Robust Detection of Corners and Corner-line links in images

Andres Solis Montero, Milos Stojmenović, Amiya Nayak
School of Information Technology and Engineering
University of Ottawa
Ottawa, Canada
{asolis, mstoj075, anayak}@site.uottawa.ca

Abstract. We define corner points in an image as the intersections among detected straight line segments, and propose an algorithm that detects corners from such a definition. Our corner detection algorithm CLDC then makes use of the LDC (Line Detection using Contours) algorithm from [19], which outputs the list of all detected line segments together with their endpoints. Each line segment is extended in a post-processing step. CLDC (Corners from LDC) then finds corners in O((*n*+*I*) log *n*) time, where *n* and *I* are the number of endpoints the intersections of line segments, respectively. Detected corners are linked via line segments that define them. Such an output of the corner detection algorithm is a novel concept. The algorithm is comparable in time complexity with other algorithms, while providing more information about the line segments in the image. CLDC is robust to image transformations, such as rotation and translations. Our CLDC is compared to some existing algorithm, and its advantages are demonstrated.

Keywords: Corner detection, lines, sweeping algorithm, shape recognition, image processing.

I. INTRODUCTION

Many computer vision approaches for extracting features and inferring image content are based on detecting some points of interest. There is a plethora of interest point definitions in literature, each motivated by particular applications. In this article, we are interested in detecting corners in images, as particular points of interest that are frequently used in stereo matching, panoramic photographs, object detection/recognition, motion tracking, image database retrieval and robot navigation. For instance, corner detection algorithms enable a fast relationship between interest points among different images of the same object. It is easier to relate pixels instead of some regions with certain heights and widths. For example, motion is ambiguous at an edge, but unambiguous at a corner. Shapes can be approximately reconstructed from their corners.

Corners in images can be defined in variety of ways. Each corner detection algorithm normally uses its own definition, and it is normally the one that closely resembles what the algorithm itself detects as corners. The goal is always to reproduce as closely as possible what a human eye would consider to be a corner. A corner can be defined as the intersection of two or more edges. Also, a corner can be defined as point where there are two dominant and different edge directions in a local neighborhood of the point, the local minimum or maximum intensity of a point, line ending or point on a curve where the curvature is locally maximal.

The repeatability and efficiency of a corner detector determines how likely it is to be useful in a real-world application. Repeatability is important because the same scene viewed from different positions should yield features which correspond to the same real-world 3D locations. Efficiency is important because this determines whether the detector combined with further processing can operate at an acceptable frame rate.

Existing corner detectors are usually not very robust and often require human supervision. Quality of a corner detector is often judged based on its ability to detect the same corner in multiples images, like different lighting, rotation, translation and other transforms. The Computational time needed to detect corners is also important, and faster algorithms are preferred.

Recent detailed surveys of a number of corner detection algorithms were given in [16, 22]. Most popular schemes are proposed by Moravec [11], Beaudet [3], Kitchen & Rosenfeld [9], Harris & Sephens [8], Kanade, Lucas and Tomasi [10, 21], Smith and Brady (SUSAN [17]), Mokhatrian and Suomela (Curvature Scale Space [12]), Trajkovic and Hedley [20], Zheng, Wang and Teoh [23], Davies [6] and Rosten, Porter and Drummond (FAST [16]). Existing algorithms are roughly divided into those that detect corners from grayscale images, and from digital curves (where contours are extracted first). Grayscale image based algorithms usually detect interest points other than corners, and mostly search for lightness and brightness contrasts in images, find local maxima in differentiate geometry operators, or compute the determinant of the Hessian matrix and second order derivates. The algorithm in [8] uses a local window in the image to determinate the average change of intensity by shifting the window in various positions. Only the Curvature Space Scale (CSS) algorithm [12] (improved by Awrangjeb and Lu [1]) explicitly uses the contour to detect corners, although by a different approach from the one described in this article.

We define corners as the intersections of detected line segments. This corner definition was also used only in [6] where lines are detected using the Hough line Transform. Our algorithm preserves the idea of a corner as the intersection of two or more straight lines. Our corner detection algorithm makes use of the LDC (Line Detection using Contours) algorithm from LDC [19]. The LDC algorithm outputs the list of all detected line segments together with their endpoints. Each line segments is extended here by E pixels before further processing. Our CLDC algorithm then finds the intersections of these extended line segments using the plane sweep algorithm, and eliminates duplicates (with the help of a kd-tree structure) by preserving only the corners obtained by the plane sweep and eliminating other corners at distance less than *E* pixels. The algorithm runs in O((*n*+*I*) log *n*) time, where *n* and *I* are the number of endpoints, and the intersections of line segments, respectively. It is comparable in time complexity with other algorithms, while providing more information about the line segments in the image. Our definition of corner as the intersection of line segments corresponds to human perception of corners. Detected corners are linked via line segments that define them. Each corner is linked to at least another corner in the image. Such an output of a corner detection algorithm is a novel concept in computer vision and image processing. Our CLDC (Corners from LDC) algorithm is robust to image transformations, such as rotation and translations. The additional benefit of our algorithm is to report corners together with the line segments they belong to.

In section 2 we present a literature review on corner detection and describe some existing techniques used in our CLDC algorithm. Section 3 explains our CLDC algorithm and the modifications to the LDC output. Experimental data in Section 4 compares our CLDC algorithm with some of existing and widely used algorithms.

II. LITERATURE REVIEW

A. *Existing Corner Detection Algorithms*

Moravec [11] defined the concept of "points of interest" as distinct regions in images in 1977 and proposed to use them to find matching regions in consecutive image frames. This low-level processing step helped to determine the existence and location of objects in the vehicle's environment. The concept in [11] included corner detectors since interest points, like corners, were the points where there is a large intensity variation in every direction. A small square window (3x3, 5x5, or 7x7 pixels) centered at *P* is shifted by one pixel in each of the eight principle directions. The intensity variation for a given shift is calculated by taking the sum of squares of intensity differences of corresponding pixels in these two windows. Intensity variation at *P* is the minimum intensity variation calculated over all shifts. The Moravec operator can be used to give a measure of cornerness to each pixel in the image. Corners are local maxima, except those with cornerness value below a threshold. This simple corner detector is now considered generally obsolete [15]. It is computationally efficient but is not rotationally invariant, is susceptible to reporting false corners along edges and at isolated pixels and is sensitive to noise.

The Harris/Plessey operator [8] allows an estimate of the intensity variation to be calculated in any direction, thus generalizing Moravec's operator. The cornerness measure is reformulated to consider the variation between the intensity variation measures in different directions. Second image derivatives are convoluted by the Gaussian window. The resulting matrix *M* contains all the differential operators describing the geometry of the image surface at a given point. The Eigenvalues of *M* will be proportional to the principle curvatures of the image surface and form a rotationally invariant description of *M*. The cornerness measure is defined via the determinant and trace of *M*. However, since the components of *M* are estimated using only the horizontal and vertical gradients, they are not rotationally invariant. Despite the high computational demand, this algorithm is widely used in practice. It is sensitive to noise since it relies on gradient information, has poor localization on many junction types, and is not rotationally invariant [15]. The significance of poor localization of certain junction types is application dependent and the widespread use of this algorithm suggests this limitation can be overcome.

Trajkovic and Hedley [20] adopt the same definition for a corner as [8, 11]: corners are points where the change of image intensity is high in all directions. Performance is improved by performing an interpixel approximation to estimate the intensity change in all directions (unlike the finite number of directions [11]). Performance and computational demand are both improved by using a multigrid approach where likely locations of corners are first found in a low resolution version of the original image. The analysis [15] shows that it is not rotationally invariant, is sensitive to noise, and responds too readily to diagonal edges.

The KLT algorithm [10, 21] is similar to [8]. The Eigenvectors of similar matrix *M* encode edge directions, and the Eigenvalues encode edge magnitudes. A corner is identified by two strong edges, as a location where the smaller of the Eigenvalues is large enough. The Eigenvectors are sorted in decreasing order, and the corners correspond to pixels in that order with values above a threshold, and with no larger value in their neighborhood.

Beaudet [3] proposed a determinant operator which has significant values only near corners. Kitchen and Rosenfeld [9] proposed methods based on the magnitude of the gradient direction, the change of direction along the edge, the angle between most similar neighbors, and turning of the fitted surface. SUSAN [17] used a circular mask for corner detection, without using derivatives. It computes the fraction of pixels within a neighborhood which have similar intensity to the center pixel. Corners can then be localized by thresholding this measure and selecting local minima. The position of the center of gravity is used to filter out false positives.

In [12], the corner points are localized at the maxima of absolute curvature of the edges. The first step is to extract the edges from the original image using a Canny detector. The corner points are tracked through multiple curvature scale levels to improve localization. Their survey (from 1998) selected corner detectors in [8, 9, 17] for having good performance.

Rosten, Porter and Drummond [16] presented a new heuristic (algorithm called FAST) for fast feature detection using machine learning. They generalize the detector, allowing it to be optimized for repeatability, with little loss of efficiency, and carry out a rigorous comparison of corner detectors based on the repeatability criterion applied to 3D scenes. The comparison demonstrates that using machine learning produces significant improvements in repeatability, yielding a detector that is both very fast and of very high quality.

It was pointed out in [14] that piecewise linear polygonal approximation with variable breakpoints will tend to locate vertices at actual corner points. These points correspond approximately to the actual or extrapolated intersections of adjacent line segments of the polygons. The algorithm in [18] extracts edges as a chain code, performs a polygonal approximation on the chains and then searches for the line segment intersections. Polygonal approximations however do not preserve many corners because they have a different goal in mind, simplifying curves by polygons.

The only previous corner detection derived from line segments was proposed in [6]. It is based on Hough Line Transform (HLT). LDC [19] discuss at length why their LDC algorithm is superior to HLT. In [6], a corner is found where lines intersect, i.e., at large maxima in Hough space [6]. A generalized Hough transform is used, which replaces each edge with a line segment. In a manner similar to chaining, a short line segment can be fitted to the edges, and the corner strength found by the change in gradient direction along the line segment. Edge detectors often fail at junctions, so corners can be defined as points where several edges at different angles end nearby.

B. *Line Detection using Contours*

LDC [19] described a line segment detection and extraction algorithm. It uses a compilation of different image processing steps such as automatic normalization, Gaussian smooth, automatic threshold, and Laplacian edge detection to extract edge contours from colour input images. Contours of each connected component are divided into short (five pixels) fragments, which are classified by their orientation into nine discrete categories. Straight lines are recognized as consecutive short segments where the same direction was repeated at least three times. Its plausible accuracy, easy implementation, simplicity, speed, parameter minimization (only selecting one in [1,3,5,7] and one in [3,5,…,19]), 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. This solution gives us a surprisingly more accurate, faster and simpler answer with fewer parameters than the widely used Hough Transform (HT) based algorithm. This line detection algorithm is robust to image transformations such as rotation, scaling and translation, and to the selection of the remaining two parameter values, while HT is very sensitive to its 7 parameter values which are also time consuming to select optimally.

C. *Line segment intersections and nearest neighbor search*

Our proposed algorithm makes use of two well know technique from computational geometry, line segment intersections and nearest neighbour search. All pair wise intersections of n line segments in a plane can be easily examined in O(n^2) by a brute-force algorithm that will check each pair of lines for their intersection. It will also return the intersections in unsorted order. To achieve a faster solution, we implemented instead a plane sweep algorithm [7]. This algorithm runs in O(n log n + I log n) where I is the number of intersections found [7]. It moves a (sweep) line across the plane to find intersection points. All intersections to the left of the sweep line have already been detected. The status of this sweep line is the set of segments currently intersecting it. This status changes as the sweep line moves to the right. Any time the sweep line passes over an event point (endpoint of a segment or an intersection point), the sweep line stops and updates this status. Because the event points are processed in sorted order, the intersections are found in sorted order. Two line segments cannot intersect unless they are next to each other. Thus, we keep track of the vertical ordering of the line segments and only test neighbouring segments for intersection.

In nearest neighbour search, given a set of N points, the goal is to find a point that is nearest to a query point q with minimal time per each search. This search can be done efficiently by using the tree properties to quickly eliminate large portions of the search space. N points are organized in a kd-tree. Every node in a kd-tree splits into sub-trees and is associated with one of the dimensions. Starting with the root node, the algorithm moves down the tree recursively, and saves the reached leaf node point as the "current best". The algorithm unwinds the recursion of the tree; if the current node is closer than the current best, then it becomes the current best. If there are nearer points on the other side of the splitting plane, the algorithm moves down the other branch of the tree from the current node looking for closer points, following the same recursive process as the entire search [5, 13] kd-tree can be built in O(N log N) time. Each search requires O(log N) time on average and O(root(N)) time in the worst case [13]. Inserting and removing nodes from the kd-tree requires O(log N) time. The kd-tree is implemented in many libraries including OpenCV/EmguCV.

III. Corner Detection Algorithm

We define a corner as the intersection of two or more segments. Further, we require that there is no other corner nearby (up to a certain threshold pixel distance E). Therefore, a corner is a structure that joins a list of segments in the image space. Typical junctions have shapes of the general form of the letters T, X, L, Y, and there are also rare junctions with five or more line segments. Each junction is the intersection of two or more line segments. The endpoints of line segments that are not close to any other line segment will be also considered as corners. Therefore a single line segment detected in an image will output two corners, one from each of its endpoints. The main problem is to determine corners that are shared by several line segments which should be reported only once.

We first apply the LDC algorithm [19] to an image to produce a set of line segments. The endpoints of each line segment are the initial set of corners. However, we would like to avoid duplicate reporting of the same corner at T, L, X and Y junctions. Also, we would like to link each unique corner with all corresponding line segments at the junction. We then first extend each line segment by E pixels on both sides, where E is a parameter. By doing this, line segments that originally did not cross each other but were in the vicinity (and to a human observer naturally correspond to a single corner) will now cross each other. These crossings can be detected by a suitable line intersection algorithm. Subsequently, other endpoints, which are near an already identified corner, can be merged with this one, using a nearest neighbor algorithm to find the distance between a candidate corner and nearest existing one to it.

Our algorithm, called CLDC (Corners from LDC algorithm), has three configuration parameters. Two come from the LDC algorithm: Gaussian Kernel Size (*GKS*) and Laplace Aperture Size (*LAS*). The third input parameter is the number of pixels *E* that each line segment is extended in both directions. No modifications will be done to the initial step of the algorithm (LDC line segment computation [19]). We will only process its output further, to compute the intersections. The CLDC algorithm is illustrated in Figure 1 on a simple example of an X junction (Fig. 1a)). The LDC algorithm originally produces 8 line segments here: *a, b, c, d, e, f, g, h* (Fig. 1b)). All of them have the junction as one endpoint, and each 'leg' of X is identified twice by the LDC algorithm [19]. This junction may physically be four different pixels or just two, if three endpoints are physically the same pixel.

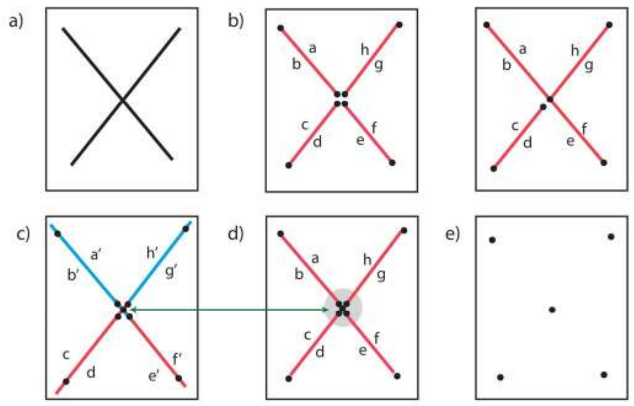


Figure 1. a) X junction; b) 8 reported line segments from LDC [19]; c) extended line segments and multiple intersection detection; d) single corner at junction by eliminating other near intersections and endpoints; e) five detected corners

Each line segment is extended, and line segments intersect at or near the junction (Fig. 1c)). All line segment intersections are discovered by applying the plane sweep algorithm. Each discovered intersection point will attempt to enter the (2-dimensional) kd-tree. However if this kd-tree already contains a corner that is near (distance at most *E* pixels) the discovered intersection then the intersection is not entered into the tree. That way, the kd-tree contains only corners, so that no two corners are near each other. Multiple occurrences of the same intersection and nearby endpoints are therefore eliminated and a single corner is identified at the junction (Fig. 1d)). During the elimination process, an already identified corner may receive more links to other line segments. The CLDC algorithm in the end reports five corners, each linked to corresponding line segments (Fig. 1e)).

After inserting all intersection points (corners) into the kd-tree, CLDC also inserts the original endpoints into the same tree. Those that are close to the new corners will be eliminated because they are closer than *E* to the already identified corners (actual intersections). After entering all intersections, the endpoints of the line segments that are not junctions will be entered into the same kd-tree without any 'resistance' from existing corners, and therefore will also become corners. That is, they generate corners from original such endpoints, not from extended ones. At the end, corners are exactly the points in the final kd-tree. The pseudo-code of the algorithm can be expressed as follows.

```
CLDC Algorithm: Input: Color Image, GKS, LGS, and E
Output: list of corners, each with a list of adjacent segments.
Begin:
   old_lines ← LDC(Image,GKS, LGS)
   new_lines ← empty; kdtree ← empty
   For each segment in old_lines:
      enlarge segment and insert it into new_lines.
   corners ← PlaneSweep(new_lines)  // may be sorted first
      (not mandatory)
   For each point from corners:
      Insert point into kdtree if the nearest corner already
      there is at distance > E, else link segment containing the
      point to corner
   For each endpoint old_corner from old_lines:
      Neighbor ← nearest corner to old_corner inside kdtree
      If distance between neighbor and old_corner < E:
         Merge adjacent list of neighbor and old_corner into
         neighbor
         add corresponding line segment links to neighbor
      Else:
         Insert old_corner in kdtree.
   Return kdtree (list of corners).
End.
```

The time complexity of the LDC algorithm [19] is O(*N*), where *N* is the number of pixels in the image. Creating the new list of segments by adding *E* pixels to its endpoints can be done in O(*s*) time, where *s* is the number of segments from LDC output. There are $n=2s$ endpoints. $N >> n$ because only representative line segments are extracted from the image. The intersections of *n* line segments can be found by the plane sweep algorithm in $O(n \log n + I \log n)$ where *I* is the number of intersections found. When $n^2 << N$, a straightforward $O(n^2)$ pairwise line segment intersection can be also applied without affecting overall time complexity. Finding the nearest corner and inserting the corners in the kd-tree will add O(*n* root(*n*)) to the complexity. Retrieving all the points from the kd-tree will be O(*n*). Finally the total complexity of our solution will be O(*N*+*I* log *n* + *n* root(*n*)). Since $N >> n$, the running time is dominated by the line segment extraction algorithm LDC [19], and the algorithm presented here does not increase time complexity.

IV. EXPERIMENTAL DATA

We compared our CLDC corner detection algorithm with some existing algorithms, by Moravec [11], Harris/Presley [8], Trajkovic [20], SUSAN [17], and FAST [16]. These algorithms selected for comparison are well-known and most frequently used in literature [15]. The comparison will be made using measures of robustness to image transformation such as rotating, blur, and adding noise. We ran two experiments.

In the first experiment we will measure the Reliability, Accuracy and Error, on a sample image from [15]. Let *D* be the exact number of corners to be detected, and let *F* be the number of corners actually reported. Reported corners are divided into true positives *P* and true negatives *N* so that *F*=*P*+*N*. The number of missing corners *M* represents corners not identified by a given algorithm. Reliability is then defined as *P/F* (thus 100% accuracy is obtained when *N*=0), accuracy is *F/D* (thus 100% accuracy means that all corners are detected), and error is defined as *N/D*.

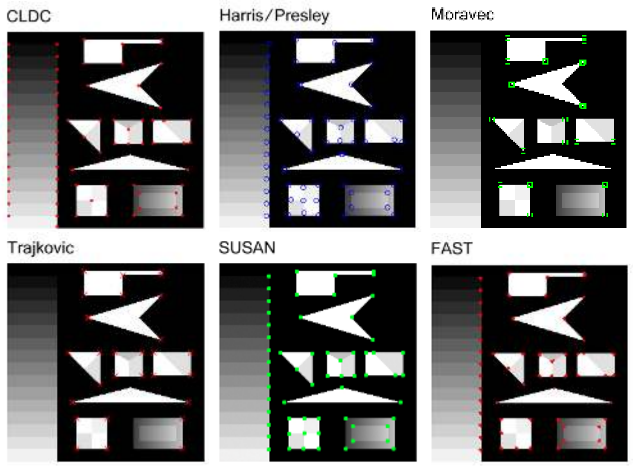


Figure 2. Sample S1: Artificial test image containing different corner types [15]

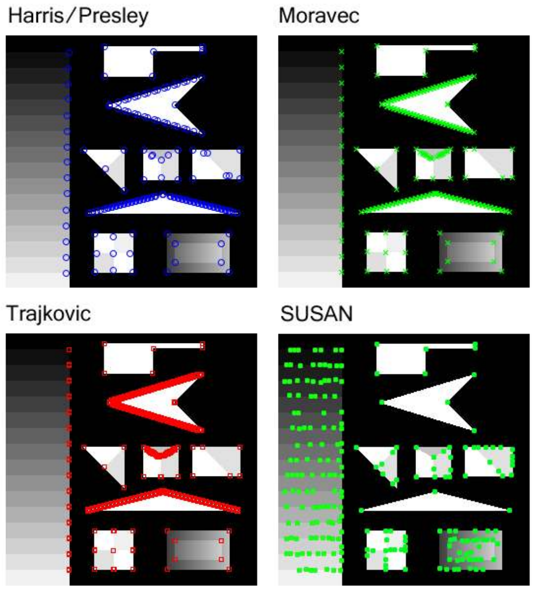


Figure 3. Sample S1b: Increasing P may drastically increase N.

Figure 3 illustrates the sensitivity of some existing algorithms to the selection of parameters. An attempt to increase *P* results in a drastic increase of *N* in sample S1b. SUSAN [17] increases *N* along horizontal and vertical lines, while others increase *N* along diagonal lines. We observe that Harris/Presley [8] and SUSAN [17] were capable of detecting more *P* and less *N* than Moravec [11] and Trajkovic [20]. FAST [16] and CLDC give a more stable output for this sample, not detecting many false negatives.

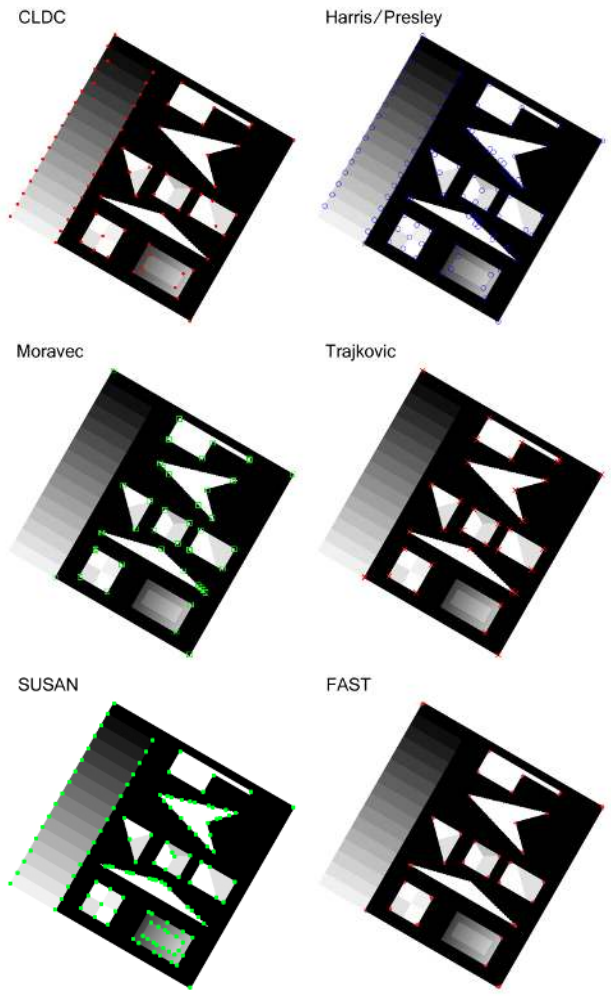


Figure 4. Sample S2: The same image rotated 30 degrees clockwise

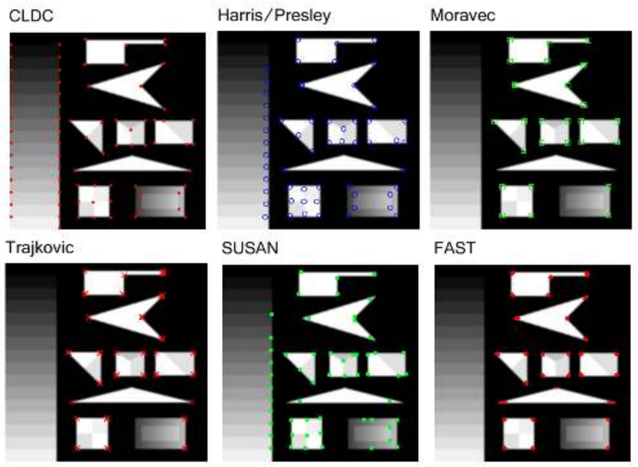


Figure 5. Sample S3: Blurring the same image

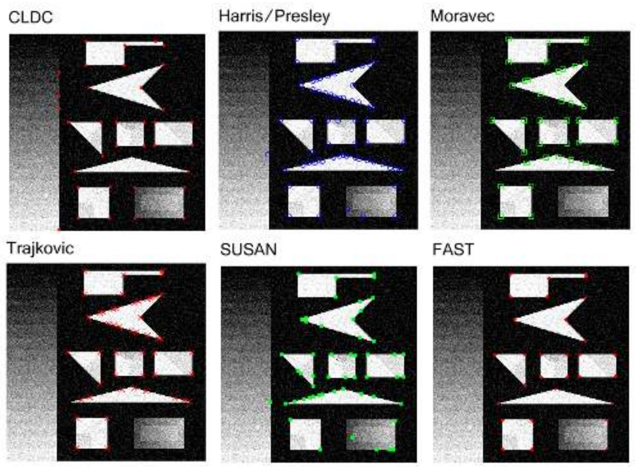


Figure 6. Sample 4: Adding Gaussian noise to the image

The number of parameters in each algorithm is similar. Trajkovic [20] has 5 input parameters, Harris/Presley [8] has 4, Moravec [11], SUSAN [17], FAST [16] and CLDC have 3 input parameters each. For examples here with CLDC, *E*=5 pixels to extend lines, GKS (Gaussian Kernel size) to blur the image was not applied, while LKS (Laplace kernel size) was applied with values of 3 or 5. All algorithms are quite fast, and there is no significant difference in speed among them. FAST [16] claimed to be the fastest one among existing ones. SUSAN [17] is slower than Moravec [11] and Trajkovic [20] while Harris/Presley [8] is the slowest one [15]. Our CLDC algorithm is based on the straight line detection algorithm LDC which is faster than the Hough Transform based alternative [SSN], and additional time to detect corner is negligible compared to the time to detect straight lines. In our experiments, the running times were about 0.16sec for SUSAN, 0.04sec for FAST, and 0.05sec for CLDC per image.

Our test set consists of a sample image (S1) which is rotated (S2), blurred (S3) and has Gaussian noise added (S4). Blurring and noise was performed on sample S1. This example shows the overall superiority of our algorithm in reliability, accuracy and error measures, compared to the solutions currently used in literature: Harris/Presley (HP), Moravec (M), Trajkovic (T), SUSAN, and FAST.

We then calculated the averages and 95% confidence intervals over the above samples, for each of the three measures and tested algorithms. The results are in Table 2.

TABLE I. RELIABILITY, ACCURACY AND ERROR RATES FOR SAMPLE IMAGES AND THEIR TRANSFORMATIONS

	D	F	M	P	N	Rel %	Acc %	Err %
S1								
CLDC	77	72	5	72	0	100	94	0
HP	77	60	18	59	1	98	77	1
M	77	28	51	26	2	93	34	3
T	77	28	49	28	0	100	36	0
SUSAN	77	61	16	61	0	100	79	0
FAST	77	58	19	58	0	100	75	0
S1b								
HP	77	158	16	61	97	39	79	126
T	77	188	16	60	128	32	78	166
M	77	164	16	60	104	37	78	135
SUSAN	77	222	16	61	161	27	79	209
S2								
CLDC	81	77	7	74	3	96	91	4
HP	81	102	3	78	24	76	96	30
M	81	43	47	33	10	77	41	12
T	81	36	47	34	2	94	42	2
SUSAN	81	119	1	80	30	67	99	37
FAST	81	33	48	33	0	100	41	0
S3								
CLDC	77	76	1	76	0	100	99	0
HP	77	57	40	37	20	65	48	26
M	77	27	50	23	4	85	30	5
T	77	59	46	31	28	53	40	36
SUSAN	77	56	29	49	7	88	64	9
FAST	77	32	47	30	2	94	39	2
S4								
CLDC	77	35	43	34	1	97	44	1
HP	77	67	46	31	36	46	40	47
M	77	39	53	24	15	62	31	19
T	77	50	52	25	25	50	32	32
SUSAN	77	53	48	29	24	55	38	31
FAST	77	30	47	30	0	100	39	0

TABLE II. AVERAGES AND CONFIDENCE INTERVALS OF TEST SET

Algorithm	Reliability		Accuracy		Error	
CLDC	0.98	±0.02	0.82	±0.25	0.01	±0.02
HP	0.71	±0.21	0.65	±0.25	0.26	±0.18
M	0.79	±0.13	0.34	±0.05	0.10	±0.07
T	0.74	±0.26	0.38	±0.04	0.18	±0.19
SUSAN	0.77	±0.20	0.70	±0.25	0.19	±0.17
FAST	0.98	±0.03	0.48	±0.18	0.01	±0.01

We define *repeatability* (*Rep*) as the ratio *Per=P12/P1* where *P1* is the number of true positives (correctly identified corners) in the original image while *P12* is the number of corners among them that are also identified as corners in the transformed image. The same definition was used in [16, 1]. This measure is automatic as it does not depend on human perception of what are true corners. Note that measures that involve human evaluation of corners were considered in [4]. Thus it measures the ratio of correctly identified corners from the original image that are also identified in the transformed image. The *joint accuracy* (*JA*) is defined as *JA=(P1+P2)/(D1+D2),* where *P* and *D* for both images are defined above, and index 1 or 2 refers to the original and transformed image, respectively. Joint accuracy measures the total amount of identified corners with respect of total number of really existing corners in both images. Finally, we also compute the *added noise rate* (*AN*) as *N2/P12*.

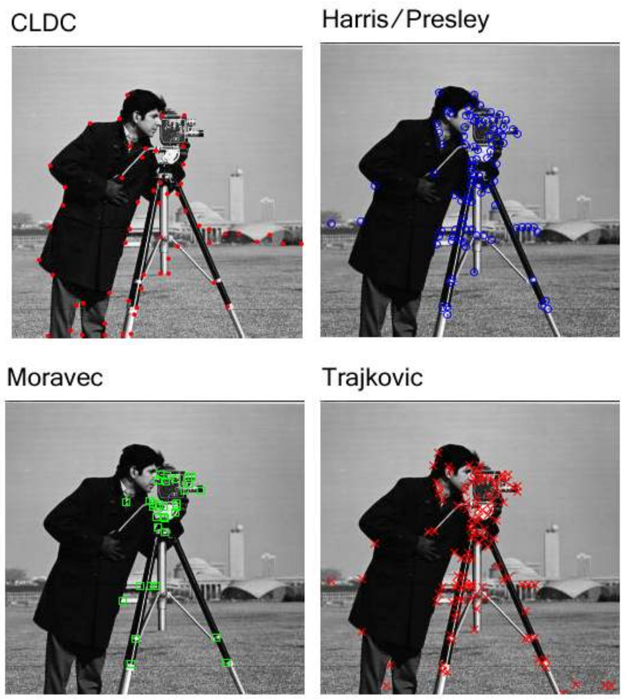
TABLE III. REPEATABILITY, JOINT ACCURACY, ADDED NOISE RATES FOR TEST SET

Algorithm	**Rotation**			**Blur**			**Noise**		
	Rep	***JA***	***AN***	***Rep***	***JA***	***AN***	***Rep***	***JA***	***AN***
CLDC	92	86	12	100	100	0	47	44	3
H	98	75	41	93	71	0	53	40	116
M	96	32	28	96	30	9	88	30	61
T	100	36	4	100	36	11	89	32	96
SUSAN	100	79	49	80	64	14	46	38	39
FAST	52	39	0	53	40	6	53	40	0

Since none of the above measures is sufficient alone, we define a combined *index* measure that appears useful as a single measure, as *index=Per*JA-AN.* Therefore the index increases with the repeatability and joint accuracy but decreases with the added noise rate. This index measure also confirms the superiority of our scheme compared to others, as seen in Table 4.

TABLE IV. AN RANKING OF ALGORITHMS

Algorithm	**Index**
CLDC	66.45 %
T	32.79 %
HP	32.67 %
SUSAN	30.04 %
FAST	20.15 %
M	3.22 %



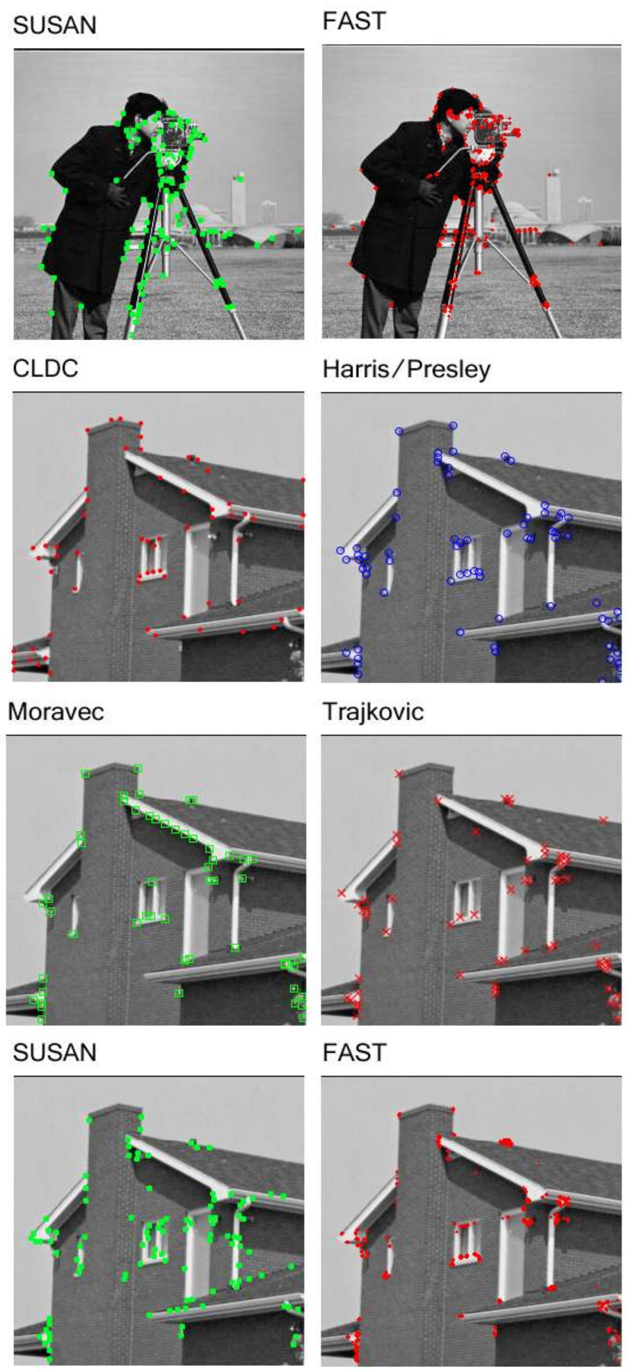


Figure 7. Corners detected from real imagery

We also show some examples from real images, with the corners detected by each method. In the first example in Figure 7, all methods except our CLDC have too many corners detected on the camera in the image which are in close proximity to each other, and apparently missing corners along the man's borders and in the background. The CLDC algorithm balances the corners along the camera, man and background, which appear evenly spread along the corresponding boundaries, thus resembling their shapes. The house image also has regions saturated with corners, yet some important corners are missing, by all methods except our CLDC.

V. CONCLUSION

We described here a simple, fast, relatively parameterless and robust algorithm for detecting corners in an image. The time complexity for our CLDC algorithm is dominated by O(N) where N is the total number of pixels, and it is then not possible, for an image as input, to have an asymptotically faster algorithm. We demonstrate its higher accuracy and repetitiveness (with respect to image transformations such as rotation, blurring, and Gaussian noise) compared to other solutions frequently used.

We defined also a novel index measure (which encompasses repeatability, added noise, and joint accuracy) that gives a combined measure of 'goodness' of an algorithm, and again show that our algorithm is better than other existing ones. Finally, our algorithm is conceptually very simple to understand, probably the simplest among all existing ones. It merely defines corners as intersections of two or more line segments, and relies on a natural extraction of line segments which extracts them as repeated edge pixels along the same direction. This is the ultimate advantage of CLDC for its use in practice.

Qualitative and quantitative properties of our CLDC algorithm, and its simplicity, provide a basis for its applications in stereo matching, object detection/recognition, motion tracking, image database retrieval, robot navigation etc. Further, it provides output not present in other algorithms, where each corner is linked directly with all line segments defining it. This enables further applications, for example extracting polygonal boundaries of all shapes in the image and shape analysis.

VI. REFERENCES

[1] M. Awrangjeb, G. Lu, Robust Image Corner Detection Based on the Chord-to-Point Distance Accumulation Technique, IEEE Transactions on Multimedia, Vol. 10, No. 6, Oct. 2008 1059-1072.

[2] Baumberg, "Reliable feature matching across widely separated views", Proceedings of IEEE Conference on Computer Vision and Pattern Recognition: pages I: 1774--1781. (2000)

[3] P.R. Beaudet, "Rotationally invariant image operators". Fourth International Conference on Pattern Recognition, pp. 579–583, 1978.

[4] Y. Bastanlar, Y. Yardimci, "Corner Validation based on Extracted Corner Properties", Computer Vision and Image Understanding, 112 (2008) 243-261.

[5] T. Cormen, C. Leiserson, R. Rivest, "Introduction to Algorithms", McGraw-Hill Companies, 768 pages. Chapter 10.March 1990.

[6] E. Davies, "Application of the Generalized Hough Transform to Corner Detection', Proc. IEE Computers and Digital Techniques, vol. 135, no. 1, pp. 49-54, 1988.

[7] M. De Berg, O. Cheong, M. Van Kreveld, M. Overmars, "Computational Geometry Algorithms and Applications", 3rd Edition. Springer, Apr. 2008.

[8] C. Harris, M. Stephens, "A Combined Corner and Edge Detector", Proc. Alvey Vision Conf., Univ. Manchester, pp. 147-151, 1988.

[9] L. Kitchen, A. Rosenfeld, "Gray-level corner detection" Pattern Recognition Letters, vol. 1, no. 2, pp. 95-102, 1982.

[10] B. Lucas, T. Kanade, "An Iterative Image Registration Technique with an Application to Stereo Vision", International Joint Conference on Artificial Intelligence, pp. 674-679, 1981.

[11] H. Moravec, "Towards Automatic Visual Obstacle Avoidance", International Joint Conference on Artificial Intelligence, pp. 584, 1977.

[12] F. Mokhtarian, R. Suomela, "Robust image corner detection through curvature scale-space," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 20, no. 12, pp. 1376–1381, 1998.

[13] J. O'Rourke, "Computational Geometry in C" (2nd Ed.), Cambridge University Press 1998.

[14] T. Pavlidis, Structural Pattern Recognition, Berlin, Heidelberg, NY: Springer-Verlag, 1977.

[15] J. Parks, J.P. Gravel, Corner detectors, University of McGill, www.cim.mcgill.ca/~dparks/CornerDetector/index.htm.

[16] E. Rosten, R. Porter, T. Drummond, "Faster and Better: A Machine Learning Approach to Corner Detection", IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 32, No. 1, pp. 105-119, January 2010.

[17] S. Smith, M. Brady, "SUSAN - A New Approach to Low Level Image Processing", International Journal of Computer Vision, Vol. 23, No. 1, pp. 45-78, 1997.

[18] H.T. Sheu, W.C. Hu, "A rotationally invariant two-phase scheme for corner detection", Pattern Recognition, Vol. 28, pp. 819-828, 1996.

[19] A. Solis-Montero, A. Nayak, M. Stojmenovic, N. Zaguia, "Robust Line Extraction Based on Repeated Segment Directions on Image Contours", IEEE Symposium: Computational Intelligence for Security and Defense Applications (CISDA), July 8-10, 2009.

[20] M. Trajkovic, M. Hedley, "Fast Corner Detection", Image and Vision Computing, Vol. 16, No. 2, pp. 75-87, 1998.

[21] C. Tomasi, T. Kanade, "Detection and Tracking of Point Features", Carnegie Mellon University Technical Report CMU-CS-91-132, April 1991.

[22] T. Tuytelaars, K. Mikolajczyk, "Local Invariant Feature Detectors: A Survey", Foundations and Trends in Computer Graphics and Vision, Vol. 3, No. 3, pp. 177–280, 2007.

[23] Z. Zheng, H. Wang and E. Teoh, "Analysis of gray level corner detection", Pattern Recognition Letters, Vol. 20, pp. 149-162, 1999.