Proceedings of the 2009 IEEE Symposium on Computational Intelligence
in Security and Defense Applications (CISDA 2009)

Robust Line Extraction Based on Repeated Segment Directions on Image Contours

Andres Solis Montero, Amiya Nayak, Milos Stojmenovic, Nejib Zaguia

Abstract— **This paper describes a new line segment detection and extraction algorithm for computer vision, image segmentation, and shape recognition applications. This is an important pre processing step in detecting, recognizing and classifying military hardware in images. This algorithm uses a compilation of different image processing steps such as normalization, Gaussian smooth, thresholding, and Laplace edge detection to extract edge contours from colour input images. Contours of each connected component are divided into short segments, which are classified by their orientation into nine discrete categories. Straight lines are recognized as the minimal number of such consecutive short segments with the same direction. This solution gives us a surprisingly more accurate, faster and simpler answer with fewer parameters than the widely used Hough Transform algorithm for detecting lines segments among any orientation and location inside images. Its easy implementation, simplicity, speed, the ability to divide an edge into straight line segments using the actual morphology of objects, inclusion of endpoint information, and the use of the OpenCV library are key features and advantages of this solution procedure. The algorithm was tested on several simple shape images as well as real pictures giving more accuracy than the actual procedures based in Hough Transform. This line detection algorithm is robust to image transformations such as rotation, scaling and translation, and to the selection of parameter values.**

Keywords— **Line detection, edge image, segmentation, images contours, linearity, shape recognition.**

I. Introduction

A straight line is the simplest basic shape and it is the topic of discussion here. Digital straight lines are image representations of straight lines. More precision would even require adding 'segments' to their full name: digital straight line segments. For brevity, they will be called simply lines in this article. Lines are one of the most common elements in images: indeed many basic shapes such as triangles, trapezoids, parallelograms, etc are composed of lines. They can be found in every human-made structure and picture. Once detected, they can be used as tools to identify more complex objects such as tanks or aircraft in images generated by satellites or unmanned aerial vehicles.

The problem of line detection and extraction in images has important applications in the fields of computer vision and image processing. Images are usually a composition of basic shapes; all of them together give us a sense of objects that can be recognized by humans. Combinations of basic shapes such as lines, squares, triangles, circles, ellipses among others create complex shapes that characterize real objects.

Lines are special cases of edges (or digital edges), which represent object boundaries and sharp changes in pixel colours or intensities in images. Edge detection is an important and well studies topic in literature. Canny [1], Sobel [2] , and Laplace [3] are some of the most known and used algorithms for extracting edges in digital image processing.

Our main goal is to find an algorithm capable of detecting significant line segments in images. It will show connected segments or portions of edges that can be considered as (straight) line segments, including their endpoints. These segments would help us recognize basic shapes in the image (this topic is out of the scope of this article). For example, for an image in Fig. 1a), our desired output is in Fig. 1b) which, in this case, rightly corresponds to the polygonization of the shape that is polygonal.

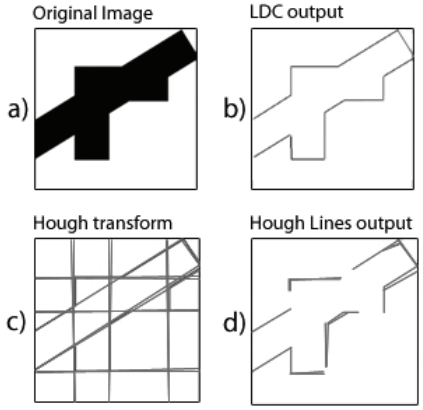

Fig.1. a) Original Image; b) Output of our proposed *LDC* algorithm; c) the Hough transform line detection algorithm output; and d) its OpenCV implementation, Hough lines output, which detects line segment endpoints.

We follow *3E* principle in our design: *easy* (to understand), *effective* (return accurate results), and *efficient* (fast), and managed to describe on such algorithm.

The existing literature on line detection concentrates around Hough transforms algorithms and its variants. They

Andres Solis Montero is with School of Information Technology & Engineering, University of Ottawa, Canada (email: asolis@ site.uottawa.ca).

Amiya Nayak is with School of Information Technology & Engineering, University of Ottawa, Canada (email: anayak@site.uottawa.ca).

Milos Stojmenovic is with School of Information Technology & Engineering, University of Ottawa, Canada (email: mstoj075@site.uottawa.ca).

Nejib Zaguia is with School of Information Technology & Engineering, University of Ottawa, Canada (email: zaguia@site.uottawa.ca).

978-1-4244-3764-1/09/$25.00 ©2009 IEEE

require a pre-processing step (sophisticated Canny edge detector) to convert an input colour image to a binary image. OpenCV library has only one algorithm for line detection, which is Hough transform based. Hough transform based algorithms detect lines, not just line segments, as shown in Fig. 1c). An extra step is required to retrieve the line segment in the image instead of a whole line across the image. An OpenCV extension of *HT* to produce connected line segments with endpoints is called Hough Lines (*HL*), see Fig. 1d). However, the outcome in this example is not 'polygonal' as two consecutive line segments are often disconnected. Our study shows that *HL* is not fully effective.

Our new algorithm, called *LDC*, finds connected components of detected edges, and produces connected lines, see Fig 1-b. In other examples we also demonstrated that *LDC* has visually better results than HL (effective algorithm). We process the input image with normalization, Gaussian smooth, Laplace edge detection, and thresholding. These steps will produce a binary image representing the edges of the object inside the image. Connected components are extracted from binary image. We then extract the consecutive boundary pixels from each connected component that create the contours of the image. Contours are divided into short segments, which are classified by their orientation into 9 discrete categories. Finally, line segments are detected by finding consecutive sequences of segments of similar slope.

We made two important changes compared to *HL*. The extraction of edges from color image is done using much simpler Laplace operator instead of sophisticated Canny edge detector from OpenCV. *HL* maps every pixel of every edge to a curve in dual space and recognizes significant accumulation. Our line recognition from edges is based on the linearity of a series of consecutive short segments of an edge, along its contour. Contours are a sequence of consecutive boundary pixels that describe outer edges and forms for any shape and object. We did not find any line detection algorithm in literature that is based on contour traversal. Both steps are greatly simplified in our *LCD* algorithm, and we give its full description here, showing also its easiness to understand.

LCD is somewhat faster than the Hough Line algorithm. More precisely, extracting edges from input image takes similar time. Our *LCD* algorithm is much faster in extracting line segments from edges than Hough lines solution. It is therefore an efficient (fast) solution.

Our new Line Detection Algorithm using Contours (called the *LDC* algorithm) is explained in Section III. It covers each step involved in the process in order to retrieve the line segments from true colour images. Section IV is dedicated to the experimental results of the comparison between *LDC* and the built-in Hough Lines (*HL*) OpenCV1 implementation. This test among the algorithms is realized over a set of 30 random images with different resolution sizes. The two tests realized for the built-in OpenCV Hough transform based algorithm, uses a probabilistic Hough transform variant [12]. It returns line segments rather than the whole lines. This variant is more efficient that the classic Hough Lines algorithm [12]. Conclusion and references complete this article.

II. Related Work

There are a number of schemes studying whether or not an edge is a line. For example, several schemes for measuring linearity of point sets (edge pixels) are described in [11]. However, if two or more lines are located on the same connected edge, these algorithms do not discover them.

Our literature review identified only one widely used line detection approach, the Hough Transform [4] method. This method maps points (pixels) from their Cartesian image space *(x, y)* extracted by an edge detecting operator to a curve in polar coordinates *(r, θ).* OpenCV uses the transform where *r* are *θ* are the distance from the origin, and the angle of the normal to a line passing through *(x, y),* respectively. These polar coordinate curves create accumulations of points in this space, of which local maxima values correspond to existing lines in the image. This method uses high complexity operations like sine and cosine calculations for each pixel. The other methods are mostly variations and extensions [1][5] of this classical one, mostly to improve their execution time. Other implementations utilize a convolution matrix and eigenvalue analysis of pictures [6].
Hough Lines uses a progressive probabilistic Hough Transform (*PPHT* [15], [17]) in order to considerably reduce the computation needed from the classical method and also give information on segment endpoints. Only a fraction of the points are accumulated, and a maximum gap parameter between lines segments is applied to connect otherwise isolated pixels into connected line segments. Other parameters include minimum gap between segments, minimal length of line segments and parameters related to the accumulator space.

III. Line Segment Detection and Linearity

This section will cover our *LDC* (Line Detection from Contour) algorithm in order to retrieve straight line segments from colour images with linear execution time in number of pixels in the image. The algorithm was implemented with OpenCV and EmguCV 2 using the C# language on the .Net platform. The experimental comparison results data between *LDC* and the two tests of the OpenCV built-in Hough Line algorithm – *HL1* and *HL2*- are in the following Section IV.

[1] OpenCV (Open Source Computer Vision) is a computer vision library originally developed by Intel. It is free for commercial and research use under a BSD license.

2 EmguCV is an Open Source cross platform – Gnu Linux, Mac, and Windows - .Net wrapper to the Intel OpenCV image-processing library. Allowing OpenCV functions to be called from .NET compatible

A. Overview of the LDC Algorithm

Our algorithm uses any (colour) image as input and produces a set of linear segments as output. Each segment is defined by two integer points (x_0, y_0) and (x_1, y_1) in the image following a discrete linearity criteria classification. More precisely, the input of our algorithm consists of: the true colour image (of any resolution) and 5 configuration parameters (to be discussed in appropriate sections). They are labelled as follows:

- Normalize (*N*),
- Gaussian kernel size (*GKS*),
- Laplace aperture size (*LAS*),
- Threshold (*TH*), and
- Minimum length contour (*MLC*).

The output of our *LDC* algorithm is a list of recognized connected lines in the image, each with its beginning and ending pixels, and all the intermediate pixels, with single pixel width. Consecutive line segments share endpoint.

In order to retrieve these segments, image processing algorithms that are commonly used in computer vision are applied to the images. Our *LDC* Algorithm is composed of 7 steps: Gray scale conversion, Normalization, Gaussian Smooth, Laplacian operator, Threshold, Contours Extraction, and Segmenting Contours. The first 6 steps used are well known image processing algorithms and the last step, Segmenting Contour, is our implementation to extract straight line segments of the objects inside the image. It is the main novelty of this article.

The first step of our algorithm (conversion from colour to gray scale image) doesn't use any configuration parameter. Normalize (*N*) is a Boolean parameter which decides whether or not we want to apply normalization process to the input image. *GKS* and *LAS* are the sizes of the Gaussian's and Laplace's kernels, respectively, to convolute the image. *TH* is the intensity value used to extract the edges after the Laplace filter was applied, and therefore it produces binary image as its output. Finally the *MLC* is the minimum amount of pixels that a contour must have to be accepted as a line in our final Segmenting Contour step.

We will first briefly describe the main idea. The colour input image is passed through several image processing algorithms. The image is converted to gray scale; possibly normalized, smoothed, edges are extracted and thresholded to produce a binary image input. Contours are then created for each candidate edge. After these steps, we scan the list of points of the contours with time complexity $O(n)$ (where n is the total number of pixels in all of the contours), looking for segments, or sections of the list of pixels which gives the notion of linearity. The final result will be a list of line segments; each segment will be represented by a starting and ending pixel position in the image space.

B. Edge Detection from colour images

Our first step is to convert the input colour image to a gray scale image using CvGray, which is an OpenCV built-in function [10]. The value of the gray pixel is a number between 0 and 255 and it's computed as the 29.9% of the red component, 58.7% of the green, and 11.4% of blue channel of the same pixel in the input image [9][12].

The normalization procedure is optional in our algorithm. If *N* is 'true' then the image is normalized to the range [0,255]. For example, if the input image has intensity range [70,201], normalization is done by first subtracting 70, then multiplying by 255/201 and rounding. Edge detection may be difficult if the image is affected by illumination problems, and normalization creates higher intensity contrasts. However sometimes it may be somewhat detrimental, introducing noise in the image.

The next two steps, Gaussian smooth and Laplace edge detection are implemented by convoluting the image (applying an appropriate filter) [13] [12]. The Gaussian smooth step is used to remove noise in the image and leave only the main contours. This blur process has the property of reducing noise and leaving the more important boundaries of the object in the image. The Gaussian kernel matrices for computing the Gaussian blur/smooth are defined by their size (small odd number *GKS* between 1 and 19).

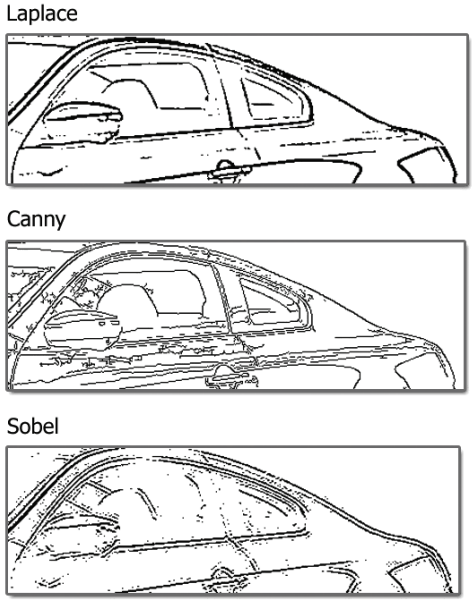

Fig.2. Edge detection algorithm examples: Laplace, Canny and Sobel. We applied a binary threshold to each image for better view of the edges.

After several tests on edge detection algorithms like Sobel, Canny, Predefined Convolution kernels, and the Laplacian operator, we found that the Laplacian operator was the one that produced the best results for our goal. It had sharper edges describing the boundaries of objects. This is illustrated on an example in Figure 2. Also, it only has one parameter to configure, the aperture size of the kernel, to apply to the image. This aperture size parameter *LAS* must be 3, 5 or 7 [12], and defines $k \times k$ kernel for $k = LAS$. This method will make the image zones brighter where edges are found.

After these steps, we apply a binary threshold *TH* to create a binary image. All the pixels with intensity values

greater than or equal to *TH* become white, representing brighter areas in the image (edges). The other pixels become black.

C. Extracting Contours

We then retrieve the list of all contours, external and internal using the findContours method of the OpenCV library. Contours are a list of sorted point's pixels inside the image where each pixel has information about the next neighbouring point. These contours surround the different white connected components in the binary input image. Because of the complex object we might find inside an image, we are going to retrieve all the internal and external contours in order to analyse them. The points inside the contours are clockwise oriented. As part of our algorithm, we introduced a threshold parameter *MLC,* to consider only contours with at least *MLC* pixels. It will filter out too short lines.

D. Repeated Segment Directions on Image Contours

Our algorithm for this step identifies line segments with any orientation inside the contours in time complexity $O(n)$, where n is the number of points in the contours we are analysing. Therefore, this final step does not compromise the execution time for the whole line segment detection algorithm. The total time complexity is dominated by the pre-processing steps that require processing of every pixel in the image, and has time complexity $O(r^2)$, where r is the number of rows/columns in the image.

1) Overview of the algorithm: Our next and final step is to analyse the list of points on each contour in order to find straight lines under a discrete linearity criterion. This produces an incomplete polygonization: the pixels not recognized as part of any line as dropped, while others are listed as line segments between two endpoints.

We will process the contour in the order given by the OpenCV built-in findContours algorithm. Each contour is divided into fragments. Each fragment contains Δ consecutive pixels. Neighbouring fragments share endpoints. A line segment consists of several consecutive fragments. Some fragments will not be associated with any line segment, and each fragment may be part of only one line segment. In our implementation, we used a fragment size of $\Delta=5$ pixels. The rest of the description is dependent on this choice; other choices require designing appropriate fragment classification scheme before detecting line segments. Because of sharing endpoints, each fragment has $\Delta+1$ pixels, including endpoints.

Our algorithm will classify each fragment using its slope, and will then attempt to identify repeated patterns of the same slope to create a (straight) line segment. A segment will keep all the information about the contour pixels and two main points: the starting pixel (x_0, y_0) and the ending pixel $(x_1, y1)$. These endpoints are the first pixel of the first fragment and the last pixel of the final fragment in detected line segment. Each fragment will be classified according to its slope into nine possible categories 0-8. This produces an integer sequence of line slopes for further processing.

2) Slope classification and fragmentation: Our classification is shown in Figure 3. Diagonal lines are discretized to produce a division of an 11x11 matrix into 9 areas, labelled 0-8. The central 3x3 region is labelled 0 and represents class 0. Classes 1-8 correspond to eight approximate directions respectively: east, south-east, south, south-west, west, north-west, north, and north-east. Each fragment is classified in the following way. The beginning of the fragment is placed in the centre of the 11x11 matrix. The fragment is classified according to the location of its endpoint in this matrix, and receives the label associated with it. Class 0 is a 'neutral' direction, or a lack of direction. It represents fragments that do not make sufficient progress in any direction, and can be treated as loops. This class was introduced due to the irregularity and noise in the edge detection steps. Progress with distances of at least 2 from the center in one of the directions suffices to classify slopes into one of the 8 classes. The location of intermediate pixels does not impact the classification. Fig. 4 shows a fragment of class 8, between two black pixels.

Figure 5 illustrates the classification process for fragments. The six fragments shown are classified as 88828. This representation will help us explain some of the criteria above.

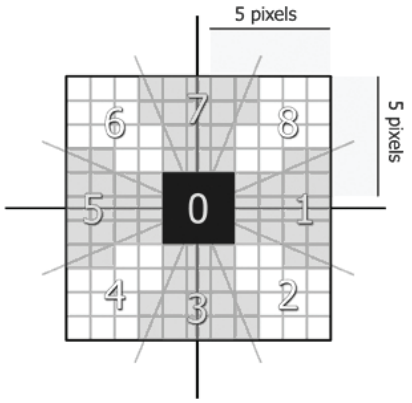

Fig.3. The 9 classes classification matrix used in our LDC algorithm.

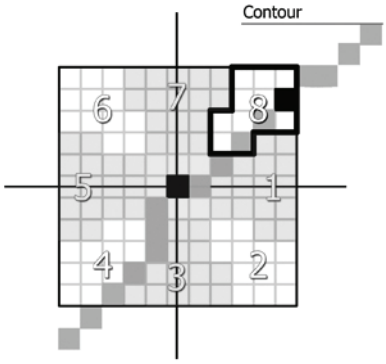

Fig.4. Two black pixels inside the contours indicate the starting and ending pixels of a fragment of class 8.

3) Merging Fragments into segments: The sequence of fragment classification values will be further processed to identify line segments, corresponding roughly to a sufficient number of repeated fragment slopes. Our exact criteria to detect line segments from fragments are:

A). A segment has at least *X* fragments of the same slope classification.

B). Two segments with the same slope classification, separated by less than *X* fragments with different classification, will create a single segment with all the fragments from and between both segments.

C). The 0 slope classification is a neutral position for the algorithm. It will not interfere in the classification of any segments; it will be treated as being same class as the previous class in the sequence.

D). If the starting fragment and final fragment of a contour have the same classification then they join in a same segment with length equal to the sum of both.

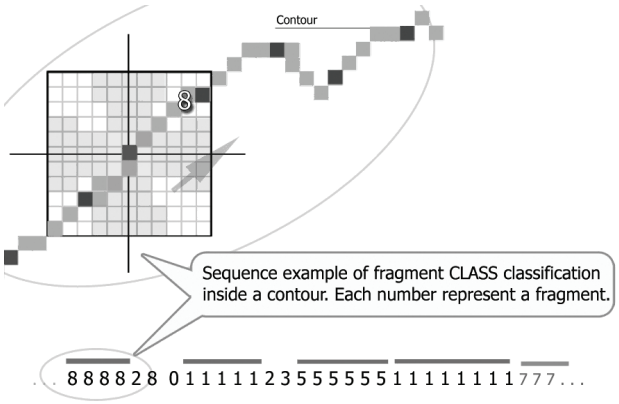

Fig.5. Line sequence example of fragments produced by LDC algorithm. Lines above the class number represent line segments recognized in the patterns. The circled area represents how the matrix is used to classify them.

Our algorithm uses an *X* value of 3; this means that at least 3 consecutive class values are needed to declare it a segment. Fig. 6 shows some examples of extracting line segments from sequences of classified fragments. In Fig. 6a), 4 lines segments are detected, based on rule *A)*, repeated patterns of 8 (line segment 888888), 1(line segment 11111), 5 (segment 555555), and 1 (segment 11111111), respectively.

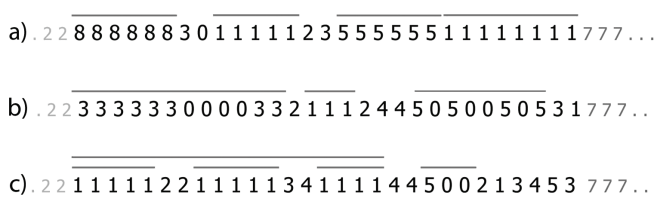

Fig. 6. Line segments extraction from sequences of classified fragments. In example a) 4 line segments are extracted using rule *A)*. Example b) uses rules *A)* and *C)* extracting 3 line segments. Example c) extracts 2 line segments using the rules *A)*, *B)*, and *C)*.

In the Fig. 6b), the sequence 0000 is inserted into the larger fragment 333333000033, using rule *C*). Similarly, 0, 00, and 0 are inserted into 5555 to produce line segment 50500505. Since the endpoint of fragment of class 0 is a neighbour of its starting point (distance 1), this naturally corresponds to a continuation of the same line segment, ignoring some 'noise' around some pixels.

In Fig. 6c), rule *B*) is applied to produce a single segment 111112211111341111 out of the initially produced segments 11111, 11111 and 1111, since the inserted 22 and 34 which are used to merge them are shorter than minimal length *X=3* needed to 'break' the sequence. In the same row, 500 is also declared a line segment (rule *C*). This corresponds to the current implementation and is subject to discussion, and can be changed later. We opted to preserve simplicity and to allow segment extensions by single pixel distances per fragment. It can be also perceived as line segments with somewhat shorter length than the minimal specified which does not necessarily mean erroneous classification.

The accuracy of any line detection algorithm is evaluated by subjective judgment. Nevertheless the appealing results are hard to deny. Fig. 7 shows the output for one input example. More examples are in Fig. 8.

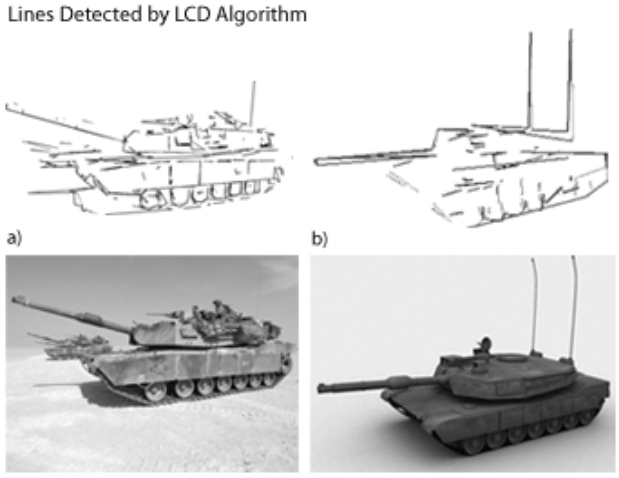

Fig.7. Line Segments detected by LDC from two tank pictures. Fig 7a) is a real photo and b) is a 3D model of a tank.

IV. Experimental Data

We compared our newly proposed *LDC* algorithm with what appears to be a widely accepted way to extract line segments in literature using the OpenCV library. Since the Canny edge detector was regularly associated with the Hough Lines [13], the comparison was made with the *GCH* algorithm, which represents a sequence of *Gray Scale* + *Canny* + *HL* (the OpenCV built-in *PPHT* Hough Lines) algorithm. In this implementation we used the Open Sources OpenCV 1.0 library and the EmguCV 1.2.2.0 project, .Net wrapper to OpenCV, and the C# language. The complete

experiment was tested under a 32 bits Intel Mobile 2.0 GHz CPU technology with 1 GB RAM laptop.

Our goal was to measure the processing time and the sets the lines detected by both algorithms. Processing time was compared separately for two steps: from colour image to edge extraction, and from edge to line segment extraction. Both *LDC* and *GCH* are then divided into corresponding parts. *LDC=GB+CS*, where *GB* stands for the sequence of Gray scale conversion from colour images, normalization, Gaussian smooth, Laplacian operator and binary thresholding, and CS is a sequence of contour extraction (which includes finding connected components) and segmenting contours. *GCH=GC+HL* consist of *GC* (Gray scale conversion followed by Canny edge detection) and *HL* (Hough Lines algorithm) which doesn't use connected components or contours to extract lines and instead maps some pixels to the dual accumulator space.

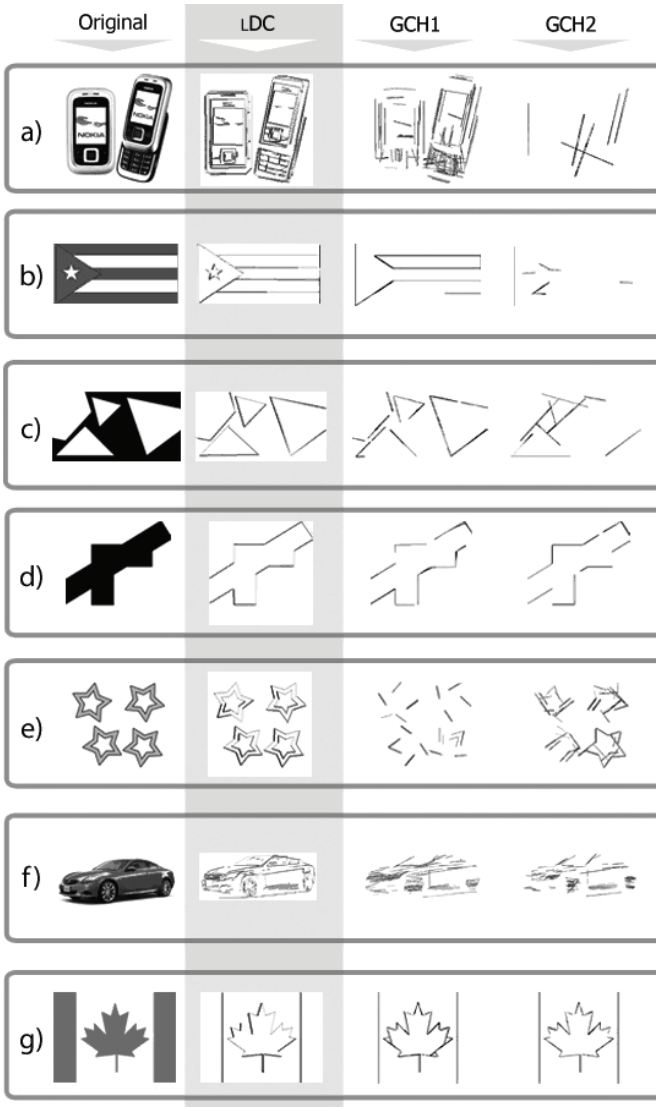

Fig. 8. Examples b), c), d), e), and g) are from the 30 image set used in the experimental data. The amount of lines detected by *LDC* is 46, 22, 24, 26, 160, and 54 respectively. Two parameter choices *HL1* and *HL2* had 13, 34, 28, 40, 62, and respectively 10, 12, 13, 82, 65 lines. Examples a) and f) show us real pictures processed for the algorithms and how our solution is closer to human classification.

The outcome of *HL* (and consequently *GCH*) was very sensitive, for all images, to the parameter choices. Even small changes in some parameters have led to quite different number of lines detected. Human intervention for obtaining satisfactory outcome in *HL* was essential and time consuming. Larger number of parameters in *GCH* also contributed to this. The best set of parameter values is picture dependent. For fairer comparison, we included two parameter choices in *GCH* for each of 30 selected pictures (24 bit-depth, 96 dpi), and only one for *LDC*. One obvious observation from this exercise was that LDC proved to be significantly more robust solution than GCH, due to significantly less sensitivity of the outcome to changes in parameter values. The picture set selected consisted of colors depth images with both planar shapes and real environment pictures. The spatial resolution of these images ranges from 256x256 to 2048x1536. Here we show average results for the set of 30 images.

The mean processing time of our *LDC* algorithm was 0.11 seconds with 0.11±0.04 of a 95% confidence level. The *GCH* mean was 0.15 seconds, with and 0.15±0.05 95% confidence interval. Therefore there exists a small but statistically insignificant difference in favour of our *LCD* algorithm since the confidence intervals overlap.

The next comparison measured processing times starting from binary images. Our *CS* is about twice faster than the competing *HL* algorithm, and the difference is statistically significant. The average processing time of *CS* was 0.03 seconds, with 95% confidence interval of 0.03±0.015. The corresponding means and confidence intervals for *HL* were 0.07 seconds, and 0.07±0.028, respectively. The processing dominant times to convert color image to binary were identical: 0.08±0.029 for *GB* and 0.08±0.025 for *GC*.

Our next and most important experiment compared the accuracy of the line detection process. Figure 8 shows 7 out of 30 images from our experimental set. In some examples the actual number of lines is obvious. Images in Fig 8c), d), e), and g) have 12, 13, 80 and 27 lines respectively, and our *LDC* algorithm retrieves exactly twice as many lines (24, 26, 160 and 54) because of easily removable duplication. The number of lines extracted by the *GCH* is quite different, sensitive to parameter choices, and in disagreement with human observation, mostly due to missing many line segments. Human classification might yield about 20 line segments for Fig. 8b). *LDC* retrieves 46 (or 23 actual ones), while two parameter choices for *GCH* yielded only 13 and 10 lines, despite the edges being intuitively obvious. The *LDC* algorithm also produces more appealing line segments for the car in Fig. 8f) and cellular phones in Fig. 8a) compared to *GCH* Hough transform based algorithm. Overall, better results for our method were obtained for every image from our set.

Finally, we computed the linearity *SNZ* [11] value for each of the line segments detected by *LDC* in the image set. The *SNZ* linearity value range is from 0 to 1 where values

nearest to 1 mean highly linear. The value is computed based on coordinates of each pixel from a given line segment. Because of discretization, perfect 1 scores are rarely possible. The 95% confidence interval for the linearity values was 0.970335 ±0.006729754. This shows that detected line segments are indeed representing lines, with almost perfect linearity score and consequently small deviation.

V. CONCLUSION

There is no widely accepted criterion for judging whether or not an edge is a (straight) line. The judgment is subjective or is based on usefulness in a particular application. We demonstrated that our algorithm achieves better accuracy in terms of the stated problem, and are closer to human classification than the Hough Lines algorithm for different input parameter configurations. Although Hough transform might be a simple algorithm to compute lines theoretically, it's not so simple in practice. Our algorithm doesn't need high complexity operations like sine and cosine, and finding local maxima points in the accumulation space. The creation of the accumulator space is a time consuming process, since each pixel is converted to a digital line. Our algorithm is considerably simpler to implement and understand, and is faster, especially in the line extraction step.

Our *LDC* solution appears also robust to any orientation and slope and invariant to image transformations such as scaling, rotation and translation. *LDC* also has smaller set of parameters. Compared to *HL*, the set of parameters used in *LDC* is more predictable and can be applied almost independently on an input picture, while *HL* requires fine tuning of parameters for each picture to arrive at satisfactory line extraction.

LCD algorithm can be further improved. Eliminating duplicate lines is easy. There are some pathological cases; for example, stair like structure may be declared a single line. The rules for accepting a line could be fine-tuned, including the change of variable X for minimal number of fragments to make a segment. The fragment size Δ may be increased and more directions could be considered. The insertion of neutral fragments into a line may be reconsidered. This and other improvements are left for the future work.

REFERENCES

[1] Canny, John. "A Computational Approach to Edge Detection". IEEE Trans. Pattern Analysis and Machine Intelligence. Vol 8, pp: 679-714. 1986.

[2] Sobel, Irwin. Feldman, Gerald. "A 3x3 Isotropic Gradient Operator for Image Processing". Unpublished 1968, cited orig. in Pattern Classification and Scene Analysis. Duda, Richard O. Hart, Peter E. Wiley, pp: 271. 1973.

[3] Gonzalez, Rafael C. Woods, Richard E. "Digital Image Processing" 3rd edition. Prentice Hall Inc, pp: 976. 2007.

[4] Duda, Richard O. Hart, Peter E. "Use of the Hough Transformation to Detect Lines and curves in pictures". Comm. ACM. Vol 15, pp: 11-15. 1972.

[5] Fernandes, Leandro A. Oliveira, Manuel M., "Real-time line detection through an improved Hough transform voting scheme". Pattern Recognition. Vol 41, issue 1, pp: 299–314. 2008.

[6] Guru, D. S. Shekar, B.H. Nagabhushan, P. "A simple and robust line detection algorithm based on small eigenvalue analysis". Pattern Recognition Letters. Vol 25, issue 1, pp: 1 – 13. 2004.

[7] Marr, David. Hildreth, Ellen.C., "Theory of Edge Detection". RoyalP(B-207), pp: 187-217. 1980.

[8] Goshtasby, Arthr A. "2-D and 3-D Image Registration ". Wiley-IEEE, pp: 280. 2005.

[9] Qiu, Peihua. "Image Processing and Jump Regression Analysis". John Wiley and Sons, pp: 305. 2005.

[10] Bradski, Gary. Kaehler, Adrian. "Learning OpenCV: Computer Vision with the OpenCV Library". O'Reilly, pp: 555 2008.

[11] Stojmenovic, Milos. Nayak, Amiya. Zunic, Jovisa. "Measuring linearity of planar points sets". Pattern Recognition 41, pp: 2503-2511 2008.

[12] Emgu CV project API Reference and Documentation. [Online] Available: http://www.emgu.com/wiki/index.php/Main_Page.

[13] Shapiro, Linda. Stockman, George. "Computer Vision" 1st edition. Prentice-Hall Inc, pp: 608. 2001.

[14] Martí, Joan. Benedi, Jose M. Mendoza, Ana M. Serrat, Joan. "Pattern Recognition and Image Analysis" 3rd Iberian Conference, IbPRIA. 2007.

[15] Kiryati, Nahum. Eldar, Yuval. Bruckstein, Alfred.M. "A probabilistic Hough transform". Pattern Recognition. Vol 24, issue 4, pp: 303 – 316. 1991.

[16] Suzuki, Kenji. Horiba, Isao. Sugie, Noboru. "Linear-time connected-component labeling based on sequential local operations". Computer Vision and Image Understanding. Image Underst. Vol 89, issue 1, pp: 1-23. 2003.

[17] Chang, Fu. Chen, Chun-Jen. Lu, Chi-Jen. "A linear-time component-labeling algorithm using contour tracing technique". Computer Vision and Image Understanding. Vol 93 issue 2, pp: 206-220. 2004.

[18] Galambos, C. Matas, J. Kittler, J. "Progressive Probabilistic Hough Transform for line detection". Computer Vision and Pattern Recognition. Vol. 1, pp: 554-560. 1999.