Introduction to Computational and Biological Vision

Project Report

Edge and Boundary Interpretation Via Relaxation labeling

Students:
Leonid Leontev
314007188
Anna Bakshi
323716399

1 Project Goals

The project aims to implement edge and boundary detection algorithm using Huffman and Clowes catalogue of possible thrihedral vertices types.
We aim to develop a program which given a picture with some object on it will resolve all possible interpretations of boundaries of the object.
Because we are using the Huffman and Clowes catalogue, the objects that can be interpreted are limited to be objects that possess only vertices that appear in the catalogue.

2 Course of Action

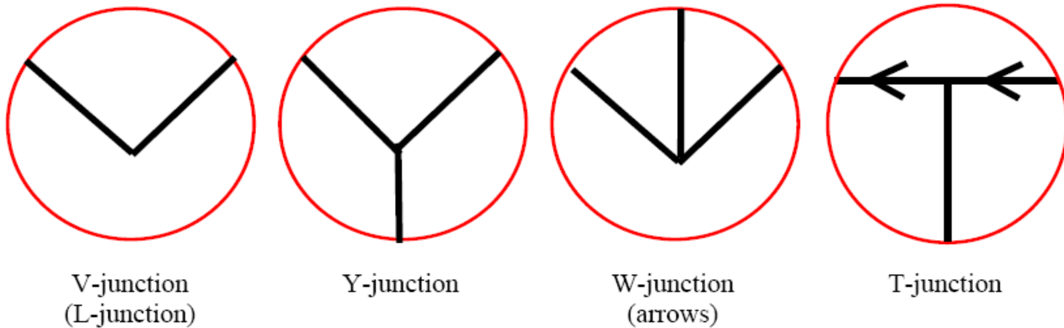
The project implements the following algorithm for edge and boundary interpretation:

Consistent line drawing labeling via relaxation labeling

1. Initial the label set for each line drawing label to {+,-,<,>}
2. Repeat at all edges concurrently until no label set decreases in size
   - If label *L* at edge *e* cannot form a consistent junction using available labels at edges intersecting *e* in a common vertex, filter *L* from *e*'s label set

To implement the algorithm having an input image we divided the problem to the following steps:

1. Resolve the corners of the object
2. Determine the types of the resolved corners according to thrihedral model of Huffman and Clowes:



3. Apply the algorithm of Consistent line drawing labeling via relaxation labeling on the vertexes from step 2.
4. Display the obtained interpretations

All the listed steps are described in detail in the implementation section.

3 Implementation

In this section the steps of the algorithm are described in detail. For each step we describe the problems and the solution we found.

1. Resolve the corners of the object

The algorithm of edge and boundary interpretation is based on operating with object's vertices, where each vertex is a corner with the line segments that are connected to it. Each vertex of the object according to the definition is located in corner. This is the reason we first needed to detect all the corners of the object.

Corner detection overview

Corner detection is an important task in various computer vision and image-understanding systems [1]. Corner detection should satisfy a number of important criteria:

- All the true corners should be detected
- No false corners should be detected
- Corner points should be well localized
- Corner detector should be robust with respect to noise
- Corner detector should be efficient

There are different methods for corner detection in literature. Each one is based on different principle. Each method addresses different problem in corner detection. For example, [2] uses a generalized Hough transform for edge detection, where the transform is needed to detect the edge lines (boundaries) of the object and the generalization is performed to cope with the corners that are not sharp. On the other hand [1] analyses the curvature scale of object's contour and extracts the points with the maxima of absolute curvature.

In the project we used the latter method for corner detection. We used the implementation of X.C. He and N.H.C. Yung, 'Curvature Scale Space Corner Detector with Adaptive Threshold and Dynamic Region of Support', Proceedings of the 17th International Conference on Pattern Recognition, 2:791-794, August 2004. The code is attached.

The method is based on CSS (The Curvature Scale-Space Technique) method.

CSS Overview

The CSS technique is suitable fro recovering invariant geometric features (curvature zero-crossing points and/or extrema) of a planar curve at multiple scales. To compute it, the curve Γ is first parameterized by the arc length parameter u:

$$\Gamma(u) = (x(u), y(u))$$

An evolved version Γ_σ of Γ can then be computed.

$$\mathbf{\Gamma}_\sigma = (X(u,\sigma), Y(u,\sigma)),$$

where

$$\mathcal{X}(u,\sigma) = x(u) \otimes g(u,\sigma) \qquad \mathcal{Y}(u,\sigma) = y(u) \otimes g(u,\sigma),$$

where $\otimes$ is the convolution operator and $g(u,\sigma)$ denotes a Gaussian of width σ. In order to find curvature zero – crossings or extrema from evolved versions of the input curve, one needs to compute curvature:

$$\kappa(u,\sigma) = \frac{\mathcal{X}_u(u,\sigma)\mathcal{Y}_{uu}(u,\sigma) - \mathcal{X}_{uu}(u,\sigma)\mathcal{Y}_u(u,\sigma)}{\left(\mathcal{X}_u(u,\sigma)^2 + \mathcal{Y}_u(u,\sigma)^2\right)^{1.5}},$$

where

$$\mathcal{X}_u(u,\sigma) = x(u) \otimes g_u(u,\sigma) \qquad \mathcal{X}_{uu}(u,\sigma) = x(u) \otimes g_{uu}(u,\sigma)$$
$$\mathcal{Y}_u(u,\sigma) = y(u) \otimes g_u(u,\sigma) \qquad \mathcal{Y}_{uu}(u,\sigma) = y(u) \otimes g_{uu}(u,\sigma).$$

CSS Outline

The corners are defined as the local maxima of the absolute value of curvature. At a very fine scale, there exist many such maxima due to noise and the digital contour. As the scale is increased, the noise is smoothed away and only the maxima corresponding to the real corners remain. The CSS corner-detection method finds the corners at these local maxima

The process of CSS image corner detection is as follows:

- Utilize the Canny edge detector to extract edges from the original image.
- Extract the edge contours from the edge image:
  —Fill the gaps in the edge contours.
  —Find the T-junctions and mark them as T-corners.
- Compute the curvature at highest scale σ_{high} and determine the corner candidates by comparing the maxima of curvature to the threshold t and the neighboring minima.
- Track the corners to the lowest scale to improve localization.
- Compare the T-corners to the corners found using the curvature procedure and remove corners which are very close.

The following is an explanation of each stage of the CSS corner detector.

Steps description:

1. Here canny edge detection was used, but it may be replaced by any other edge detector
2. The canny edge detector can cause gaps at T-junctions and the corners may not be found with the CSS method.
   - If the endpoint is nearly connected to another endpoint, fill the gap and continue the extraction
   - If the endpoint is nearly connected to an edge contour, but not to another endpoint, mark this point as a T-junction corner.

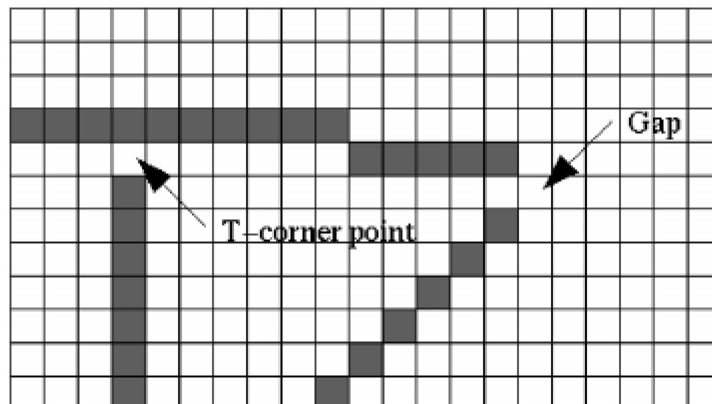


Figure1: Two cases of gaps in the edge contours

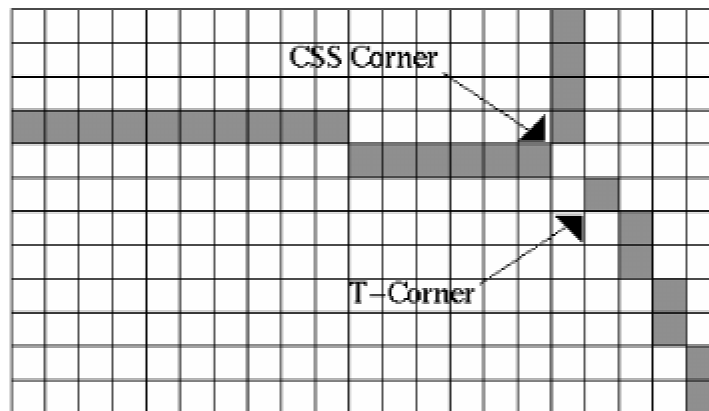
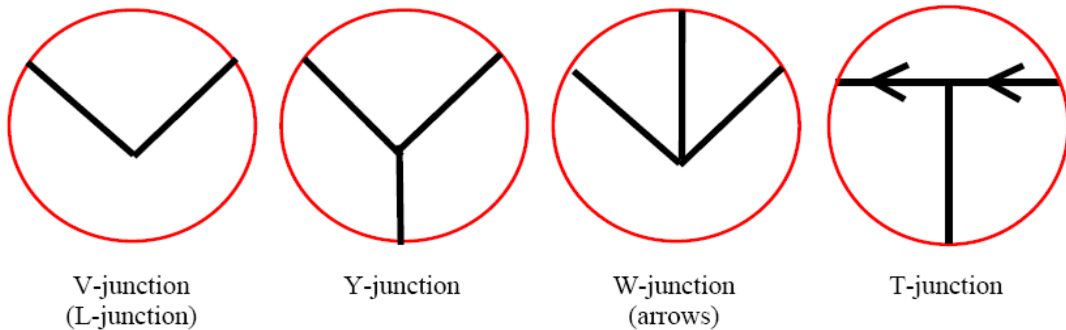


Figure 2: case where one corner is marked twice

3. The edge contours are extracted from the edge image and the absolute value of curvature is computed at the initial scale sigma(high). The local maxima of absolute curvature are the possible candidates for corner points. A local maximum is either a corner, the top value of a rounded corner or a peak due to noise. The latter two should not be detected as corners. The curvature of a real corner point has a higher value than that of a rounded corner or noise. The corner points are also compared to the two neighboring local minima. The curvature of a corner should be twice that of one of the neighboring local minima. This is because when the shape of the contour is very round, contour curvature can be above the threshold t. The threshold t depends on sigma(high) used and it is set according to it.
4. After the initial corner points are located, tracking is introduced to the detection. As the corners were detected at scale sigma(high), the corner localization might not be good. The curvature is computed at a lower scale and the corner candidates examined in a small neighborhood of the previous corners. Corner locations are updated, if needed, in this neighborhood. Tracking is continued until scale is very low. This process gives very good localization. No thresholding is needed in the tracking. The number of corners is determined at the initial sigma(high), and tracking only changes the localization, not the number of corners. Tracking improves the localization of the corners. Corners do not move dramatically during tracking and only a few other curvature values need to be computed.
5. As described before, corners are declared using two methods and, in some cases, the two methods mark the same corner. In Figure 2, the case where one corner is marked twice is shown. The edge extraction algorithm examines a small neighborhood when it arrives at the end of a contour. The corner in Figure 2 is a Y junction and it is marked twice. The CSS method finds a corner on the continuous contour and the edge extraction algorithm marks a T-corner at the end of the other contour as it is nearly connected to a continuous edge contour. The final part of the algorithm is to examine the points marked by the edge-extraction algorithm. These T-junction corners are compared to the corner points found with the CSS method and if they are very close to each other, the T-junction corners are removed.

Thus we used the above algorithm for corner detection; the input of the algorithm was the initial Image and the output is the list of coordinates of the detected corners. Now we can move to the next step.

2. **Determine the types of the resolved corners according to thrihedral model of Huffman and Clowes.**
   At this step we have detected all corners of the object. Now we need to resolve the types of vertices to which each corner belongs. Because we are working with Huffman and Clowes catalogue of thrihedral vertices the types of vertices are limited and what we need is to assign to each corner the appropriate type. The vertices types are as follows:



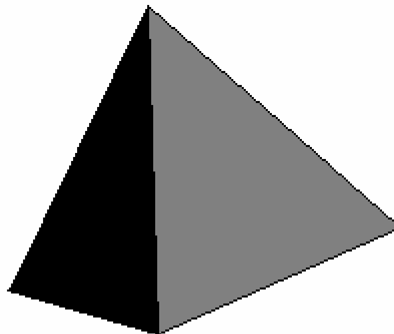
V-junction (L-junction) | Y-junction | W-junction (arrows) | T-junction

We divided the problem to the following steps:
1. For each corner detect the line segments that are connected to it
2. For each detected vertex (which is a corner and the line segments that are attached to it) determine the type of the corner and order the lines in clockwise order

Steps description:

1. **Detect the line segments that are connected to the corner**
   Idea:
   1.1. First we apply canny edge detector on the image.
   1.2. On the resulting binary image we apply Hough transform for lines detection (Matlab implementation was used)
   1.3. For each corner we scan the neighborhood in radius of some predefined threshold to detect the edge points of segments that are connected to this corner

   Problematic issues:
   Step 1.1
   The resulting image after edge detection smoothes the corners and in some cases even cuts the edges, additionally the edge detector adds some extra edges (as might be seen from the following figures):



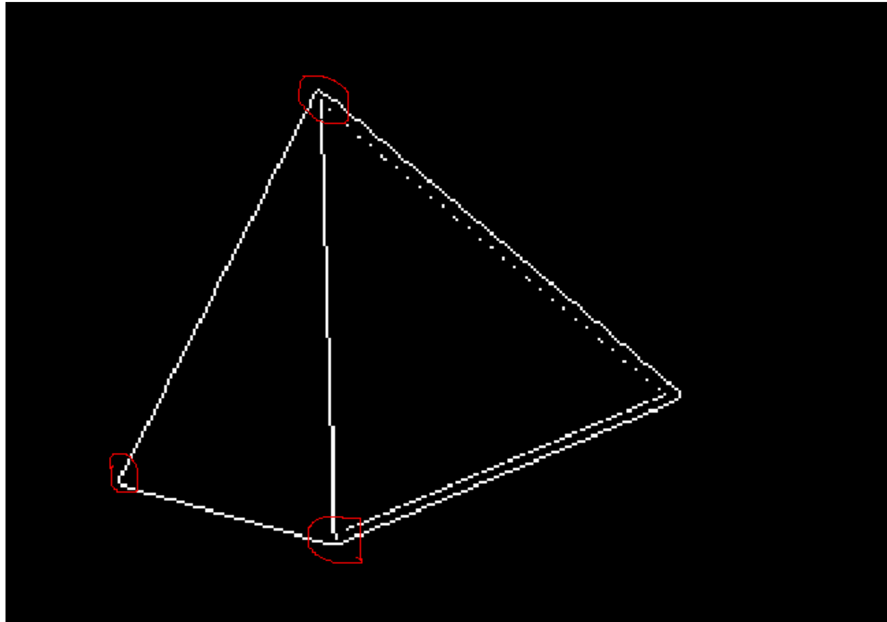


Figure 4: The pyramid object after canny edge detector

To avoid the case of adding extra lines we require the input object to possess a sharp contrast between the object faces, additionally the rib that connect two faces is the meeting of two faces of different colors, the rib should not be signified by some different third color:

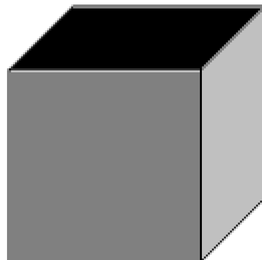


Figure 5: Cube with ribs whose colors are different from the colors of two faces the rib connects

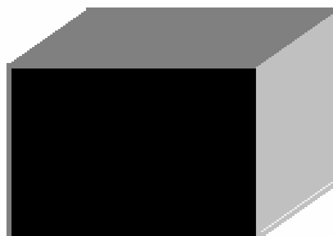


Figure 6: Cube object with ribs whose color is not different from the colors of two faces the rib connects

Step 1.2
It is difficult to draw digital image with absolutely straight lines. As an example you can see Figure 4, the middle rib of the pyramid consists of 5 short segments. Thus we do not need all the segments the Hough transform detects, but only the longest.
The problem is to determine a correct thresholds on the longest line segments, they are vary from object to object, additionally, even in the same object there might be either legal short and long segments, so the global threshold will be no good.
To eliminate the short liens that are not valid we performed the following:
After Hough transform we pass through all the received line segments and if we find two that are similar then the longest one is left and the shortest is removed.
Similarity test:
Let S1 be a line segment with rho1 and theta1, and S2 be a line segment with rho2 and theta 2,
If (rho1 – rho2) < rho_thershold && (theta1 – theta 2) < theta_thershold
Then the segments belong to the same edge

Step 1.3
The output of Hough transform is a set of detected segments with start and end edge point for each line. Now having the segments and the corners coordinates we scan the neighborhood (the radius is predefined by some threshold) of each corner and seek for the line segments whose end points lie within the circle defined by the radius.
It is important to distinguish between two cases:

1. The line segment is connected with one of it edge points to the corner
2. The corner lies in the interior of line segment

The first case was checked in the following way:
Given and edge point (Ex, Ey) and corner point (Cx, Cy) the edge belongs to the corner if the distance between two points is less or equal to the predefined threshold

Threshold >= sqrt((Ex-Cx)^2 +(Ey - Cy)^2)

This threshold is not good enough for all objects, as might be seen from the examples section Results.

To check the second case we used the following test:
Let L be a current line segment with edge points (Ax, Ay), (Bx, By), and C be the current corner with coordinates (Cx, Cy). Consider the following triangle:
(A,C,B):
If the angle $\angle CAB \text{ and } \angle CBA \geq 90°$ then corner C does not lie in the interior of Segment L, else it does. The corners are calculated according to cosine theorem:

$$c^2 = a^2 + b^2 - 2ab\cos C$$
$$\text{where } C \text{ is the corner in front of corner } c$$

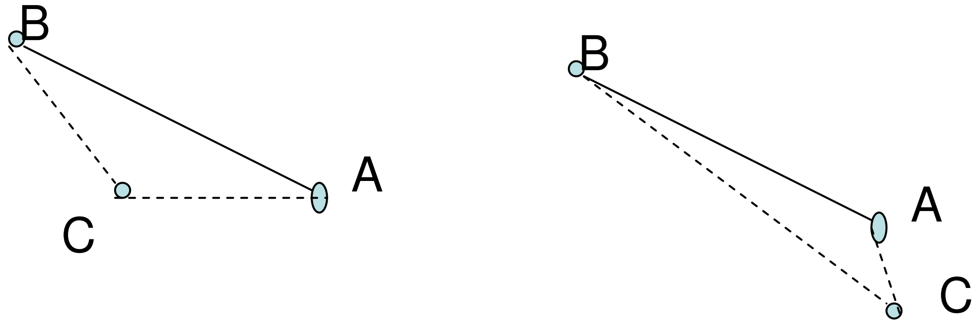


Figure 11: The left example demonstrates the case when corner C lies on the interior of line AB. The right figure demonstrates the case when corner C does not lie on the interior of the line AB.

For each corner we save the detected lines and the type of line connection to the corner (Line's edge point that connects the line to the corner (in the first case), or in the second case variable that specifies that the corner lies in the interior of the line)

The main problem in this stage is to define the proper thresholds so one corner is connected to maximum three line segments. We do not handle the case when more than three lines or less than one line are connected to the corner. We assume that the lines are detected properly and we take reasonable thresholds.
Now we have the vertices and can move to the next step.

2. Determine the type of the corner and order the lines in clockwise order

At this step we have vertices with line segments that are connected to them and the information about the connection type. We need to assign to each vertex the appropriate label according to thrihedral model catalogue of Huffman and Clowes. Additionally, for each vertex we need to order the connected line segments in a clockwise order. We need the ordering for the next steps when we apply the catalogue labels to the vertexes. For example consider the w vertex with its three line segments. On the Figure 6 the ordering is presented. Notice that according to catalogue shown in Figure 7, line segment 2 cannot posses the depth label at all, also line segments 1 and 3 may possess only depth labels that are pointed left. For other vertexes we have similar limitations.

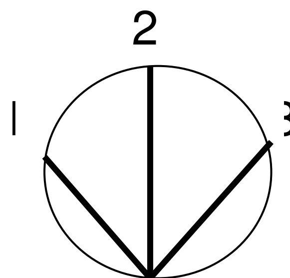


Figure 6: Example of ordering the line segments of w vertex

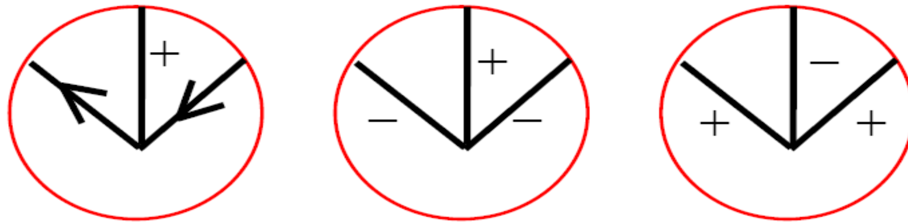


Figure 7: All possible labeling for w vertex according to Huffman and Clowes catalogue

To accomplish this step we perform the following operations:

Idea:

2.1. For each line segment in vertex determine the angel with respect to x aces
2.2. Sort all line segments in each vertex according to angles in ascending order (clockwise order)
2.3. Assign the types to the labels and perform the final sort of the lines inside each vertex

Problematic issues:

Step 2.1

For each vertex we calculate the angles of line segments associated with this vertex. The angles are calculated in the following manner:

Given a line segment with its edge points: (Ax, Ay), (Bx, By) where A is the point of connection to the corner, the angle of this segment with respect to x aces (the coordinates of the edge points were given according to this aces) is:

$$atan(Ax-Bx, Ay - By)$$

Step 2.2

For each vertex sort the line segments in ascending order which is in clockwise order. Figure 7 demonstrates the possible orderings of three special cases. All other cases are simpler.

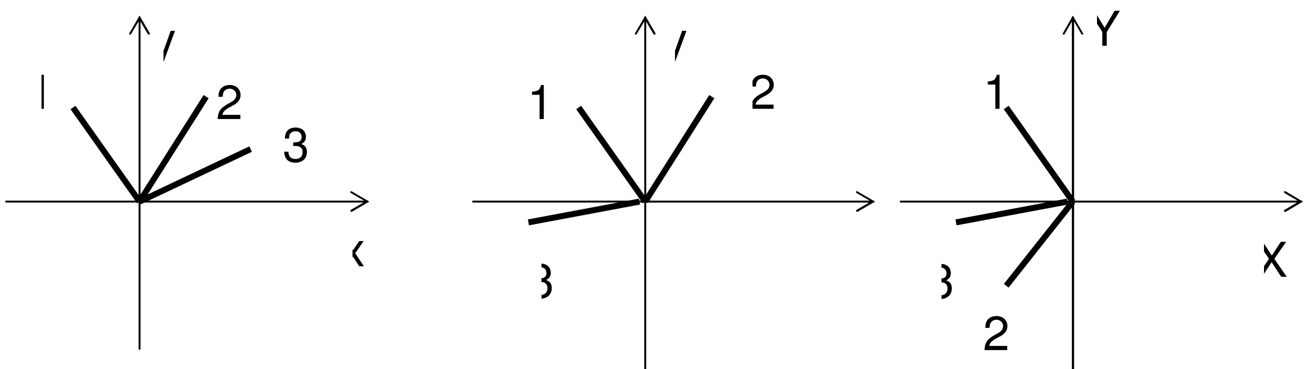


Figure 7: The example of three cases of ordering the line segment of vertex. The numbers specify the obtained order

Step 2.3

After the step 2.2 there is a preliminary order of line segments in each vertex

The types of vertexes and the final order are determined by the following ways:
Vertexes with two lines:

If the type of connection of one of the lines is T-junction (the interior of the line contains the corner)

first is the line that is connected with its edge points to the corner, and the second will be the line that contains the corner in its interior

Else

The vertex is a V-junction and the order is as follows:
Compute the difference between angels of the line segments, if the difference is greater than pi, then switch the order of lines (this solves the vertex illustrated in Figure 8), else leave as is.

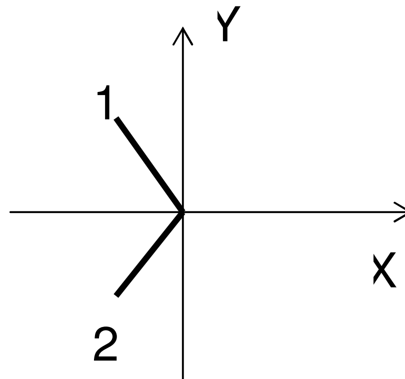


Figure 8: Example of V-junction, the numbers specify the order

Vertexes with three lines:

Denote the angle of line (from step 2.2) segment to be (1) where 1 specifies the order number of the line obtained at step 2.1

If (1) – (3) > 180 and (1) – (2) <180 and (2)-(3)<180
Then it is a Y-junction and the order remains as is
Else this is a W-junction
If (1) – (3) > 180 and (1) – (2) <180
then the case is the middle graph on Figure 7, switch the order: 3-> 1; 1->2;2->3
if (1) – (3) > 180 and (2)-(3)<180
then the case is the left graph on Figure 7, switch the order 2->1;3->2;1->3
Else the case is the right graph on Figure 7 and the order stays as is

Now we have vertexes with their types and order of line segments connected to each vertex. Now we can apply the algorithm of relaxation labeling on the vertexes.

3. Apply the algorithm of Consistent line drawing labeling via relaxation labeling on the vertexes

The algorithm is implemented as follows:

1. Assign for each line segment all possible labels:

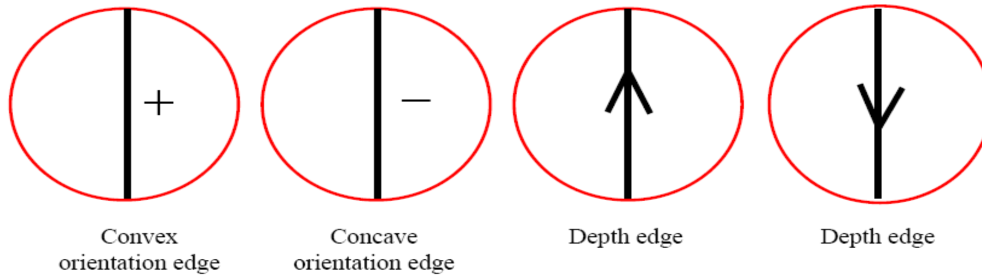


Figure 9: All possible labels that can be assigned to line segment

2. For each vertex perform:
   For each line that is connected to the vertex remove the labels from line that are not valid according to Huffman and Clowes catalogue.

Step 2 is iterative and is performed until no label can be removed

Problematic Issues:
Step 1

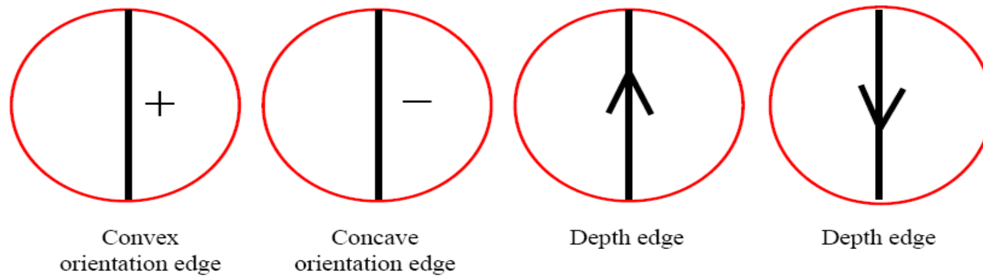
When assigning the labels of depth edge to line we need to specify the direction of the depth edge. The problem was that fro each vertex from step 2 the same label can possess a different direction depending upon the position of the vertex. To solve this problem we define the absolute direction for each line:

Given a line segment with edge points (Ax,Ay), (Bx, By), where the points order is as they were returned from Hough transform. Let (Ax,Ay) be the first point and (Bx, By) be the second point, then there are two direction from (Ax,Ay) to (Bx, By) and the opposite.

This absolute direction is used at step 2.

When dealing with a T-junction we could not distinguish between the cases:

4. **Display the obtained interpretations**
   At this step we display near the center of each line segment the valid labels, according to catalogue



Additionally, all the detected vertexes of the image are displayed and an image with all detected lines is displayed.

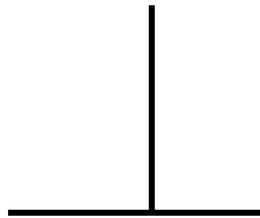
4 Results

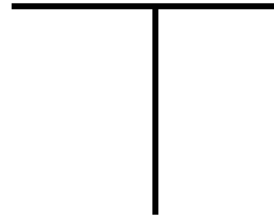
The result are far from satisfying. The main problem is that the algorithm itself performs good only if the input object was clear enough so all the correct object line segments were detected correctly.

Our implemen

Input limitations:

1. Input images have to contain objects with vertexes from catalogue only
2. The ribs of the objects must not be colored
3. There must be a sharp contrast between the colors of object's faces
4. We did not solve the problem of T-junction mentioned earlier, thus the object must not contain the T-junction of the type:





Inputs:

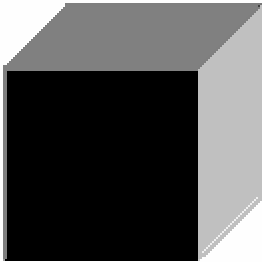


Figure 10: Input object cube

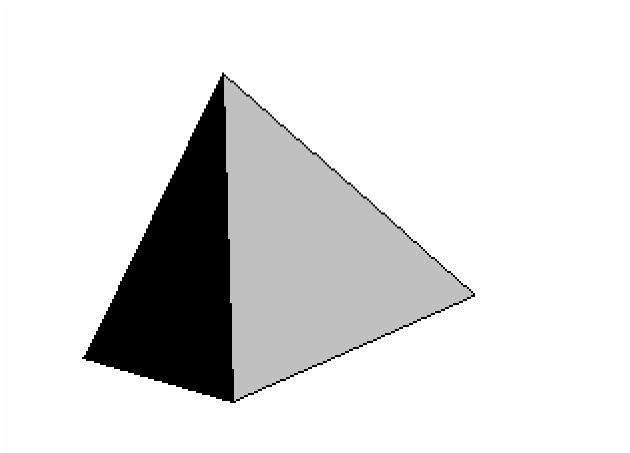


Figure 11: Input object pyramid

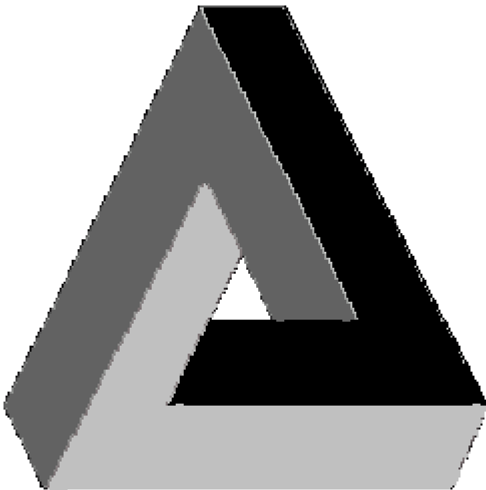
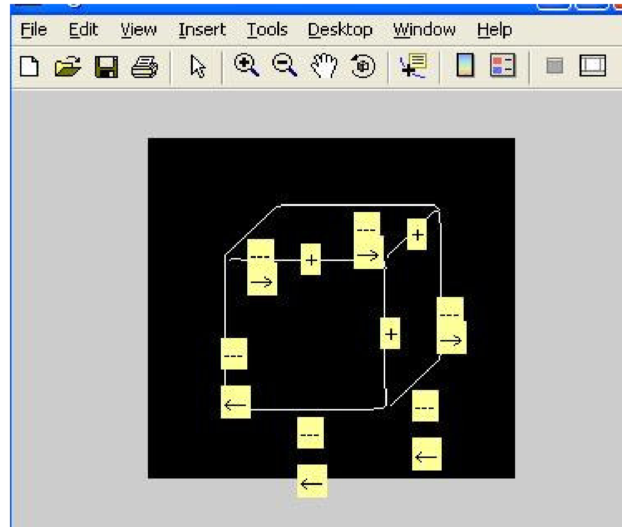
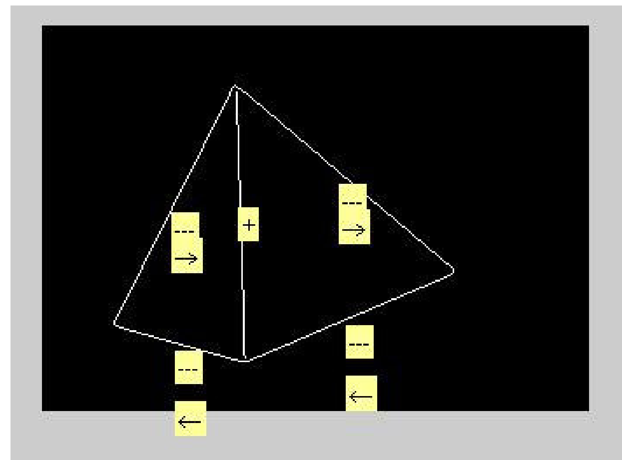


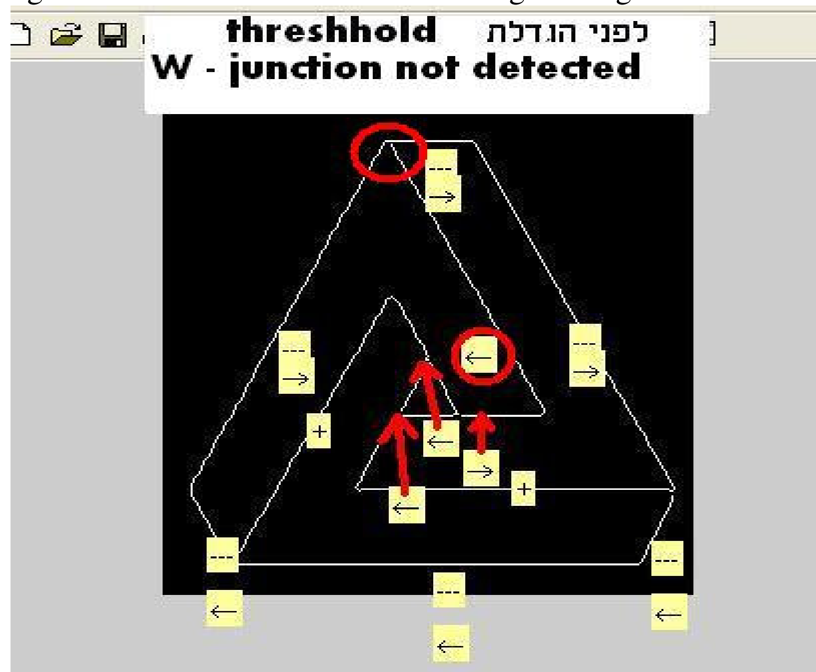
Figure 12: Input object triangle

Outputs:



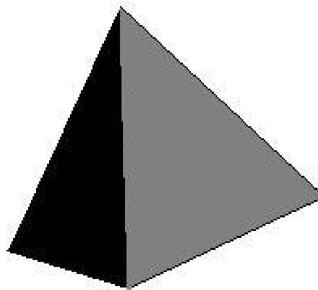


Triangle with the same threshold for finding line segments of the corner



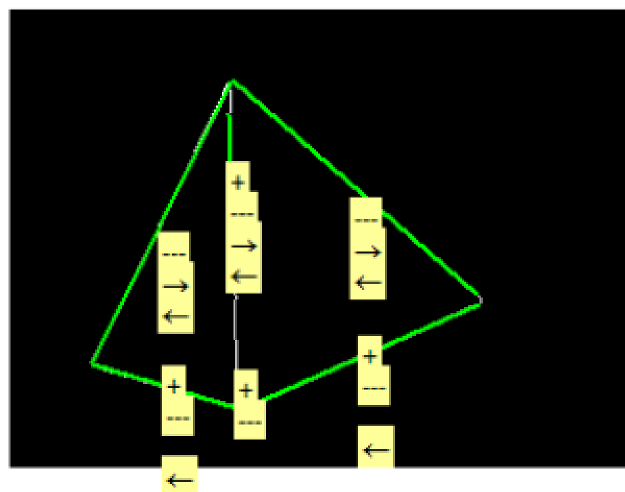
Triangle with bigger threshold for finding line segments of the corner

If the following example demonstrates a bad input image:



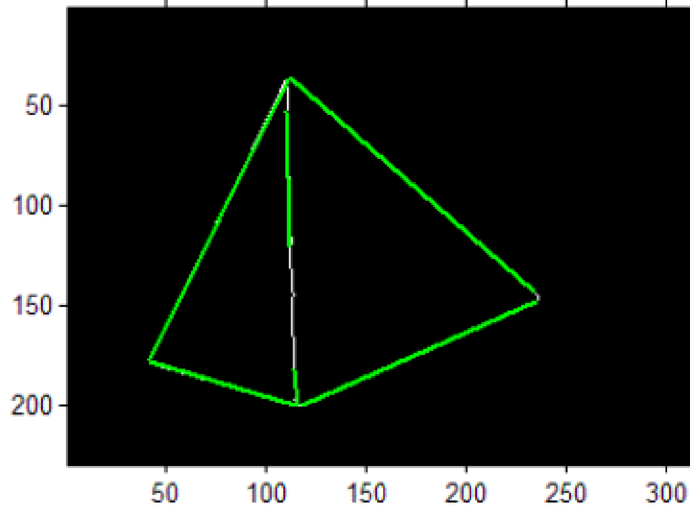
The middle line is not straight enough, this leads to the following output:

Final Result

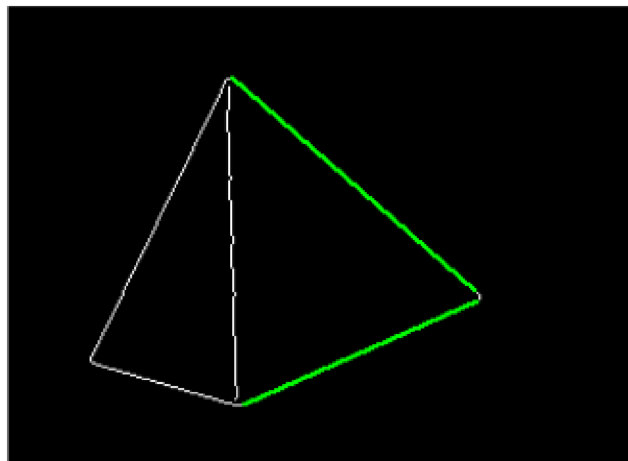


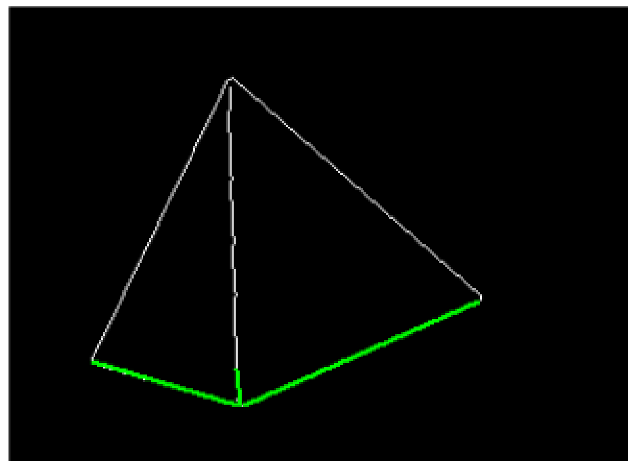
The output is not correct, and the reason is that Hough transform did no find a straight line that is

long enough and lies on the middle line thus two vertexes the highest and the lowest were not recognized as connected. The following picture demonstrates all found lines:

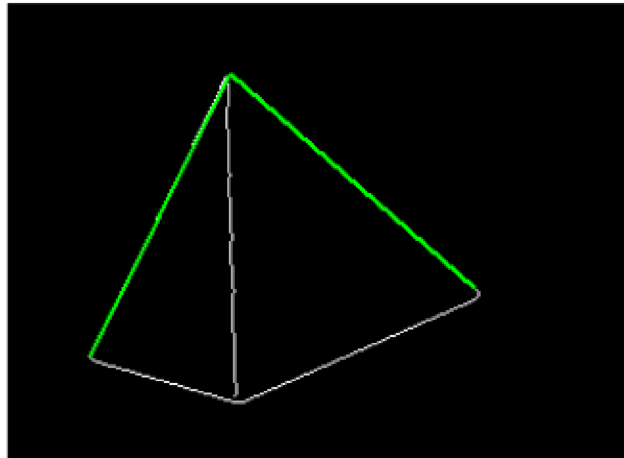


The vertexes were detected correctly (with respect to the found lines):





V - junction



This is not a real W- junction, but according to the detected lines it is.

5 Program Code

Corner detector (we used the existing implementation)

```
function [cout,marked_img]=corner(varargin)


%   CORNER Find corners in intensity image.
%
%       CORNER works by the following step:
%       1.  Apply the Canny edge detector to the gray level image and obtain a
%       binary edge-map.
%       2.  Extract the edge contours from the edge-map, fill the gaps in the
%       contours.
%       3.  Compute curvature at a low scale for each contour to retain all
%       true corners.
%       4.  All of the curvature local maxima are considered as corner
%       candidates, then rounded corners and false corners due to boundary
%       noise and details were eliminated.
%       5.  End points of line mode curve were added as corner, if they are not
%       close to the above detected corners.
%
%       Syntax :
%       [cout,marked_img]=corner(I,C,T_angle,sig,H,L,Endpiont,Gap_size)
%
%       Input :
%       I -  the input image, it could be gray, color or binary image. If I is
%           empty([]), input image can be get from a open file dialog box.
%       C -  denotes the minimum ratio of major axis to minor axis of an ellipse,
%           whose vertex could be detected as a corner by proposed detector.
%           The default value is 1.5.
%       T_angle -  denotes the maximum obtuse angle that a corner can have when
%           it is detected as a true corner, default value is 162.
%       Sig -  denotes the standard deviation of the Gaussian filter when
%           computeing curvature. The default sig is 3.
%       H,L -  high and low threshold of Canny edge detector. The default value
%           is 0.35 and 0.
%       Endpoint -  a flag to control whether add the end points of a curve
%           as corner, 1 means Yes and 0 means No. The default value is 1.
%       Gap_size -  a paremeter use to fill the gaps in the contours, the gap
%           not more than gap_size were filled in this stage. The default
```

```
%            Gap_size is 1 pixels.
%
%        Output :
%        cout -  a position pair list of detected corners in the input image.
%        marked_image -  image with detected corner marked.
%
%        Examples
%        -------
%        I = imread('alumgrns.tif');
%        cout = corner(I,[],[],[],0.2);
%
%        [cout, marked_image] = corner;
%
%        cout = corner([],1.6,155);
%
%
%    Composed by He Xiaochen
%    HKU EEE Dept. ITSR, Apr. 2005
%
%    Algorithm is derived from :
%        X.C. He and N.H.C. Yung, Ў°Curvature Scale Space Corner Detector with
%        Adaptive Threshold and Dynamic Region of SupportЎ±, Proceedings of the
%        17th International Conference on Pattern Recognition, 2:791-794, August
2004.
%    Improved algorithm is included in Ў°A Corner Detector based on Global and Local
%    Curvature PropertiesЎ±and submitted to Pattern Recognition.


[I,C,T_angle,sig,H,L,Endpoint,Gap_size] = parse_inputs(varargin{:});




if size(I,3)==3
    I=rgb2gray(I); % Transform RGB image to a Gray one.
end

tic
 BW=EDGE(I,'canny',[L,H]);  % Detect edges
%BW=EDGE(I,'prewitt',[L,H]);  % Detect edges
time_for_detecting_edge=toc

tic
[curve,curve_start,curve_end,curve_mode,curve_num]=extract_curve(BW,Gap_size);  %
Extract curves
time_for_extracting_curve=toc

%size(curve{2})
curve_num

tic
cout=get_corner(curve,curve_start,curve_end,curve_mode,curve_num,BW,sig,Endpoint,C,
T_angle); % Detect corners
time_for_detecting_corner=toc

img=I;
for i=1:size(cout,1)
    img=mark(img,cout(i,1),cout(i,2),5);
end
```

```
marked_img=img;
figure(2)
imshow(marked_img);
title('Detected corners')
imwrite(marked_img,'corner.jpg');

junctions = JunctionType(cout, BW);


function
[curve,curve_start,curve_end,curve_mode,cur_num]=extract_curve(BW,Gap_size)

%   Function to extract curves from binary edge map, if the endpoint of a
%   contour is nearly connected to another endpoint, fill the gap and continue
%   the extraction. The default gap size is 1 pixles.

[L,W]=size(BW);
BW1=zeros(L+2*Gap_size,W+2*Gap_size);
BW_edge=zeros(L,W);
BW1(Gap_size+1:Gap_size+L,Gap_size+1:Gap_size+W)=BW;
[r,c]=find(BW1==1);
cur_num=0;

while size(r,1)>0
    point=[r(1),c(1)];
    cur=point;
    BW1(point(1),point(2))=0;
    [I,J]=find(BW1(point(1)-Gap_size:point(1)+Gap_size,point(2)-
Gap_size:point(2)+Gap_size)==1);
    while size(I,1)>0
        dist=(I-Gap_size-1).^2+(J-Gap_size-1).^2;
        [min_dist,index]=min(dist);
        point=point+[I(index),J(index)]-Gap_size-1;
        cur=[cur;point];
        BW1(point(1),point(2))=0;
        [I,J]=find(BW1(point(1)-Gap_size:point(1)+Gap_size,point(2)-
Gap_size:point(2)+Gap_size)==1);
    end

    % Extract edge towards another direction
    point=[r(1),c(1)];
    BW1(point(1),point(2))=0;
    [I,J]=find(BW1(point(1)-Gap_size:point(1)+Gap_size,point(2)-
Gap_size:point(2)+Gap_size)==1);
    while size(I,1)>0
        dist=(I-Gap_size-1).^2+(J-Gap_size-1).^2;
        [min_dist,index]=min(dist);
        point=point+[I(index),J(index)]-Gap_size-1;
        cur=[point;cur];
        BW1(point(1),point(2))=0;
        [I,J]=find(BW1(point(1)-Gap_size:point(1)+Gap_size,point(2)-
Gap_size:point(2)+Gap_size)==1);
    end

    if size(cur,1)>(size(BW,1)+size(BW,2))/25
        cur_num=cur_num+1;
        curve{cur_num}=cur-Gap_size;
    end
    [r,c]=find(BW1==1);
```

```
end

for i=1:cur_num
    curve_start(i,:)=curve{i}(1,:);
    curve_end(i,:)=curve{i}(size(curve{i},1),:);
    if (curve_start(i,1)-curve_end(i,1))^2+...
        (curve_start(i,2)-curve_end(i,2))^2<=32
        curve_mode(i,:)='loop';
    else
        curve_mode(i,:)='line';
    end

    BW_edge(curve{i}(:,1)+(curve{i}(:,2)-1)*L)=1;
end
figure(1)
imshow(~BW_edge)
title('Edge map')
imwrite(~BW_edge,'edge.jpg');


function
cout=get_corner(curve,curve_start,curve_end,curve_mode,curve_num,BW,sig,Endpoint,C,
T_angle)

corner_num=0;
cout=[];

GaussianDieOff = .0001;
pw = 1:30;
ssq = sig*sig;
width = max(find(exp(-(pw.*pw)/(2*ssq))>GaussianDieOff));
if isempty(width)
    width = 1;
end
t = (-width:width);
gau = exp(-(t.*t)/(2*ssq))/(2*pi*ssq);
gau=gau/sum(gau);

for i=1:curve_num;
    x=curve{i}(:,1);
    y=curve{i}(:,2);
    W=width;
    L=size(x,1);
    if L>W

        % Calculate curvature
        if curve_mode(i,:)=='loop'
            x1=[x(L-W+1:L);x;x(1:W)];
            y1=[y(L-W+1:L);y;y(1:W)];
        else
            x1=[ones(W,1)*2*x(1)-x(W+1:-1:2);x;ones(W,1)*2*x(L)-x(L-1:-1:L-W)];
            y1=[ones(W,1)*2*y(1)-y(W+1:-1:2);y;ones(W,1)*2*y(L)-y(L-1:-1:L-W)];
        end

        xx=conv(x1,gau);
        xx=xx(W+1:L+3*W);
        yy=conv(y1,gau);
        yy=yy(W+1:L+3*W);
```

```
        Xu=[xx(2)-xx(1) ; (xx(3:L+2*W)-xx(1:L+2*W-2))/2 ; xx(L+2*W)-xx(L+2*W-1)];
        Yu=[yy(2)-yy(1) ; (yy(3:L+2*W)-yy(1:L+2*W-2))/2 ; yy(L+2*W)-yy(L+2*W-1)];
        Xuu=[Xu(2)-Xu(1) ; (Xu(3:L+2*W)-Xu(1:L+2*W-2))/2 ; Xu(L+2*W)-Xu(L+2*W-1)];
        Yuu=[Yu(2)-Yu(1) ; (Yu(3:L+2*W)-Yu(1:L+2*W-2))/2 ; Yu(L+2*W)-Yu(L+2*W-1)];
        K=abs((Xu.*Yuu-Xuu.*Yu)./((Xu.*Xu+Yu.*Yu).^1.5));
        K=ceil(K*100)/100;

        % Find curvature local maxima as corner candidates
        extremum=[];
        N=size(K,1);
        n=0;
        Search=1;

        for j=1:N-1
            if (K(j+1)-K(j))*Search>0
                n=n+1;
                extremum(n)=j;  % In extremum, odd points is minima and even points
is maxima
                Search=-Search;
            end
        end
        if mod(size(extremum,2),2)==0
            n=n+1;
            extremum(n)=N;
        end

        n=size(extremum,2);
        flag=ones(size(extremum));

        % Compare with adaptive local threshold to remove round corners
        for j=2:2:n
            %I=find(K(extremum(j-1):extremum(j+1))==max(K(extremum(j-
1):extremum(j+1))));
            %extremum(j)=extremum(j-1)+round(mean(I))-1; % Regard middle point of
plateaus as maxima

            [x,index1]=min(K(extremum(j):-1:extremum(j-1)));
            [x,index2]=min(K(extremum(j):extremum(j+1)));
            ROS=K(extremum(j)-index1+1:extremum(j)+index2-1);
            K_thre(j)=C*mean(ROS);
            if K(extremum(j))<K_thre(j)
                flag(j)=0;
            end
        end
        extremum=extremum(2:2:n);
        flag=flag(2:2:n);
        extremum=extremum(find(flag==1));

        % Check corner angle to remove false corners due to boundary noise and
trivial details
        flag=0;
        smoothed_curve=[xx,yy];
        while sum(flag==0)>0
            n=size(extremum,2);
            flag=ones(size(extremum));
            for j=1:n
                if j==1 & j==n
                    ang=curve_tangent(smoothed_curve(1:L+2*W,:),extremum(j));
                elseif j==1
```

```
ang=curve_tangent(smoothed_curve(1:extremum(j+1),:),extremum(j));
                elseif j==n
                    ang=curve_tangent(smoothed_curve(extremum(j-
1):L+2*W,:),extremum(j)-extremum(j-1)+1);
                else
                    ang=curve_tangent(smoothed_curve(extremum(j-
1):extremum(j+1),:),extremum(j)-extremum(j-1)+1);
                end
                if ang>T_angle & ang<(360-T_angle)
                    flag(j)=0;
                end
            end

            if size(extremum,2)==0
                extremum=[];
            else
                extremum=extremum(find(flag~=0));
            end
        end

        extremum=extremum-W;
        extremum=extremum(find(extremum>0 & extremum<=L));
        n=size(extremum,2);
        for j=1:n
            corner_num=corner_num+1;
            cout(corner_num,:)=curve{i}(extremum(j),:);
        end
    end
end


% Add Endpoints
if Endpoint
    for i=1:curve_num
        if size(curve{i},1)>0 & curve_mode(i,:)=='line'

            % Start point compare with detected corners
            compare_corner=cout-ones(size(cout,1),1)*curve_start(i,:);
            compare_corner=compare_corner.^2;
            compare_corner=compare_corner(:,1)+compare_corner(:,2);
            if min(compare_corner)>25       % Add end points far from detected
corners
                corner_num=corner_num+1;
                cout(corner_num,:)=curve_start(i,:);
            end

            % End point compare with detected corners
            compare_corner=cout-ones(size(cout,1),1)*curve_end(i,:);
            compare_corner=compare_corner.^2;
            compare_corner=compare_corner(:,1)+compare_corner(:,2);
            if min(compare_corner)>25
                corner_num=corner_num+1;
                cout(corner_num,:)=curve_end(i,:);
            end
        end
    end
end
```

```
function ang=curve_tangent(cur,center)

for i=1:2
    if i==1
        curve=cur(center:-1:1,:);
    else
        curve=cur(center:size(cur,1),:);
    end
    L=size(curve,1);

    if L>3
        if sum(curve(1,:)~=curve(L,:))~=0
            M=ceil(L/2);
            x1=curve(1,1);
            y1=curve(1,2);
            x2=curve(M,1);
            y2=curve(M,2);
            x3=curve(L,1);
            y3=curve(L,2);
        else
            M1=ceil(L/3);
            M2=ceil(2*L/3);
            x1=curve(1,1);
            y1=curve(1,2);
            x2=curve(M1,1);
            y2=curve(M1,2);
            x3=curve(M2,1);
            y3=curve(M2,2);
        end

        if abs((x1-x2)*(y1-y3)-(x1-x3)*(y1-y2))<1e-8  % straight line
            tangent_direction=angle(complex(curve(L,1)-curve(1,1),curve(L,2)-
curve(1,2)));
        else
            % Fit a circle
            x0 = 1/2*(-y1*x2^2+y3*x2^2-y3*y1^2-y3*x1^2-y2*y3^2+x3^2*y1+y2*y1^2-
y2*x3^2-y2^2*y1+y2*x1^2+y3^2*y1+y2^2*y3)/(-y1*x2+y1*x3+y3*x2+x1*y2-x1*y3-x3*y2);
            y0 = -1/2*(x1^2*x2-x1^2*x3+y1^2*x2-y1^2*x3+x1*x3^2-x1*x2^2-x3^2*x2-
y3^2*x2+x3*y2^2+x1*y3^2-x1*y2^2+x3*x2^2)/(-y1*x2+y1*x3+y3*x2+x1*y2-x1*y3-x3*y2);
            % R = (x0-x1)^2+(y0-y1)^2;

            radius_direction=angle(complex(x0-x1,y0-y1));
            adjacent_direction=angle(complex(x2-x1,y2-y1));
            tangent_direction=sign(sin(adjacent_direction-
radius_direction))*pi/2+radius_direction;
        end

    else % very short line
        tangent_direction=angle(complex(curve(L,1)-curve(1,1),curve(L,2)-
curve(1,2)));
    end
    direction(i)=tangent_direction*180/pi;
end
ang=abs(direction(1)-direction(2));



function img1=mark(img,x,y,w)
```

```
[M,N,C]=size(img);
img1=img;

if isa(img,'logical')
    img1(max(1,x-floor(w/2)):min(M,x+floor(w/2)),max(1,y-
floor(w/2)):min(N,y+floor(w/2)),:)=...
        (img1(max(1,x-floor(w/2)):min(M,x+floor(w/2)),max(1,y-
floor(w/2)):min(N,y+floor(w/2)),:)<1);
    img1(x-floor(w/2)+1:x+floor(w/2)-1,y-floor(w/2)+1:y+floor(w/2)-1,:)=...
        img(x-floor(w/2)+1:x+floor(w/2)-1,y-floor(w/2)+1:y+floor(w/2)-1,:);
else
    img1(max(1,x-floor(w/2)):min(M,x+floor(w/2)),max(1,y-
floor(w/2)):min(N,y+floor(w/2)),:)=...
        (img1(max(1,x-floor(w/2)):min(M,x+floor(w/2)),max(1,y-
floor(w/2)):min(N,y+floor(w/2)),:)<128)*255;
    img1(x-floor(w/2)+1:x+floor(w/2)-1,y-floor(w/2)+1:y+floor(w/2)-1,:)=...
        img(x-floor(w/2)+1:x+floor(w/2)-1,y-floor(w/2)+1:y+floor(w/2)-1,:);
end


function [I,C,T_angle,sig,H,L,Endpoint,Gap_size] = parse_inputs(varargin);

error(nargchk(0,8,nargin));

Para=[1.5,162,3,0.35,0,1,1]; %Default experience value;

if nargin>=2
    I=varargin{1};
    for i=2:nargin
        if size(varargin{i},1)>0
            Para(i-1)=varargin{i};
        end
    end
end

if nargin==1
    I=varargin{1};
end

if nargin==0 | size(I,1)==0
    [fname,dire]=uigetfile('*.bmp;*.jpg;*.gif','Open the image to be detected');
    I=imread([dire,fname]);
end

C=Para(1);
T_angle=Para(2);
sig=Para(3);
H=Para(4);
L=Para(5);
Endpoint=Para(6);
Gap_size=Para(7);
```

Our Code:

Main method:

```
function [res]=JunctionType(corners, Image)
% define window size for line offset from corner
thres = 15;
% Find lines

[H,T,R] = hough(Image);
corners_size = size(corners);
P  = houghpeaks(H,30,'threshold',ceil(0.1*max(H(:))));
tmp_lines = houghlines(Image,T,R,P,'FillGap',10,'MinLength',15);
% lines = houghlines(Image,T,R,P,'FillGap',30,'MinLength',25);

% figure(3);
% imshow(Image), hold on
% axis on, axis normal, hold on;
% for i = 1:length(tmp_lines)
% xy = [tmp_lines(i).point1;tmp_lines(i).point2];
% plot(xy(:,1),xy(:,2),'LineWidth',2,'Color','green');
% end
%
% %% Printing detected lines one by one
% for i = 1:length(tmp_lines)
%     figure(i + 10);
% imshow(Image), hold on
% axis on, axis normal, hold on;
% %for i = 4:4
% %  xy = [lines(1).point1;lines(1).point2];
% % plot(xy(:,1),xy(:,2),'LineWidth',2,'Color','green');
%
% xy = [tmp_lines(i).point1;tmp_lines(i).point2];
% plot(xy(:,1),xy(:,2),'LineWidth',2,'Color','green');
%
% end


lines = remove_redundantOld(tmp_lines, Image);




%% Printing detected lines
figure(3);
title('Lines detected by Hough after removing redundancies');
imshow(Image), hold on
axis on, axis normal, hold on;
for i = 1:length(lines)
xy = [lines(i).point1;lines(i).point2];
plot(xy(:,1),xy(:,2),'LineWidth',2,'Color','green');
end

% %% Printing detected lines one by one
% for i = 1:length(lines)
%     figure(i + 10);
% imshow(Image), hold on
% axis on, axis normal, hold on;
% %for i = 4:4
% %  xy = [lines(1).point1;lines(1).point2];
```

```
% % plot(xy(:,1),xy(:,2),'LineWidth',2,'Color','green');
%
% xy = [lines(i).point1;lines(i).point2];
% plot(xy(:,1),xy(:,2),'LineWidth',2,'Color','green');
%
% end


% Find lines, that lies on corners
xlines = struct('labels', {}, 'corner1', {}, 'corner2', {}, 'cornert', {});
for i = 1:length(lines)
    xlines{i}.corner1 = 0;
    xlines{i}.corner2 = 0;
    xlines{i}.cornert = 0;
    xlines{i}.labels = [1:4];
end
junctions = struct('corner_x', {}, 'corner_y', {},  'ilines' ,{}, 'type', {});
% junction type:
%                                        1 - V junction
%                                        2 - W junction
%                                        3 - Y junction
%                                        4 - T junction
for l = 1:corners_size(1)
    % line1 -  picture line index
    % line2 - picture line connection to corner type: 1 - point1, 2 -
    % point2, 3 - T - connection
    % line 3 - picture line angle (from -pi to pi)
    junctions{l}.ilines = zeros(3,1);
    junctions{l}.type = 0;
    junctions{l}.corner_x = corners(l, 2);
    junctions{l}.corner_y = corners(l, 1);
    line = 1;
    corner_num = l;
    for k = 1:length(lines)
        xy = [lines(k).point1; lines(k).point2];
        found = 1;
        dist = sqrt(((junctions{l}.corner_x)-xy(1,1))^2 + (junctions{l}.corner_y -
xy(1,2))^2 );
        if(thres >= dist)
                                 %junctions{l}.ilines(:,line) = [k;1; tan2((xy(2,1) -
xy(1,1)), (xy(2,2) - xy(1,2)))];
                                 junctions{l}.ilines(:,line) = [k;1; 0];
                                 line = line + 1;
                                 found= 0;
                                 xlines = add_xline(l, k, xlines);
          end
          dist = sqrt(((junctions{l}.corner_x)-xy(2,1))^2 + (junctions{l}.corner_y
- xy(2,2))^2 );
          if(thres >= dist && found ==1)
                                %junctions{l}.ilines(:, line) = [k;2;tan2((xy(1,1) -
xy(2,1)), (xy(1,2) - xy(2,2)))];
                                junctions{l}.ilines(:, line) = [k;2;0];
                                line = line + 1;
                                found = 0;
                                xlines = add_xline(l, k, xlines);
           end
        if (found ==1)
            det_matr = [1 xy(1,1) xy(1,2);1 xy(2,1)  xy(2,2); 1
junctions{l}.corner_x  junctions{l}.corner_y];
            dist =abs(det(det_matr));
```

```
            if(dist <= 200 & opposite(xy(1,1), xy(1,2), xy(2,1), xy(2,2),
junctions{l}.corner_x, junctions{l}.corner_y) == 1)
                junctions{l}.ilines(:,line) = [k;3;0];
                line = line + 1;
                found = 0;
                xlines = add_txline(l, k, xlines);
            end
        end
   end
end

junctions = sort_and_type_junctions_lines(junctions, lines);

%% Print first selected junction with lines and type
% junction type:
%                                       1 - V junction
%                                       2 - W junction
%                                       3 - Y junction
%                                       4 - T junction

for j = 1:length(junctions)
    figure;
    imshow(Image); hold on;
    mylines = junctions{j}.ilines;
    type = junctions{j}.type;
    if (type == 1)
        title('V - junction');
    end
    if (type == 2)
        title('W - junction');
    end
    if (type == 3)
        title('Y - junction');
    end
    if (type == 4)
        title('T - junction');
    end
    [stam, ilength] = size(mylines);
    for i = 1:ilength
        xy = [lines(mylines(1, i)).point1; lines(mylines(1, i)).point2];
        plot(xy(:,1),xy(:,2),'LineWidth',2,'Color','green');
    end
end


%% Set line labels
xlines = set_labels(junctions, lines, xlines, Image);
%% Print final result
figure;
imshow(Image); hold on;
title('Final Result');
for i= 1:length(lines)
    ilabels = xlines{i}.labels;
    xy = [lines(i).point1; lines(i).point2];
    plot(xy(:,1),xy(:,2),'LineWidth',2,'Color','green');
    if(ilabels(1) == 1)
        text(0.5*(xy(1,1) + xy(2,1)), 0.5*(xy(1,2) + xy(2,2)) , '+'
,'BackgroundColor',[1 1 .6]);
    end
    if(ilabels(2) == 2)
```

```
        text(0.5*(xy(1,1) + xy(2,1)) , 0.5*(xy(1,2) + xy(2,2)) + 15, '---'
,'BackgroundColor',[1 1 .6]);
    end
    if(ilabels(3) == 3)
        text(0.5*(xy(1,1) + xy(2,1)) , 0.5*(xy(1,2) + xy(2,2)) + 30, '\rightarrow'
,'BackgroundColor',[1 1 .6]);
    end
    if(ilabels(4) == 4)
        text(0.5*(xy(1,1) + xy(2,1)) , 0.5*(xy(1,2) + xy(2,2)) + 45, '\leftarrow'
,'BackgroundColor',[1 1 .6]);
    end
end
res = 0;
end

function[ans] = opposite(point1x, point1y, point2x, point2y, testx, testy)
ans = 0;
a = sqrt((point2x - point1x)^2 + (point2y - point1y)^2);
b = sqrt((point2x - testx)^2 + (point2y - testy)^2);
c = sqrt((testx - point1x)^2 + (testy - point1y)^2);
cosb = (a^2 + c^2 - b^2) / (2 * a * c);
cosc = (a^2 + b^2 - c^2) / (2 * a * b);
if (cosb > 0 & cosc > 0)
    ans = 1;
end
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% xlines = struct('labels', {}, 'corner1', {}, 'corner2', {}, 'cornert', {});
% junctions = struct('corner_x', {}, 'corner_y', {},  'ilines' ,{}, 'type', {});
% Labels:
%           1 = +
%           2 = -
%           3 = ->
%           4 = <-
% junction type:
%                                       1 - V junction
%                                       2 - W junction
%                                       3 - Y junction
%                                       4 - T junction
function[xlines] = set_labels(junctions, lines, xlines, Image)
changed = 1;
while(changed > 0)
    changed = 0;
    for i = 1:length(junctions)
        if ( junctions{i}.type == 1)
            [tmp, xlines] = deal_v_junction(junctions{i}, xlines);
            changed = changed + tmp;
%            plot_figure(Image, lines, xlines);
        end
        if ( junctions{i}.type == 2)
            [tmp, xlines] = deal_w_junction(junctions{i}, xlines);
            changed = changed + tmp;
%            plot_figure(Image, lines, xlines);
        end
        if ( junctions{i}.type == 3)
           [tmp, xlines] = deal_y_junction(junctions{i}, xlines);
           changed = changed + tmp;
%           plot_figure(Image, lines, xlines);
        end
```

```
        if ( junctions{i}.type == 4)
          [tmp, xlines] = deal_t_junction(junctions{i}, xlines);
          changed = changed + tmp;
%           plot_figure(Image, lines, xlines);
        end
    end % for i = 1:length(junctions)
end % while(changed == 1)
end

function[stam] = plot_figure(Image, lines, xlines)
figure;
imshow(Image); hold on;
title('Current Iteration');

for i= 1:length(lines)
    ilabels = xlines{i}.labels;
    xy = [lines(i).point1; lines(i).point2];
    if(ilabels(1) == 1)
        text(0.5*(xy(1,1) + xy(2,1)), 0.5*(xy(1,2) + xy(2,2)) , '+'
,'BackgroundColor',[1 1 .6]);
    end
    if(ilabels(2) == 2)
        text(0.5*(xy(1,1) + xy(2,1)) , 0.5*(xy(1,2) + xy(2,2)) + 15, '---'
,'BackgroundColor',[1 1 .6]);
    end
    if(ilabels(3) == 3)
        text(0.5*(xy(1,1) + xy(2,1)) , 0.5*(xy(1,2) + xy(2,2)) + 30, '\rightarrow'
,'BackgroundColor',[1 1 .6]);
    end
    if(ilabels(4) == 4)
        text(0.5*(xy(1,1) + xy(2,1)) , 0.5*(xy(1,2) + xy(2,2)) + 45, '\leftarrow'
,'BackgroundColor',[1 1 .6]);
    end
end
end



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% xlines = struct('labels', {}, 'corner1', {}, 'corner2', {}, 'cornert', {});
% junctions = struct('corner_x', {}, 'corner_y', {},  'ilines' ,{}, 'type', {});
% Labels:
%           1 = +
%           2 = -
%           3 = ->
%           4 = <-
function[changed, xlines] = deal_t_junction(junction, xlines)
lines = junction.ilines(1,:);
if (junction.ilines(2,1) == 1)
    in1 = 4;
    out1 = 3;
else
    in1 = 3;
    out1 = 4;
end

changed = 0;

% Foot of T line (lines 1)
```

```
if ( xlines{lines(1)}.labels(in1) ~= 0 | xlines{lines(1)}.labels(1) ~= 0 |
xlines{lines(1)}.labels(2) ~= 0 )
    changed = 1;
    xlines{lines(1)}.labels(1) = 0;
    xlines{lines(1)}.labels(2) = 0;
    xlines{lines(1)}.labels(in1) = 0;
end
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% xliness = struct('labels', {}, 'corner1', {}, 'corner2', {}, 'cornert', {});
% junctions = struct('corner_x', {}, 'corner_y', {},  'ilines' ,{}, 'type', {});
% Labels:
%           1 = +
%           2 = -
%           3 = ->
%           4 = <-
function[changed, xlines] = deal_y_junction(junction, xlines)
lines = junction.ilines(1,:);
if (junction.ilines(2,1) == 1)
    out1 = 3;
    in1 = 4;
else
    out1 = 4;
    in1 = 3;
end
if (junction.ilines(2,2) == 1)
    in2 = 4;
    out2 = 3;
else
    in2 = 3;
    out2 = 4;
end
if (junction.ilines(2,3) == 1)
    in3 = 4;
    out3 = 3;
else
    in3 = 3;
    out3 = 4;
end

changed = 0;

if (( xlines{lines(1)}.labels(1) == 0 & (xlines{lines(2)}.labels(1) ~= 0 |
xlines{lines(3)}.labels(1) ~= 0 )) | ( xlines{lines(2)}.labels(1) == 0 &
(xlines{lines(1)}.labels(1) ~= 0 | xlines{lines(3)}.labels(1) ~= 0 )) | (
xlines{lines(3)}.labels(1) == 0 & (xlines{lines(2)}.labels(1) ~= 0 |
xlines{lines(1)}.labels(1) ~= 0 )))
    changed = 1;
    xlines{lines(2)}.labels(1) = 0;
    xlines{lines(3)}.labels(1) = 0;
    xlines{lines(1)}.labels(1) = 0;
end

if ( xlines{lines(1)}.labels(in1) == 0 & xlines{lines(2)}.labels(out2) ~= 0 )
    changed = 1;
    xlines{lines(2)}.labels(out2) = 0;
end
if ( xlines{lines(2)}.labels(in2) == 0 & xlines{lines(3)}.labels(out3) ~= 0 )
```

```
        changed = 1;
        xlines{lines(3)}.labels(out3) = 0;
    end
    if ( xlines{lines(3)}.labels(in3) == 0 & xlines{lines(1)}.labels(out1) ~= 0 )
        changed = 1;
        xlines{lines(1)}.labels(out1) = 0;
    end

    if ( xlines{lines(2)}.labels(out2) == 0 & xlines{lines(1)}.labels(in1) ~= 0 )
        changed = 1;
        xlines{lines(1)}.labels(in1) = 0;
    end
    if ( xlines{lines(3)}.labels(out3) == 0 & xlines{lines(2)}.labels(in2) ~= 0 )
        changed = 1;
        xlines{lines(2)}.labels(in2) = 0;
    end
    if ( xlines{lines(1)}.labels(out1) == 0 & xlines{lines(3)}.labels(in3) ~= 0 )
        changed = 1;
        xlines{lines(3)}.labels(in3) = 0;
    end

    if (xlines{lines(1)}.labels(2) == 0 & (xlines{lines(2)}.labels(in2) ~= 0 |
    xlines{lines(3)}.labels(out3) ~= 0 ))
        changed = 1;
        xlines{lines(2)}.labels(in2) = 0 ;
        xlines{lines(3)}.labels(out3) = 0;
    end
    if (xlines{lines(2)}.labels(2) == 0 & (xlines{lines(3)}.labels(in3) ~= 0 |
    xlines{lines(1)}.labels(out1) ~= 0 ))
        changed = 1;
        xlines{lines(3)}.labels(in3) = 0 ;
        xlines{lines(1)}.labels(out1) = 0;
    end
    if (xlines{lines(3)}.labels(2) == 0 & (xlines{lines(1)}.labels(in2) ~= 0 |
    xlines{lines(2)}.labels(out2) ~= 0 ))
        changed = 1;
        xlines{lines(1)}.labels(in1) = 0 ;
        xlines{lines(2)}.labels(out2) = 0 ;
    end

    if (xlines{lines(1)}.labels(2) == 0 & xlines{lines(1)}.labels(in1) == 0 &
    xlines{lines(1)}.labels(out1) == 0 & (xlines{lines(2)}.labels(2) ~= 0 |
    xlines{lines(2)}.labels(in2) ~= 0 | xlines{lines(2)}.labels(out2) ~= 0 |
    xlines{lines(3)}.labels(2) ~= 0 | xlines{lines(3)}.labels(in3) ~= 0 |
    xlines{lines(3)}.labels(out3) ~= 0 ))
        changed = 1;
        xlines{lines(2)}.labels(2) = 0;
        xlines{lines(2)}.labels(in2) = 0;
        xlines{lines(2)}.labels(out2) = 0;
        xlines{lines(3)}.labels(3) = 0;
        xlines{lines(3)}.labels(in3) = 0;
        xlines{lines(3)}.labels(out3) = 0;
    end
    if (xlines{lines(2)}.labels(2) == 0 & xlines{lines(2)}.labels(in2) == 0 &
    xlines{lines(2)}.labels(out2) == 0 & (xlines{lines(1)}.labels(2) ~= 0 |
    xlines{lines(1)}.labels(in1) ~= 0 | xlines{lines(1)}.labels(out1) ~= 0 |
    xlines{lines(3)}.labels(2) ~= 0 | xlines{lines(3)}.labels(in3) ~= 0 |
    xlines{lines(3)}.labels(out3) ~= 0 ))
        changed = 1;
        xlines{lines(1)}.labels(2) = 0;
        xlines{lines(1)}.labels(in1) = 0;
```

```
    xlines{lines(1)}.labels(out1) = 0;
    xlines{lines(3)}.labels(2) = 0;
    xlines{lines(3)}.labels(in3) = 0;
    xlines{lines(3)}.labels(out3) = 0;
end
if (xlines{lines(3)}.labels(2) == 0 & xlines{lines(3)}.labels(in3) == 0 &
xlines{lines(3)}.labels(out3) == 0 & (xlines{lines(2)}.labels(2) ~= 0 |
xlines{lines(2)}.labels(in2) ~= 0 | xlines{lines(2)}.labels(out2) ~= 0 |
xlines{lines(1)}.labels(2) ~= 0 | xlines{lines(1)}.labels(in1) ~= 0 |
xlines{lines(1)}.labels(out1) ~= 0 ))
    changed = 1;
    xlines{lines(2)}.labels(2) = 0;
    xlines{lines(2)}.labels(in2) = 0;
    xlines{lines(2)}.labels(out2) = 0;
    xlines{lines(1)}.labels(3) = 0;
    xlines{lines(1)}.labels(in1) = 0;
    xlines{lines(1)}.labels(out1) = 0;
end



end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% xlines = struct('labels', {}, 'corner1', {}, 'corner2', {}, 'cornert', {});
% junctions = struct('corner_x', {}, 'corner_y', {},  'ilines' ,{}, 'type', {});
% Labels:
%            1 = +
%            2 = -
%            3 = p1 -> p2
%            4 = p2 -> p1
function[changed, xlines] = deal_v_junction(junction, xlines)
lines = junction.ilines(1,:);
if (junction.ilines(2,1) == 1)
    out1 = 3;
    in1 = 4;
else
    out1 = 4;
    in1 = 3;
end
if (junction.ilines(2,2) == 1)
    in2 = 4;
    out2 = 3;
else
    in2 = 3;
    out2 = 4;
end

changed = 0;
% Left line
if ( xlines{lines(1)}.labels(in1) == 0 & xlines{lines(2)}.labels(1) ~= 0 )
    changed = 1;
    xlines{lines(2)}.labels(1) = 0;
end
if ( xlines{lines(1)}.labels(in1) == 0 & xlines{lines(1)}.labels(1) == 0 &
xlines{lines(2)}.labels(out2) ~= 0 )
    changed = 1;
    xlines{lines(2)}.labels(out2) = 0;
end
```

```
if ( xlines{lines(1)}.labels(out1) == 0 & xlines{lines(2)}.labels(2) ~= 0 )
    changed = 1;
    xlines{lines(2)}.labels(2) = 0;
end
if ( xlines{lines(1)}.labels(out1) == 0 & xlines{lines(1)}.labels(2) == 0 &
xlines{lines(2)}.labels(in2) ~= 0 )
    changed = 1;
    xlines{lines(2)}.labels(in2) = 0;
end

% Right line
if ( xlines{lines(2)}.labels(out2) == 0 & xlines{lines(1)}.labels(1) ~= 0 )
    changed = 1;
    xlines{lines(1)}.labels(1) = 0;
end
if ( xlines{lines(2)}.labels(out2) == 0 & xlines{lines(2)}.labels(1) == 0 &
xlines{lines(1)}.labels(in1) ~= 0 )
    changed = 1;
    xlines{lines(1)}.labels(in1) = 0;
end
if ( xlines{lines(2)}.labels(in2) == 0 & xlines{lines(1)}.labels(2) ~= 0 )
    changed = 1;
    xlines{lines(1)}.labels(2) = 0;
end
if ( xlines{lines(2)}.labels(in2) == 0 & xlines{lines(2)}.labels(2) == 0 &
xlines{lines(1)}.labels(out1) ~= 0 )
    changed = 1;
    xlines{lines(1)}.labels(out1) = 0;
end
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% xlines = struct('labels', {}, 'corner1', {}, 'corner2', {}, 'cornert', {});
% junctions = struct('corner_x', {}, 'corner_y', {},  'ilines' ,{}, 'type', {});
% Labels:
%           1 = +
%           2 = -
%           3 = ->
%           4 = <-
function[changed, xlines] = deal_w_junction(junction, xlines)
lines = junction.ilines(1,:);
if (junction.ilines(2,1) == 1)
    in1 = 4;
    out1 = 3;
else
    in1 = 3;
    out1 = 4;
end
if (junction.ilines(2,2) == 1)
    in2 = 4;
    out2 = 3;
else
    in2 = 3;
    out2 = 4;
end
if (junction.ilines(2,3) == 1)
    in3 = 4;
    out3 = 3;
else
    in3 = 3;
```

```
    out3 = 4;
end

changed = 0;

if ( xlines{lines(1)}.labels(in1) ~= 0)
    xlines{lines(1)}.labels(in1) = 0;
end
if ( xlines{lines(3)}.labels(out3) ~= 0)
    xlines{lines(3)}.labels(out3) = 0;
end

if ( xlines{lines(2)}.labels(in2) ~= 0 | xlines{lines(2)}.labels(out2) ~= 0 )
    changed = 1;
    xlines{lines(2)}.labels(in2) = 0;
    xlines{lines(2)}.labels(out2) = 0;
end % if ( labels(3) ~= 0)

% First line
if ( xlines{lines(1)}.labels(1) == 0 & (xlines{lines(2)}.labels(2) ~= 0 |
xlines{lines(3)}.labels(1) ~= 0) )
    changed = 1;
    xlines{lines(2)}.labels(2) = 0;
    xlines{lines(3)}.labels(1) = 0;
end % if ( xlines{lines(1)}.labels(1) == 0 )
if ( xlines{lines(1)}.labels(2) == 0 & xlines{lines(3)}.labels(2) ~= 0 )
    changed = 1;
    xlines{lines(3)}.labels(2) = 0;
end % if ( xlines{lines(1)}.labels(1) == 0 )
if ( xlines{lines(1)}.labels(out1) == 0 & xlines{lines(3)}.labels(in3) ~= 0)
    changed = 1;
    xlines{lines(3)}.labels(in3) = 0;
end % if ( xlines{lines(1)}.labels(1) == 0 )
if ( xlines{lines(1)}.labels(2) == 0 & xlines{lines(1)}.labels(out1) == 0 &
xlines{lines(2)}.labels(1) ~= 0)
    changed = 1;
    xlines{lines(2)}.labels(1) = 0;
end % if ( xlines{lines(1)}.labels(1) == 0 )
% Middle line
if ( xlines{lines(2)}.labels(1) == 0 & ( xlines{lines(1)}.labels(2) ~= 0 |
xlines{lines(1)}.labels(out1) ~= 0 | xlines{lines(3)}.labels(2) ~= 0 |
xlines{lines(3)}.labels(in3) ~= 0))
    changed = 1;
    xlines{lines(1)}.labels(2) = 0;
    xlines{lines(1)}.labels(out1) = 0;
    xlines{lines(3)}.labels(2) = 0;
    xlines{lines(3)}.labels(in3) = 0;
end
if ( xlines{lines(2)}.labels(2) == 0 & ( xlines{lines(1)}.labels(1) ~= 0 |
xlines{lines(3)}.labels(1) ~= 0 ))
    changed = 1;
    xlines{lines(1)}.labels(1) = 0;
    xlines{lines(3)}.labels(1) = 0;
end
% Third line
if ( xlines{lines(3)}.labels(1) == 0 & (xlines{lines(2)}.labels(2) ~= 0 |
xlines{lines(1)}.labels(1) ~= 0) )
    changed = 1;
    xlines{lines(2)}.labels(2) = 0;
    xlines{lines(1)}.labels(1) = 0;
```

```
end % if ( xlines{lines(1)}.labels(1) == 0 )
if ( xlines{lines(3)}.labels(2) == 0 & xlines{lines(1)}.labels(2) ~= 0 )
    changed = 1;
    xlines{lines(1)}.labels(2) = 0;
end % if ( xlines{lines(1)}.labels(1) == 0 )
if ( xlines{lines(3)}.labels(in3) == 0 & xlines{lines(1)}.labels(out1) ~= 0)
    changed = 1;
    xlines{lines(1)}.labels(out1) = 0;
end % if ( xlines{lines(1)}.labels(1) == 0 )
if ( xlines{lines(3)}.labels(2) == 0 & xlines{lines(3)}.labels(in3) == 0 &
xlines{lines(2)}.labels(1) ~= 0)
    changed = 1;
    xlines{lines(2)}.labels(1) = 0;
end % if ( xline{lines(1)}.labels(1) == 0 )

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% xlines = struct('xline', lines, 'labels' {}, 'corner1', {}, 'corner2', {},
cornert, {});
function[xlines] = add_xline(junction_index, line_index, xlines)
if(xlines{line_index}.corner1 == 0)
    xlines{line_index}.corner2 = junction_index;
else
    xlines{line_index}.corner1 = junction_index;
    xlines{line_index}.labels = zeros(1,4);
end
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% xlines = struct('xline', lines, 'labels' {}, 'corner1', {}, 'corner2', {},
cornert, {});
function[xlines] = add_txline(junction_index, line_index, xlines)
xlines{line_index}.cornert = junction_index;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% junction type:
%                                       1 - V junction
%                                       2 - W junction
%                                       3 - Y junction
%                                       4 - T junction
% junctions = struct('corner_x', {}, 'corner_y', {},  'ilines' ,{}, 'type', {});
function[junctions] = sort_and_type_junctions_lines(junctions, lines)
for l = 1:length(junctions)
    junctions{l} = calculate_line_angles(junctions{l}, lines);
    junctions{l} = sort_lines_and_set_types(junctions{l});
end % for l = 1:length(junctions);
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function[junction] = sort_lines_and_set_types(junction)
[stam,lines_number] = size(junction.ilines);
    for i = 1:lines_number
        for j = 1:lines_number-1
            if (junction.ilines(3, j) < junction.ilines(3, j+1))
                collumn = junction.ilines(:, j);
                junction.ilines(:, j) = junction.ilines(:, j+1);
                junction.ilines(:, j+1) = collumn;
```

```
            end % if (junction.ilines(3, j) < junction.ilines(3, j+1))
        end % for j = i:lines_number-1
    end % for i = 1:lines_number
    if (lines_number == 2)
        junction = set_2_lines_corner_type(junction);
    else
        if(lines_number == 3)
            junction = set_3_lines_corner_type(junction);
        end
    end % if
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function[junction] = set_3_lines_corner_type(junction)
if (junction.ilines(3,1) - junction.ilines(3,3) > pi & junction.ilines(3,1) -
junction.ilines(3,2) < pi & junction.ilines(3,2) - junction.ilines(3,3) < pi)
    junction.type = 3;
else
    junction.type = 2;
    if(junction.ilines(3,1) - junction.ilines(3,3) > pi & junction.ilines(3,1) -
junction.ilines(3,2) < pi)
        column = junction.ilines(:, 3);
        junction.ilines(:, 3) = junction.ilines(:, 2);
        junction.ilines(:, 2) = junction.ilines(:, 1);
        junction.ilines(:, 1) = column;
    else
        if (junction.ilines(3,1) - junction.ilines(3,3) > pi & junction.ilines(3,2)
- junction.ilines(3,3) < pi)
            column = junction.ilines(:, 1);
            junction.ilines(:, 1) = junction.ilines(:, 2);
            junction.ilines(:, 2) = junction.ilines(:, 3);
            junction.ilines(:, 3) = column;
        end
    end % if(junction.ilines(3,1) - junction.ilines(3,3) > pi &
junction.ilines(3,1) - junction.ilines(3,2) < pi)
end % if (junction.ilines(3,1) - junction.ilines(3,3) > pi)
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function[junction] = set_2_lines_corner_type(junction)
if ( junction.ilines(2,1) == 3 | junction.ilines(2,2) == 3)
    junction.type = 4;
    if ( junction.ilines(2,1) == 3)
        column = junction.ilines(:, 1);
        junction.ilines(:, 1) = junction.ilines(:, 2);
        junction.ilines(:, 2) = column;
    end
else % V - junction
    junction.type = 1;
    if (junction.ilines(3,1) - junction.ilines(3,2) > pi)
        column = junction.ilines(:, 1);
        junction.ilines(:, 1) = junction.ilines(:, 2);
        junction.ilines(:, 2) = column;
    end
end
end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function[junction] = calculate_line_angles(junction, lines)
[stam,lines_number] = size(junction.ilines);
for i = 1:lines_number
    if ( junction.ilines(2,i) == 1) % line i connected with the first point to
junction.
        xy = [lines(junction.ilines(1,i)).point1;
lines(junction.ilines(1,i)).point2];
    else
        xy = [lines(junction.ilines(1,i)).point2;
lines(junction.ilines(1,i)).point1];
    end% if line i connected with the first point to junction.
    angle = atan2(xy(2,2) - xy(1,2), xy(2,1) - xy(1,1));
    if (angle ~= pi)
        angle = -1 * angle;
    end
    junction.ilines(3, i) = angle;
end % for i = 1:lines_number
end
```

Removal of redundant line segments:

```
function [res] = remove_redundantOld(segs, Image)
 theta_thres = 10;
 rho_thres = 20;

for i = 1:length(segs)
    line_size(i) = get_size(segs(i));
end

[ordered_sizes,ordered_seq]= sort(line_size,'descend');

segments = segs(ordered_seq);

tmp_lines = segments;
vals = zeros(2,length(segments));
for i = 1:length(segments)
    vals(:,i) = [segments(i).theta;segments(i).rho];
end
if 0
    res = segments;
else

        negative_rho = find(vals(2,:) < 0);
        vals(2,negative_rho) = vals(2,negative_rho).*-1;

        negative_theta = zeros(1,1);

        counter = 1;
        for i= negative_rho
          if vals(1,i) < 0
              negative_theta(counter) = i;
              counter = counter+1;
          end
        end

        positive_theta = zeros(1,1);

        counter = 1;
        for i= negative_rho
```

```
          if vals(1,i) >= 0
              positive_theta(counter) = i;
              counter = counter+1;
          end
        end


      if negative_theta ~=0
       vals(1,negative_theta) = vals(1,negative_theta)+180;
      end

      if positive_theta ~=0
       vals(1,positive_theta) = vals(1,positive_theta)-180;
      end
      counter = 1;


    while (counter < length(vals))
        d1 = abs(vals(1,(counter+1):end) - vals(1,counter));
        d2 = abs(vals(2,(counter+1):end) - vals(2,counter));

        d3 = abs(2*90+ vals(1,(counter+1):end) - vals(1,counter));

        ind = intersect(find(d1 < theta_thres | d3 < theta_thres ),find(d2 <
rho_thres));
        ind = ind + counter;

        segments(ind) = [];
        vals(:,ind) = [];
        counter = counter + 1;
    end
    res = segments;
end





function size = get_size(line)
        xy = [line.point1;line.point2];
        x1 = xy(1,1);
        y1 = xy(1,2);
        x2 = xy(2,1);
        y2 = xy(2,2);
        size = sqrt((x1-x2)^2 +(y1-y2)^2);
```

6 Conclusions

To implement the algorithm of edge and boundary interpretation is a difficult and challenging task. The main problem is caused by the fact that input images are not ideal. And it is highly difficult to define general thresholds that will be suitable for all input objects.

The improvements need to be made in object preprocessing in order to obtain the proper vertexes.

7 References

[1] IEEE Transactions on Pattern Analysis and Machine Intelligence, Farzin Mokhtarian and Riku Suomela, Vol. 20, NO. 12, December 1998

[2] Application of the generalized Hough transform to corner detection, E.R Davies, MA, DPhill, CPhys, FInstP

[3] A local edge detector used for finding corners, Fet Shen, Han Wang, School of Electrical and Electronic Engineering, Nanyang Technological University, Singapure

[4] Site of introduction to computational and biological vision, University Ben Gurion of the Negev. www.cs.bgu.ac.il/~icbv061