the Precision (P), Recall (R), Intersection over Union (IoU), F-Score (F_sc), Average Precision (AP) and bounded Average Precision (bAP).

	Original YUD annotations						YorkUrban-LineSegment annotations					
Method	P	R	IoU	F_sc	AP	bAP	P	R	IoU	F_sc	AP	bAP
LSD	0.26	0.53	0.59	0.34	17.33	34.78	0.68	0.52	0.67	0.58	**38.23**	76.35
EDLines	0.27	0.60	0.64	0.36	18.48	43.42	0.67	0.56	0.68	**0.60**	36.61	76.88
AG3line	0.27	**0.67**	0.69	0.37	21.02	42.32	0.60	**0.62**	0.66	0.60	34.69	69.06
ELSED	**0.32**	0.66	**0.71**	**0.41**	**21.17**	**44.94**	0.68	0.53	0.68	0.58	31.01	71.69
ELSED-NJ	-	-	-	-	-	-	**0.71**	0.51	**0.69**	0.59	32.43	**76.93**
MCMLSD	0.26	0.77	0.74	0.37	21.98	37.50	0.55	**0.62**	0.62	0.57	30.24	57.71
Linelet	0.24	0.60	0.63	0.33	21.37	43.62	0.62	0.58	0.66	**0.59**	**39.35**	**75.52**
HAWP	**0.49**	**0.60**	**0.83**	**0.51**	**38.53**	**51.41**	0.65	0.30	0.70	0.40	33.48	56.84
F-Clip(HG1)	0.47	0.42	0.80	0.42	31.40	40.35	0.67	0.22	**0.72**	0.32	32.13	50.06
F-Clip(HR)	0.53	0.47	0.82	0.47	34.74	44.77	**0.69**	0.22	0.72	0.33	32.78	51.15



(a) Wireframe (b) York Urban Dataset

Fig. 10. Wireframe parsing evaluation.

scale, rotation or lighting. In this subsection we evaluate the detectors' repeatability. Given two images of the same scene under different conditions, the capacity to detect the same segments in both situations. Specifically, given two images, we define line segments repeatability as the ratio between the length of 1-to-1 segment matches and the total length of segments detected in both images. We take into account only the segments located in the part of the scene present in both images, adjusting their endpoints accordingly.

We use the images of HPatches [35] where the repeatability of segment detections in images $\mathcal{A}$ and $\mathcal{B}$ is computed as:

$$\text{repeatability} = \frac{\sum_{i,j \in \mathtt{A}^*} \mathbf{x}_i^{\mathcal{A}} \cap_{\mathbf{x}_i^{\mathcal{A}}} \mathbf{x}_j^{\mathcal{A}|\mathcal{B}}}{\sum_i \left|\mathbf{x}_i^{\mathcal{A}}\right| + \sum_j \left|\mathbf{x}_j^{\mathcal{A}|\mathcal{B}}\right|} + \frac{\sum_{i,j \in \mathtt{B}^*} \mathbf{x}_i^{\mathcal{B}} \cap_{\mathbf{x}_i^{\mathcal{B}}} \mathbf{x}_j^{\mathcal{B}|\mathcal{A}}}{\sum_i \left|\mathbf{x}_i^{\mathcal{B}}\right| + \sum_j \left|\mathbf{x}_j^{\mathcal{B}|\mathcal{A}}\right|} \quad (3)$$

Where $\mathbf{x}^{\mathcal{I}}$ are the segments detected in image $\mathcal{I}$ and $\mathbf{x}^{\mathcal{I}|\mathcal{J}}$ the segments detected in image $\mathcal{J}$, projected to $\mathcal{I}$ using the homography between them. The matching matrix $\mathtt{A}^*$ contains the 1-to-1 segment assignations between segments $\mathbf{x}^{\mathcal{A}}$ and $\mathbf{x}^{\mathcal{A}|\mathcal{B}}$ obtained with the matching process described in Section 4.1 and $\mathtt{B}^*$ the one between $\mathbf{x}^{\mathcal{B}}$ and $\mathbf{x}^{\mathcal{B}|\mathcal{A}}$.

In Table 2 the second column also shows the repeatability in terms of the number of matched segments. We employ here $\lambda_{overlap} = 0.5$ and $\lambda_{dist} = 5$ according to Pautrat et al. [26]. It both cases ELSED obtains the most repeatable results. This is because EED provides stability to the edge detection and also because the jump strategy is able to overcome small discontinuities that cause other local methods to fail. We have also observed that deep models like SOLD^2 or HAWP get highly repeatable results in some scenes and very bad results in others, this is possible because they have been trained in a quite specific problem (the wireframe parsing for indoor scenes) and do not generalize well to the diverse images of Hpatches.

Table 2
Mean repeatability of each segment detector in the HPatches data set. The higher the repeatability, the more robust the detector.

Method	Length Repeatability	Num Segs. Repeatability
LSD	0.53934	0.48707
EDLines	0.54352	0.47242
AG3line	0.43582	0.40002
ELSED	**0.55928**	**0.50285**
MCMLSD	0.50322	0.39440
Linelet	0.52102	0.43427
HAWP	0.40056	0.41248
SOLD^2	0.46161	0.47957
F-Clip (HG1)	0.39784	0.43560
F-Clip (HR)	0.40969	0.43413

4.4. Ablation study

We start with the simplest version of ELSED: no discontinuity jumps and no validation step (see Table 3). This version of the algorithm obtains the worst results (IoU= 0.646 and F_sc= 0.360 in Table 3) for long segment detections. When the validation is activated, the precision increases from 0.271 to 0.307 whereas the recall remains high (0.59).

If we now add the discontinuity jump component with a fixed length of 5 pixels it removes some small detection errors. For example, in the second row and column of Fig. 11 the broken segments of the wall in the left, with the added fixed-length jump

Table 3
Results of the ablation study using our 1-to-1 evaluation protocol over the original YUD annotations. Best results in **bold**. Execution times measured on Intel Core i7.

Configuration			Last P-R point metrics				Global metrics	
Jumps	Jump Val.	Seg. Val.	P	R	IoU	F_sc	bAP	Time (ms)
None			0.271	0.609	0.646	0.360	43.06	**2.95**
None		✓	0.307	0.590	0.651	0.388	44.27	4.16
Fixed (5 px)			0.278	**0.702**	**0.712**	0.383	42.46	3.25
Fixed (5 px)	✓		0.282	0.667	0.694	0.381	43.62	3.48
Multiple (5,7,9 px)	✓		0.285	0.686	0.706	0.387	43.28	4.16
Multiple (5,7,9 px)	✓	✓	**0.320**	0.664	0.711	**0.415**	**44.94**	5.38

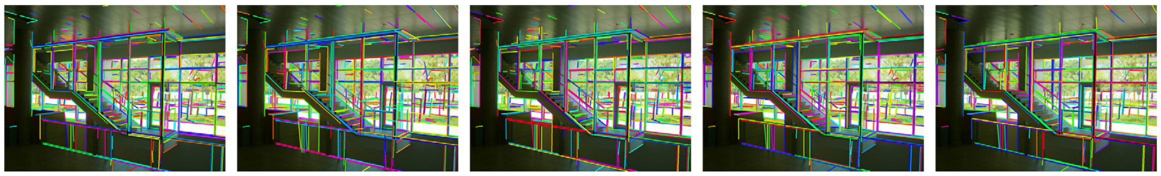


Fig. 11. From left to right: No Validation no jumps, No validation with fixed size jumps (5 px) and no jump validation, No validation with fixed size jumps (5 px) and jump validation, No validation with multi-size jumps (5, 7, 9 px) and jump validation, Validation with multi-size jumps (5, 7, 9 px) and jump validation

Table 4
Executions times for different state of art line segment detectors on different processors. Results are the average processing time per image, in the YUD [33] images with size 640 × 480.

Method	Intel Core i7	Snapdragon	Exynox
LSD	36.51 (±1.60)	58.68 (±0.81)	390.91 (±0.92)
EDLines	7.64 (±0.33)	13.79 (±0.15)	65.79 (±0.16)
AG3line	13.04 (±0.76)	18.57 (±0.20)	100.54 (±0.19)
ELSED-NJ	**4.18 (±0.23)**	**8.28 (±0.03)**	**45.84 (±0.02)**
ELSED	5.38 (±0.30)	10.20 (±0.07)	59.99 (±0.16)
MCMLSD	4.68K (±1.78K)	**GeForce GTX 1050**	
Linelet	20.9K (±10.1K)		
HAWP	12.4K (±0.8K)	212.25 (±8.35)	
SOLD^2	3.17K (±0.52K)	417.72 (±6.78)	
F-Clip HR	7.94K (±0.15K)	47.39 (±1.46)	
F-Clip HG1	6.78K (±0.22K)	11.00 (±0.47)	

capability are detected as a unique segment. On the other hand, now the algorithm performs some incorrect jumps going beyond the endpoint of the segment. This effect can be observed in the results of Table 3 where the Recall takes a big leap (from 0.609 to 0.702) and the Precision also increases moderately (from 0.271 to 0.278). To fix the problem with incorrect jumps, we add the validation of the jump destination region of Section 3.2 (see the well-fitted endpoints of the third column in Fig. 11).

On the other hand, a jump of 5 pixels is not big enough if the discontinuity is large. In this case, the jump validation is performed over pixels on the discontinuity. Thus, since discontinuities contain gradients in different directions to the normal of the segment, the jump validation fails. This is the reason why we add the multi-length jumps in the fourth column in Fig. 11. With it, we can deal with longer segment discontinuities. The last step is the validation of the whole detected segment (see Section 3.3) shown in the last column of Fig. 11. Segment validation increases the precision (from 0.285 to 0.320) with a small penalty in the recall (from 0.686 to 0.664).

4.5. *Efficiency evaluation*

Nowadays CV applications not only require good accuracy, but also fast execution times and low energy consumption. This experiment measures the average execution time in the images of YUD data set (Table 4) which contains 101 images with a resolution of 640 × 480 pixels. Execution time is measured on four different platforms: a laptop with an Intel Core i7 8750H CPU, 12 cores and 16GB of RAM; a smartphone *Samsung J5 2017* with an Exynox Octa S CPU, 8 cores and 2GB of RAM; a smartphone *One Plus 7 Pro* with Snapdragon 855 CPU, 8 cores and 6GB of RAM and a GPU GeForce GTX 1050 with 4GB of RAM.

We use the implementation provided by the authors of each method: LSD, EDLines AG3line and ELSED in C++, Linelet and MCMLSD in Matlab and HAWP, SOLD^2 and F-Clip in Python. ELSED is implemented in C++ with Python bindings. To keep fast execution times we compute only L1 gradient norm, which is faster than L2, and predominant gradient direction (vertical or horizontal). In EED we fit the segments with a least squares approach oriented vertically or horizontally that we compute incrementally and, if possible, we reuse the top element from $\mathcal{D}_{stack}$ to avoid memory reallocation.

In all platforms, ELSED is around 2× faster than AG3line, and EDLines, 6× faster than LSD and much faster than MCMLSD and Linelet. DL methods are designed to run in the GPU, however GPU may not always be available in some platforms like drones, IoT or mobile phones and when it is, it usually involves unaffordable energy consumption. Looking at the CPU times, the DL methods need between 2300× (HAWP) and 1200× (F-Clip HG1) more computation than ELSED. Moreover, even when we run the DL methods on a laptop GPU (Geforce GTX 1050) ELSED is still faster than any of the methods. Therefore, for limited platforms, ELSED represents the best segment detector, as it is not only faster, but also detects better than the other efficient methods (see Table 1) obtaining the most repeatable group of segments (Table 2).

5. Conclusions

In this paper we have introduced ELSED, a general-purpose, fast and flexible line segment detector. It processes a 640x480 image in less than 6 ms on a regular PC and around 10 ms on a modern smartphone. This efficiency arises as a result of joining the processes of edge drawing and segment detection in one single step, with an Enhance Edge Drawing (EED) algorithm conceived for the problem of line segment detection.

ELSED also includes a scheme to jump over discontinuities. This endows our method with a flexible strategy to cope with different segment length requirements and improves its robustness against occlusions, shadows and glitches, which make all efficient methods to break down. This is important, for example, in a problem such as Vanishing Point estimation, where long and accurate segments are required.

In segment-based reconstruction, repeatability is a key feature desired in detectors. We have also introduced a repeatability metric and experimentally shown that ELSED is the top performer.

Overall, ELSED is the fastest and most repeatable segment detector in the literature. It is however less accurate than other DL-based competitors, which are computationally orders of magnitude less efficient. Yet, since it is a general purpose detector, it exhibits good performance on different data sets, achieving an AP at the same level as other algorithms orders of magnitude slower. These properties make it ideal for real-time applications like Visual Odometry, SLAM, or self-localization in resource-limited devices.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This work was supported by Doctorado Industrial grant DI-16-08966 and MINECO project TIN2016-75982-C2-2-R.

References

[1] Y. Zhou, H. Qi, Y. Zhai, Q. Sun, Z. Chen, L.-Y. Wei, Y. Ma, Learning to reconstruct 3D manhattan wireframes from a single image, in: Proc. of Int. Conf. on Comp. Vis., 2019, pp. 7698–7707.
[2] P. Miraldo, T. Dias, S. Ramalingam, A minimal closed-form solution for multi--perspective pose estimation using points and lines, in: Proc. European Conf. on Comp. Vis., Springer, 2018, pp. 490–507.
[3] Y. Li, N. Brasch, Y. Wang, N. Navab, F. Tombari, Structure-SLAM: low-drift monocular SLAM in indoor environments, IEEE Rob. Autom. Lett. 5 (4) (2020) 6583–6590.
[4] R. Gomez-Ojeda, F.-A. Moreno, D. Zuniga-Noël, D. Scaramuzza, J. Gonzalez-Jimenez, PL-SLAM: a stereo SLAM system through the combination of points and line segments, IEEE Trans. Rob. 35 (3) (2019) 734–746.
[5] R. Gomez-Ojeda, J. Briales, J. Gonzalez-Jimenez, PL-SVO: semi-direct monocular visual odometry by combining points and line segments, in: Proc. of Int. Conf. on Intell. Robots Systems, IEEE, 2016, pp. 4211–4216.
[6] J. Lezama, R. Grompone von Gioi, G. Randall, J.-M. Morel, Finding vanishing points via point alignments in image primal and dual domains, in: Proc. Conf. on Comp. Vis. and Pattern Recogn., 2014, pp. 509–515.
[7] S. Iago, M. Enrique, J.M.B. Luis Baumela, FSG: a statistical approach to line detection via fast segments grouping, in: Proc. of Int. Conf. on Intell. Robots Systems, IEEE, 2018, pp. 97–102.
[8] Y. Tian, C. Zhang, S. Jiang, J. Zhang, W. Duan, Noncontact cable force estimation with unmanned aerial vehicle and computer vision, Comput.-Aided Civ. Infrastruct. Eng. 36 (1) (2021) 73–88.
[9] C. Liu, R. Abergel, Y. Gousseau, F. Tupin, LSDSAR, a Markovian a contrario framework for line segment detection in SAR images, Pattern Recognit. 98 (2020) 107034.
[10] I. Suárez, J.M. Buenaposada, L. Baumela, Revisiting binary local image description for resource limited devices, IEEE Rob. Autom. Lett. 6 (4) (2021) 8317–8324.
[11] R.G. von Gioi, J. Jakubowicz, J.M. Morel, G. Randall, LSD: a fast line segment detector with a false detection control, IEEE Trans. Pattern Anal. Match. Intell. 32 (4) (2010) 722–732.
[12] C. Topal, C. Akinlar, Edge drawing: a combined real-time edge and segment detector, J. Vis. Commun. Image Represent. 23 (6) (2012) 862–872.
[13] C. Akinlar, C. Topal, EDLines: a real-time line segment detector with a false detection control, Pattern Recognit. Lett. 32 (13) (2011) 1633–1642.
[14] Y. Zhang, D. Wei, Y. Li, AG3line: active grouping and geometry-gradient combined validation for fast line segment extraction, Pattern Recognit. 113 (2021) 107834.
[15] J. Matas, C. Galambos, J. Kittler, Robust detection of lines using the progressive probabilistic hough transform, Comp. Vis. Image Understanding 78 (1) (2000) 119–137.
[16] R. Tal, J.H. Elder, An accurate method for line detection and manhattan frame estimation, in: Proc. of Asian Conf. on Comp. Vis., Springer, 2012, pp. 580–593.
[17] E.J. Almazan, R. Tal, Y. Qian, J.H. Elder, MCSLD: a dynamic programming approach to line segment detection, in: Proc. Conf. on Comp. Vis. and Pattern Recogn., IEEE, 2017, pp. 5854–5862.
[18] W. Ding, W. Wang, X. Li, OTLines: a novel line-detection algorithm without the interference of smooth curves, Pattern Recognit. 53 (2016) 238–258.
[19] N.-G. Cho, A. Yuille, S.-W. Lee, A novel linelet-based representation for line segment detection, IEEE Trans. Pattern Anal. Match. Intell. (2017).
[20] K. Huang, Y. Wang, Z. Zhou, T. Ding, S. Gao, Y. Ma, Learning to parse wireframes in images of man-made environments, in: Proc. Conf. on Comp. Vis. and Pattern Recogn., 2018, pp. 626–635.
[21] N. Xue, S. Bai, F. Wang, G.-S. Xia, T. Wu, L. Zhang, Learning attraction field representation for robust line segment detection, in: Proc. Conf. on Comp. Vis. and Pattern Recogn., 2019, pp. 1595–1603.
[22] Y. Zhou, H. Qi, Y. Ma, End-to-end wireframe parsing, in: Proc. of Int. Conf. on Comp. Vis., 2019, pp. 962–971.
[23] N. Xue, T. Wu, S. Bai, F. Wang, G.-S. Xia, L. Zhang, P.H.S. Torr, Holistically-attracted wireframe parsing, in: Proc. Conf. on Comp. Vis. and Pattern Recogn., 2020, pp. 2788–2797.
[24] X. Dai, X. Yuan, H. Gong, Y. Ma, Fully convolutional line parsing,2021arXiv: 2104.11207
[25] Y. Xu, W. Xu, D. Cheung, Z. Tu, Line segment detection using transformers without edges, in: Proc. Conf. on Comp. Vis. and Pattern Recognit., 2021, pp. 4257–4266.
[26] R. Pautrat, J.-T. Lin, V. Larsson, M.R. Oswald, M. Pollefeys, SOLD^2: self-supervised occlusion-aware line description and detection, in: Proc. Conf. on Comp. Vis. and Pattern Recognit., 2021, pp. 11368–11378.
[27] J. Canny, A computational approach to edge detection, in: Readings in Comp. Comp., Elsevier, 1987, pp. 184–203.
[28] D.H. Ballard, Generalizing the hough transform to detect arbitrary shapes, in: Readings in Comp. Comp., Elsevier, 1987, pp. 714–725.
[29] N. Kiryati, Y. Eldar, A.M. Bruckstein, A probabilistic hough transform, Pattern Recognit. 24 (4) (1991) 303–316.
[30] Z. Xu, B.-S. Shin, R. Klette, Closed form line-segment extraction using the hough transform, Pattern Recognit. 48 (12) (2015) 4012–4023.
[31] J.H. Elder, S.W. Zucker, Local scale control for edge detection and blur estimation, IEEE Trans. Pattern Anal. Match. Intell. 20 (7) (1998) 699–716.
[32] A. Desolneux, L. Moisan, J.-M. Morel, Meaningful alignments, Int. J. Comp. Vis. 40 (1) (2000) 7–23.
[33] P. Denis, J.H. Elder, F.J. Estrada, Efficient edge-based methods for estimating manhattan frames in urban imagery, in: Proc. European Conf. on Comp. Vis., Springer, 2008, pp. 197–210.
[34] K. Huang, Y. Wang, Z. Zhou, T. Ding, S. Gao, Y. Ma, Learning to parse wireframes in images of man-made environments, in: Proc. Conf. on Comp. Vis. and Pattern Recogn., 2018, pp. 626–635.
[35] V. Balntas, K. Lenc, A. Vedaldi, K. Mikolajczyk, HPatches: a benchmark and evaluation of handcrafted and learned local descriptors, in: Proc. Conf. on Comp. Vis. and Pattern Recogn., 2017, pp. 5173–5182.

Iago Suárez received the BS degree in 2015 from Universidade da Coruña, MS degree in 2016 and PhD degree in 2021 from the Universidad Politécnica de Madrid (UPM). He has co-founded or participated in several startups: XOIA Software Development and The Graffter where he actually works as Computer Vision engineer. He is also a teaching assistant in the courses of Pattern Recognition and Deep Learning from UPM. He works in the intersection of computer vision and machine learning. His research interests include efficient deep learning, image matching and localization, local features, line segment detection and vanishing point estimation, SLAM and SfM.

José M. Buenaposada received the BS and MS degrees in 1999 and the PhD degree in 2005 both in computer science from the Universidad Politécnica de Madrid. Since 2003 he has been working at the Universidad Rey Juan Carlos, Spain, and from 2008 to 2018 he was a Contratado Doctor (Associate Professor equivalent) and from december 2018 he is a Profesor Titular (Associate Professor equivalent) at the same University. He is member of the Computer Vision and GAVAB research group at Universidad Rey Juan Carlos. He is also an external member of the Computer Perception Group at Universidad Politécnica de Madrid. His research interests include image alignment, face image analysis, object detection and efficient computer vision.

Luis Baumela received the BS and MS degrees in 1989 and the PhD degree in 1995, all in computer science from the Universidad Politécnica de Madrid. From 1989 to 1992, he was an engineer at Telefónica's R&D labs. From 1997 to 2016, he has been an Associate Professor, and since 2016 he is a Professor of computer science at the ETSI Informáticos of the Universidad Politécnica de Madrid, where he leads the Computer Perception Group. His research interests include image alignment, face image analysis, and medical imaging.