Automatic Line Detection

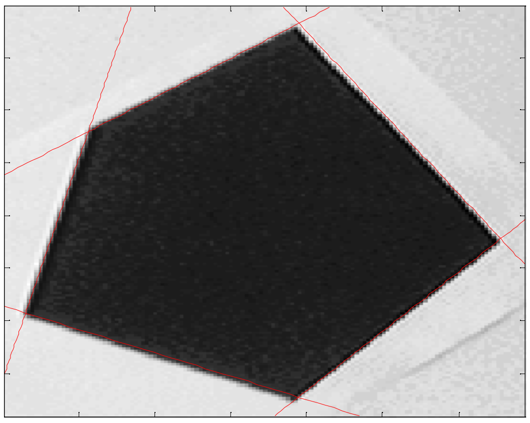

Table of Contents

Introduction

Knowledge about the lines in an image is useful in many applications, e.g. in Computer vision. If the equations of the same line in several 2D images are known, then it is possible to get the coordinates of the 3D object giving rise to the 2D images.

To manually extract the line information from an image can be very tiring and time-consuming especially if there are many lines in the image. An automatic method is preferable, but is not as trivial as edge detection since one has to determine which edge point belongs to which line, if any. The Hough-transform makes this separation possible and is the method we have used in our program for automatic line detection.

This project was performed as a part of the examination for the Computer Vision course given by the Mathematical Imaging Group at the University of Lund during the summer of 1999.

Theory of the Hough Transform

The Hough transform (HT), named after Paul Hough who patented the method in 1962, is a powerful global method for detecting edges. It transforms between the Cartesian space and a parameter space in which a straight line (or other boundary formulation) can be defined.

Let's consider the case where we have straight lines in an image. We first note that for every point (x_i, y_i) in that image, all the straight lines passing through that point satisfy Equation 1 for varying values of line slope and intercept (m,c), see Figure 1.

$$y_i = mx_i + c \qquad \text{Equation 1}$$

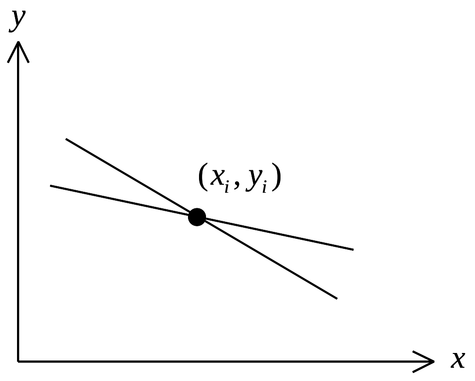

Figure 1: Lines through a point in the Cartesian domain.

Now if we reverse our variables and look instead at the values of (m,c) as a function of the image point coordinates (x_i, y_i), then Equation 1 becomes:

$$c = y_i - mx_i \qquad \text{Equation 2}$$

Equation 2 describes a straight line on a graph of c against m as shown in Figure 2.

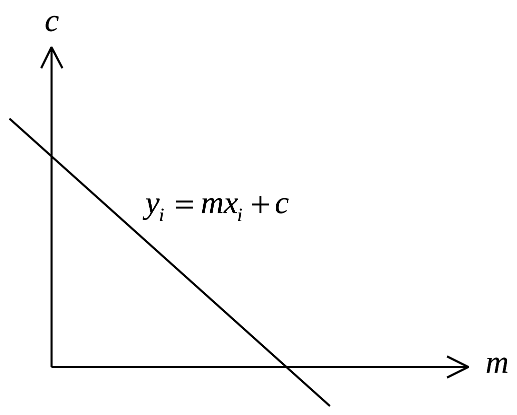

Figure 2: The (m,c) domain.

At this point, it is easy to see that each different line through the point (x_i, y_i) corresponds to one of the points on the line in the (m,c) space.

Now, consider two pixels P1 and P2, which lie on the same line in the (x, y) space. For each pixel, we can represent all the possible lines through it by a single line in the (m,c) space. Thus a line in the (x, y) space that passes through both pixels must lie on the intersection of the two lines in the (m,c) space, which represent the two pixels. This means that all pixels which lie on the same line in the (x, y) space are represented by lines which all pass through a single point in the (m,c) space, see Figure 3 and Figure 4.

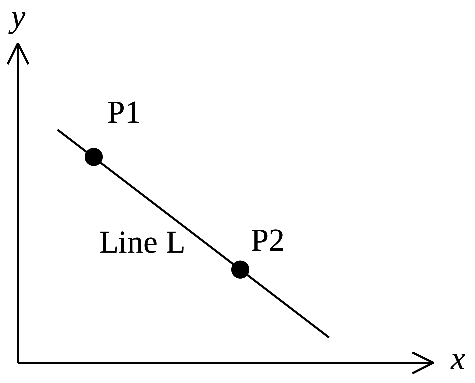

Figure 3: Points on the same line.

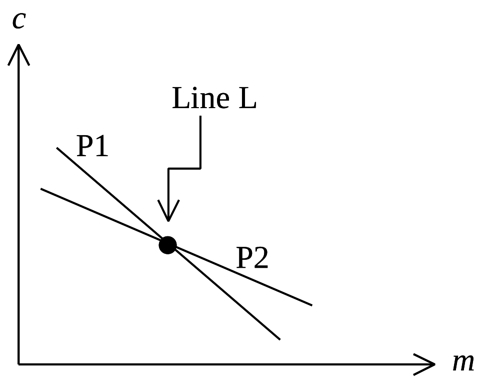

Figure 4: The mapping of P1 and P2 from Cartesian space to the (m,c) space.

Following the discussion above, we now can describe an algorithm for detecting lines in images. The steps are as follows:

1. Find all the edge points in the image using any suitable edge detection scheme.
2. Quantize the (m,c) space into a two-dimensional matrix H with appropriate quantization levels.
3. Initialize the matrix H to zero.

4. Each element of H matrix, $H(m_i, c_i)$, which is found to correspond to an edge point is incremented by 1. The result is a histogram or a vote matrix showing the frequency of edge points corresponding to certain (m,c) values (i.e. points lying on a common line).
5. The histogram H is thresholded where only the large valued elements are taken. These elements correspond to lines in the original image.

Advantages and Disadvantages

The advantage of the Hough transform is that the pixels lying on one line need not all be contiguous. This can be very useful when trying to detect lines with short breaks in them due to noise, or when objects are partially occluded.

As for the disadvantages of the Hough transform, one is that it can give misleading results when objects happen to be aligned by chance. This clearly shows another disadvantage which is that the detected lines are infinite lines described by their (m,c) values, rather than finite lines with defined end points.

Practical Issues

To avoid the problem of infinite *m* values which occurs when vertical lines exist in the image, the alternative formulation shown in Equation 3 can be used to describe a line, see Figure 5.

$$x \cos q + y \sin q = r \qquad \text{Equation 3}$$

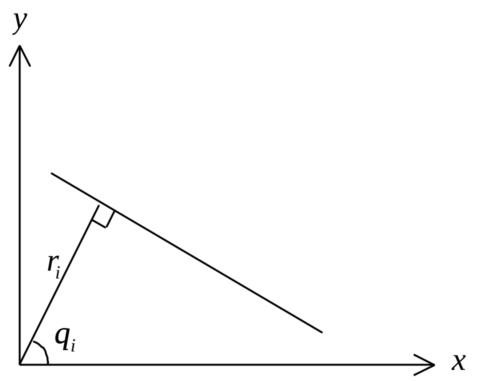

Figure 5: The representation of a line in the (x, y) space using (r_i, q_i).

This, however, means that a point in (x, y) space is now represented by a curve in (r, q) space rather than a straight line.

Note that the Hough transform can be used to detect shapes in an image other than straight lines such as circles for example. In that case, the only difference is that the method will have to be modified to use a threedimentional matrix H (with the three parameters: the radius, the x and y coordinates of the centre). Nevertheless, due to the increased complexity of the method for more complicated curves, it is practically of use only for simple ones.

Our Implementation

Overview

We have developed a graphical user interface (GUI) program using MATALB 5.3. This program allows the user to:

- Select an image (JPEG, TIFF, BMP, PNG, HDF, PCX, XWD, and PGM). See CVimage.
- Apply edge detection to a selected image using different gradient kernels (Sobel, Prewitt, Roberts), sub-pixel resolution, or other methods such as: Canny or looking for zero crossings after filtering the image with a Laplacian of Gaussian filter. See CVedge.
- Perform Hough transform on the detected edges. The user can specify the intended resolution for the resulting vote histogram. See CVhough.
- Extract plausible lines from the vote histogram matrix. The user can specify a vote threshold value that will effectively control the number of selected lines. See CVunhough.
- Sample the detected line equations and plot the lines on the image. See CVline.

Details

Following is a description of the main functions in our implementation. The prototype of each function is included as an aid to the reader. For the complete listing of the functions, refer to the Appendix.

CVimage [function I=CVimage;]

This function produces an open-file dialog box, and prompts the user to select an image file. The image file string is examined and then the appropriate image reading function is called. The function that is used to read the image file is either readpgm(…) or MATLABs imread(…). The image is then, if needed, converted to a gray image and then normalized to a matrix of values ranging from zero to one. This matrix is returned by the function in the variable I. Figure 6 shows an example of such an image.

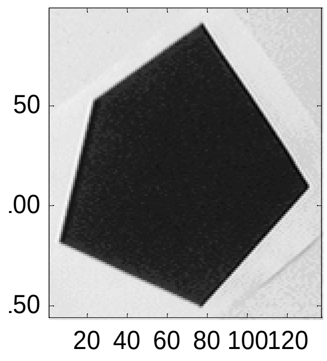

Figure 6: Loaded Image.

CVedge [function edgedata=CVedge(I,M,T,A);]

This function finds the coordinates of the pixels with high gradient values in the supplied normalized image I. These coordinates are returned in the matrix edgedata, where the first row contains the x-coordinates and the second contains the y-coordinates. How high the gradient should be for a pixel to be selected is specified by a threshold value T that ranges between zero and one. The input argument M is either 1 or 2. If M is 1 then the sub-pixel edge detection is used and the A value represents the width of the smoothing kernel. If M is 2 then MATALBs edge(…) function is used and A specifies the method to be used (e.g. 'Sobel', 'zerocross', 'canny'). An example of detected edges is shown in Figure 7.

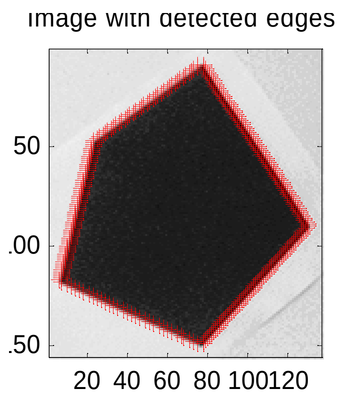

Figure 7: Image with detected edges.

CVhough [function [H,m,b]=CVhough(edgedata,nT,nS);]

As explained previously, the Hough transform of an image results in a number of votes for a set of parameterized lines. The parameters used to describe the lines are the orientation (of a line segment extending from the origin and normal to the desired line) and the length of this segment. Since quantization of the line parameters is inevitable, we have allowed the user to specify the number of orientations (nT) and distances (nS). Suitable default values are selected based on the image dimensions if the user chooses to ignore these parameters. Unlike nT and nS, edgedata (explained in CVedge) is a necessary input argument. With the nT, nS, and edgedata we proceed as follows.

We find the different quantized orientations. And then for each edgedata coordinate we calculate the distance parameter for all the orientations according to Equation 3, and end up with a matrix of dimensions: length(edgedata)×nS.

A certain mapping is then done to convert a specific orientation-distance pair to a column-row index into the vote histogram matrix. Basically it is a affine mapping for the orientation: 0→1 & pi-pi/nT→nT and another affine mapping for the distances: min(S)→1 & max(S)→nS, with the values in between rounded to an integer.

The final step is to find how many edge points belong to a specific orientation-distance pair. This was done in two ways, the first is to loop over all the distance values and sum up the edge data having specific angles. The other way was to use 3D matrices and perform this counting without the use of loops. It turned out that the 3D matrices took a lot of memory and resulted in slower performance (the code for 3D matrices is included as comments in the Appendix). Having completed the vote counting, this function returns the vote matrix H, and two other parameters m and b, that describe the distance mapping and will be used later on to find the real values of the line parameters. Below is an example of a vote histogram illustrated in 2D and 3D plots, see Figure 8 and Figure 9. Notice the five peaks in the histogram reflecting the five edges of the pentagon image.

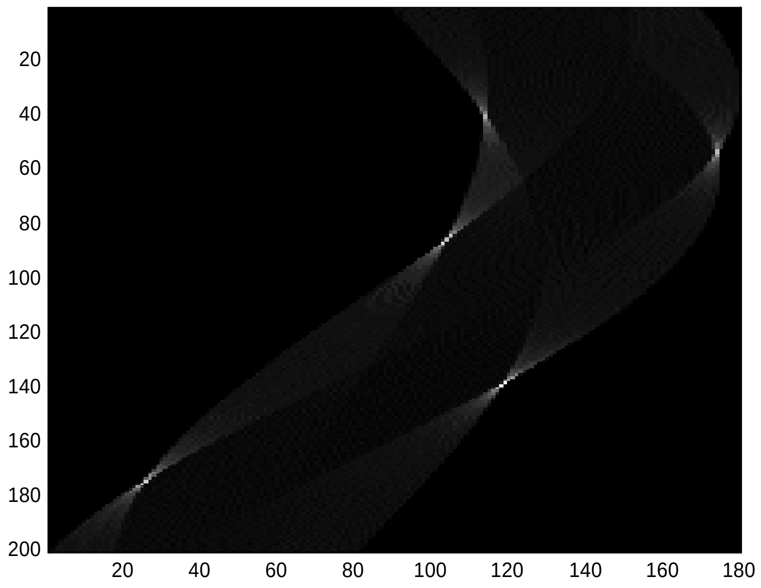

Figure 8: Vote histogram.

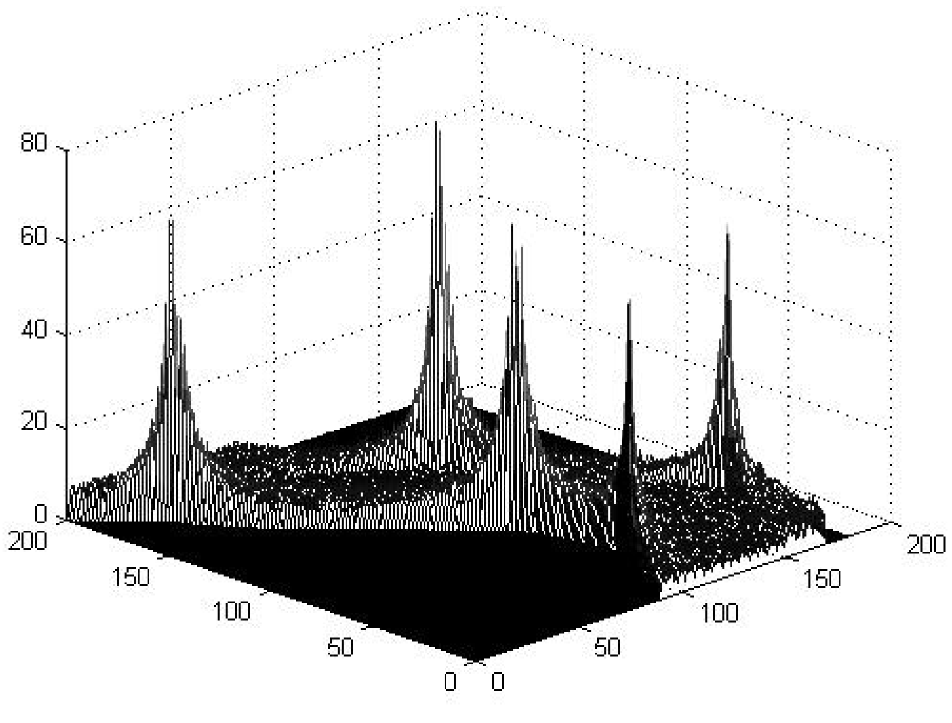

Figure 9: Vote histogram in 3D.

CVunhough [function [SL,TL,intSL,intTL]=CVunhough(H,m,b,P);]

The aim of this function is to search through the vote matrix and select the lines with high votes, and then convert the vote row-column indices to true orientation-distance values.

Different methods for selecting lines with high votes from the H matrix were examined. The problem is that many of the neighboring points in the Hough domain are essentially the same line in the image. One idea was to smooth the vote matrix and then perform thresholding. Another was to threshold first and then proceed with a clustering analysis to combine the neighboring points into one cluster and select the mean of this cluster as a the detected line. Eventually, we chose to use the following strategy. We first threshold the vote matrix using the value P which has values between 0 and 1 (either provided as an input argument or a default value is used) and thus obtain a binary matrix with the ones donating lines that have high vote count and zeros otherwise. Now, in order to combine the neighboring points into one we first dilate the binary vote matrix and then label the different resulting regions as different lines. Then we assign one line to each region with its parameters equal to the mean of the parameters of all the points in that region. Figure 10 shows five selected regions after thresholding and dilating the vote histogram. Figure 11 shows the detected lines after averaging the regions.

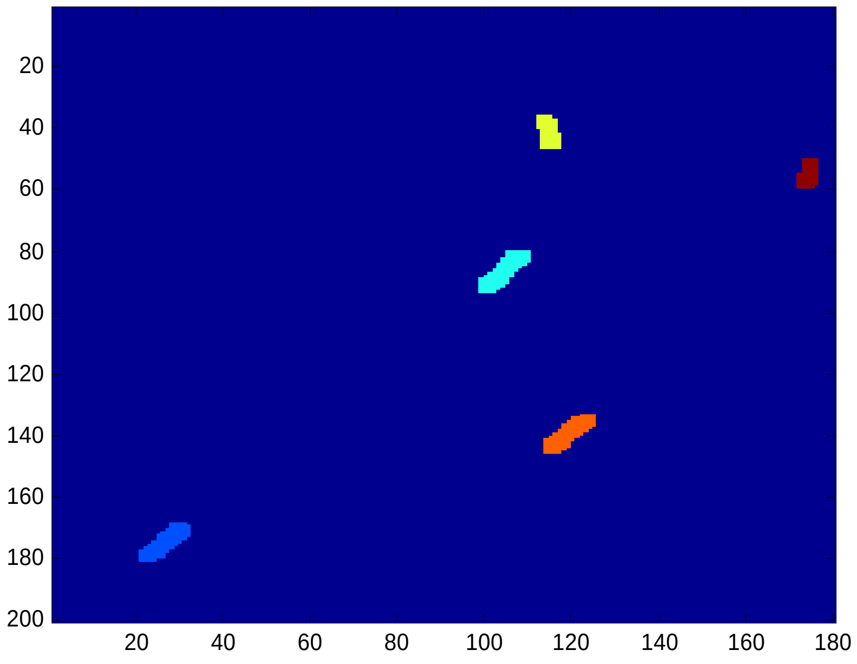

Figure 10: After thresholding and dilating the vote histogram, five regions are obtained. Each region will be assigned to one line.

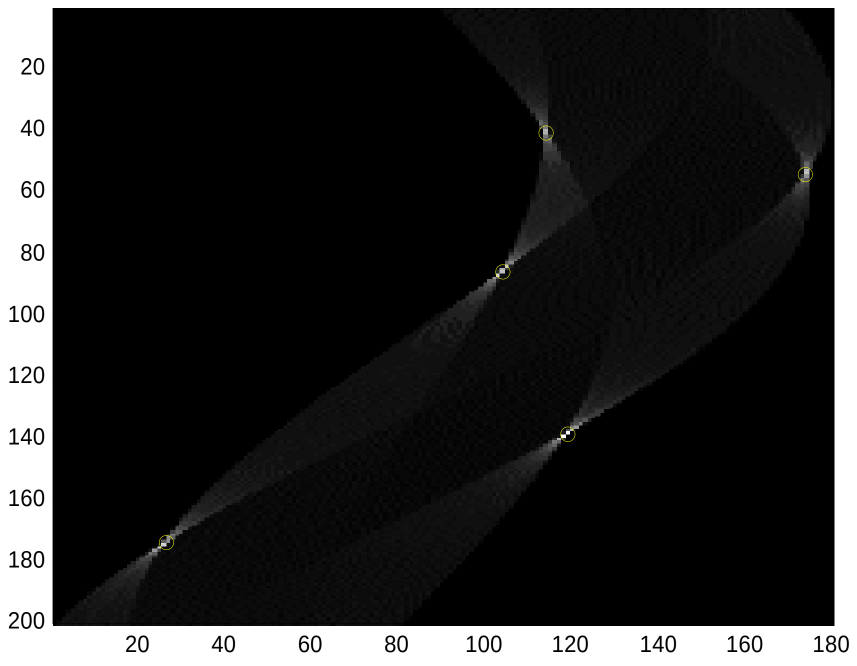

Figure 11: The detected lines marked with ‘o’on the vote histogram.

Since we now work with the indices of the vote matrix, we need to convert these indices to true orientation-distance values and hence we make use of the m and b parameters as explained in CVhough.

The values returned from this function are the distance-orientation pairs for all the selected lines (in real values: SL and TL, and also in row-column entries into the vote matrix: intSL and intTL). This function also prints the parameters of the selected lines in [a, b, c] format.

CVline [function [XL,YL]=CVline(SL,TL,X);]

This function samples the line equations with orientation-distance parameters specified in SL and TL for the range of x-coordinate values specified in X. It returns the x and y coordinates in XL and YL. Figure 12 shows the detected lines plotted over the original image.

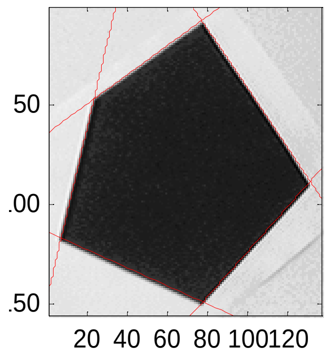

Figure 12: Detected lines.

Graphical User Interface

All the above functionality is combined in a user-friendly manner by producing the 'Line Detection Using Hough Transform' GUI. The GUI is shown below in Figure 13.

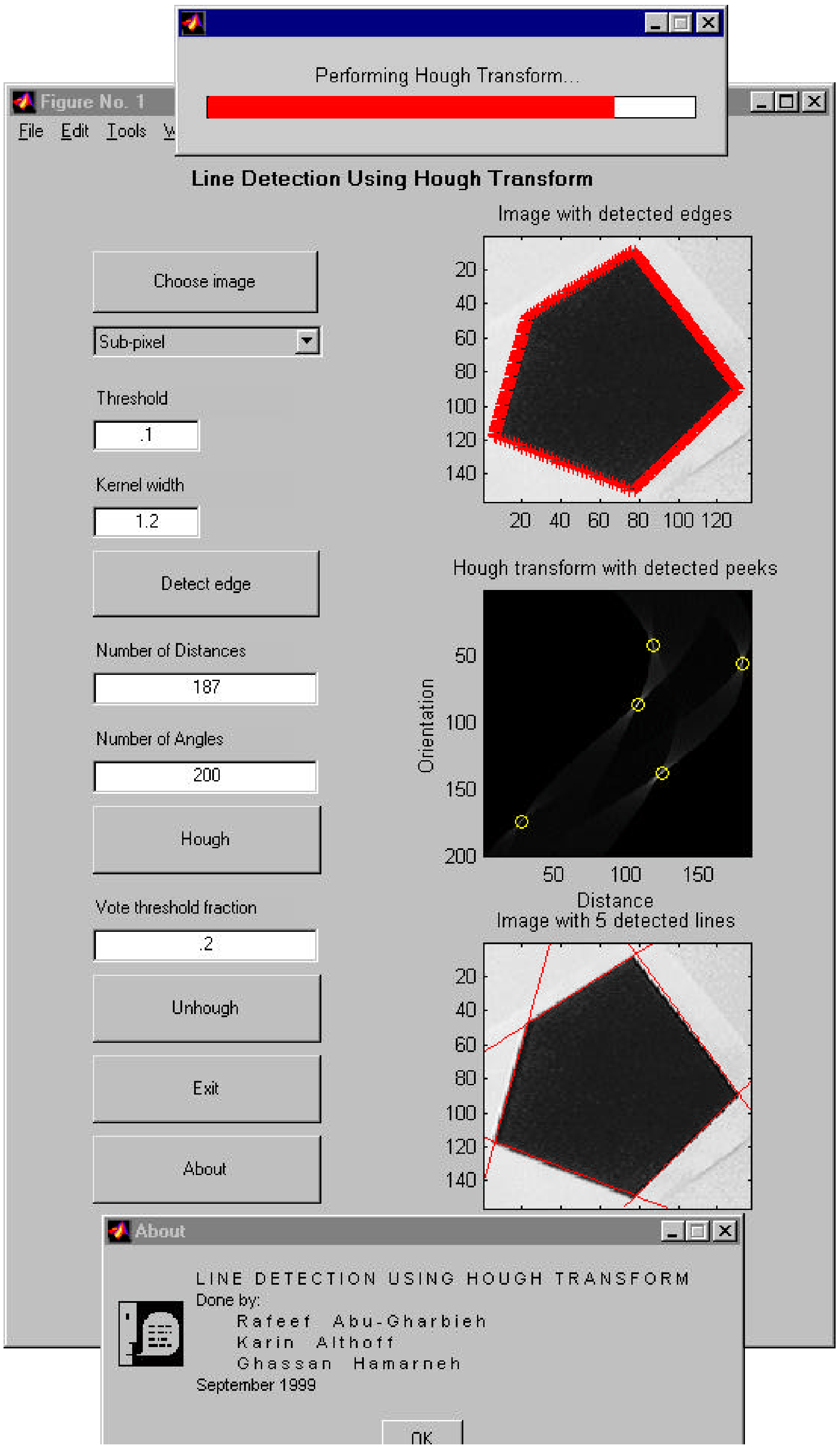

Figure 13: Illustration of our graphical user interface.

A typical sequence of events while using the GUI might be as follows. The user presses 'Choose image', a dialog box appears, and the user browses and selects the image file desired. The image is displayed in the top subplot. Then the user specifies the type of edge detection method from the dropdown list box, and then fills in the parameters' text boxes that are updated dynamically based on the edge detection method. The user then presses the 'Detect edge' button. The user tries different edge detection methods until the result is satisfactory. The detected edges are overlaid on the original image on the top subplot. Then the user specifies the parameters for the Hough transform and presses the 'Hough' button. The resulting vote matrix is displayed in the middle subplot. Finally the user presses 'Unhough' which will result in drawing the image with the detected lines in the bottom subplot, and also in marking the selected lines on the vote matrix in the middle subplot.

Results

In this section we present some examples illustrating our implementation of automatic line detection using the Hough transform.

For each of the following four figures, we display the three subplots produced by our GUI program. The original image with its detected edge points is shown in the top subplot. The middle subplot displays the vote histogram with the selected lines overlaid. The bottom subplot shows the original image with the detected lines added.

Figure Number	**Edge Detection Method**	**Threshol d**	**Kernel Width**	**Number of Distances**	**Number of Orientations**	**Vote Fraction**
14	Canny	0.3	-	100	200	0.50
15	Canny	0.8	-	836	200	0.50
16	Canny	0.1	-	331	200	0.25
17	Sub-pixel	0.1	1.2	300	200	0.20

Table 1: The input parameters used to produce the results shown in Figures 14-17.

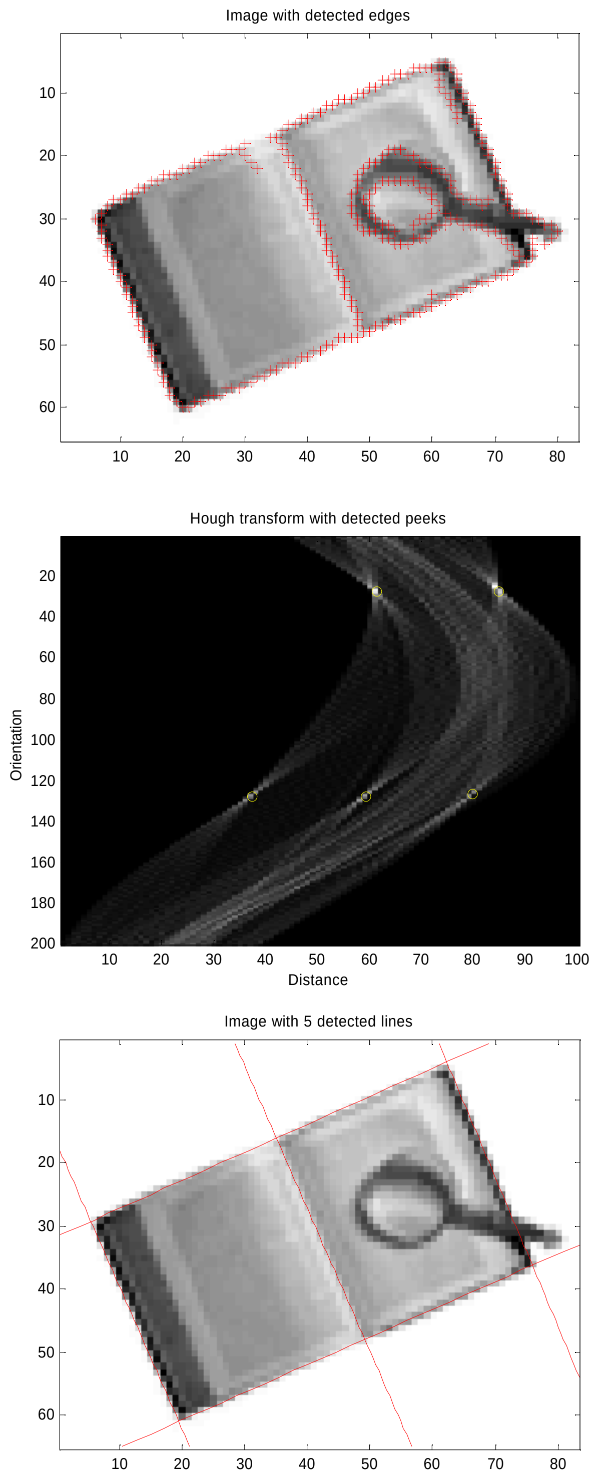

Figure 14: Top; image with detected edges, middle; vote histogram with selected lines, bottom; image with detected lines.

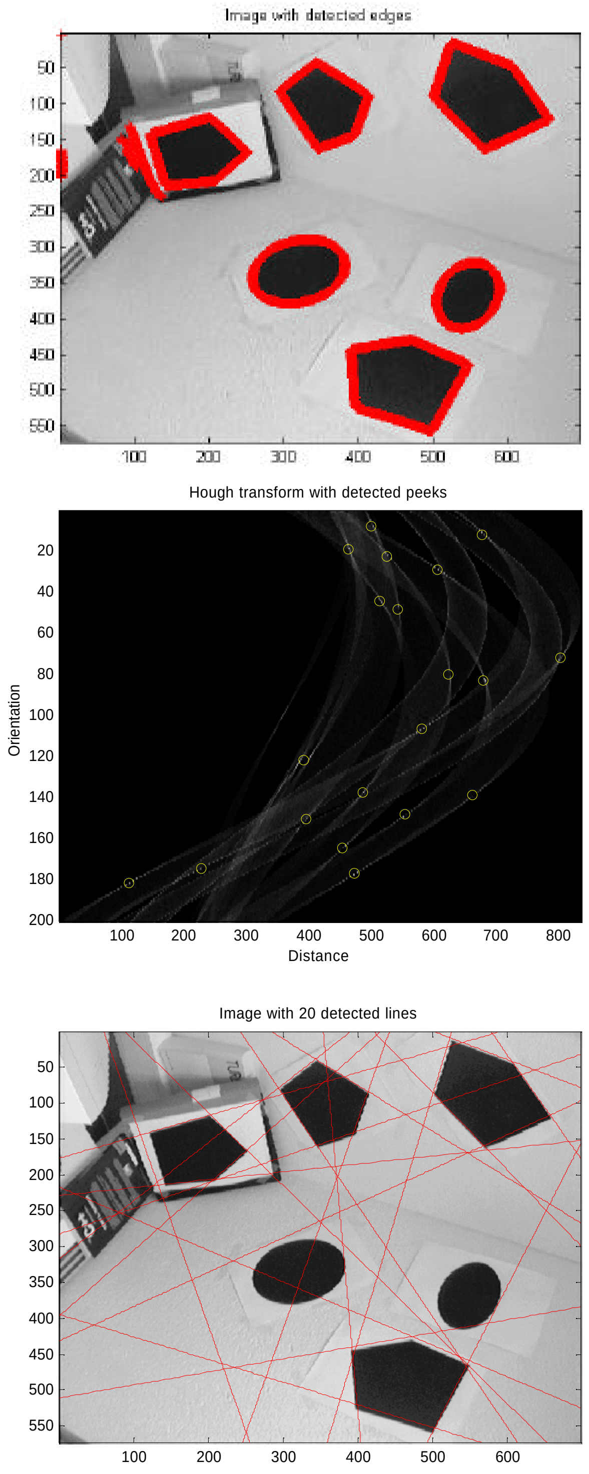

Figure 15: Top; image with detected edges, middle; vote histogram with selected lines, bottom; image with detected lines.

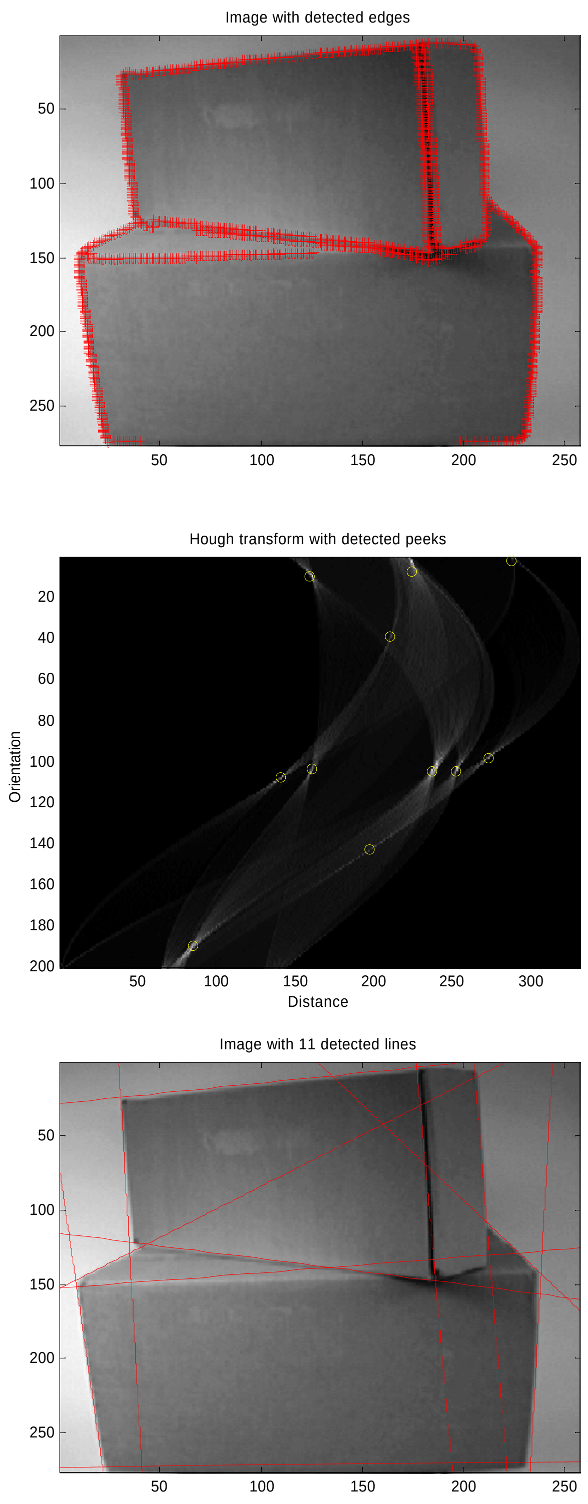

Figure 16: Top; image with detected edges, middle; vote histogram with selected lines, bottom; image with detected lines.

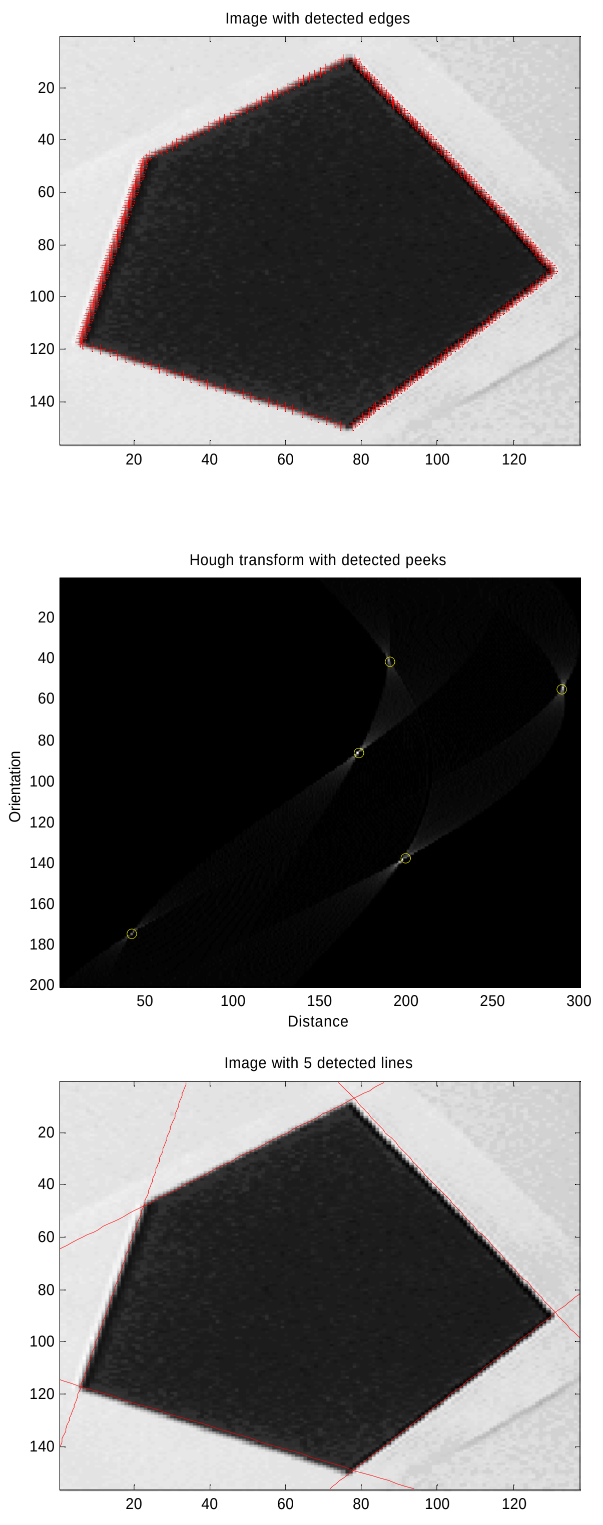

Figure 17: Top; image with detected edges, middle; vote histogram with selected lines, bottom; image with detected lines.

Appendix - MATLAB code

The MATLAB code for the: CVimage, CVedge, CVhough, CVunhough, and CVline functions are included below. The GUI developed also requires some callback functions: CBimage, CBedge, CBhough, CBunhough, CBabout, and CBexit. Other external dependency functions are: rita, calcw123_1, rowedges, and readpgm. The callback and the external functions are not included here.

CVimage

```
function I=CVimage;
%function I=CVimage;
%CVimage returns a normalized (0-1) grey scale image given the path
%  I  returned image
%
%  See also: CVimage, CVhough, CVunhough, CVedge, CVline, CVproj


[F,P]=uigetfile('*.pgm;*.jpg;*.jpeg;*.tif;*.tiff;*.bmp;*.png;*.hdf;*.
pcx;*.xwd','Choose Image');

if F==0
   I=[];
else
   PF=[P,F];
   ext=PF(findstr(PF,'.')+1:end);

   if strcmp(ext,'pgm')
      I  = readpgm(PF);
   else %matlab image types
      [Im,MAP]=imread(PF);
      I = ind2gray(Im,MAP);
   end
   I = I/max(I(:));
end
```

CVedge

```
function edgedata=CVedge(I,M,T,A);
%CVedge finds the coordinates of the edges in an image
%
%function edgedata=CVedge(I,M,T,A);
%  M  1:subpixel
%  T     T:threshold = T
%  A     A:width of smoothing kernel = B
%  M  2:edge function
%  T     T:threshold
%  A     A:method ex. 'Sobel', 'Roberts'
%  edgedata a 2-row matrix, with the x and y coordinates of the edges
%
%  See also: CVimage, CVhough, CVunhough, CVedge, CVline, CVproj

if M>2
   error('M should be 1(subpixel) or 2(edge)');
elseif M==1 %SUBPIXEL
   edgedata=[];
   for rownr = 1:size(I,1);
      row = I(rownr,:);
      edgeposfine=rowedges(row,A,T);
      edgedata=[edgedata
[edgeposfine;rownr*ones(size(edgeposfine))]];
   end;
elseif M==2 %EDGE
   switch A
   case 1,
      meth='sobel';
   case 2,
      meth='prewitt';
   case 3,
      meth='roberts';
   case 4,
      meth='log';
   case 5,
      meth='zerocross';
   case 6,
      meth='canny';
   otherwise,
      error('edge method values only 1 through 6');
   end
   E=edge(I,meth,T);
   [r,c]=find(E);
   edgedata=[c';r'];
end
```

CVhough

```
function [H,m,b]=CVhough(edgedata,nT,nS)
%CVhough Hough transform of a binary matrix
%
%function [H,m,b]=CVhough(edgedata,nT,nS)
%  edgedata a 2-row matrix, with the x and y coordinates of the edges
%  nT number of orientations(thetas)~ 1/orientation resolution
%  nS number of distances           ~ 1/distance resolution
%  H votes histogram
%  m and b: distance mapping parameters
%           [1..nS] = [Smin...Smax]*m + b%
%
%  See also: CVimage, CVhough, CVunhough, CVedge, CVline, CVproj

MAXDIST=1.2;


if nargin<1
   error('require at least one input argument: binary image')
elseif nargin<2
   warning('defualt value of 200 assigned to number of orientations
nT')
   nT=200;
   warning(['defualt  value  of',  max(edgedata(:))*MAXDIST,  'assigned
to number of orientations nS'])
   nS=max(edgedata(:))*MAXDIST;
elseif nargin<3
   warning(['defualt  value  of',  max(edgedata(:))*MAXDIST,  'assigned
to number of orientations nS'])
   nS=max(edgedata(:))*MAXDIST;
end

row=edgedata(2,:)';
col=edgedata(1,:)';

%defining the range of the orientations of line
Ts=[0:pi/nT:pi-pi/nT]';
```

```
%cos and sin of all the angles
CsT=cos(Ts);
SnT=sin(Ts);

%solving for distances for all orientations at all nonzero pixels
%size of S is: [length(row) , length(Ts)]
S=row*CsT' + col*SnT';

%mapping:
%  Smin = min(S(:))--> 1
%  Smax = max(S(:))--> nS
%gives (y=mx+b):
%  m=(nS-1)/(Smax-Smin)
%  b=(Smax-nS*Smin)/(Smax-Smin)
%and then round it and get rounded mapped S:rmS

Smin=min(S(:));
Smax=max(S(:));
m =(nS-1)/(Smax-Smin);
b =(Smax-nS*Smin)/(Smax-Smin);
rmS=round(m*S + b);

%Note:   H is [nT,nS]
%        rmS is [nP,nT]  nP:number of edge points

H=[];
hw=waitbar(0,'Performing Hough Transform...');
for k=1:nS,
   isEq=(rmS==k);
   %  H=[H,sum(isEq)'];  %sum(isEq)  1 x nT
   H(:,k)=sum(isEq)';
   waitbar(k/nS,hw);
end
close(hw);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  USING 3D MATRICES
%
%  we tried to calculate the votes for all the S,T pairs
%  in the hough transform using 3D matrices and without
%  using for loops, but 3D matrices took a lot of memory
%  and resulted in slower performance. (See below...)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%counting number of specific distances for each angle
%first produce a 3D matrix with repS(:,:,i)=i, where i=1:nS

%TMP% repS=shiftdim(repmat(1:nS,[nT,1,length(row)]),2);

%then we repeat the rmS matrix and get a 3D reprmS(:,:,i)=rmS, where
i=1:nS

%TMP% reprmS=repmat(rmS,[1 1 nS]);

%then we compare repS with reprmS
%dim1=#nonzeros pixels
%dim2=number of orientations nT
%dim3=number of distances nS

%TMP% isEq=(repS==reprmS);

%we sum up the ones for each direction at each distance
%and obtain H(histogram of votes) of size [nT,nS]
```

```
%TMP% H=squeeze(sum(isEq,1));
```

CVunhough

```
function [SL,TL,intSL,intTL]=CVunhough(H,m,b,P)
%CVunhough finds lines from a Hough histogram
%function [SL,TL,intSL,intTL]=CVunhough(H,m,b,P)
%  H votes histogram of size [nT,nS]
%  P percentage threshold
%  m and b: distance mapping parameters
%           [1..nS] = [Smin...Smax]*m + b
%  SL distances of selected lines
%  TL orientations of selected lines
%  intSL distances of selected lines after mapping to 1:nS
%  intTL orientations of selected lines after mapping to 1:nT
%
%  See also: CVimage, CVhough, CVunhough, CVedge, CVline, CVproj

DILATEFRAC=.02;


if nargin<3
   error('require at least 3 input arguments: histogram matrix H, 2
distance mapping parameters m & b');
elseif nargin<4
   warning('defualt value of 0.7 assigned to percentage threshold
P');
   P=0.7;
end

[nT,nS]=size(H);

%locate the peeks in the histogram with
%votes more than P*100% of the highest vote
%note: r~orientation, c~distance
%TMP% [r,c]=find(H>=P*max(H(:)));

%Theshhold H --> TH
TH=im2bw(H,P*max(H(:)));

%Morphological
H1=dilate(TH,ones(round(DILATEFRAC*size(H))),1);

%Labeling
L=bwlabel(H1,8);
n=max(L(:));

%for the n lines found above
%we collect the indices into the Hough votes matrix
intSL=[]; intTL=[];
for k=1:n,
   [r,c]=find(L==k);
   intTL=[intTL; mean(r)];
   intSL=[intSL; mean(c)];
end


%remap r to orientation, we want
%  r=1  --> 0 rad  and
%  r=nT --> pi-pi/nT rad
TL=(intTL-1)*pi/nT;
```

```
%remap c to distance
%in hough we mapped (y=mx+b):
%  [1..nS] = [Smin...Smax]*m + b
%now we need to reverse mapping
%  x=(y-b)/m
%where
%  m=(nS-1)/(Smax-Smin)
%  b=(Smax-nS*Smin)/(Smax-Smin)

SL=(intSL-b)/m;




%printing the lines on the matlab workspace
disp('Selected lines in the form [a b c]')
disp('-----------------------------------')
for k=1:n,
   a=num2str(cos(TL(k)));
   b=num2str(sin(TL(k)));
   c=num2str(-SL(k));
   disp(['Line  ',num2str(k),'/',num2str(n),':  [  ',a,'  ',b,'  ',c,'
]']);
end
```

CVline

```
function [XL,YL]=CVline(SL,TL,X)
%CVline converts lines in distance-orientation representation
%       to x and y coordinates for a given range of X
%
%function [XL,YL]=CVline(SL,TL,X)
%  X range of X (e.g X=1:50;)
%  SL distances of selected lines
%  TL orientations of selected lines
%  XL X's of generated lines
%  YL Y's of generated lines
%
%  See also: CVimage, CVhough, CVunhough, CVedge, CVline, CVproj

X=X(:)'; %make X a row vector

%s=x cos(th) + y sin(th) ==> %y=(s - x cos(th))/sin(th)
YL=((repmat(SL,1,length(X))-cos(TL)*X)./...
   sin(repmat(TL,1,length(X))))';
XL=repmat(X,size(YL,2),1)';
```