Proc. International Conference on Advanced Computing, Networking and Security (ADCONS 2011), Dec. 16 – 18, 2011, Springer Publications

Detection of straight lines using Rule Directed Pixel Comparison (RDPC) Method

Anand T V[1], Madhu S. Nair[2], Rao Tatavarti [3]

[1] Department of Computer Science, Rajagiri College of Social Sciences, Kalamassery, Kochi 683104, Kerala, India
e-mail: anandtvenu@gmail.com

[2] Department of Computer Science, University of Kerala, Kariavattom, Thiruvananthapuram 695581, Kerala, India
e-mail: madhu_s_nair2001@yahoo.com

[3] Department of Civil Engineering, Gayatri Vidya Parishad College of Engineering, Madhurawada, Visakhapatnam 530048, Andhra Pradesh, India
e-mail: rtatavarti@gmail.com

Abstract. A simple and efficient algorithm, based on Rule Directed Pixel Comparison, RDPC method, is proposed for detecting straight line segments in an edge image, based on certain specific rules, scanning column wise and labelling done in accordance with the application of rules. Four rules are formulated to detect the edge pixels which are part of straight lines with each straight line having two threshold values, minimum line length and minimum line level length. A comparison of the resultant image is made with Standard Hough Transform and the algorithm proposed by Guru et al (2004).

1 Introduction

Object recognition and scene analysis in computer vision requires detection and extraction of straight edges or lines in image, which is a well known and challenging problem in image processing. Towards robust line detection, significant work has been carried out for the last two decades, which can be broadly classified into four categories. Statistical based (Mansouri et al., 1987 [1]; Guru, et. al., 2004 [2]), gradient based (Burns et al., 1986; Nelson, 1994 [3]), pixel connectivity-edge linking based (Nevatia and Babu, 1980 [4]) and Hough Transform (HT) based (Duda and Hart, 1972 [6]) models.

The statistical based method of Mansouri et al. (1987) [1] is a hypothesize-and-test algorithm to extract line segments of specified lengths by hypothesizing their existence utilizing local information. Line detection algorithm of Guru et al. (2004) [2] is also a statistical method in which Eigenvalue of the edge pixels covered by a mask of an appropriate size is estimated. The mask is moved pixel by pixel from the top left corner to the bottom right corner, so that each edge pixel has a number of Eigenvalues because of the overlapping of masks. Out of these Eigenvalues, the smallest one is assigned and is compared with pre-defined threshold value to consider it as a straight line. If the smallest Eigenvalue is smaller than the predefined value it will be a part of straight line. However, the choosing of an appropriate size of the mask and presence of excessive noise pixels can damage the performance of this method.

The gradient approach of Burns et al., (1986) [3], explores the gradient magnitude and orientation properties of, each pixel for the purpose of detecting line segments in an image. The principle of this approach is to utilize the gradient direction to partition the image into a set of support regions. This method is further improvised by Nelson (1994) [5]. However, Nelson's (1994) algorithm sometimes fails to identify even linear edges, if they happen to be parts of curve segments.

Algorithms based on the idea of finding out local edge pixels, linking of the found pixels into a contour on the basis of proximity and orientation, and then segmenting the contours into relatively straight line pieces were also proposed. The method proposed by Nevatia and Babu (1980) [4] is a pixel connectivity-edge linking algorithm widely used in several applications which involve extraction of continuous line segments. The main advantage of pixel connectivity-edge linking algorithms is that the connectivity among all the pixels which are identified as linear edge pixels is very much ensured. Because of this fact, these methods are indeed said to outperform other line detection methods.

The Hough transform (HT) is used to find imperfect instances of objects within a certain class of shapes by a voting procedure. This voting procedure is carried out in a parameter space, from which the object candidates are obtained as local maxima in a so-called accumulator space that is explicitly constructed by the algorithm for computing the Hough transform. It has some inherent limitations such as high computing time, unwieldy memory requirement, low peaks for short lines and incapability in preserving edge pixel connectivity. In order to enhance the applicability of the standard Hough transform to various domains, several improved versions were proposed. They are Fast HT (Li et al., 1986 [7]), Adaptive HT (Illingworth and Kittler, 1987 [8]), Combinatorial HT (Ben-Tzvi and Sandler, 1990 [9]), Hierarchical HT (Princen et al., 1990 [10]) and Multi-resolution HT (Atiquzzaman, 1992 [11]). In all these improved techniques, the complexity involved in the process of deciding local peaks is reduced. However, these models still require a complete scan of the entire mage for pixel transformation.

Against this background, we describe a simple and efficient algorithm for straight line detection. As discussed in the following section, the algorithm extracts straight lines from edge image using newly proposed rules. First we give labels to each and every edge pixels by applying the rules. The label specifies whether a pixel is a part of a line, if yes it also tells in which orientation. This method uses four vectors for storing the labels.

2 Rule Directed Pixel Comparison (RDPC) Method

RDPC method is a method for transforming a given edge image into an image which contains only the linear edge pixels, that which is a part of straight lines. In this method, detection of straight lines is based on certain newly proposed rules.

This method has two stages. In the first stage, on each non-zero pixel in the edge image, the proposed rules are applied and directions are labelled. In the second stage, the extraction of straight line is carried out according to those labels.

The proposed method scans the input image from top left corner to bottom right corner in column major form. For each pixel, decision is made on the orientation of straight lines and labels them according to their orientations. A straight line may have any of the four orientations: horizontal, vertical, main diagonal and anti diagonal; besides which there are slight varying orientations.

This method uses four vectors to indicate the orientation of each non zero edge pixel in the straight line: horizontal, vertical, main diagonal and ant diagonal. Each vector has size of *1 X N*, where *N* is the total number of edge pixels in the edge image. These vectors have either one or zero value in corresponding positions. This act as the label of the pixel.

At first, an assumption is made that all the pixels of the image are not a part of a straight line; i.e., vectors are initialized to zero. During the processing of each pixel and its decision making, the values in the vector may change from '0' to '1', if it is a part of a straight line. For example, if we find that one pixel, *f(i,j)* is a part of a horizontal line then we change the label of corresponding position in horizontal vector from '0' to '1'. Likewise, it is done for all types of lines in their corresponding vectors.

2.1 PHASE I: Labelling using newly proposed rules

For applying the rules, we consider one pixel at a time in column major form; taking each column from left to right. The labelling of each pixel is done by the application of newly proposed rules. Before processing we pad the edge image on all sides.

Suppose *f(i, j)* is an edge pixel, where *f* is the padded edge image, and *i* and *j* specify the (x, y) position of the current pixel. The pixel and its neighbour are shown in Fig.1.

f(i-1, j-1)	f(i-1, j)	f(i-1, j+1)
f(i, j-1)	f(i, j)	f(i, j+1)
f(i+1, j-1)	f(i+1, j)	f(i+1, j+1)

Fig.1. Neighbours of f (i,j)

There are four rules that are essential for the labelling of edge pixels in straight lines in the proposed method. *First rule* is pertains to the merit of the straight lines to be detected by this method. *Second and third rules* are help in the detection of starting pixel as well as the orientation of the lines. *Fourth* one is a special rule which ensures the detection and relative labelling of each and every edge pixel.

2.1.1 First rule

All straight lines that can be extracted should have two threshold values: minimum line length *(min_line_length)* and minimum line level length *(min_linelevel_length);* $1 < min\_linelevel\_length \leq min\_line\_length$. The *min_line_length* specifies the minimum number of edge pixels in considering the object as a straight line that can be extracted by this method; whereas the *min_linelevel_length* is the specific number of edge pixels present in each level of the extracted straight line.

2.1.2 Second rule

If *f(i, j)* is part of a straight line and the six neighbours of *f(i,j),* viz., *f(i-1,j-1), f(i-1,j), f(i,j-1), f(i+1,j-1), f(i+1,j) and f(i-1, j+1)* are non edge pixel, then *f(i,j)* is the starting pixel of a straight line as shown in Fig 2.

If all the neighbours of *f(i,j)* are non edge pixel then it may be a noise pixel. If one out of these six is an edge pixel, then *Rule 4* is to be used for further processing.

2.1.3 Third rule

In order to check the orientation of the straight line to which the current pixel belongs, a comparison of participation is made with the subsequent pixels in all the four directions, vertical, horizontal, main diagonal and anti diagonal; provided, only one direction at a time.

For checking whether a pixel, *f(i,j)*, is a part of a horizontal line we assess pixel by pixel, whether it is edge or non edge, in horizontal direction, up to *f(i,j+n)* where $n = min\_line\_length$. If $n < min\_line\_length$, then there is no perfect horizontal line; it may be a horizontal line with several levels, as in the Fig 2(a)**.** If the last pixel of the current level of the particular line is *f(p,q)*, then the comparison level is changed to *f(p-1,q+1)* or to *f(p+1,q+1);* and the comparison is proceeded to $(q+n)^{th}$ pixel where $n \geq min\_linelevel\_length$. If there are more than two levels then, the change of level should be either in an increased or in a decreased level of the *x* position of *f(i,j)* and never in a zigzag manner. When the total number of pixels in all levels is greater than or equal to *min_line_length*, then there is no need of further comparison and change the label of the starting pixel *f(i,j),* in corresponding position in horizontal vector.

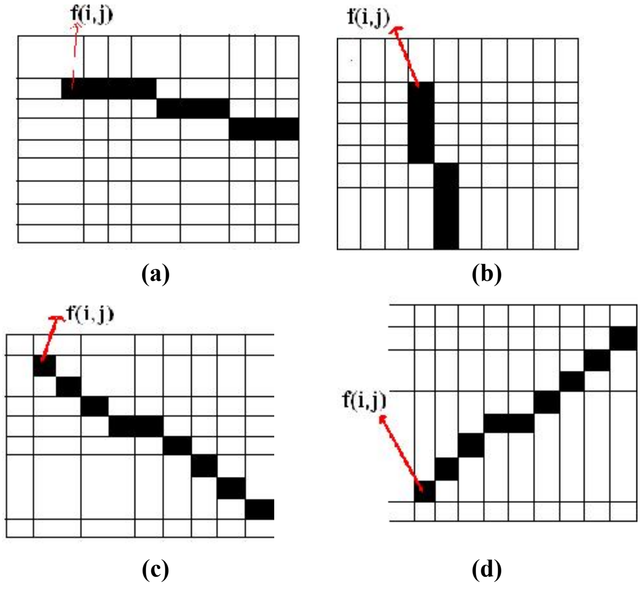


Fig 2: Example of four directions of lines (a) horizontal line (b) vertical line (c) main diagonal (d) anti diagonal. *f(i,j)* is our current pixel. In Fig (a), (b) and (c), the 6 neighbours of *f(i,j): f(i-1, j-1), f(i-1, j), f(i, j-1), f(i+1, j-1), f(i+1,j), f(i-1,j+1)* are non edge pixel so we can apply the *Rule 2 & 3.* But in Fig (d) *f(i-1,j+1)* is an edge pixel so we use *Rule 4*

The same procedure will follow in the case of vertical, main diagonal and anti diagonal straight lines; with some modifications in processing. For checking whether a pixel, *f(i,j)*, is a part of a vertical line we assess pixel by pixel, whether it is edge or non edge, in vertical direction, up to *f(i+n,j)* where *n* is equal to the *min_linel_length*. If *n< min_line_length*, then there is no perfect vertical line; it may be a vertical line with several levels, as in the Fig 2(b). If the last pixel of the current level of the particular line is *f(p,q),* then the comparison level is changed to *f(p+1,q-1)* or to *f(p+1, q+1)*; and the comparison is proceeded to *(p+n)*th pixel where *n* ≥ *min_linelevel_length.* If there are more than two levels then, the change of level should be either in an increased or a decreased level in *y* position of *f(i,j)* and never in a zigzag manner. When the total number of pixels in all levels is greater than or equal to *min_line_length*., then there is no need of further comparison and change the label of the starting pixel *f(i,j),* in corresponding position in vertical vector.

For checking whether a pixel, *f(i,j)*, is a part of a main diagonal line we assess pixel by pixel, whether it is edge or non edge, in main diagonal direction, up to *f(i+n,j+n)* where *n* = *min_line_length*. If *n< min_line_length*, then there is no perfect main diagonal line; it may be a main diagonal line with several levels, as in the Fig 2(c). If the last pixel of the current level of the particular line is *f(p,q)*, then the comparison level is changed to *f(p+1,q)* or to *f(p,q+1)*; and the comparison is proceeded to either *(p+n)*th or *(q+n)*th pixel, as the case it may be, where *n* ≥ *min_linelevel_length.* If there are more than two levels then, the change of level should be either an increase in *x* position or in *y* position of *f(i,j)* never both in a straight line. When the total number of pixels in all levels is greater than or equal to *min_line_length*., then there is no need of further comparison and change the label of the starting pixel *f(i,j),* in corresponding position in main diagonal vector.

For checking whether a pixel, *f(i, j)*, is a part of a anti diagonal line we assess pixel by pixel, whether it is edge or non edge, in anti diagonal direction, up to *f(i-n,j+n)* where n = *min_line_length*. If $n<$ *min_line_length*, then there is no perfect anti diagonal; it may be an anti diagonal line with several levels, as in the Fig 2(d). If the last pixel of the current level of the particular line is *f(p,q)*, then the comparison level is changed to *f(p-1,q)* or to *f(p,q+1)*; and the comparison is proceeded to either $(p-n)^{th}$ or $(q+n)^{th}$ pixel, as the case it may be, where $n \geq$ *min_linelevel_length.* If there are more than two levels then, the change of level should be either an increase in *x* position or decrease in *y* position of *f(i,j)* never both in a straight line. When the total number of pixels in all levels is greater than or equal to *min_line_length*., then there is no need of further comparison and change the label of the starting pixel *f(i,j),* in corresponding position in anti diagonal vector.

After applying the above rules 1, 2 and 3, we can decide whether any straight line is originating from the identified starting pixel.

2.1.4 Fourth Rule:

This rule is used to decide whether the current edge pixel is a part of horizontal/ vertical/ main diagonal/ anti diagonal line.

In order to decide the current edge pixel *f(i,j)*, as a part of horizontal line, the labels of edge pixels, out of the three neighbourhood pixels *f(i-1,j-1)*, *f(i,j-1)* and *f(i+1,j-1)* are reviewed. If *f(i,j-1)* is labelled as a part of a horizontal line then *f(i,j)* will be the part of that perfect horizontal line as in Fig 3(a). If *f(i-1,j-1)* or *f(i+1,j-1)* is labelled as a part of a horizontal line as in Fig 3(b) and 3(c), then *f(i,j)* will be a part of horizontal line only if the level which includes *f(i,j)* has *min_linelevel_length*. If we can't decide this edge pixel as a part of line, by the above conditions, then there is a chance *f(i,j)* is a starting pixel of a horizontal line. To confirm it as a starting pixel *Rule 3* is used.

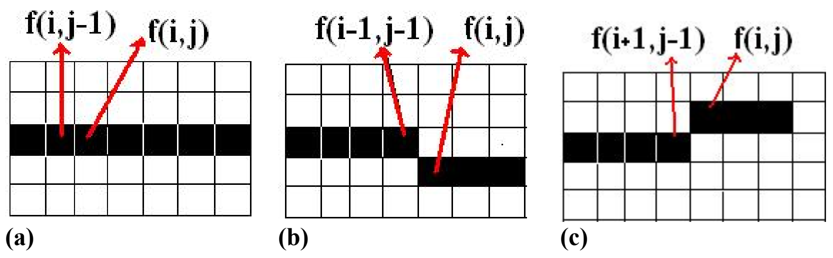


Fig 3: Example of horizontal lines. (a) Perfect horizontal line (b), (c) horizontal lines like step structure.

In order to decide the current edge pixel *f(i,j)*, as a part of vertical line, the labels of edge pixels, out of the three neighbourhood pixels *f(i-1, j-1), f(i-1, j) and f(i-1, j+1)* are reviewed. If *f(i-1,j)* is labelled as a part of a vertical line then *f(i,j)* will be the part of that perfect vertical line as in Fig 4(a). If *f (i-1,j-1)* is labelled as a part of a vertical line then *f(i,j)* will be a part of vertical line only if the level which includes *f(i,j)* has *min_linelevel_length* as in Fig 4(b). Here we use column major so *f(i-1,j+1)* is not yet labelled. To check whether there exists a line in that direction as in Fig 4(c) we need extra comparison. We compare pixels present in both directions, that is *f(i-1,j+1), f(i-2,j+1),… and f(i+1,j), f(i+2, j)…* along with *min_linelevel_length* and *min_line_length*. If we can't decide this edge pixel as a part of line, by the above conditions, then there is a chance *f(i,j)* is a starting pixel of a vertical line. To confirm it as a starting pixel *Rule 3* is used.

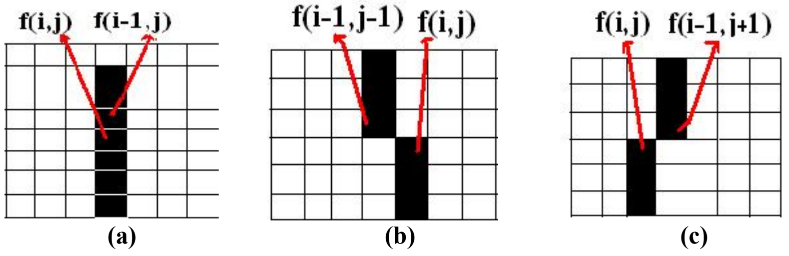


Fig. 4: Example of vertical lines. (a) Perfect vertical line (b), (c) vertical lines like step structure.

In order to decide the current edge pixel *f(i,j)*, as a part of main diagonal line, the labels of edge pixels, out of the three neighbourhood pixels *f(i-1,j-1), f(i-1, j)* and *f(i,j-1)* are reviewed. If *f(i-1,j-1)*, is labelled as a part of a main diagonal line then *f(i,j)* will be the part of that perfect main diagonal as in Fig 5(a). If *f(i-1,j)* and *f(i,j-1)* is labelled as a part of a main diagonal as in Fig 5(b) and 5(c), then *f(i,j)* will be a part of main diagonal only if the level which includes *f(i,j)* has *min_linelevel_length.* If we can't decide this edge pixel as a part of line, by the above conditions, then there is a chance *f(i,j)* is a starting pixel of a main diagonal. To confirm it as a starting pixel *Rule 3* is used.

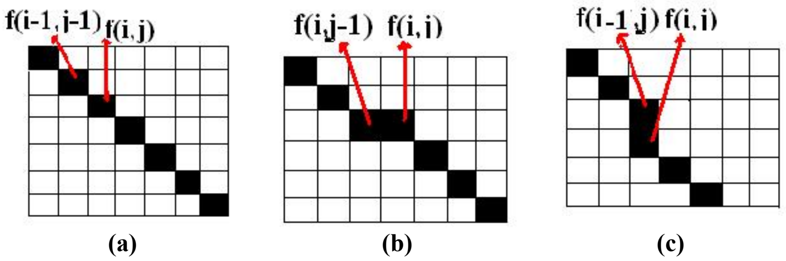


Fig. 5: Example of main diagonal lines. (a) Perfect main diagonal (b), (c) lines with several levels.

In order to decide the current edge pixel *f(i,j),* as a part of anti diagonal, the labels of edge pixels, out of the three neighbourhood pixels *f(i,j-1), f(i+1,j-1)* and *f(i+1,j)* are reviewed. If *f(i+1,j-1)* is labelled as a part of an anti diagonal line then *f(i, j)* will be a part of that anti diagonal as in Fig 6(a). If *f(i,j-1)* is labelled as a part of an anti diagonal then *f(i,j)* will be a part of anti diagonal only if the level which includes *f(i,j)* has *min_linelevel_length* as in Fig 6(b). *f(i+1,j)* is not yet labelled since we are moving column wise. To check whether there exists a line in that direction we use the label of *f(i+2,j-1)*, that is already given. If *f(i+2,j-1)* is a part of a anti diagonal line and *f(i+1,j)* is an edge pixel and the level which include *f(i,j)* has *min_linelevel_length*, then *f(i,j)* is a part of the anti diagonal line as in Fig 6(c). If we can't decide this edge pixel as a part of line, by the above conditions, then there is a chance *f(i,j)* is a starting pixel of anti diagonal. To confirm it as a starting pixel *Rule 3* is used.

If *f(i,j)* is a part of a straight line then we change the label according to the application of these above said rules.

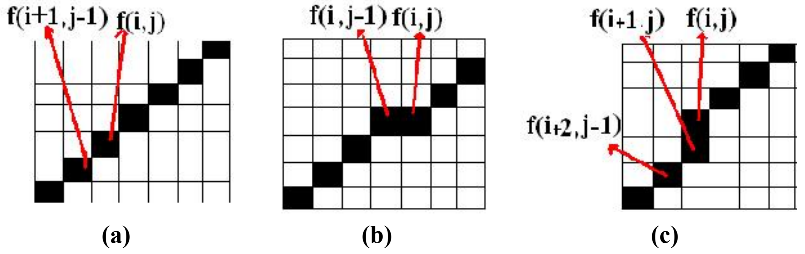


Fig 6: Example of anti diagonal lines. (a) Perfect antidiagonal (b), (c) lines with several levels.

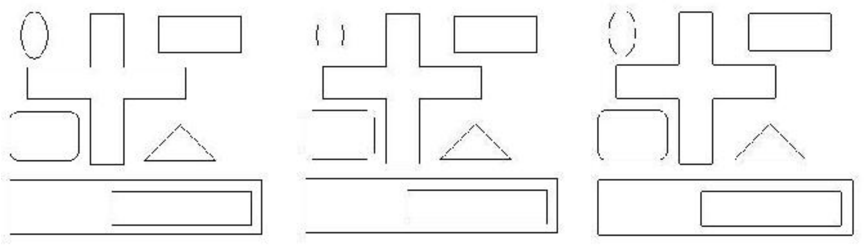
2.2 PHASE II: Extraction

The next phase is extraction of straight lines using the labels given for the edge pixels. Let *G* be our output image, which has same size of original edge image. The implementation of second phase begins with the initialization of all pixels of the output image *G* to zero and then scanning of the four vectors. If a pixel *f(i,j)* has a value 1 in corresponding positions in any of four vectors then we set the value one in *G(i,j)*.

3 Experimental Results

This section presents the results of the experiments conducted to show the performance of the proposed algorithm. The method has been implemented in the MATLAB on a Celeron-D 2.66 GHz with 1024 X 768 resolutions. Experiments on both synthetic and real images are conducted. We use the threshold value *min_linelevel_length*=3 and *min_line_length*=7. This value depends on the specific application. If we want to detect small lines and curves we decrease the values, but if need only lengthy lines this value should be large. In this experiment, the Eigenvalue threshold 0.1257 is used at a window size 7, as proposed in Guru et al (2004), for the extraction of straight lines.

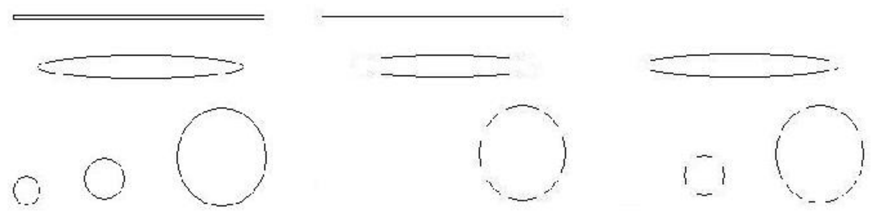
We compared our algorithm to the one that is proposed by Guru et al. (2004) with a synthetic image, Fig 7(a). There is a very conspicuous difference between the proposed one and that of Guru et al. Fig 7(b) is having more clarity in the case of straight line extraction, especially that of corners of the rounded rectangle and circle.



(a) **(b)** **(c)**

Fig 7: (a) synthetic image (b) the result of proposed method (c) the result of Guru et al.

In the comparison of second set of images, it is found that the two nearly placed parallel straight lines are not detected in their algorithm; while our algorithm detects them as shown in Fig 8. Besides that their algorithm detected too many parts of the circles and ellipse that are not straight lines, but ours detect only the straight part of the circles and ellipse.



(a) **(b)** **(c)**

Fig 8: (a) synthetic image (b) the result of proposed method (c) the result of Guru et al.

Three real standard images, House, Gantry Crane and Circuit are used for the testing of the new algorithm and result is compared with that of Guru et al and standard Hough Transform. For getting an edge image from original image, Canny edge detector (1986) [12] is used. Later, extraction of the straight lines is carried out using the above mentioned three algorithms

In the case of House image, all the results are given below. The edge image is Fig 9(b), resulted image of newly proposed algorithm is in Fig 9(c) and image by Guru et al's method Fig 9(d). To reveal the robustness of the proposed method during image transformation such as rotation, an experiment is conducted and the images are Fig

9(e) and 9(f). Efficient extraction is of straight lines alone is made by the new algorithm.

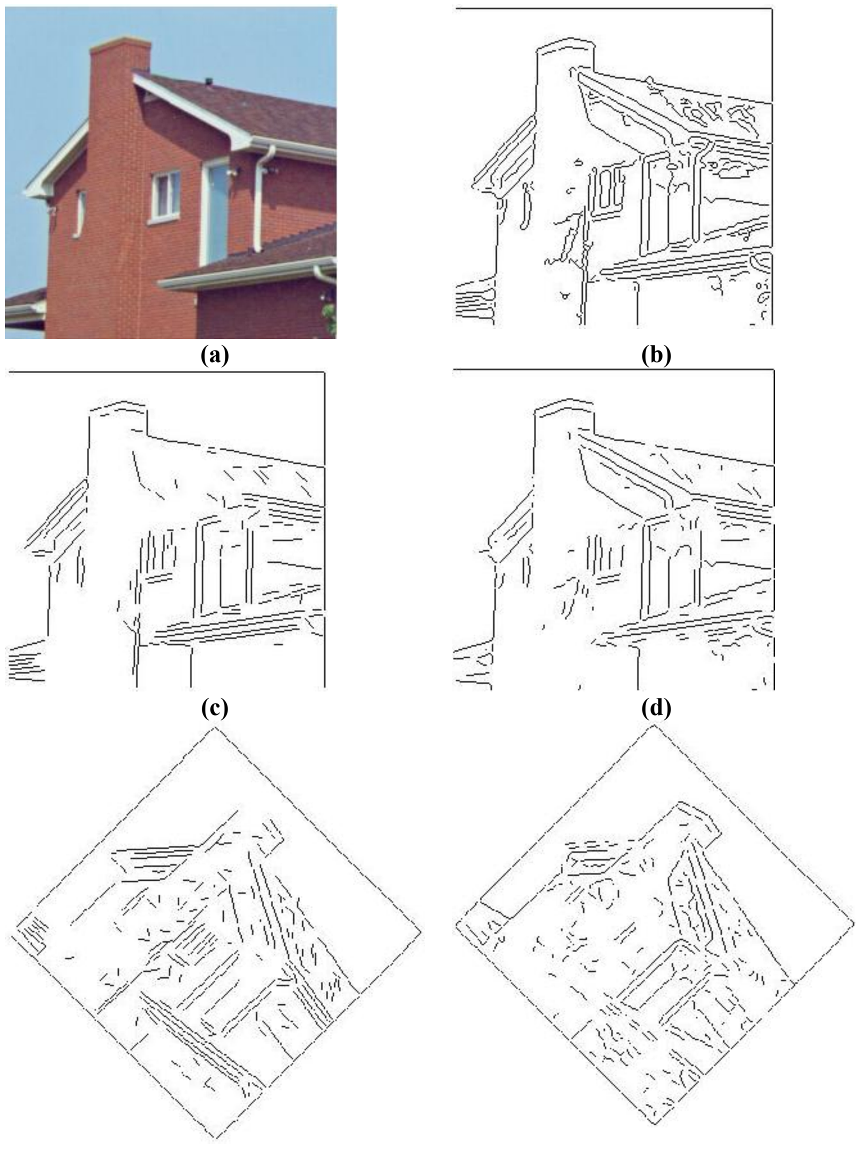
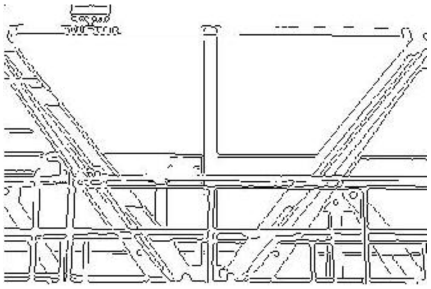
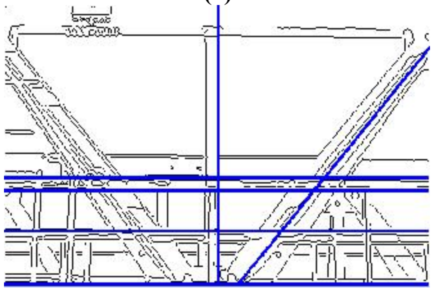


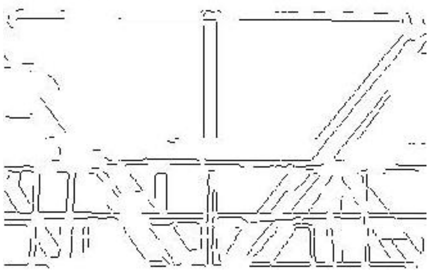
Fig 9: (a) House Image, (b) edge image, (c) result of proposed method (d) result of Guru et al (e) result of proposed method in rotated image of house (f) result of Eigenvalue based line detection in rotated image of house

Gantry Crane image and Circuit image are also tested using the same procedure. In these two cases comparison also is made with standard Hough transform. There is a very conspicuous difference between the proposed one and that of Guru et al. as shown in Fig 10(e) and 11(e), is having more clarity in the case of straight line extraction. In Guru et al's method some part of nearly placed parallel straight lines are not detected, while in our method it is detected. Fig 10(c) and 11(c) shows the result of HT. The blue line shows the pixels that are part of straight lines are grouped into line segments. Analysis of the images supports the efficiency of the newly proposed algorithm on straight line extraction in comparison with those mentioned above.

(a)


(b)


(c)


(d)

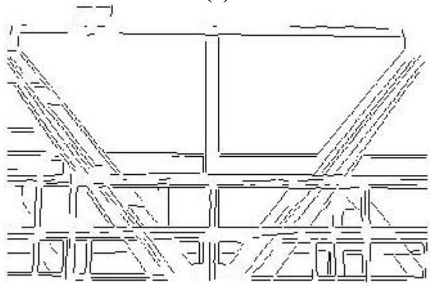

(e)

Fig 10: (a) Gantry Crane Image, (b) edge image, (c) result of standard Hough transform (d) result of Guru et al (e) result of proposed method

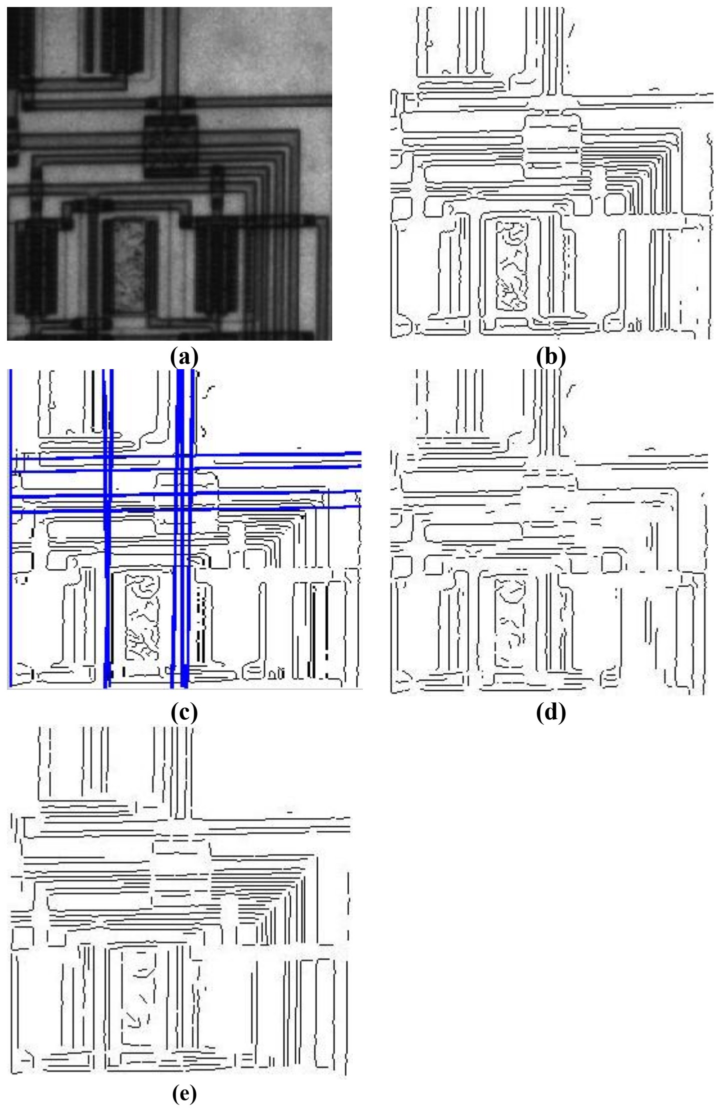


Fig 11: (a) Circuit Image, (b) edge image, (c) result of standard Hough transform (d) result of Guru et al (e) result of proposed method

4 Conclusions:

In the proposed new algorithm, detection of straight lines is based on a set of rules. The main advantage of the Rule Directed Pixel Comparison Method is its ability to identify the orientation of each edge pixel, and therefore detect more straight lines than other algorithms reported in literature. We have demonstrated how the algorithm can effectively detect even two finely separated parallel straight lines.

The proposed algorithm is simple and efficient. Especially, it is good for an edge image that has a lot of straight edges. Our algorithm avoids mask processing. Solving the problem of circles and ellipses will be future work for improving the performance of our algorithm.

References:

1. Mansouri, A., Malowany, S., Levine, M.D., 1987. Line detection in digital pictures: A hypothesis prediction/verification paradigm. Comput. Vision Graphics Image Process.40, 95–114.
2. Guru, D.S., Shekar, B.H., Nagabhushan, P., 2004. A simple and robust line detection algorithm based on small eigenvalue analysis. Pattern Recognition Lett. 25, 1–13.
3. Burns, J.B., Hanson, A.R., Riseman, E.M., 1986. Extracting straight lines. IEEE Trans. Pattern Anal. Machine Intell. 8(4), 425–455
4. Nevatia, R., Babu, K.R., 1980. Linear feature extraction and description. Comput. Vision Graphics Image Process. 13, 257–269.
5. Nelson, R.C., 1994. Finding line segments by stick growing. IEEE Trans. Pattern Anal. Machine Intell. 16 (5), 519–523.
6. Duda, R.O., Hart, P.E., 1972. Use of Hough transformation to detect lines and curves in pictures. Commun. ACM 15 (1), 11–15.
7. Li, H., Lavin, M.A., Le Master, R.J., 1986. Fast Hough transform: A hierarchical approach. Comput. Vision Graphics Image Process. 36, 139–161.
8. Illingworth, J., Kittler, J., 1987. The adaptive Hough transform. IEEE Trans. Pattern Anal. Machine Intell. 9 (5), 690–698.
9. Ben-Tzvi, D., Sandler, M.B., 1990. A combinatorial Hough transform. Pattern Recognition Lett. 11 (3), 167–174.
10. Princen, J., Illingworth, J., Kittler, J., 1990. A hierarchical approach to line extraction based on the Hough transform. Comput. Vision Graphics Image Process. 52 (1), 57–77.
11. Atiquzzaman, M., 1992. Multiresolution Hough transform—an efficient method of detecting patterns in images. IEEE Trans. Pattern Anal. Machine Intell. 14 (11), 1090–1095.
12. Canny, J.F., 1986. A computational approach to edge detection. IEEE Trans. Pattern Anal. Machine Intell. 8 (6), 679–698.