Junction and Corner Detection through the Extraction and Analysis of Line Segments

Christian Perwass*

Institut für Informatik und Praktische Mathematik
Christian-Albrechts-Universität zu Kiel
Christian-Albrechts-Platz 4, 24118 Kiel, Germany
chp@ks.informatik.uni-kiel.de

Abstract. An algorithm is presented that analyzes the edge structure in images locally, using a geometric approach. A local edge structure that can be interpreted as a corner or a junction is assumed to be representable by a set of line segments. In a first step a segmentation of the local edge structure into line segments is evaluated. This leads to a graph model of the local edge structure, which can be analyzed further using a combinatorial method. The result is a classification as corner or junction together with the absolute orientation and internal structure, like the opening angle of a corner, or the angles between the legs of a junction. Results on synthetic and real data are given.

1 Introduction

In many areas of Computer Vision the detection of feature points in an image plays an important role. For example, in the field of object recognition the use of "key features" for an object shows some promising results [12, 9]. Also tracking and 3D-reconstruction algorithms use feature points in images. These feature points are quite often corners, or more specifically, intrinsically two dimensional (i2D) structures. The advantage of i2D structures over intrinsically one dimensional (i1D) structures, i.e. edges, is that they can be identified with a specific position in an image, whereas i1D structures only allow for a localization along one direction.

Many algorithms have been developed to detect corners and edges, see e.g. [7, 10, 8, 13, 6]. At a signal level, edge detectors basically locate places in an image where the image gradient is high. Different types of edges may also be distinguished by evaluating the local phase of the image signal (cf. [5]). In order to detect i2D structures, usually the image gradients within an image patch are combined in some way. One method often used is the summation of the structure tensor over an image patch. The rank of the resultant matrix then indicates whether the image gradient vectors in the respective image patch span a 1D-space or a 2D-space. In the first case an edge is present in the image patch, since all gradients point (approximately) in the same direction. In the second

* This work has been supported by DFG Graduiertenkolleg No. 357 and by EC Grant IST-2001-3422 (VISATEC).

case a corner or junction must be present, since the gradients point in two or more different directions. For example, the Förstner operator [7] and the corner detector by Harris and Stevens [10] are based on this principle.

Methods using the structure tensor in this way can only distinguish between i1D and i2D structures. A further distinction between i2D signals is not possible, since the structure tensor can at most encode two directions. Nevertheless, it would be advantageous to distinguish between i2D image structures like corners, line crossings, y-junctions, t-junctions and ψ-junctions and also to measure the orientations of their different parts. Consequently, there has been some effort to analyze i2D structures further, see e.g. [1, 4, 11].

In this paper we propose a method to extract the type of i2D image structures and to evaluate their parameters, like the opening angle of a corner, for example. Instead of analyzing the image gradients directly, we use a two step approach. In a first step the image gradients are used to find edges. At this step an appropriate scale and smoothing has to be selected for the given image. The result of this first step is an image containing only the edges of the initial image.

In a second step the local geometry of the edges is analyzed. This analysis is again split into a number of steps. First we observe that the line structures we are interested in can be represented by a pair of conics in a useful way. Note that we do not use the form of the fitted conics directly to analyze the image structure, as for example in [15], but consider their intersections instead. From this a weighted graph representing the local edge structure can be constructed. Which particular structure is present may then be deduced from a combinatorial analysis of the graph.

The remainder of this paper is structured as follows. First we discuss the fitting of conic pairs to edge data and how this can be used to extract a graph representing the local edge geometry. The next part is dedicated to the combinatorial analysis of the extracted graph. This is followed by the presentation of experimental results and some concluding remarks.

2 Fitting Conic Pairs

As mentioned in the introduction, an image is reduced to a set of edge pixels in an initial preprocessing step. The assumption we then make is that within a local area the edge pixels can be segmented into a set of line segments. A corner, for example, consists of two line segments that meet in the local area. Even though we are looking for line segments, it is not obvious how to fit lines to the data, since it is in general not known how many line segments are present. A y-junction, for example, has three line segments, while a corner or a t-junction only have two.

The basic idea we follow here is to perform an eigenvector analysis of the edge data in a local area. However, instead of using the data directly we first transform it to some other space, where the eigenvectors represent conics. From a geometric point of view, we try to find the conics that best fit the data. How

this can be used to segment the data into line segments will be described later. First the embedding used and the eigenvector analysis are discussed.

As mentioned before, the edge geometry is evaluated locally. That is, given an edge image, we move a window of comparatively small size over it and try to analyze the local edge geometry within the window for all window positions. For each window position the pixel coordinates of the edge points are transformed to coordinates relative to the center of the window, such that the top left corner of the local area is at position $(-1, 1)$ and the bottom right corner at the position $(1, -1)$. The main reason for this transformation is to improve the numerical stability of the algorithm. Let the position vector of the i^{th} edge point in the local area in transformed coordinates be denoted by the column vector $\mathsf{w}_i = (u_i, v_i)^\mathsf{T}$. These position vectors are embedded in a 6D-vector space of symmetric matrices, which allows us to fit conics to the set of edge points. The details of this embedding are as follows.

2.1 The Vector Space of Conics

It is well known that given a symmetric 3×3 matrix A, the set of vectors $\mathsf{x} = (x, y, 1)^\mathsf{T}$ that satisfy $\mathsf{x}^\mathsf{T}\ \mathsf{A}\ \mathsf{x} = 0$, lie on a conic. This can also be written using the scalar product of matrices, denoted here by $\cdot$, as $\left(\mathsf{x}\mathsf{x}^\mathsf{T}\right)\ \cdot\ \mathsf{A} = 0$. It makes therefore sense to define a vector space of symmetric matrices in the following way. If a_{ij} denotes the component of matrix A at row i and column j, we can define the transformation $\mathcal{T}$ that maps elements of $\mathbb{R}^{3\times3}$ to $\mathbb{R}^6$ as

$$\mathcal{T}\ :\ \mathsf{A} \in \mathbb{R}^{3\times3} \mapsto \left(a_{13},\, a_{23},\, \tfrac{1}{\sqrt{2}}\, a_{33},\, \tfrac{1}{\sqrt{2}}\, a_{11},\, \tfrac{1}{\sqrt{2}}\, a_{22},\, a_{12}\right)^\mathsf{T}\ \in \mathbb{R}^6. \tag{1}$$

A vector $\mathsf{x} \in \mathbb{R}^3$ may now be embedded in the same six dimensional space via

$$\mathbf{x} := \mathcal{T}\left(\mathsf{x}\mathsf{x}^\mathsf{T}\right) = \left(x,\, y,\, \tfrac{1}{\sqrt{2}},\, \tfrac{1}{\sqrt{2}}\, x^2,\, \tfrac{1}{\sqrt{2}}\, y^2,\, x\, y\right)^\mathsf{T}\ \in \mathbb{R}^6. \tag{2}$$

If we define $\mathbf{a} := \mathcal{T}(\mathsf{A})$, then $\mathsf{x}^\mathsf{T}\ \mathsf{A}\ \mathsf{x} = 0$ can be written as the scalar product $\mathbf{x}^\mathsf{T}\,\mathbf{a} = 0$. Finding the vector $\mathbf{a}$ that best satisfies this equation for a set of points $\{\mathbf{x}_i\}$ is usually called the algebraic estimation of a conic [2].

In the following we will denote the 6D-vector space in which 2D-conics may be represented by $\mathbb{D}^2 \equiv \mathbb{R}^6$. A 2D-vector $(x, y) \in \mathbb{R}^2$ is transformed to $\mathbb{D}^2$ by the function

$$\mathcal{D}\ :\quad (x, y) \in \mathbb{R}^2\ \mapsto\ \left(x,\, y,\, \tfrac{1}{\sqrt{2}},\, \tfrac{1}{\sqrt{2}}\, x^2,\, \tfrac{1}{\sqrt{2}}\, y^2,\, xy\right) \in \mathbb{D}^2. \tag{3}$$

2.2 The Eigenvector Analysis

In order to analyze the edge data, we embed the data vectors $\{\mathsf{w}_i\}$ in the vector space of symmetric matrices as described above, i.e. $\mathbf{w}_i := \mathcal{D}(\mathsf{w}_i)$. Note that we use a different font to distinguish image vectors $\mathsf{w}_i \in \mathbb{R}^2$ and their embedding $\mathbf{w}_i \in \mathbb{D}^2$. Denote by $\mathbf{W}$ the matrix constructed from the $\{\mathbf{w}_i\}$ as $\mathbf{W} = (\mathbf{w}_1, \ldots, \mathbf{w}_N)^\mathsf{T}$, where N is the number of data vectors. A conic $\mathbf{a} = \mathcal{T}(\mathsf{A})$

that minimizes $\|\mathbf{W}\,\mathbf{a}\|^2$ is then a best fit to the data in an algebraic sense. The key to our algorithm is not just to look at the best fit but at the *two* eigenvectors of $\mathbf{W}$ with the smallest eigenvalues.

In order to obtain real valued eigenvalues and orthogonal eigenvectors, we evaluate the eigenvectors and eigenvalues of $\mathbf{W}$ by performing a singular value decomposition (SVD) on $\mathbf{W}^\mathsf{T}\,\mathbf{W}$, which is symmetric. The singular vectors are then simply the eigenvectors and the square root of the singular values gives the eigenvalues of $\mathbf{W}$.

If $\mathbf{W}$ has two small eigenvalues, this means that the whole subspace spanned by the corresponding eigenvectors is a good fit to the data. In mathematical terms this can be written as follows. Throughout this text we will use $\mathbf{c}_1, \mathbf{c}_2$ to denote the two eigenvectors with smallest eigenvalues of $\mathbf{W}$. For any $\alpha, \beta \in \mathbb{R}$, $\mathbf{c} = \alpha\,\mathbf{c}_1 + \beta\,\mathbf{c}_2$ is a good fit to the data. This may also be termed a pencil of conics. The base points that define this pencil of conics are those that lie on all conics in this pencil. These points are simply the intersection points of the conics $\mathbf{c}_1$ and $\mathbf{c}_2$. It therefore seems sensible that the intersection points of $\mathbf{c}_1$ and $\mathbf{c}_2$ also contain important information about the structure of the data from which $\mathbf{W}$ was constructed. An example that this is indeed the case can be seen in figure 1. The dots in this figure represent the data points and the two hyperbolas are the conics represented by the two eigenvectors of the corresponding $\mathbf{W}$ matrix with the smallest eigenvalues. It can immediately be seen that each conic by itself does not represent the data distribution. However, their intersection points lie exactly in the clusters formed by the data.

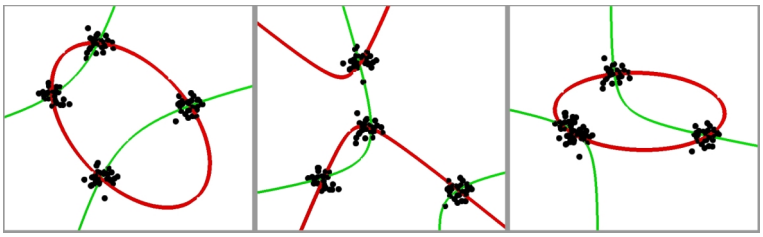

Fig. 1. Examples of four and three clusters of data points and the conics represented by the two eigenvectors with the smallest eigenvalues.

By intersecting conics $\mathbf{c}_1$ and $\mathbf{c}_2$, we basically try to represent the data in terms of up to four points. In effect, this is not much different from a principal component analysis (PCA). The space of intersections of eigenvectors of $\mathbf{W}$ may actually be expressed as the vector space of bivectors in a Clifford algebra over the vector space of symmetric matrices. In this sense, the space of intersections may be regarded as a kind of "second order" null space of $\mathbf{W}$. See [14] for more details.

2.3 Analyzing Image Data

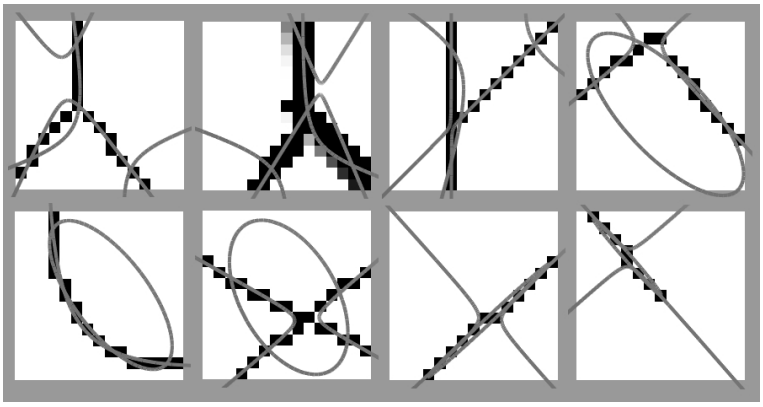

Fig. 2. Examples of typical image structures.

The type of data that we want to analyze with the above described method, are sets of a few line segments, like those shown in figure 2. A standard PCA approach on 2D position vectors will not be of any use in this case, since this would not allow us to distinguish between differently oriented line segments in the same data subspace. Instead we observe that the intersections of conics $\mathbf{c}_1$ and $\mathbf{c}_2$, represent the data in a very useful way, which can be seen in figure 2, where the two conics are drawn on top of the image structures. The images show that the intersection points of $\mathbf{c}_1$ and $\mathbf{c}_2$ lie on the line segments, and that the line segments always lie approximately between two intersection points. Unfortunately, we cannot give an analytic proof that this always has to be the case. However, we can give a motivation for this behavior.

First of all consider the case where only two line segments are present as in corners, crossings and t-junctions. By fitting projective conics to the data, line pairs can be represented well, since they are simply degenerate conics. Hence, the best fitting projective conic will approximate the actual structure in the image. The next best fitting conic is orthogonal to the first and also has to pass somehow through the data, since it is still a fairly good fit. Therefore, the two conics have to intersect on the line segments.

The more complex case is the one where three different line segments meet in a single point, as is the case for y-junctions and ψ-junctions. In this case one pair of line segments can be represented by one branch of one conic, and the last line segment by one branch of the other conic. Hence, the two conics again have to meet on or near the line segments.

In the following we will denote the set of intersection points of $\mathbf{c}_1$ and $\mathbf{c}_2$ in $\mathbb{R}^2$ as $\mathbb{S}_E \subset \mathbb{R}^2$. We use the subscript E for $\mathbb{S}$, since $\mathbb{S}_E$ contains the intersection

points in Euclidean space $\mathbb{R}^2$. If $|\mathbb{S}_E| = 4$, that is, $\mathbf{c}_1$ and $\mathbf{c}_2$ intersect in four points, then there are six unique point pairs between which lines could occur. Typically, only a few of these lines are actually present in the image, though. Therefore, we are not finished once we have found $\mathbb{S}_E$. We also have to check which of the possible six lines have support in the image. Once we have identified such a subset, the last step will be to analyze the extracted line segments and to decide which type of structure, if any, is currently present.

2.4 Intersection of Conics

Finding the intersection points of two 2D conics is not trivial. In general one has to solve a polynomial equation of degree at most four. The method we use is described in detail in [14]. In short, given two conics we find a linear combination of them that represents a degenerate conic, for example a line pair. This degenerate conic then also passes through the intersection points of the two initial conics. This allows us to evaluate the intersection points of the two conics by evaluating the intersection of the degenerate conic with one of the initial conics. This is much simpler than solving a polynomial of degree four, since it results in two polynomial equations of degree two. The only numerically sensitive operation we have to use is the evaluation of eigenvectors and eigenvalues, for which many stable numerical algorithms exist.

2.5 Finding Support for Lines

Given the set of intersection points $\mathbb{S}_E$ of two conics, the question now is which of the $\binom{|\mathbb{S}_E|}{2}$ lines, is actually present in the data, if any. The basic idea is as follows: the number of data points along a line segment should be at least as high as the separation between the two corresponding intersection points measured in pixels. Since the data points give the coordinates of edge pixels, this condition basically says that there is a closed line of pixels between two intersection points. In order to weaken this condition somewhat, we use the following mathematical approach to implement the idea.

Denote by $\mathbb{W} \subset \mathbb{R}^2$ the set of data points, i.e. the set of edge pixels in a local area. Furthermore, let $N = |\mathbb{W}|$ be the number of data points. We take as distance between a data point and a line segment the orthogonal separation of the point from the line segment, if the data point projects onto the line segment. If it does not, then the distance is taken as infinity. The latter condition implements the idea that a data point that does not project onto a line segment should not count at all towards the support of a line segment.

The support of the j^{th} line segment is then given by

$$q_j^{sup} = \sum_{i=1}^{N} \exp\left(-\frac{1}{2}\left(\frac{d_{ij}}{\lambda\, d_{pix}}\right)^2\right), \tag{4}$$

where $d_{ij} \in \mathbb{R}$ is the distance measure between data point i and line segment j, $d_{pix} \in \mathbb{R}$ gives the width of a pixel, and $\lambda \in \mathbb{R}$ is a scale factor. When $d_{ij} = 0$,

then a data point lies directly on the line segment in question. This will then add unity to the support measure q_j^{sup}. The factor λ sets the support data points off the line segment add towards q_j^{sup}. If $d_{ij} \to \infty$, then this will add nothing to q_j^{sup}, i.e. the corresponding data point adds no support to the respective line segment.

In order to decide whether an evaluated support measure q_j^{sup} represents good or bad support for a line segment, we have to evaluate the support that could ideally be expected for the line segment. Ideal support for a line segment means, that the maximum number of pixels possible along the line segment were present. If this is the case, the value of q_j^{sup} will be just this number of pixels. Since we only count those data points that appear between the end points of the line segment, the value q_j^{exp} we should expect for q_j^{sup} can be evaluated as

$$q_j^{exp} := \frac{1}{d_{pix}} \max\left\{ |r_j^1|, |r_j^2| \right\} - 1, \tag{5}$$

where $\mathsf{r}_j := (r_j^1, r_j^2)$ is the direction vector of the j^{th} line segment.

If $q_j^{sup} \geq q_j^{exp}$ we can be sure that the j^{th} line segment has good support in the image. If, however, $q_j^{sup} < q_j^{exp}$ we should give the respective line segment a lower confidence value. The final quality measure for a line segment is therefore evaluated as

$$q_j := \begin{cases} \exp\left(-\frac{1}{2} \left(\frac{q_j^{sup} - q_j^{exp}}{\tau\, q_j^{exp}} \right)^2 \right) : & q_j^{sup} < q_j^{exp} \\ 1 : & q_j^{sup} \geq q_j^{exp} \end{cases}, \tag{6}$$

where $\tau \in \mathbb{R}$ gives a measure of how close q_j^{sup} has to be to q_j^{exp} in order for it to give a high q_j value.

Every $q_j \in [0, 1]$ gives a measure of support for a line segment. The closer the value of q_j to unity, the more likely it is that the respective line segment is also present in the local image area under inspection. Which particular structure is present in the local image area depends on the combination of line segments with good support. It is therefore useful to collect the separate support measures in a support matrix. Let us denote the support value of the line segment between intersection points $\mathsf{s}_i \in \mathbb{S}_E$ and $\mathsf{s}_j \in \mathbb{S}_E$ by $q_{i,j} = q_{j,i}$. The support matrix Q is now defined as

$$\mathsf{Q} := \begin{pmatrix} 0 & q_{1,2} & q_{1,3} & q_{1,4} \\ q_{2,1} & 0 & q_{2,3} & q_{2,4} \\ q_{3,1} & q_{3,2} & 0 & q_{3,4} \\ q_{4,1} & q_{4,2} & q_{4,3} & 0 \end{pmatrix} \tag{7}$$

We can also regard Q as a weight matrix, giving the weights of the edges of a fully connected graph with four vertices. Note that if less than four conic intersection points are found, Q is reduced accordingly.

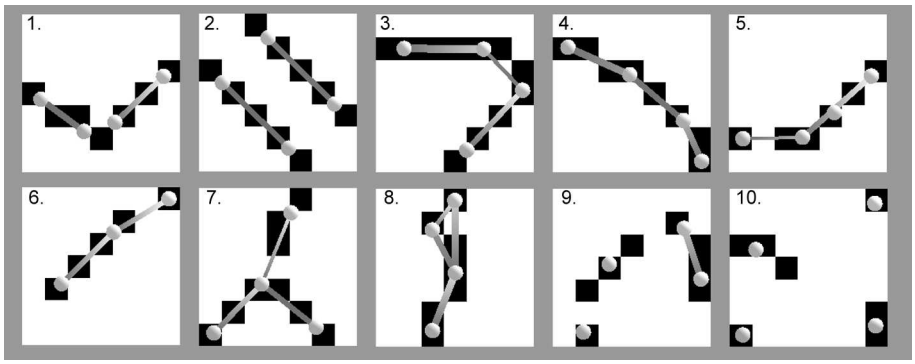

Fig. 3. Examples of analyzed image structures.

3 Analyzing the Line Segments

After the support for the set of possible line segments has been evaluated, we still have to analyze the set of lines and decide on the type of image structure that is present. Figure 3 shows a set of typical structures that are encountered. In this collection of images the round points represent the conic intersection points found and the lines drawn show those lines for which sufficient support was found. The thicker a line, the higher its support value as evaluated in equation (6).

Images 1 and 2 of figure 3 show line pairs. The structures in images 3, 4 and 5 will be called 4-chains. The structure in image 6 is called a 3-chain and image 7 shows a star. The remaining images show spurious structures. An example not shown here is that of a line. When a line is the only element in the local area that is analyzed, the four conic intersection points also lie almost on that line. In the following we will neglect such structures and concentrate on the detection of corners and junctions. The structures we will interpret are thus a line pair, a 4-chain, a 3-chain and a star.

Given a set of intersection points and line segments, the next step is to test the line segment structure for one of the different patterns described above. We will describe the method used with the help of an example. Figure 4 shows the intersection points and the line segments with their respective weights found for an image structure. Let Q denote the support quality matrix for this structure as defined in equation (7). In this case, the values $\mathsf{Q}_{1,2}$, $\mathsf{Q}_{1,3}$ and $\mathsf{Q}_{1,4}$ are close to unity and the values $\mathsf{Q}_{2,3}$, $\mathsf{Q}_{3,4}$ and $\mathsf{Q}_{4,2}$ are close to zero. We can therefore evaluate a measure of confidence that the present structure is a star as

$$C = \left(\mathsf{Q}_{1,2}\,\mathsf{Q}_{1,3}\,\mathsf{Q}_{1,4}\right)\left(1 - \mathsf{Q}_{2,3}\,\mathsf{Q}_{3,4}\,\mathsf{Q}_{4,2}\right) \tag{8}$$

That is, we have to test for a positive and a negative pattern. Since the numbering of the intersection points is arbitrary, the above measure will in general have to be evaluated for all permutations of $\{1, 2, 3, 4\}$. In order to formulate this

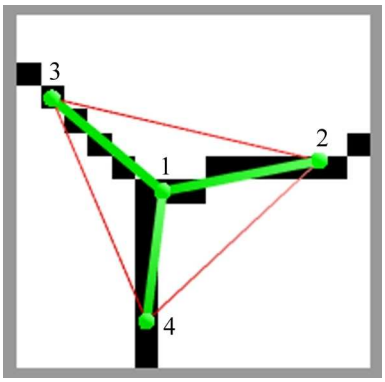

Fig. 4. Examples of a junction structure.

mathematically, let us denote by $\mathbf{i}$ and index vector defined as $\mathbf{i} := (i_1, i_2, i_3, i_4)$. We can then define a positive (p^+) and a negative (p^-) pattern that we expect for a particular structure. In the case of the star structure, these patterns are

$$\begin{aligned} p^+(\mathbf{i}) &= \Big((i_1, i_2),\, (i_1, i_3),\, (i_1, i_4)\Big), \\ p^-(\mathbf{i}) &= \Big((i_2, i_3),\, (i_3, i_4),\, (i_2, i_4)\Big). \end{aligned} \tag{9}$$

In the following let p_k^+ denote the kth index pair of p^+, and analogously for p^-. In order to improve the readability of the following formulas, we will also write $\mathsf{Q}[i_1,\, i_2]$ in order to denote the element Q_{i_1,i_2}. The confidence value for the star pattern for a particular $\mathbf{i}$ may then be written as

$$C\Big(p^+(\mathbf{i}),\, p^-(\mathbf{i})\Big) = \left(\prod_k Q[p_k^+(\mathbf{i})]\right)\left(1 - \prod_l Q[p_l^-(\mathbf{i})]\right) \tag{10}$$

The permutation of $\mathbf{i}$ that gives the largest value of $C(p^+(\mathbf{i}),\, p^-(\mathbf{i}))$ then allows us to evaluate the central point of the star (i_1) and the three end points (i_2, i_3, i_4). We will denote this value of $\mathbf{i}$ as $\hat{\mathbf{i}}$, with

$$\hat{\mathbf{i}} = \arg \max_{\mathbf{i} \in \mathrm{perm}\{(1,2,3,4)\}} C\Big(p^+(\mathbf{i}),\, p^-(\mathbf{i})\Big), \tag{11}$$

where $\mathrm{perm}\{(1, 2, 3, 4)\}$ denotes the set of index vectors of permutations of $(1, 2, 3, 4)$.

For each structure that we would like to test for, we can define a positive (p^+) and a negative (p^-) pattern and then evaluate the confidence value $C(p^+(\hat{\mathbf{i}}),\, p^-(\hat{\mathbf{i}}))$ on a given Q matrix. In our implementation of the algorithm we test for the star, the 4-chain, the 3-chain, the 3-chain with a disjoint point and the line pair. Typical examples of these structures are shown in figure 5. Note

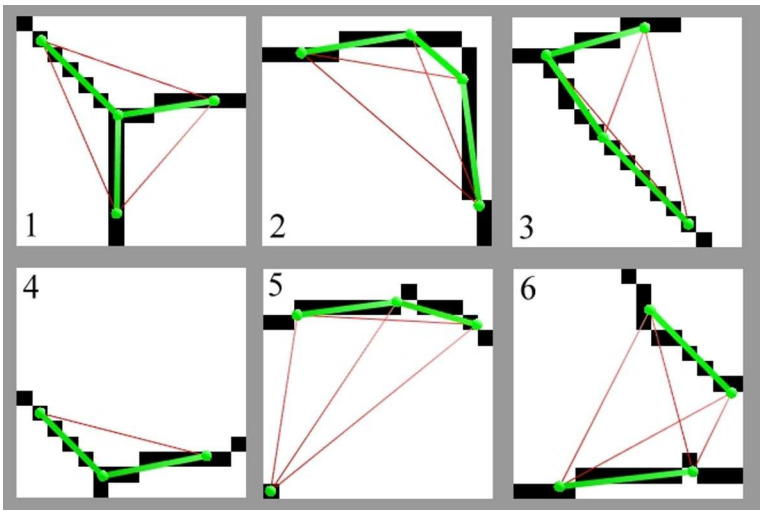

Fig. 5. Examples of structures tested for. 1. Star, 2. & 3. 4-Chain, 4. & 5. 3-Chain, 6. Line Pair.

that image 5 shows a 3-chain with a disjoint point and images 2 and 3 both show 4-chains. The latter two structures should be interpreted in different ways. While image 2 can be interpreted as a double corner, image 3 should be interpreted as a single corner. This shows that by finding the best matching structure to a local image area, we still cannot make a final decision on what the structure represents.

3.1 Analyzing Line Structures

For each of the structures we test for, we obtain a confidence value $C(p^+(\hat{\mathbf{i}}), p^-(\hat{\mathbf{i}}))$. The structure with the highest confidence value is then analyzed further to decide whether it represents a corner, a double corner or a junction. One could also test for a curve, a line or a line pair, but in this text we are mainly interested in finding corners and junctions.

The Star. The star can be interpreted immediately. Since we have $\hat{\mathbf{i}}$ we know which of the intersection points is the central point and which are the three edge points. From this the position of the junction in the image and the angles between the legs can be readily evaluated. If the angle between two legs is nearly 180 degrees one may also call the junction a t-junction and otherwise a y-junction.

The 4-Chain. A 4-chain can represent a number of different entities: a corner, a double corner, a curve and a line. The difference between these entities cannot be defined strictly in general. Which structure is present depends on the angles between the legs of the 4-chain. Here thresholds have to be set that are

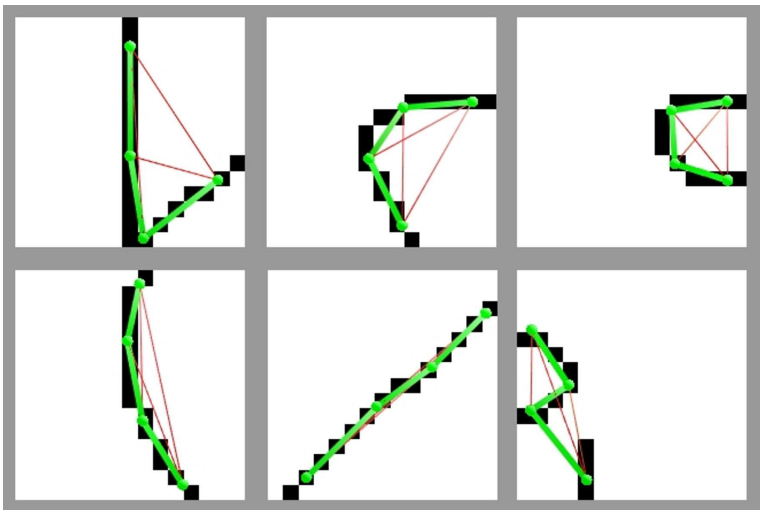

Fig. 6. Examples of entities that can be described by a 4-chain. From top-left to bottom-right: corner, double corner, double corner, curve, line, "snake".

most appropriate for the current application. In this text we will concentrate on distinguishing between corners and junctions. Therefore, a curve will also be interpreted as a corner with large opening angles. See figure 6 for examples of these structures.

Two angles (α_1, α_2) can be evaluated between the three legs of a 4-chain. Since we always take the smaller angle between two line segments, we have to make sure that the present 4-chain does not have a form as in the bottom-right image of figure 6, which we will call a "snake". This can be checked by evaluating the cross products of the directions of the line segment pairs from which the angles are evaluated. If the resultant vectors point in opposite directions, then the 4-chain describes a snake.

The other structures are distinguished using α_1 and α_2 as follows.

- **Line**, $\alpha_1 > 170°$ and $\alpha_2 > 170°$.
- **Double Corner**, $\alpha_1 < 150°$ and $\alpha_2 < 150°$.
- **Corner**, in all other cases. The corner is given by the two line segments with the smaller angle between them.

The 3-Chain. A 3-chain either describes a corner or a line. It usually appears if one of the intersection points of the conics lies outside the local image area under investigation and is thus neglected. A 3-chain can also appear if two intersection points are so close to each other that they are combined into a single point.

The Line Pair. A line pair either describes two disjunct lines which we will not interpret further, or a crossing of two lines. These two cases can be

distinguished quite easily by evaluating the intersection point of the two lines given by the extension of the line segments. If the intersection point lies on both line segments, then we found a crossing.

3.2 Translation Invariance of Structure Analysis

Using the analysis described in the previous sections, we can obtain for each local area in an image an interpretation of the area's structure, where we distinguish between corners and junctions. For every corner we obtain its location, its opening angle and its orientation. Junctions may be separated into y-junctions, t-junctions and crossings. For each of these we also obtain their location, orientation and angles.

The same structure found in one local area is also likely to be found in neighboring areas, whereby each time the structure has the same absolute position. This follows, since the method described here is translation and rotation invariant for one data set. In a real image, however, translating a local area will remove some edge points from the local area and others will appear. For some examples of translation invariance see [14].

Nevertheless, typically a particular corner or junction may not only be found at one particular test position. Instead, strong structures are likely to appear for a set of neighboring test positions. This offers the possibility of applying a clustering procedure to the corners and junctions found, in order to stabilize the output of the algorithm. However, this has so far not been implemented.

4 Experiments

Before the structure analysis algorithm can be applied, the edges of an image have to be extracted. This was done using the Canny edge detector [3]. The initial image and the result of the edge detection can be seen in figure 7. The algorithm was applied to this edge image, whereby a test window of 15×15 pixels was moved over the image in steps of two pixels. The factor λ from equation (4) was set to 0.1 and the factor τ from equation (6) to 0.2.

Recall that equation (11) gives a confidence value for a structure. This confidence can be used to measure the confidence we can have in a corner or junction found. The junctions found are shown in figure 8. Here the left image shows all junctions and the right image only those junctions with a confidence value of 0.90 or higher. The images in figure 9 show those corners with a confidence value of 0.99 or higher and an opening angle between 0 and 150 degrees, and 0 and 110 degrees, for the left and right image, respectively.

From the images shown here it can be seen that the algorithm finds all important corners and also gives a good measure of their opening angle. Furthermore, almost all junctions were found. Junctions that were not detected have fairly large gaps in their contour with respect to the size of the test window. Three spurious junction were found. These false positives occurred at places where the gap between two separate structures became so small that they appear locally as

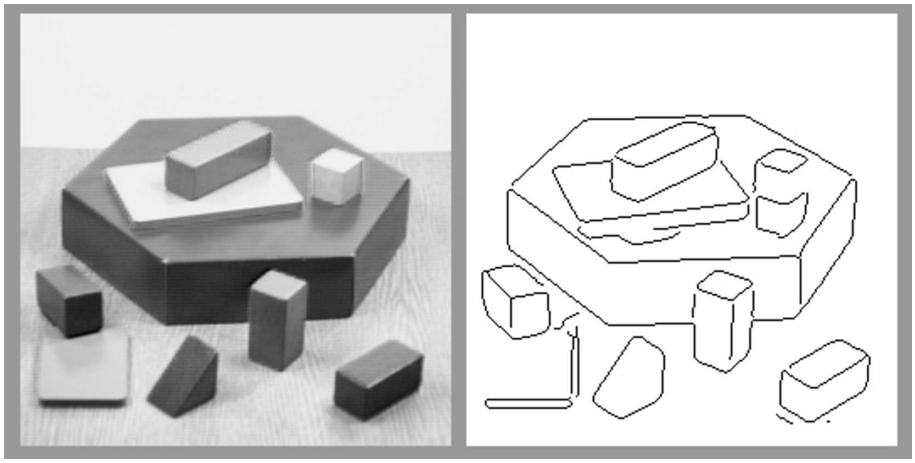

Fig. 7. Example image "blox" (left), and the extracted edges (right).

one structure with some missing pixels. The problem that manifests itself here is, that within a small test window, structures can only be interpreted locally. Global relationships are not taken into account which leads to false positives and false negatives.

The two main problems the algorithm faces are the following:

- Corners and junctions only become apparent at a particular scale. If the scale is chosen too small, many spurious corners may be found. If it chosen too large, too much structure may be present in a test window such that the algorithm fails.
- Edges may be incomplete. If there are only small gaps in the edges, the algorithm may still give good results. However, if the gaps become too large with respect to the test window, structures will not be detected correctly. Here the balance has to be found between bridging gaps and detecting corners and junction where there are none.

The effect noise in the initial image has on the algorithm depends largely on the noise sensitivity of the initial edge detection. The more robust the edge detection, the better the algorithm will work. Note that more experimental results can be found in [14].

5 Conclusions

We have presented an algorithm that uses conics to analyze local image structure. The main idea is to fit intersections of conics to the data. It was found that these intersections can represent local image structures in a very useful way: the line segments that make up the local image structure lie between intersection point pairs. This basically reduces the search space of possible line segments to at most

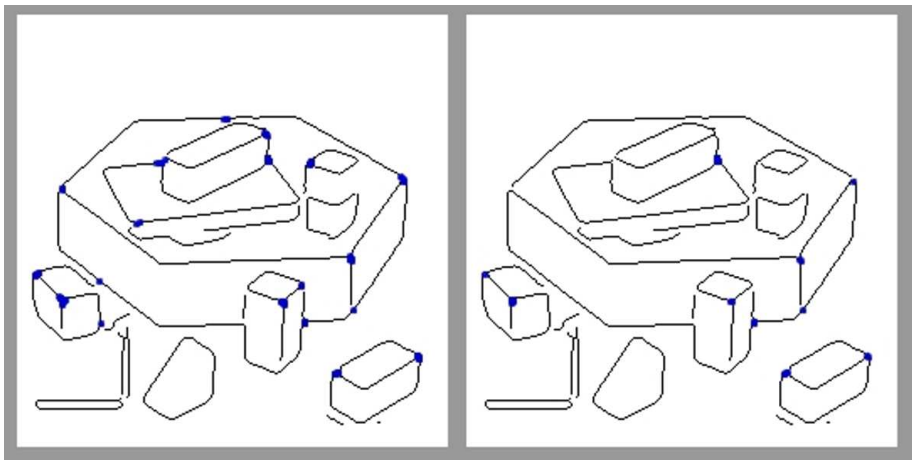

Fig. 8. Detected junctions in example image "blox", with confidence value ≥ 0.50 (left) and ≥ 0.90 (right).

six specific ones. It was then shown that through a combinatorial analysis of the resultant graph it is possible to extract corners and junctions from an image and to evaluate their parameters.

Compared with algorithms that use the gradient field directly for the corner and junction detection, the algorithm presented here does not work directly on the image data. Instead, an initial edge detection abstracts somewhat from it. This means that the edge detection algorithm has to deal with most of the noise present in an image. The type of noise the analysis algorithm then has to be able to cope with are incomplete edges. Clearly, the better the edge detection, the better the results the analysis algorithm generates.

A potential advantage of the presented algorithm over standard corner detectors is that it can distinguish between different types of i2D structures. Furthermore, it can be used to extract parameters of the local image structures, like the opening angle of a corner.

References

1. S. Baker, S. K. Nayar, and H. Murase. Parametric feature detection. *IJCV*, 27(1):27–50, 1998.
2. F.L. Bookstein. Fitting conic sections to scattered data. *Comp. Graph. Image Proc.*, 9:56–71, 1979.
3. J. Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8(6), November 1986.
4. M.A. Cazorla, F. Escolano, R. Rizo, and D. Gallardo. Bayesian models for finding and grouping junctions. In *Second International Workshop on Energy Minimization Methods in Computer Vision and Pattern Recognition*, 1999. York.
5. M. Felsberg and G. Sommer. The monogenic signal. *IEEE Transactions on Signal Processing*, 49(12):3136–3144, December 2001.

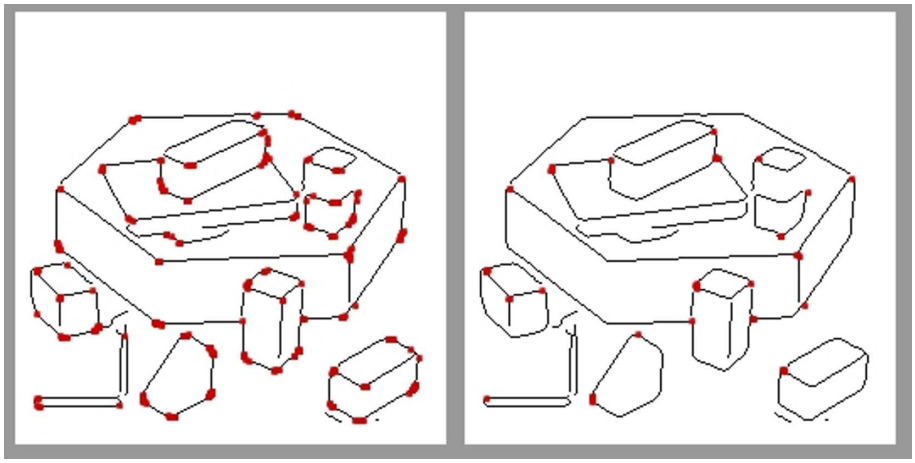

Fig. 9. Detected corners in example image "blox", with confidence value ≥ 0.90 and opening angles between 0 and 150 degrees (left), and 0 and 110 degrees (right).

6. M. Felsberg and G. Sommer. Image features based on a new approach to 2D rotation invariant quadrature filters. In A. Heyden, G. Sparr, M. Nielsen, and P. Johansen, editors, *Computer Vision, ECCV02, Kopenhagen, 2002*, volume 2350 of *LNCS*, pages 369–383. Springer, 2002.
7. W. Förstner. A feature based correspondence algorithm for image matching. *Intl. Arch. of Photogrammetry and Remote Sensing*, 26:150–166, 1986.
8. W. Förstner. A framework for low level feature extraction. In J. O. Eklundh, editor, *Computer Vision - ECCV'94*, volume 2 of *LNCS 801*, pages 383–394. Springer-Verlag, 1994.
9. Gösta H. Granlund and Anders Moe. Unrestricted recognition of 3-D objects using multi-level triplet invariants. In *Proceedings of the Cognitive Vision Workshop*, Zürich, Switzerland, September 2002. URL: http://www.vision.ethz.ch/cogvis02/.
10. C. G. Harris and M. J. Stevens. A combined corner and edge detector. In *Proc. of 4th Alvey Vision Conference*, 1988.
11. U. Köthe. Edge and junction detection with an improved structure tensor. In B. Michaelis and G. Krell, editors, *Pattern Recognition*, LNCS 2781, pages 25–32. Springer-Verlag, 2003.
12. D. G. Lowe. Local feature view clustering for 3d object recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 682–688, 2001.
13. F. Mokhtarian and R. Suomela. Curvature scale space for robust image corner detection. In *Proc. International Conference on Pattern Recognition*, pages 1819–1821, 1998.
14. C. Perwass. Analysis of local image structure using intersections of conics. Technical Report Number 0403, Christian-Albrechts-Universität zu Kiel, Institut für Informatik und Praktische Mathematik, July 2004.
15. M. Shpitalni and H. Lipson. Classification of sketch strokes and corner detection using conic sections and adaptive clustering. *Trans. of ASME J. of Mechanical Design*, 119(2):131–135, 1997.