ABSTRACT:

The **Hough transform** is a feature extraction technique used in image analysis, computer vision, and digital image processing. The purpose of the technique is to find imperfect instances of objects within a certain class of shapes by a voting procedure. This voting procedure is carried out in a parameter space, from which object candidates are obtained as local maxima in a so-called accumulator space that is explicitly constructed by the algorithm for computing the Hough transform.

The classical Hough transform was concerned with the identification of lines in the image, but later the Hough transform has been extended to identifying positions of arbitrary shapes, most commonly circles or ellipses.

TECHNICAL DISCUSSION

Line Detection

1. Change the image to grey scale and resize if required
2. Detect edges using Canny edge detection
3. Specify the range of theta which will be used (1:180)
4. Specify the range of rho by determining the diagonal of the image which will be the maximum Rho
5. Determine rho and theta of the lines passing through each point lying on the edge detected by Canny [rho=x*cos(theta)+y*sin(theta)]
6. A voting matrix is used to store the number of times of each rho and theta voted by each point
7. Find the peaks (the rho and theta of the line which got the maximum votes) in the Voting Matrix
8. Plot those lines on the image after transforming the polar coordinates to Cartesian

Circle Detection

1. Change the image to grey scale and resize if required
2. Detect edges using Canny edge detection
3. The range of theta which is used (0:180)

4. Specify the range of radius which will be used in the image
5. Get a and b by using equation[a=x+ r*cos (theta) , b=y +r*sin(theta)]
6. A voting matrix is used to store the number of times of each a and b and r voted by each point
7. Find the peaks (the a and b and r of the circle which got the maximum votes) in the Voting Matrix
8. Plot those circles on the image

Ellipse Edge detection

1. Change the image to grey scale and resize if required
2. Detect edges using Canny edge detection
3. The range of theta which is used (0:180)
4. Specify the range of major axis which will be used in the image
5. Fit an ellipse by examining all possible major axes (all pairs of points) and getting the minor axis using Hough transform.
6. A voting matrix is used
7. Find the peaks in the Voting Matrix
8. Plot those ellipses on the image

RESULTS:

LINE DETECTION

IMAGE 1

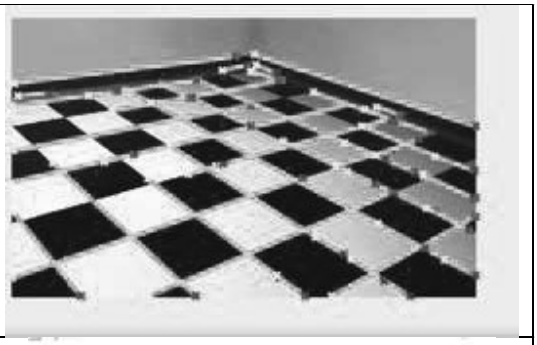
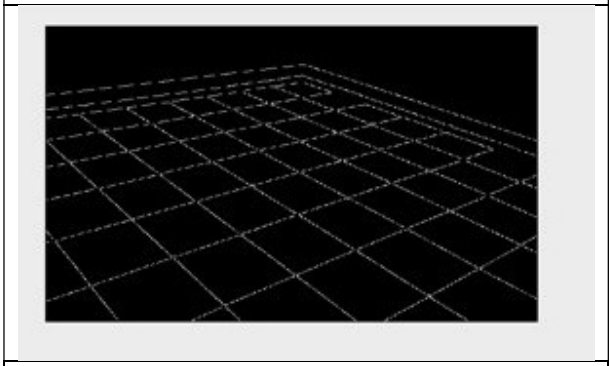
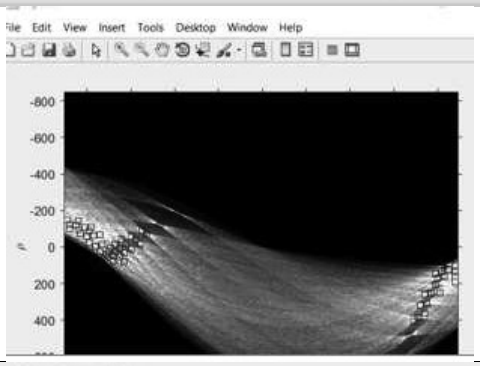
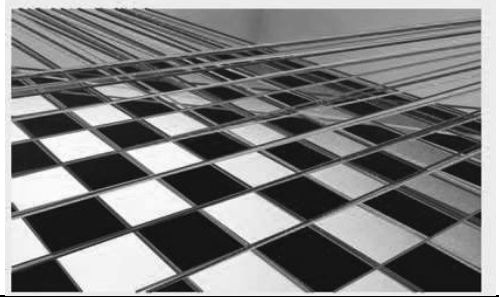
MATLAB available line code With edge threshold= 0.078	
	
OUR MATLAB CODE WITH THRESHOLD=100	

IMAGE 2

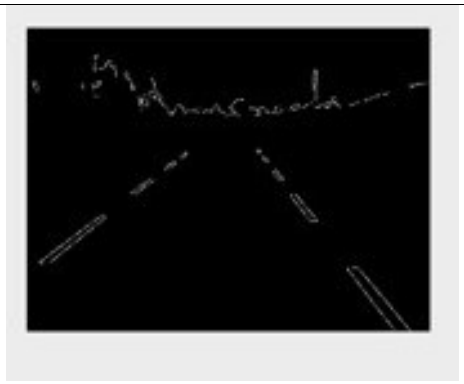
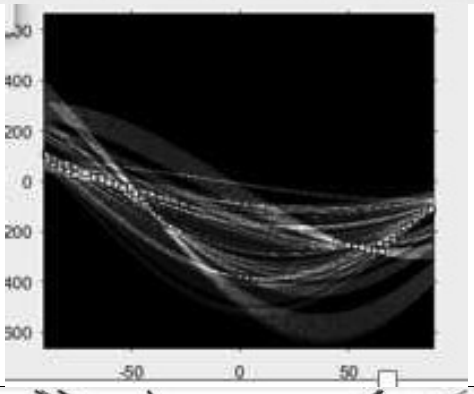
MATLAB available line code With edge threshold= 0.55	
	
OUR MATLAB CODE WITH THRESHOLD=155	

IMAGE 3

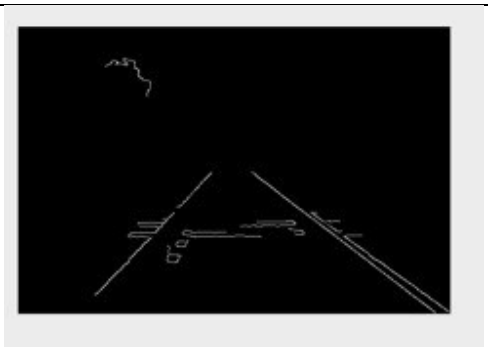
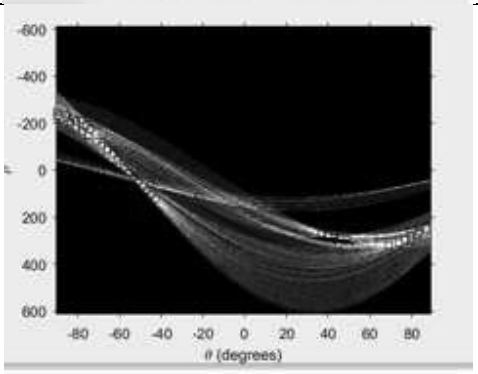
MATLAB available line code With edge threshold= 0.75	
	
OUR MATLAB CODE WITH THRESHOLD=175	

IMAGE 4

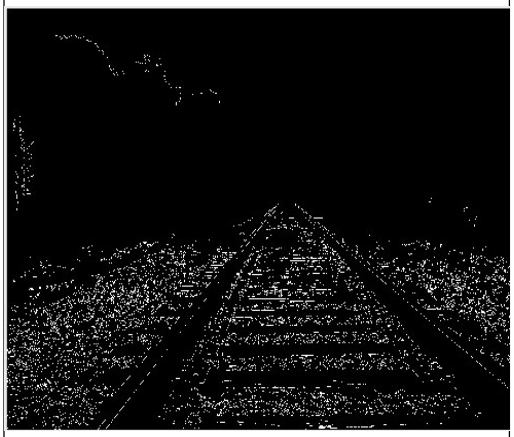
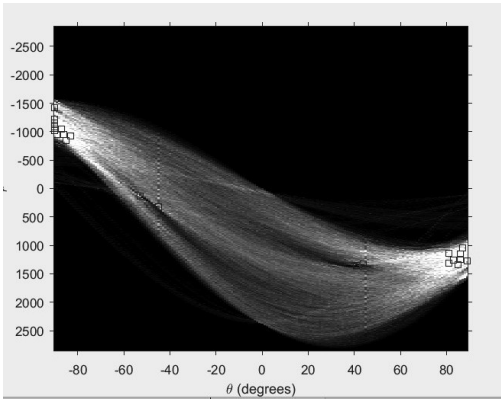
MATLAB available line code With edge threshold= 0.5	
	
OUR MATLAB CODE WITH THRESHOLD= 250	

IMAGE 5

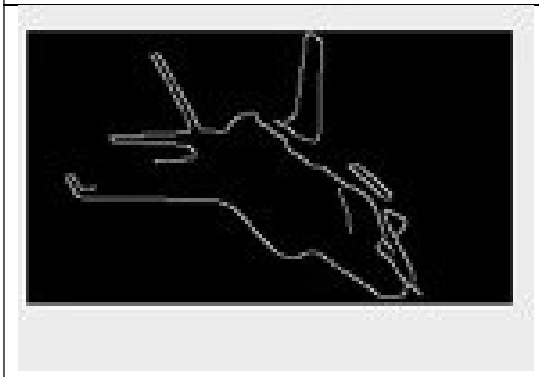
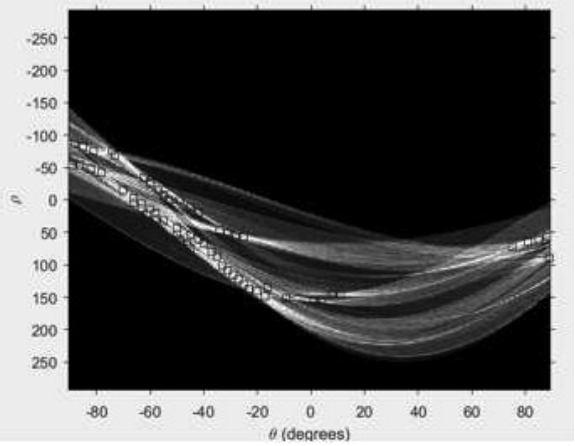
MATLAB available line code With edge threshold= 0.5	
	
OUR MATLAB CODE WITH VOTING THRESHOLD= 45	

IMAGE 6

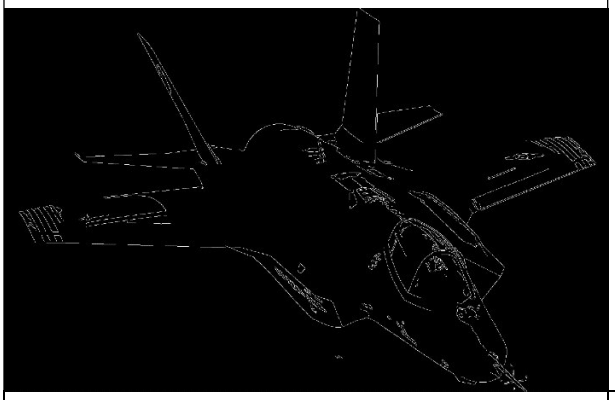
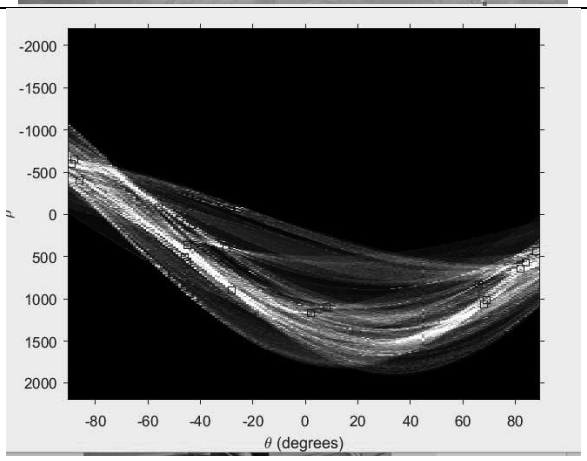
MATLAB available line code With edge threshold= 0.15	
	
OUR MATLAB CODE WITH VOTING THRESHOLD= 60	

IMAGE 7

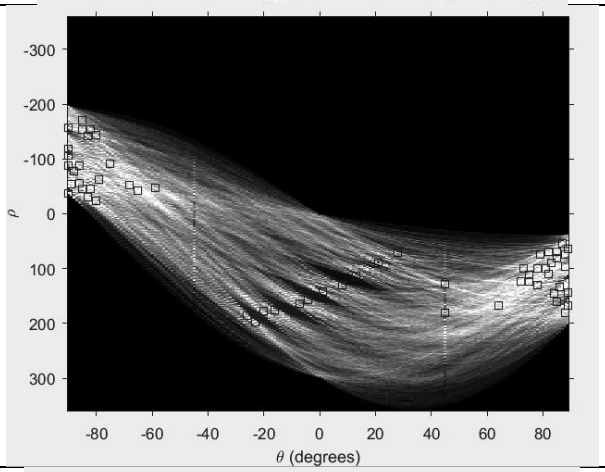
MATLAB available line code With edge threshold= 0.2	
	
OUR MATLAB CODE WITH VOTING THRESHOLD= 60	

IMAGE 8

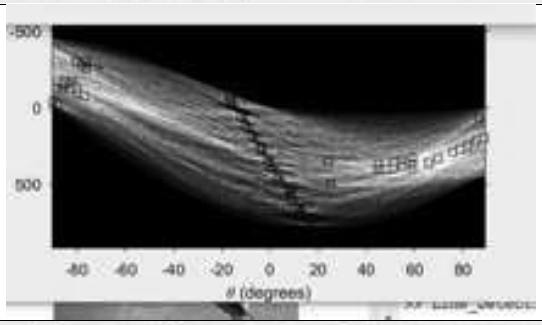
MATLAB available line code With edge threshold= 0.2	
	
OUR MATLAB CODE WITH VOTING THRESHOLD= 155	

IMAGE 9

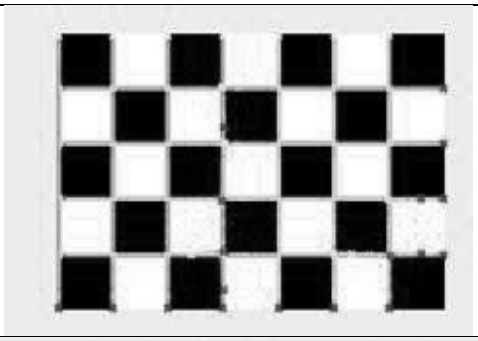
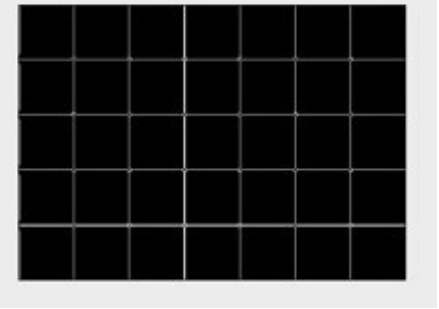
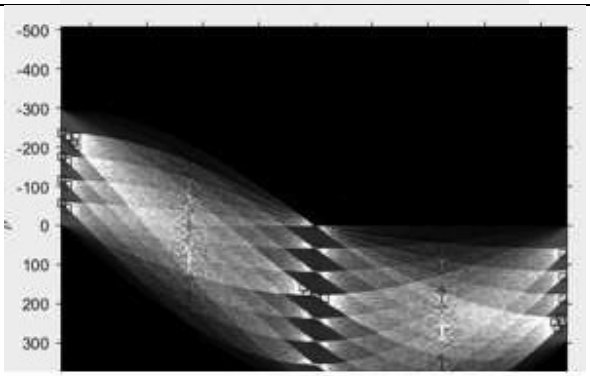
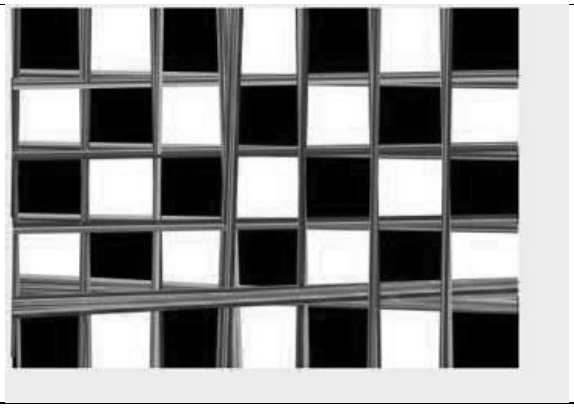
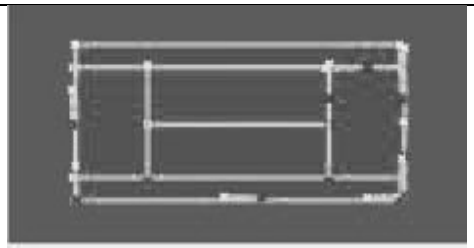
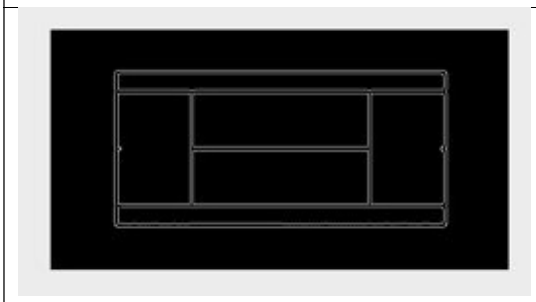
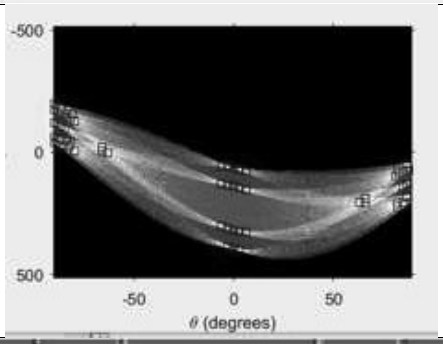
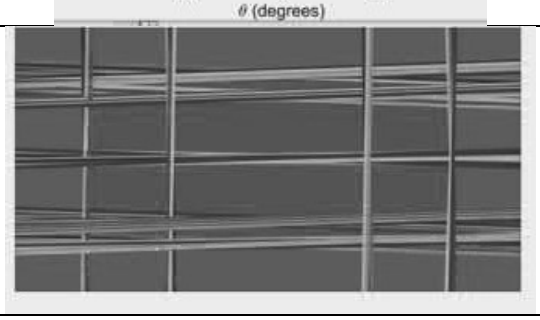
MATLAB available line code With edge threshold= 0.078	
	
OUR MATLAB CODE WITH VOTING THRESHOLD= 50	

IMAGE 10

MATLAB available line code With edge threshold= 0.5	
	
OUR MATLAB CODE WITH VOTING THRESHOLD= 60	

STAIR DETECTION

IMAGE 1 WITH THRESHOLD

0.6 FOR CANNY EDGE DETECTION

85 FOR VOTING MATRIX

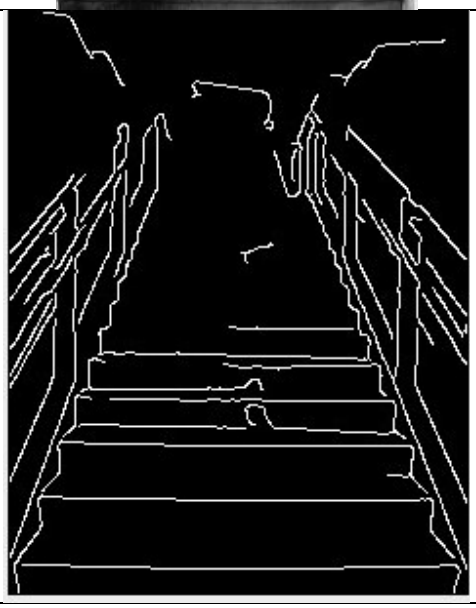


IMAGE 2 WITH THRESHOLD

0.3 FOR CANNY EDGE DETECTION

70 FOR VOTING MATRIX

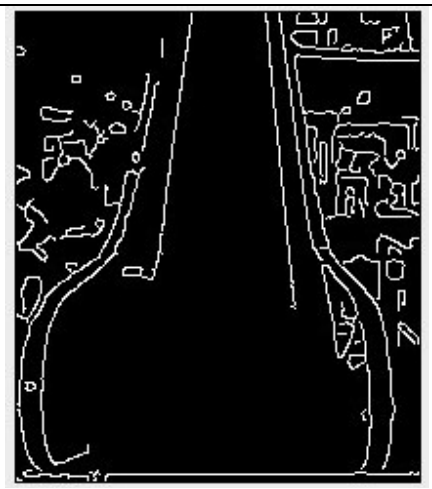


IMAGE 3 WITH THRESHOLD

0.4 FOR CANNY EDGE DETECTION

140 FOR VOTING MATRIX

IMAGE 4 WITH THRESHOLD

0.5 FOR CANNY EDGE DETECTION

130 FOR VOTING MATRIX

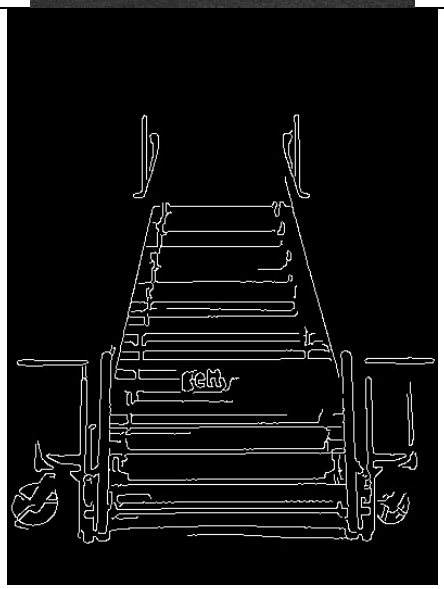
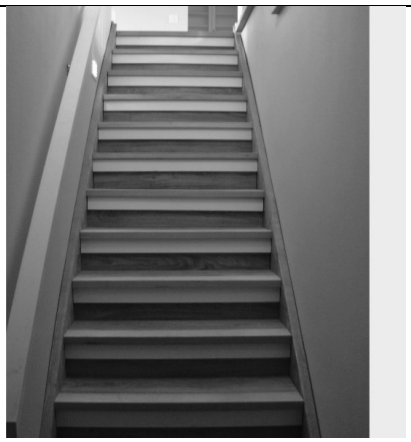


IMAGE 5 WITH THRESHOLD

0.4 FOR CANNY EDGE DETECTION

350 FOR VOTING MATRIX



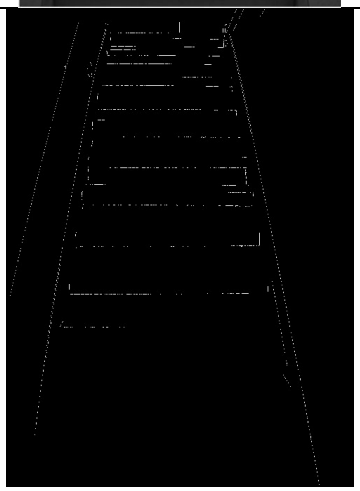


IMAGE 6 WITH THRESHOLD

0.4 FOR CANNY EDGE DETECTION

150 FOR VOTING MATRIX

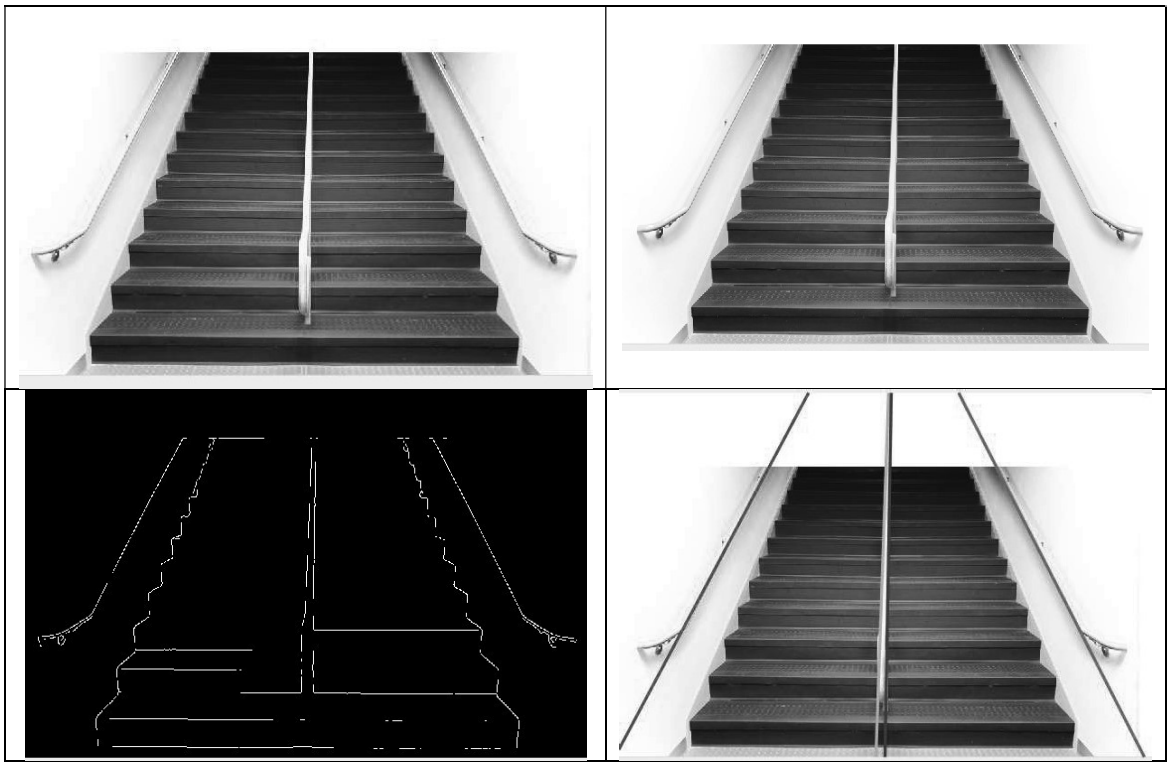


IMAGE 7 WITH THRESHOLD

0.1 FOR CANNY EDGE DETECTION

100 FOR VOTING MATRIX

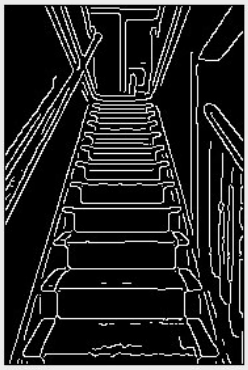
	
	

IMAGE 8 WITH THRESHOLD

0.2 FOR CANNY EDGE DETECTION

300 FOR VOTING MATRIX

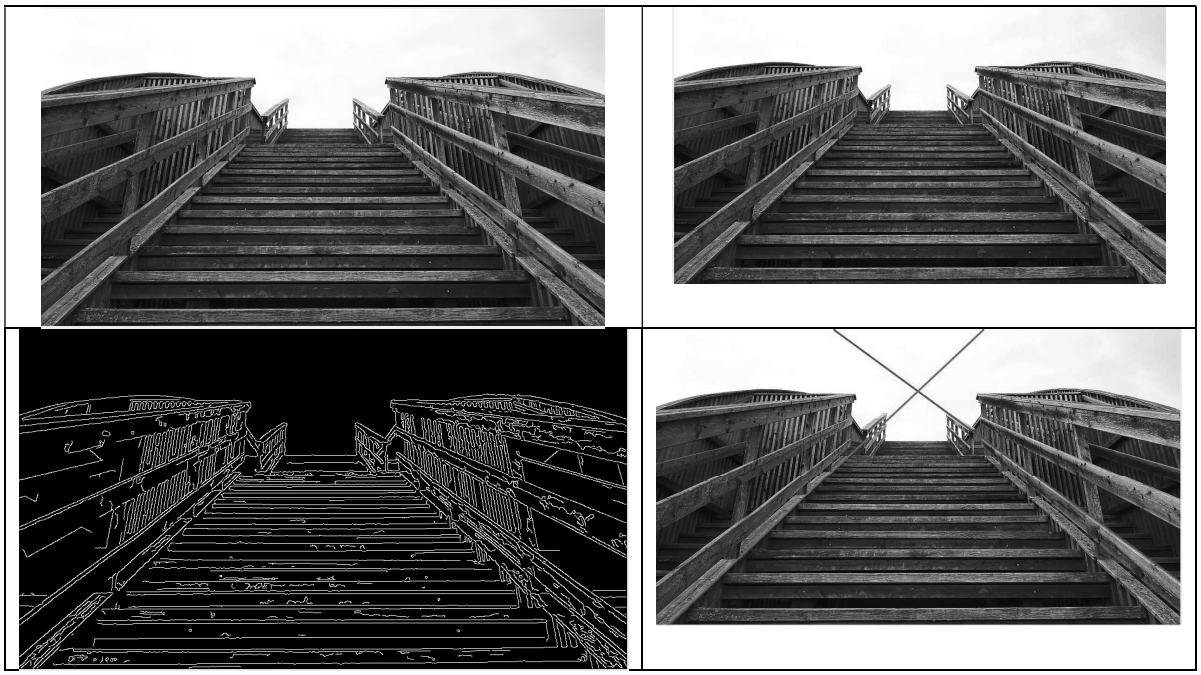


IMAGE 9 WITH THRESHOLD

0.2 FOR CANNY EDGE DETECTION

200 FOR VOTING MATRIX

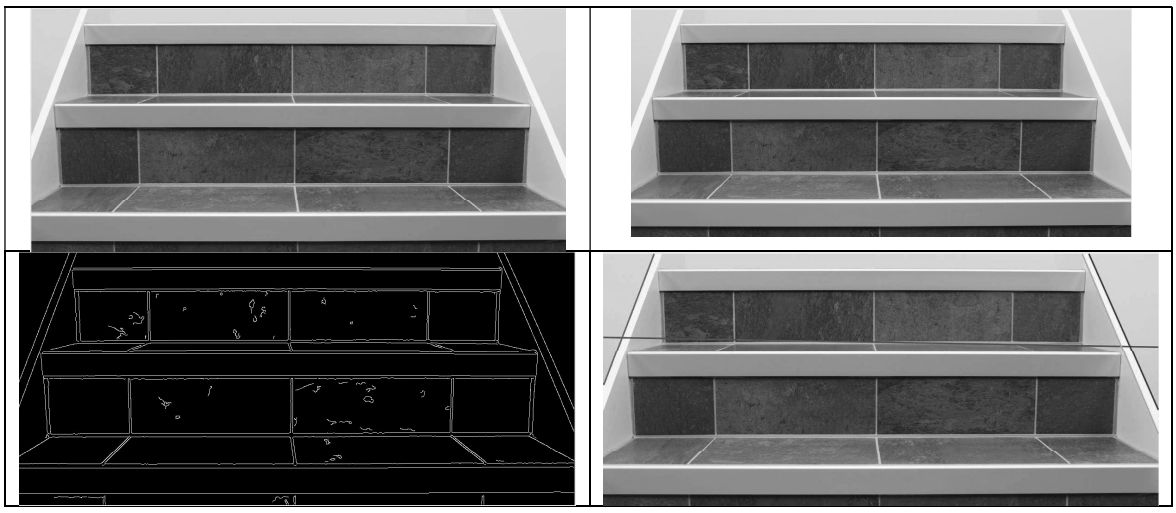
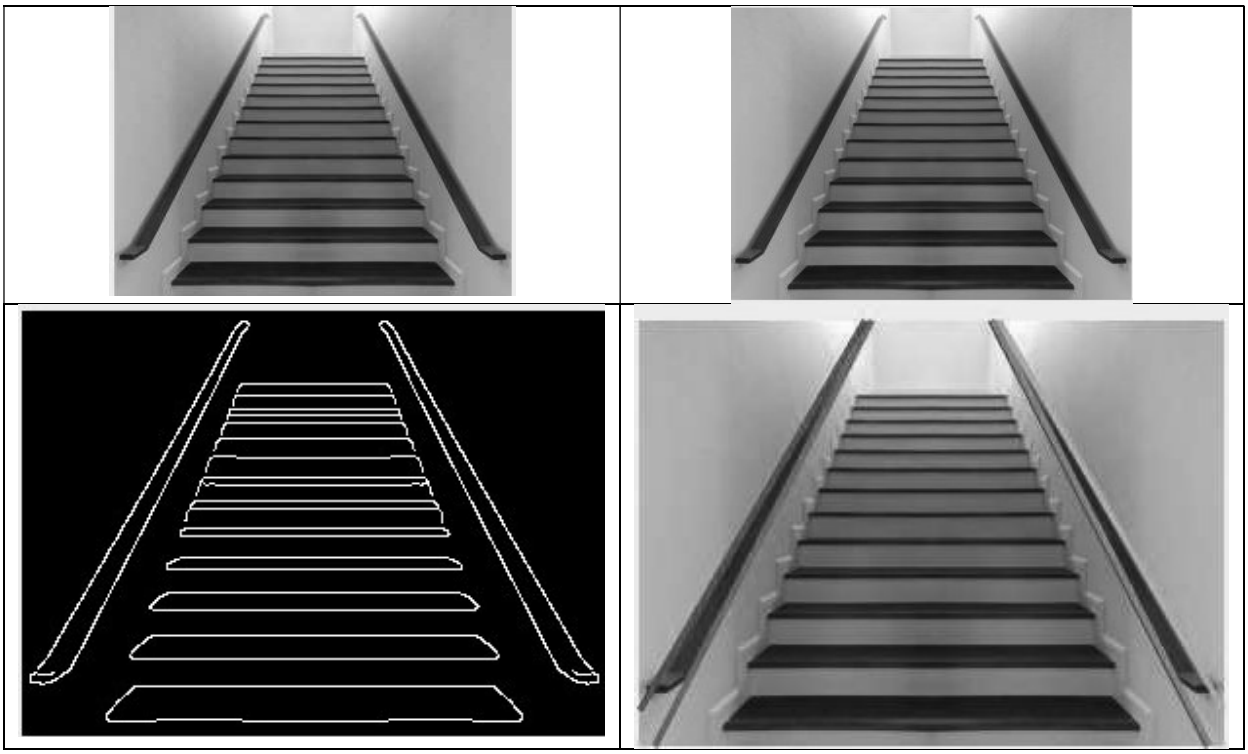


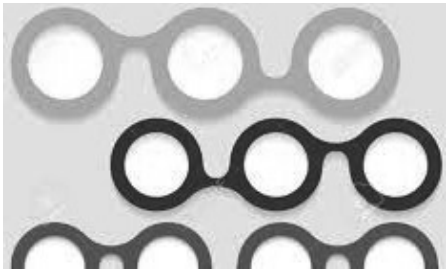
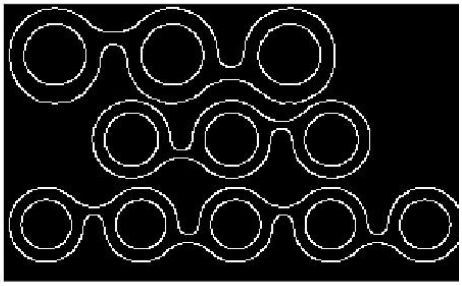
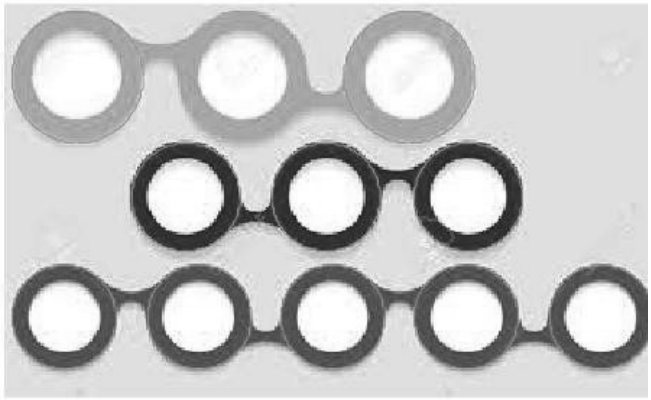
IMAGE 10 WITH THRESHOLD

0.5 FOR CANNY EDGE DETECTION

100 FOR VOTING MATRIX



CIRCLE DETECTION

IMAGE 1 MATLAB available line code	
	
	
OUR CODE	

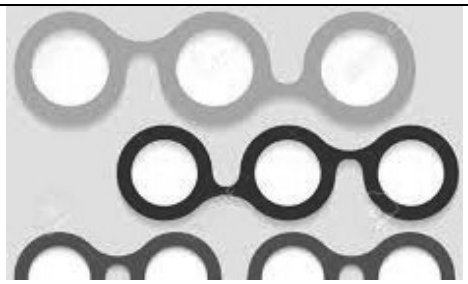
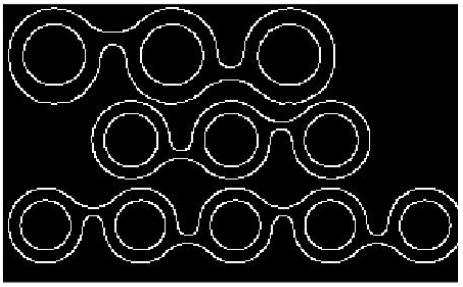
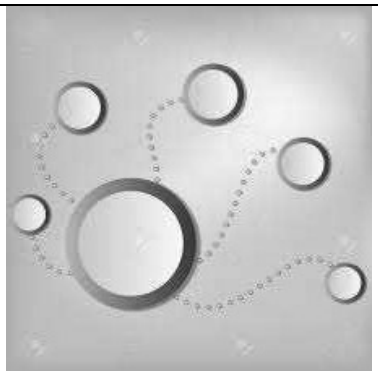
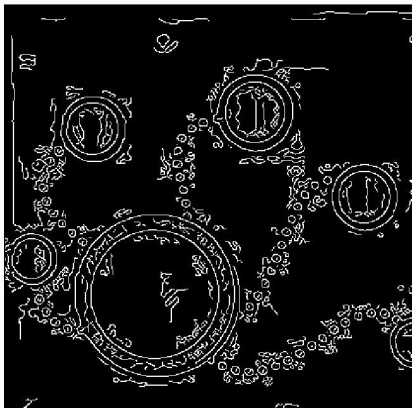
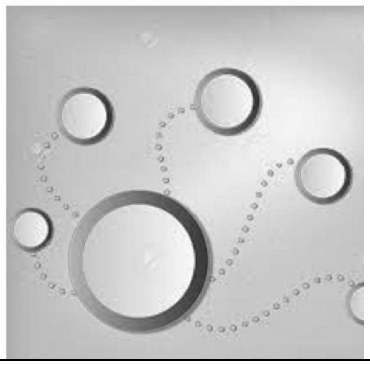
	

IMAGE 2

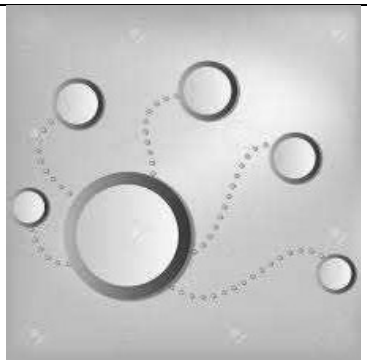
MATLAB available line code

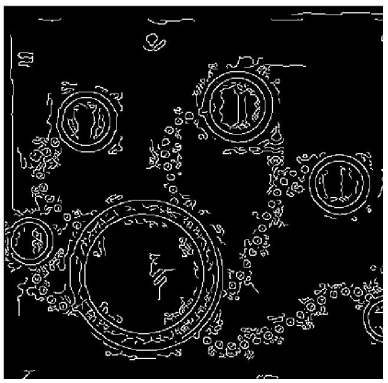






OUR CODE





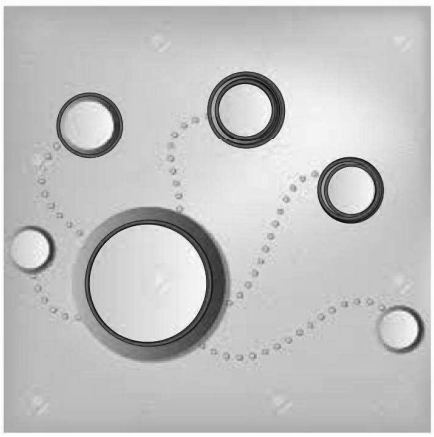
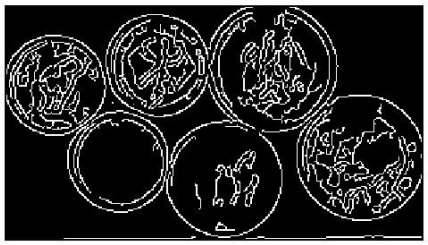


IMAGE 3

MATLAB available line code



OUR CODE

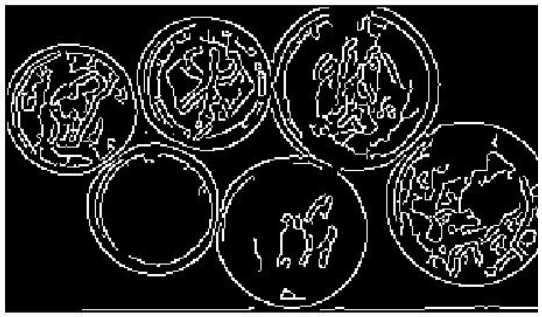
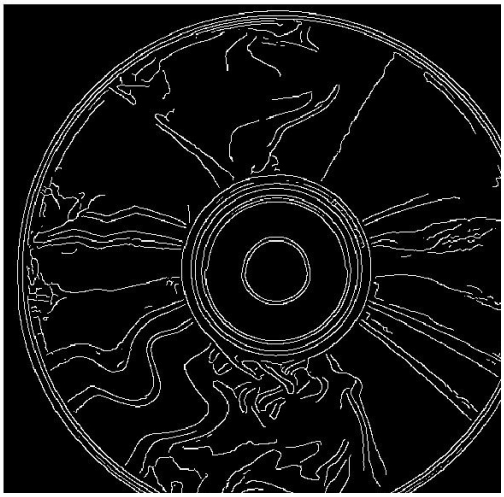


IMAGE 4

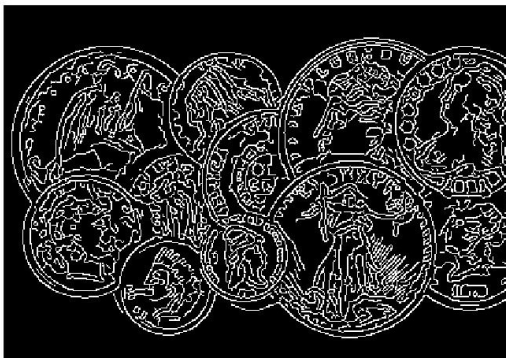
MATLAB available line code



OUR CODE

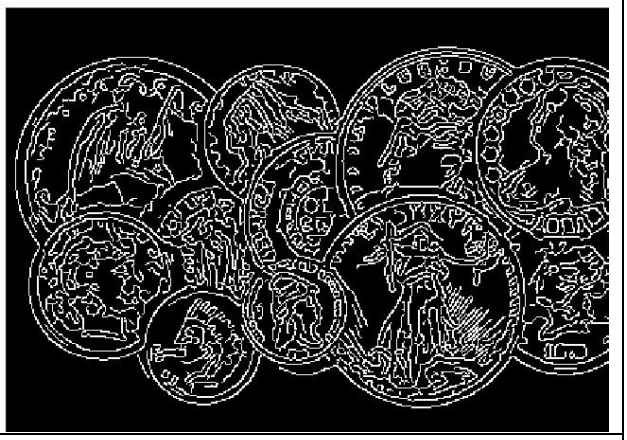
IMAGE 5

MATLAB available line code



OUR CODE

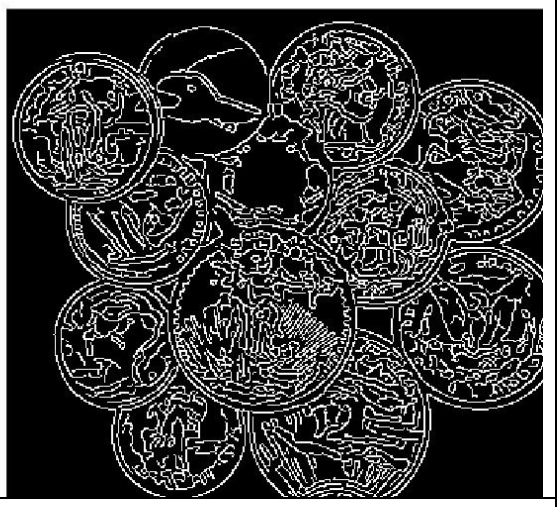

1821
1875




1821
1875

IMAGE 6

MATLAB available line code



OUR CODE

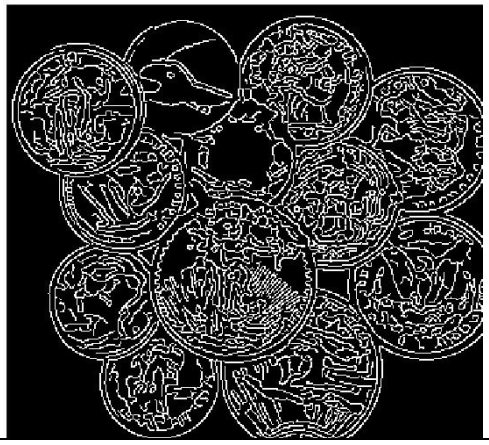
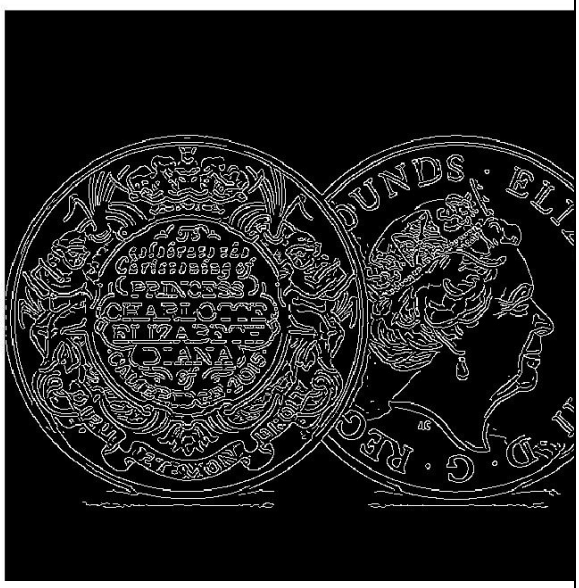


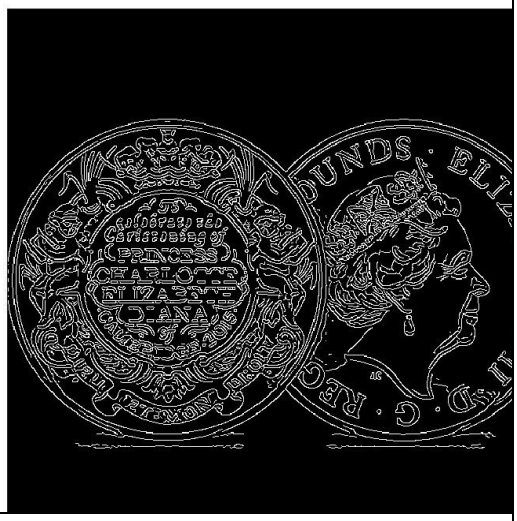
IMAGE 7

MATLAB available line code



OUR CODE

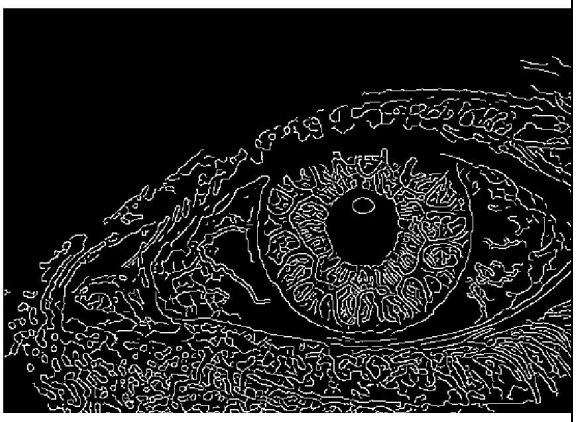

To celebrate the Christening of
PRINCESS
CHARLOTTE
ELIZABETH
DIANA
of
CAMBRIDGE
2015
DIEU ET MON DROIT
ELIZABETH II D·G·REG·F·D·
JC


To celebrated the Christening of
PRINCESS
CHARLOTTE
ELIZABETH
DIANA
of
CAMBRIDGE
2015
DIEU ET MON DROIT
JC


To celebrate the Christening of
PRINCESS
CHARLOTTE
ELIZABETH
DIANA
of
CAMBRIDGE
2015
DIEU ET MON DROIT
JC

IMAGE 8

MATLAB available line code



OUR CODE

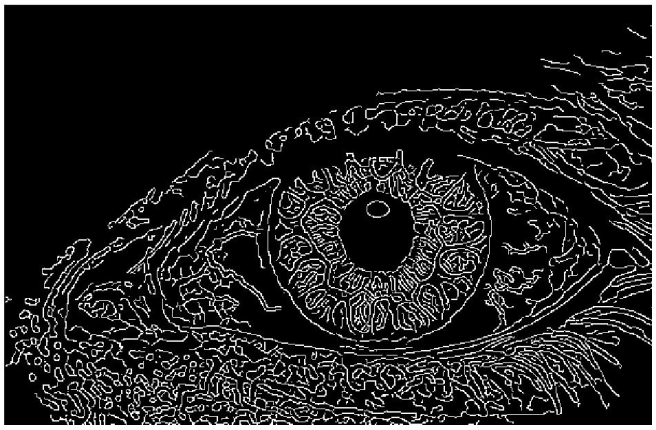


IMAGE 9

MATLAB available line code

OUR CODE

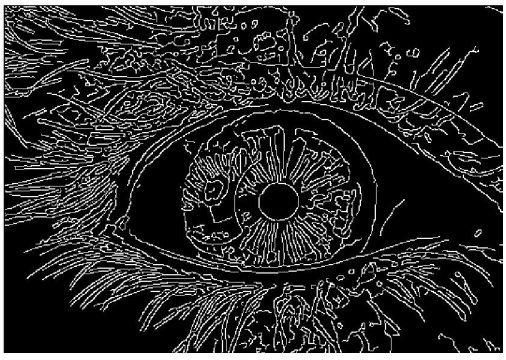
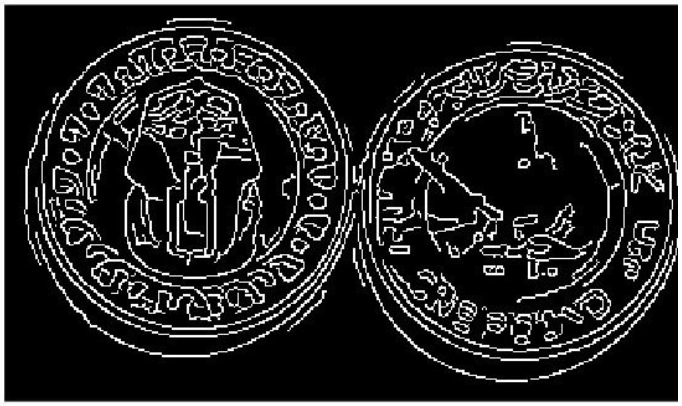


IMAGE 10

MATLAB available line code



OUR CODE

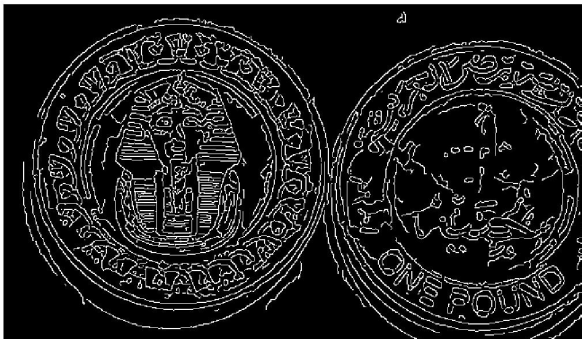


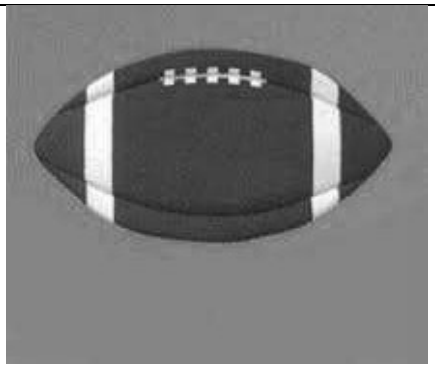
IMAGE 11

MATLAB available line code

OUR CODE

ELLIPSE DETECTION

IMAGE 1



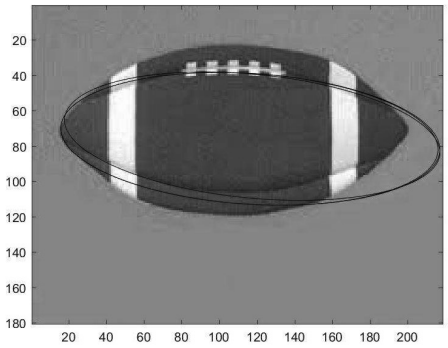


IMAGE 2

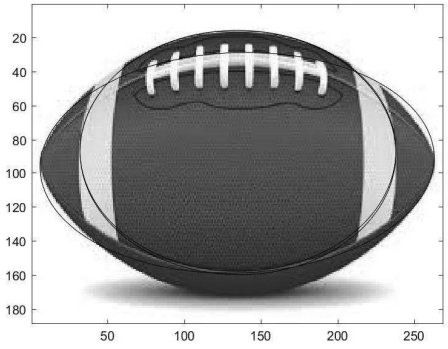


IMAGE 3

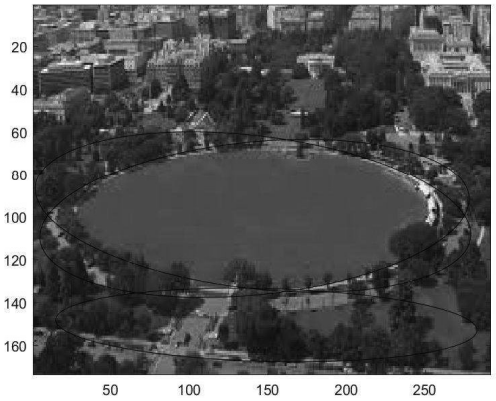


IMAGE 4

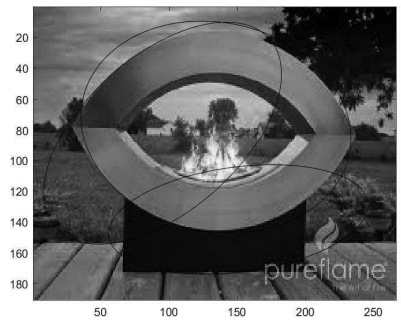
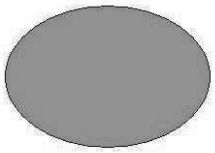


IMAGE 5

IMAGE 6



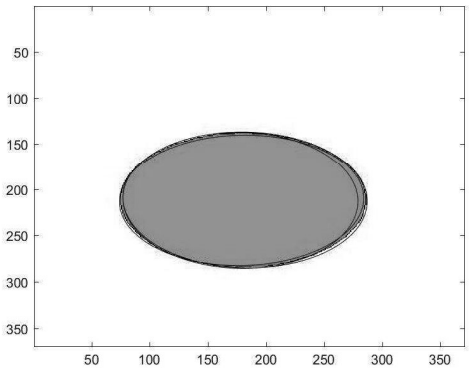


IMAGE 7

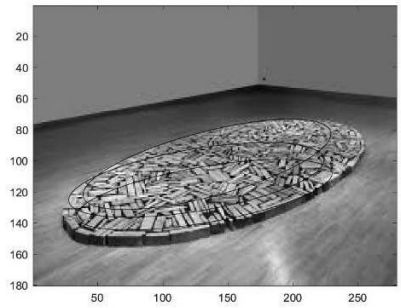
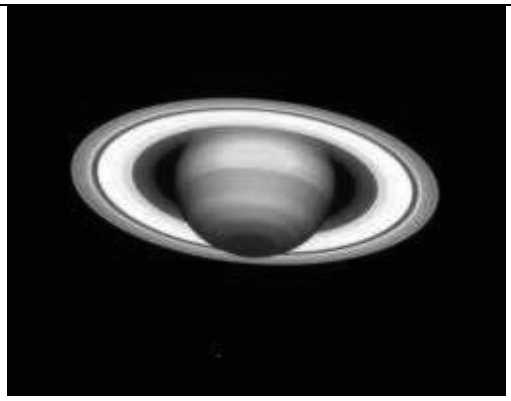


IMAGE 8



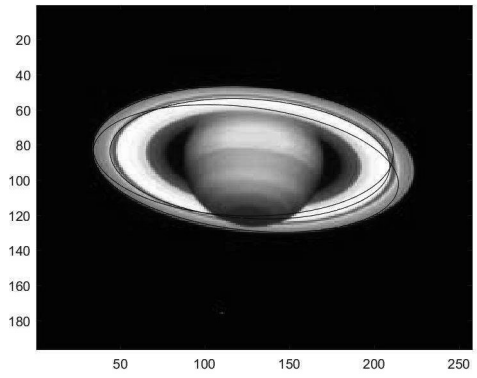


IMAGE 9

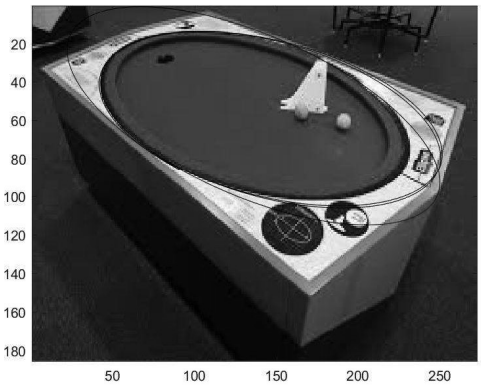
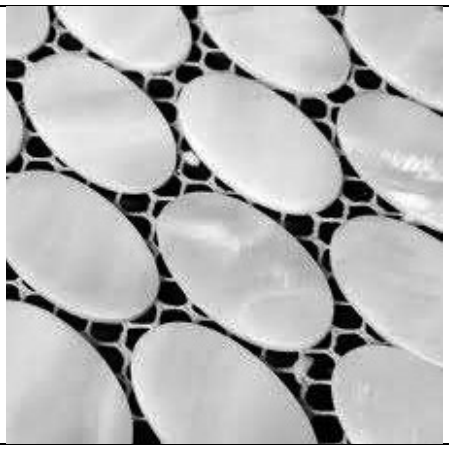
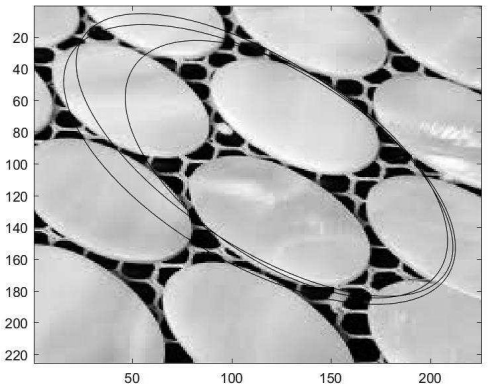


IMAGE 10




20
40
60
80
100
120
140
160
180
200
220
50
100
150
200

APPENDIX

1. Our MATLAB code for line detection

```
 %read the image
image = imread('2.jpg');
%image=imresize(image,0.5);
figure,imshow(image);
%convert the image to grey
gr=rgb2gray(image);
figure,imshow(gr);
%use canny for edge detection and  the threshold changes from
one image to another
img=edge(gr,'Canny',0.6);
figure,imshow(img);
[H,W]=size(img);
%get the maxrho which is the diagonal so as to get the range of
rho
maxrho=round(sqrt(H^2+W^2));
%specify theta and rho range
theta_range=1:180;
rho_range=-maxrho:1:maxrho;
%accumulator for voting matrix
acc=zeros(length(rho_range),length(theta_range));
%pass by each pt in the image to find if it is on the edge
detected by canny ..if so then calculate rho and theta of all
pts passing through this point and store it in the accumulator
matrix
for i=1:H
    for j=1:W
     if img(i,j)~=0
         for theta=1:180
                 distance=round(i*cosd(theta)+j*sind(theta));
                 acc_dis=(distance+maxrho)+1;
                 acc(acc_dis,theta)=acc(acc_dis,theta)+1;
         end
     end
    end
end
%specify the threshold which will be used to choose the local
max votings in the accumulator matrix and store those voting as
1 in another matrix named regmax
regmax=zeros(size(acc));
[p,q]=size(acc);
for d=1:p
    for e=1:q
        if acc(d,e)>85
```

```
            regmax(d,e)=1;
        end
    end
end
figure,imshow(image);
hold on
%pass by the pts in regmax matrix and if the pt=1 then see the
corresponding rho and theta at this pt and plot this line in
cartesian coordinates x&y
[x,y]=size(regmax);
for row=1:x
    for col=1:y
        if regmax(row,col)~=0
          r =row-maxrho-1;
          th=col;
            if th~=180
                    yt=1:W;
                    x0=(-sind(th)/cosd(th))*yt + (r /
cosd(th));
                    plot(yt,x0,'LineWidth',2);
            else
                line([r r], [1 H], 'LineWidth',1 );
            end
        end
    end
end
```

2. MATLAB available line detection code

```
I = imread('House.png');
%image=imresize(I,0.5);
Edge_Threshold = 0.2;
BW = edge(rgb2gray(I),'canny',Edge_Threshold);
figure,imshow(BW);

[H,theta,rho] = hough(BW);
figure
imshow(imadjust(mat2gray(H)),[],'XData',theta,'YData',rho,'Init
ialMagnification','fit');
xlabel('\theta (degrees)')
```

```
ylabel('\rho')
axis on
axis normal
hold on

P = houghpeaks(H,60,'Threshold',ceil(0.2*max(H(:))));
x = theta(P(:,2));
y = rho(P(:,1));
plot(x,y,'s','color','blue');

lines = houghlines(BW,theta,rho,P,'FillGap',15,'MinLength',35);
%Create a plot that displays the original image with the lines
%superimposed on it.
figure, imshow(I), hold on
max_len= 0;
for k = 1:length(lines)
xy = [lines(k).point1; lines(k).point2];
plot(xy(:,1),xy(:,2),'LineWidth',3,'Color','green');
% Plot beginnings and ends of lines
plot(xy(1,1),xy(1,2),'x','LineWidth',2,'Color','yellow');
plot(xy(2,1),xy(2,2),'x','LineWidth',2,'Color','red');
% Determine the endpoints of the longest line segment
len = norm(lines(k).point1 -lines(k).point2);
if ( len> max_len)
max_len= len;
xy_long= xy;
end
end
```

3. Our MATLAB function for circle detection

Circle.m

```
clear                                   %Clear the command Window
img = 'mine1.jpg';                      %Set the image name
A=imread(img);                          %Read the image file
scale = 0.25 ;                          %set the scale for image
resizing
L = imresize(A,scale);                  %Resize the image
N = 25;                                 %Set the number of maximum
peaks that I want
circleDetect2(L,img,[80 300],N);        %Detect and plot the
circles
```

circleDetect2.m

```
function circleDetect2(I,imageName,radii_range,N)  % Function
Declaration
```

```
ed = edge ( ( rgb2gray ( I ) ), 'canny');           %Edge
detection using canny and grayscale image
figure();
imshow(ed);
saveas(gcf,strcat( 'out_Edge_me_' , imageName));    %Display
the edge detection and save it

[sizex,sizey]=size(ed);                             %Get the
size of the edge image and create a voting matrix of zeros as
initial
voting=zeros(sizex,sizex,sizex);

%Itterate on the edge image to make the voting matrix
    for x=(1:sizex)
        for y=(1:sizey)
            if ed(x,y)== 1      %if there is and edge apply the
following
                for r=(radii_range(1):radii_range(2))+1
%for radii in the range
                    for theta= 0:180
%for theta within 0-180
                        a=round(x+r*cosd(theta));
%Calculate a and b
                        b=round(y+r*sind(theta));
                        if a>0 && a<sizex && b>0 && b<sizex
%if the a and b are within the image increment the specified
voting matrix with one
                            voting(a,b,r-radii_range(1)) =
(voting(a,b,r-radii_range(1))+1) ;
                        end

                    end
                end
            end
        end
    end
    [a,b,R] = Max3D(voting,N); %Call the function to get the
number of maximum peaks specified by N

    figure();
    imshow(I);          %Show the original image
    for p = 1:N         %Plot the circles on the image
        viscircles([b(p) a(p)],R(p)+radii_range(1));
    end
    saveas(gcf, strcat('out_me_' , imageName));
end
```

Max3D.m

```
function [ a,b,R ] = Max3D( voting,N )
```

```
max = voting (1,1,1);   %Set the initial value for max
a0 = 1;                 %Set the initial values for a0 and b0
and R0
b0 = 1;
R0 = 1;
[h,w,t] = size(voting);
for p = 1:N         %itterate to get the maximum
    for x = 2 : h
        for y = 2 : w
            for z = 2 : t
                if voting(x,y,z) > max
                    max = voting(x,y,z);
                    a0 = x;
                    b0 = y;
                    R0 = z;
                end
            end
        end
    end
    voting(a0,b0,R0)= 0; % delete the maximum and store it in
the output matrix
    max=voting (1,1,1);
    a(p)=a0;
    b(p)=b0;
    R(p)=R0;
end


end
```

4. MATLAB available circle detection code

circle_hough.m

```
function [h, margin] = circle_hough(b, rrange, varargin)
%CIRCLE_HOUGH Hough transform for circles
%   [H, MARGIN] = CIRCLE_HOUGH(B, RADII) takes a binary 2-D
image B and a
%   vector RADII giving the radii of circles to detect. It
returns the 3-D
%   accumulator array H, and an integer MARGIN such that
H(I,J,K) contains
%   the number of votes for the circle centred at B(I-MARGIN,
J-MARGIN),
%   with radius RADII(K). Circles which pass through B but
whose centres
%   are outside B receive votes.
%
%   [H, MARGIN] = CIRCLE_HOUGH(B, RADII, opt1, ...) allows
options to be
%   set. Each option is a string, which if included has the
following
%   effect:
%
%   'same' returns only the part of H corresponding to centre
positions
%   within the image. In this case H(:,:,k) has the same
dimensions as B,
%   and MARGIN is 0. This option should not be used if circles
whose
%   centres are outside the image are to be detected.
%
%   'normalise' multiplies each slice of H, H(:,:,K), by
1/RADII(K). This
%   may be useful because larger circles get more votes,
roughly in
%   proportion to their radius.
%
%   The spatial resolution of the accumulator is the same as
the spatial
%   resolution of the original image. Smoothing the accumulator
array
%   allows the effective resolution to be controlled, and this
is probably
%   essential for sensitivity to circles of arbitrary radius if
the spacing
%   between radii is greater than 1. If time or memory
requirements are a
%   problem, a generalisation of this function to allow larger
bins to be
%   used from the start would be worthwhile.
%
```

```
%   Each feature in B is allowed 1 vote for each circle. This
function
%   could easily be generalised to allow weighted features.
%
%   See also CIRCLEPOINTS, CIRCLE_HOUGHPEAKS, CIRCLE_HOUGHDEMO

% Copyright David Young 2008, 2010

% argument checking
opts = {'same' 'normalise'};
narginchk(2, 2+length(opts));
validateattributes(rrange, {'double'}, {'real' 'positive'
'vector'});
if ~all(ismember(varargin, opts))
    error('Unrecognised option');
end

% get indices of non-zero features of b
[featR, featC] = find(b);

% set up accumulator array - with a margin to avoid need for
bounds checking
[nr, nc] = size(b);
nradii = length(rrange);
margin = ceil(max(rrange));
nrh = nr + 2*margin;        % increase size of accumulator
nch = nc + 2*margin;
h = zeros(nrh*nch*nradii, 1, 'uint32');  % 1-D for now, uint32
a touch faster

% get templates for circles at all radii - these specify
accumulator
% elements to increment relative to a given feature
tempR = []; tempC = []; tempRad = [];
for i = 1:nradii
    [tR, tC] = circlepoints(rrange(i));
    tempR = [tempR tR]; %#ok<*AGROW>
    tempC = [tempC tC];
    tempRad = [tempRad repmat(i, 1, length(tR))];
end

% Convert offsets into linear indices into h - this is similar
to sub2ind.
% Take care to avoid negative elements in either of these so
can use
% uint32, which speeds up processing by a factor of more than 3
(in version
% 7.5.0.342)!
tempInd = uint32( tempR+margin + nrh*(tempC+margin) +
nrh*nch*(tempRad-1) );
```

```
featInd = uint32( featR' + nrh*(featC-1)' );

% Loop over features
for f = featInd
    % shift template to be centred on this feature
    incI = tempInd + f;
    % and update the accumulator
    h(incI) = h(incI) + 1;
end

% Reshape h, convert to double, and apply options
h = reshape(double(h), nrh, nch, nradii);

if ismember('same', varargin)
    h = h(1+margin:end-margin, 1+margin:end-margin, :);
    margin = 0;
end

if ismember('normalise', varargin)
    h = bsxfun(@rdivide, h, reshape(rrange, 1, 1,
length(rrange)));
end

end
```

circle houghpeaks.m

```
function peaks = circle_houghpeaks(h, radii, varargin)
%CIRCLE_HOUGHPEAKS finds peaks in the output of CIRCLE_HOUGH
%   PEAKS = CIRCLE_HOUGHPEAKS(H, RADII, MARGIN, OPTIONS)
locates the
%   positions of peaks in the output of CIRCLE_HOUGH. The
result PEAKS is a
%   3 x N array, where each column gives the position and
radius of a
%   possible circle in the original array. The first row of
PEAKS has the
%   x-coordinates, the second row has the y-coordinates, and
the third row
%   has the radii.
%
%   H is the 3D accumulator array returned by CIRCLE_HOUGH.
%
%   RADII is the array of radii which was passed as an argument
to
%   CIRCLE_HOUGH.
%
%   MARGIN is optional, and may be omitted if the 'same' option
was used
```

```
%   with CIRCLE_HOUGH. Otherwise, it should be the second
result returned
%   by CIRCLE_HOUGH.
%
%   OPTIONS is a comma-separated list of parameter/value pairs,
with the
%   following effects:
%
%   'Smoothxy' causes each x-y layer of H to be smoothed before
peak
%   detection using a 2D Gaussian kernel whose "sigma"
parameter is given
%   by the value of this argument.
%
%   'Smoothr' causes each radius column of H to be smoothed
before peak
%   detection using a 1D Gaussian kernel whose "sigma"
parameter is given
%   by the value of this argument.
%
%       Note: Smoothing may be useful to locate peaks in noisy
accumulator
%       arrays. However, it may also cause the performance to
deteriorate
%       if H contains sharp peaks. It is most likely to be
useful if
%       neighbourhood suppression (see below) is not used.
%
%       Both smoothing operations use reflecting boundary
conditions to
%       compute values close to the boundaries.
%
%   'Threshold' sets the minimum number of votes (after any
smoothing)
%   needed for a peak to be counted. The default is 0.5 * the
maximum value
%   in H.
%
%   'Npeaks' sets the maximum number of peaks to be found. The
highest
%   NPEAKS peaks are returned, unless the threshold causes
fewer than
%   NPEAKS peaks to be available.
%
%   'Nhoodxy' must be followed by an odd integer, which sets a
minimum
%   spatial separation between peaks. See below for a more
precise
%   statement. The default is 1.
%
```

```
%   'Nhoodr' must be followed by an odd integer, which sets a
minimum
%   separation in radius between peaks. See below for a more
precise
%   statement. The default is 1.
%
%       When a peak has been found, no other peak with a
position within an
%       NHOODXY x NHOODXY x NHOODR box centred on the first
peak will be
%       detected. Peaks are found sequentially; for example,
after the
%       highest peak has been found, the second will be found
at the
%       largest value in H excepting the exclusion box found
the first
%       peak. This is similar to the mechanism provided by the
Toolbox
%       function HOUGHPEAKS.
%
%       If both the 'Nhoodxy' and 'Nhoodr' options are omitted,
the effect
%       is not quite the same as setting each of them to 1.
Instead of a
%       sequential algorithm with repeated passes over H, the
Toolbox
%       function IMREGIONALMAX is used. This may produce
slightly different
%       results, since an above-threshold point adjacent to a
peak will
%       appear as an independent peak using the sequential
suppression
%       algorithm, but will not be a local maximum.
%
%   See also CIRCLE_HOUGH, CIRCLE_HOUGHDEMO

% check arguments
params = checkargs(h, radii, varargin{:});

% smooth the accumulator - xy
if params.smoothxy > 0
    [m, hsize] = gaussmask1d(params.smoothxy);
    % smooth each dimension separately, with reflection
    h = cat(1, h(hsize:-1:1,:,:), h, h(end:-1:end-
hsize+1,:,:));
    h = convn(h, reshape(m, length(m), 1, 1), 'valid');

    h = cat(2, h(:,hsize:-1:1,:), h, h(:,end:-1:end-
hsize+1,:));
    h = convn(h, reshape(m, 1, length(m), 1), 'valid');
```

```
end

% smooth the accumulator - r
if params.smoothr > 0
    [m, hsize] = gaussmask1d(params.smoothr);
    h = cat(3, h(:,:,hsize:-1:1), h, h(:,:,end:-1:end-
hsize+1));
    h = convn(h, reshape(m, 1, 1, length(m)), 'valid');
end

% set threshold
if isempty(params.threshold)
    params.threshold = 0.5 * max(h(:));
end

if isempty(params.nhoodxy) && isempty(params.nhoodr)
    % First approach to peak finding: local maxima

    % find the maxima
    maxarr = imregionalmax(h);

    maxarr = maxarr & h >= params.threshold;

    % get array indices
    peakind = find(maxarr);
    [y, x, rind] = ind2sub(size(h), peakind);
    peaks = [x'; y'; radii(rind)];

    % get strongest peaks
    if ~isempty(params.npeaks) && params.npeaks < size(peaks,2)
        [~, ind] = sort(h(peakind), 'descend');
        ind = ind(1:params.npeaks);
        peaks = peaks(:, ind);
    end

else
    % Second approach: iterative global max with suppression
    if isempty(params.nhoodxy)
        params.nhoodxy = 1;
    elseif isempty(params.nhoodr)
        params.nhoodr = 1;
    end
    nhood2 = ([params.nhoodxy params.nhoodxy params.nhoodr]-1)
/ 2;

    if isempty(params.npeaks)
        maxpks = 0;
        peaks = zeros(3, round(numel(h)/100));  % preallocate
    else
        maxpks = params.npeaks;
        peaks = zeros(3, maxpks);  % preallocate
```

```
    end

    np = 0;
    while true
        [r, c, k, v] = max3(h);
        % stop if peak height below threshold
        if v < params.threshold || v == 0
            break;
        end
        np = np + 1;
        peaks(:, np) = [c; r; radii(k)];
        % stop if done enough peaks
        if np == maxpks
            break;
        end
        % suppress this peak
        r0 = max([1 1 1], [r c k]-nhood2);
        r1 = min(size(h), [r c k]+nhood2);
        h(r0(1):r1(1), r0(2):r1(2), r0(3):r1(3)) = 0;
    end
    peaks(:, np+1:end) = [];   % trim
end

% adjust for margin
if params.margin > 0
    peaks([1 2], :) = peaks([1 2], :) - params.margin;
end
end

function params = checkargs(h, radii, varargin)
% Argument checking
ip = inputParser;

% required
htest = @(h) validateattributes(h, {'double'}, {'real'
'nonnegative' 'nonsparse'});
ip.addRequired('h', htest);
rtest = @(radii) validateattributes(radii, {'double'}, {'real'
'positive' 'vector'});
ip.addRequired('radii', rtest);

% optional
mtest = @(n) validateattributes(n, {'double'}, {'real'
'nonnegative' 'integer' 'scalar'});
ip.addOptional('margin', 0, mtest);

% parameter/value pairs
stest = @(s) validateattributes(s, {'double'}, {'real'
'nonnegative' 'scalar'});
ip.addParamValue('smoothxy', 0, stest);
```

```
ip.addParamValue('smoothr', 0, stest);
ip.addParamValue('threshold', [], stest);
nptest = @(n) validateattributes(n, {'double'}, {'real'
'positive' 'integer' 'scalar'});
ip.addParamValue('npeaks', [], nptest);
nhtest = @(n) validateattributes(n, {'double'}, {'odd'
'positive' 'scalar'});
ip.addParamValue('nhoodxy', [], nhtest);
ip.addParamValue('nhoodr', [], nhtest);
ip.parse(h, radii, varargin{:});
params = ip.Results;
end

function [m, hsize] = gaussmask1d(sigma)
% truncated 1D Gaussian mask
hsize = ceil(2.5*sigma);  % reasonable truncation
x = (-hsize:hsize) / (sqrt(2) * sigma);
m = exp(-x.^2);
m = m / sum(m);  % normalise
end

function [r, c, k, v] = max3(h)
% location and value of global maximum of a 3D array
[vr, r] = max(h);
[vc, c] = max(vr);
[v, k] = max(vc);
c = c(1, 1, k);
r = r(1, c, k);
end
```

circlepoints.m

```
function [x, y] = circlepoints(r)
%CIRCLEPOINTS  Returns integer points close to a circle
%   [X, Y] = CIRCLEPOINTS(R) where R is a scalar returns
coordinates of
%   integer points close to a circle of radius R, such that
none is
%   repeated and there are no gaps in the circle (under 8-
connectivity).
%
%   If R is a row vector, a circle is generated for each
element of R and
%   the points concatenated.

%   Copyright David Young 2010

x = [];
y = [];
```

```
for rad = r
    [xp, yp] = circlepoints1(rad);
    x = [x xp];
    y = [y yp];
end

end

function [x, y] = circlepoints1(r)
% Get number of rows needed to cover 1/8 of the circle
l = round(r/sqrt(2));
if round(sqrt(r.^2 - l.^2)) < l   % if crosses diagonal
    l = l-1;
end
% generate coords for 1/8 of the circle, a dot on each row
x0 = 0:l;
y0 = round(sqrt(r.^2 - x0.^2));
% Check for overlap
if y0(end) == l
    l2 = l;
else
    l2 = l+1;
end
% assemble first quadrant
x = [x0 y0(l2:-1:2)];
y = [y0 x0(l2:-1:2)];
% add next quadrant
x0 = [x y];
y0 = [y -x];
% assemble full circle
x = [x0 -x0];
y = [y0 -y0];
end
```

Circular Hough Transform Demonstration

David Young

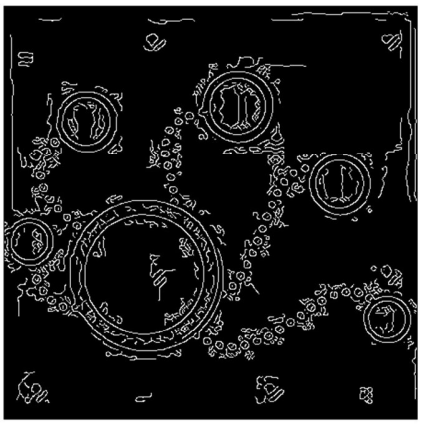
Demonstrates the use of `circle_hough` and `circle_houghpeaks` to find circular objects in an image.

Uses the Image Processing Toolbox

Setup

Reads an example image, gets its edges and displays them

```
A = imread('C2.jpg');
scale = 2 ;
B = imresize(A,scale);
im = rgb2gray(B);
e = edge(im, 'canny');
imshow(e);
saveas(gcf, 'out_C2_Edge.jpg'); %save the image file
```



Carry out the HT

The circles round the coins have radii in the 20-30 pixels range. To make sure we cover the range, we search radii from 15 to 40 pixels, in steps of 1 pixel.

We select the 'same' option to simplify later processing, and the 'normalise' option to avoid a bias towards finding larger circles.

```
radii = 20:1:100;
h = circle_hough(e, radii, 'same', 'normalise');
```

Find some peaks in the accumulator

We use the neighbourhood-suppression method of peak finding to ensure that we find spatially separated circles. We select the 10 most prominent peaks, because as it happens we can see that there are 10 coins to find.

```
peaks = circle_houghpeaks(h, radii, 'nhoodxy', 15, 'nhoodr', 21, 'npeaks', 10);
```

Look at the results

We draw the circles found on the image, using both the positions and the radii stored in the `peaks` array. The `circlepoints` function is convenient for this - it is also used by `circle_hough` so comes with it.

```
imshow(B);
hold on;
for peak = peaks
[x, y] = circlepoints(peak(3));
plot(x+peak(1), y+peak(2), 'g-');
end
saveas(gcf, 'out_C2.jpg'); %save the image file
hold off
```

5. Open source code for ellipse detection

Main File

```
I = imread('E10.jpg');           %Read Image file
E = edge(rgb2gray(I),'canny');  %Get edge using canny
edge detection
%E = edge(I,'canny');

% override some default parameters
params.minMajorAxis = 20;
params.maxMajorAxis = 400;
% note that the edge (or gradient) image is used
bestFits = ellipseDetection(E, params); %detect ellipse
using ellipseDetection function
fprintf('Output %d best fits.\n', size(bestFits,1));
figure;
image(I);
%ellipse drawing implementation:
http://www.mathworks.com/matlabcentral/fileexchange/289
ellipse(bestFits(:,3),bestFits(:,4),bestFits(:,5)*pi/18
0,bestFits(:,1),bestFits(:,2),'k'); %Draw the Ellipse
saveas(gcf, 'out_E10.jpg'); %save the image file
```

```
Ellipse Detection Function
function bestFits = ellipseDetection(img, params)
% ellipseDetection: Ellipse detection
%
% Overview:
% --------
% Fits an ellipse by examining all possible major axes
(all pairs of points) and
% getting the minor axis using Hough transform. The
algorithm complexity depends on
% the number of valid non-zero points, therefore it is
beneficial to provide as many
% restrictions in the "params" input arguments as
possible if there is any prior
```

```
% knowledge about the problem.
%
% The code is reasonably fast due to (optional)
randomization and full code vectorization.
% However, as the algorithm needs to compute pairwise
point distances, it can be quite memory
% intensive. If you get out of memory errors, either
downsample the input image or somehow
% decrease the number of non-zero points in it.
% It can deal with big amount of noise but can have
severe problem with occlusions (major axis
% end points need to be visible)
%
% Input arguments:
% --------
% img
%   - One-channel input image (greyscale or binary).
% params
%   - Parameters of the algorithm:
%       minMajorAxis: Minimal length of major axis
accepted.
%       maxMajorAxis: Maximal length of major axis
accepted.
%        rotation, rotationSpan: Specification of
restriction on the angle of the major axis in degrees.
%                                If rotationSpan is in
(0,90), only angles within [rotation-rotationSpan,
%                               rotation+rotationSpan]
are accepted.
%       minAspectRatio: Minimal aspect ratio of an
ellipse (in (0,1))
%       randomize: Subsampling of all possible point
pairs. Instead of examining all N*N pairs, runs
%                  only on N*randomize pairs. If 0,
randomization is turned off.
%       numBest: Top numBest to return
%       uniformWeights: Used to prefer some points over
others. If false, accumulator points are weighted
%                       by their grey intensity in the
image. If true, the input image is regarded as binary.
%       smoothStddev: In order to provide more
stability of the solution, the accumulator is convolved
with
```

```
%                       a gaussian kernel. This parameter
specifies its standard deviation in pixels.
%
% Return value:
% --------
% Returns a matrix of best fits. Each row (there are
params.numBest of them) contains six elements:
% [x0 y0 a b alpha score] being the center of the
ellipse, its major and minor axis, its angle in degrees
and score.
%
% Based on:
% --------
% - "A New Efficient Ellipse Detection Method"
(Yonghong Xie Qiang , Qiang Ji / 2002)
% - random subsampling inspired by "Randomized Hough
Transform for Ellipse Detection with Result Clustering"
%   (CA Basca, M Talos, R Brad / 2005)
%
% Update log:
% --------
% 1.1: More memory efficient code, better
documentation, more parameters, more solutions
possible, example code.
% 1.0: Initial version
%
%
% Author: Martin Simonovsky
% e-mail: <mys007@seznam.cz>
% Release: 1.1
% Release date: 25.7.2013
%
%
% --------
%
% Redistribution and use in source and binary forms,
with or without
% modification, are permitted provided that the
following conditions are
% met:
%
%     * Redistributions of source code must retain the
above copyright
```

```
%       notice, this list of conditions and the
following disclaimer.
%     * Redistributions in binary form must reproduce
the above copyright
%       notice, this list of conditions and the
following disclaimer in
%       the documentation and/or other materials
provided with the distribution
%
% THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS
AND CONTRIBUTORS "AS IS"
% AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT
NOT LIMITED TO, THE
% IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
A PARTICULAR PURPOSE
% ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER
OR CONTRIBUTORS BE
% LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
EXEMPLARY, OR
% CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
PROCUREMENT OF
% SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
PROFITS; OR BUSINESS
% INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
LIABILITY, WHETHER IN
% CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
NEGLIGENCE OR OTHERWISE)
% ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE,
EVEN IF ADVISED OF THE
% POSSIBILITY OF SUCH DAMAGE.

    % default values
    if nargin==1; params=[]; end
    % - parameters to contrain the search
    if ~isfield(params,'minMajorAxis');
params.minMajorAxis = 10; end
    if ~isfield(params,'maxMajorAxis');
params.maxMajorAxis = 200; end
    if ~isfield(params,'rotation');
params.rotation = 0; end
    if ~isfield(params,'rotationSpan');
params.rotationSpan = 0; end
    if ~isfield(params,'minAspectRatio');
params.minAspectRatio = 0.1; end
```

```
    if ~isfield(params,'randomize');
params.randomize = 2; end
    % - others
    if ~isfield(params,'numBest');
params.numBest = 3; end
    if ~isfield(params,'uniformWeights');
params.uniformWeights = true; end
    if ~isfield(params,'smoothStddev');
params.smoothStddev = 1; end

    eps = 0.0001;
    bestFits = zeros(params.numBest,6);
    params.rotationSpan = min(params.rotationSpan, 90);
    H = fspecial('gaussian', [params.smoothStddev*6 1],
params.smoothStddev);

    [Y,X]=find(img);
    Y = single(Y); X = single(X);
    N = length(Y);

    fprintf('Possible major axes: %d * %d = %d\n', N,
N, N*N);

    % compute pairwise distances between points (memory
intensive!) and filter
    % TODO: do this block-wise, just appending the
filtered results (I,J)
    distsSq = bsxfun(@minus,X,X').^2 +
bsxfun(@minus,Y,Y').^2;
    [I,J] = find(distsSq>=params.minMajorAxis^2 &
distsSq<=params.maxMajorAxis^2);
    idx = I<J;
    I = uint32(I(idx)); J = uint32(J(idx));

    fprintf('..after distance constraint: %d\n',
length(I));

    % compute pairwise angles and filter
    if params.rotationSpan>0
        tangents = (Y(I)-Y(J)) ./ (X(I)-X(J));
        tanLo = tand(params.rotation-
params.rotationSpan);
        tanHi =
tand(params.rotation+params.rotationSpan);
```

```
        if tanLo<tanHi
            idx = tangents > tanLo & tangents < tanHi;
        else
            idx = tangents > tanLo | tangents < tanHi;
        end
        I = I(idx); J = J(idx);
        fprintf('..after angular constraint: %d\n',
length(I));
    else
        fprintf('..angular constraint not used\n');
    end

    npairs = length(I);

    % compute random choice and filter
    if params.randomize>0
        perm = randperm(npairs);
        pairSubset =
perm(1:min(npairs,N*params.randomize));
        clear perm;
        fprintf('..after randomization: %d\n',
length(pairSubset));
    else
        pairSubset = 1:npairs;
    end

    % check out all hypotheses
    for p=pairSubset
        x1=X(I(p)); y1=Y(I(p));
        x2=X(J(p)); y2=Y(J(p));

        %compute center & major axis
        x0=(x1+x2)/2; y0=(y1+y2)/2;
        aSq = distsSq(I(p),J(p))/4;
        thirdPtDistsSq = (X-x0).^2 + (Y-y0).^2;
        K = thirdPtDistsSq <= aSq; % (otherwise the
formulae in paper do not work)

        %get minor ax propositions for all other points
        fSq = (X(K)-x2).^2 + (Y(K)-y2).^2;
        cosTau = (aSq + thirdPtDistsSq(K) - fSq) ./
(2*sqrt(aSq*thirdPtDistsSq(K)));
        cosTau = min(1,max(-1,cosTau)); %inexact float
arithmetic?!
```

```
        sinTauSq = 1 - cosTau.^2;
        b = sqrt( (aSq * thirdPtDistsSq(K) .* sinTauSq)
./ (aSq - thirdPtDistsSq(K) .* cosTau.^2 + eps) );

        %proper bins for b
        idxs = ceil(b+eps);

        if params.uniformWeights
            weights = 1;
        else
            weights =
img(sub2ind(size(img),Y(K),X(K)));
        end
        accumulator = accumarray(idxs, weights,
[params.maxMajorAxis 1]);

        %a bit of smoothing and finding the most busy
bin
        accumulator = conv(accumulator,H,'same');

accumulator(1:ceil(sqrt(aSq)*params.minAspectRatio)) =
0;
        [score, idx] = max(accumulator);

        %keeping only the params.numBest best
hypothesis (no non-maxima suppresion)
        if (bestFits(end,end) < score)
            bestFits(end,:) = [x0 y0 sqrt(aSq) idx
atand((y1-y2)/(x1-x2)) score];
            if params.numBest>1
                [~,si]=sort(bestFits(:,end),'descend');
                bestFits = bestFits(si,:);
            end
        end
    end
end


Ellipse.m (the plotting depends on it)
function h=ellipse(ra,rb,ang,x0,y0,C,Nb)
% Ellipse adds ellipses to the current plot
%
% ELLIPSE(ra,rb,ang,x0,y0) adds an ellipse with
semimajor axis of ra,
```

```
% a semimajor axis of radius rb, a semimajor axis of
ang, centered at
% the point x0,y0.
%
% The length of ra, rb, and ang should be the same.
% If ra is a vector of length L and x0,y0 scalars, L
ellipses
% are added at point x0,y0.
% If ra is a scalar and x0,y0 vectors of length M, M
ellipse are with the same
% radii are added at the points x0,y0.
% If ra, x0, y0 are vectors of the same length L=M, M
ellipses are added.
% If ra is a vector of length L and x0, y0 are  vectors
of length
% M~=L, L*M ellipses are added, at each point x0,y0, L
ellipses of radius ra.
%
% ELLIPSE(ra,rb,ang,x0,y0,C)
% adds ellipses of color C. C may be a string
('r','b',...) or the RGB value.
% If no color is specified, it makes automatic use of
the colors specified by
% the axes ColorOrder property. For several circles C
may be a vector.
%
% ELLIPSE(ra,rb,ang,x0,y0,C,Nb), Nb specifies the
number of points
% used to draw the ellipse. The default value is 300.
Nb may be used
% for each ellipse individually.
%
% h=ELLIPSE(...) returns the handles to the ellipses.
%
% as a sample of how ellipse works, the following
produces a red ellipse
% tipped up at a 45 deg axis from the x axis
% ellipse(1,2,pi/8,1,1,'r')
%
% note that if ra=rb, ELLIPSE plots a circle
%

% written by D.G. Long, Brigham Young University, based
on the
```

```
% CIRCLES.m original
% written by Peter Blattner, Institute of
Microtechnology, University of
% Neuchatel, Switzerland, blattner@imt.unine.ch


% Check the number of input arguments

if nargin<1,
  ra=[];
end;
if nargin<2,
  rb=[];
end;
if nargin<3,
  ang=[];
end;

%if nargin==1,
%  error('Not enough arguments');
%end;

if nargin<5,
  x0=[];
  y0=[];
end;

if nargin<6,
  C=[];
end

if nargin<7,
  Nb=[];
end

% set up the default values

if isempty(ra),ra=1;end;
if isempty(rb),rb=1;end;
if isempty(ang),ang=0;end;
if isempty(x0),x0=0;end;
if isempty(y0),y0=0;end;
if isempty(Nb),Nb=300;end;
if isempty(C),C=get(gca,'colororder');end;
```

```
% work on the variable sizes

x0=x0(:);
y0=y0(:);
ra=ra(:);
rb=rb(:);
ang=ang(:);
Nb=Nb(:);

if isstr(C),C=C(:);end;

if length(ra)~=length(rb),
  error('length(ra)~=length(rb)');
end;
if length(x0)~=length(y0),
  error('length(x0)~=length(y0)');
end;

% how many inscribed elllipses are plotted

if length(ra)~=length(x0)
  maxk=length(ra)*length(x0);
else
  maxk=length(ra);
end;

% drawing loop

for k=1:maxk

  if length(x0)==1
    xpos=x0;
    ypos=y0;
    radm=ra(k);
    radn=rb(k);
    if length(ang)==1
      an=ang;
    else
      an=ang(k);
    end;
  elseif length(ra)==1
    xpos=x0(k);
    ypos=y0(k);
```

```
    radm=ra;
    radn=rb;
    an=ang;
  elseif length(x0)==length(ra)
    xpos=x0(k);
    ypos=y0(k);
    radm=ra(k);
    radn=rb(k);
    an=ang(k)
  else
    rada=ra(fix((k-1)/size(x0,1))+1);
    radb=rb(fix((k-1)/size(x0,1))+1);
    an=ang(fix((k-1)/size(x0,1))+1);
    xpos=x0(rem(k-1,size(x0,1))+1);
    ypos=y0(rem(k-1,size(y0,1))+1);
  end;

  co=cos(an);
  si=sin(an);
  the=linspace(0,2*pi,Nb(rem(k-1,size(Nb,1))+1,:)+1);
%  x=radm*cos(the)*co-si*radn*sin(the)+xpos;
%  y=radm*cos(the)*si+co*radn*sin(the)+ypos;
  h(k)=line(radm*cos(the)*co-
si*radn*sin(the)+xpos,radm*cos(the)*si+co*radn*sin(the)
+ypos);
  set(h(k),'color',C(rem(k-1,size(C,1))+1,:));

end;
```