Implementation of various Computer Vision algorithms

Submitted by: **Pranav Kakkar**
401703019
Mentored by: **Dr. Shailendra Tiwari**

ACKNOWLEDGEMENTS

This research was technically supported by my respected professor of Computer Vision **Dr. Shailendra Tiwari.**

Edge Detection:

Canny Filter:-

Code:

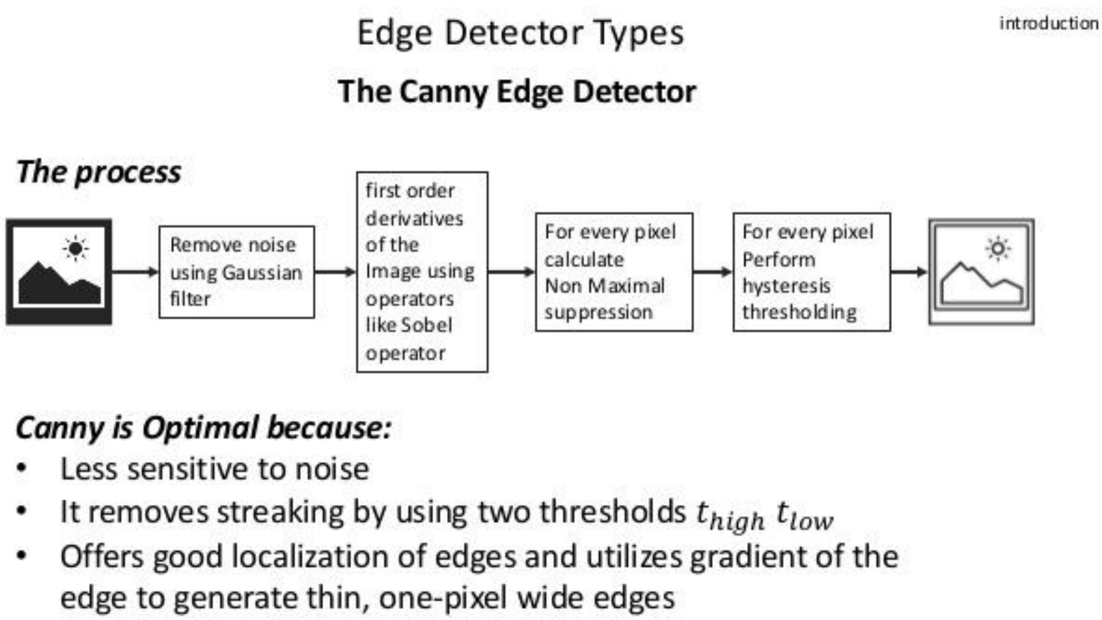
```
clear all;
clc;
%Input image
img = imread ('canny.jpg');
%Show input image
figure, imshow(img);
img = rgb2gray(img);
img = double (img);
%Value for Thresholding
T_Low = 0.075;
T_High = 0.175;
%Gaussian Filter Coefficient
B = [2, 4, 5, 4, 2; 4, 9, 12, 9, 4;5, 12, 15, 12, 5;4, 9,
12, 9, 4;2, 4, 5, 4, 2 ];
B = 1/159.* B;
%Convolution of image by Gaussian Coefficient
A=conv2(img, B, 'same');
%Filter for horizontal and vertical direction
```

```
KGx = [-1, 0, 1; -2, 0, 2; -1, 0, 1];
KGy = [1, 2, 1; 0, 0, 0; -1, -2, -1];
%Convolution by image by horizontal and vertical filter
Filtered_X = conv2(A, KGx, 'same');
Filtered_Y = conv2(A, KGy, 'same');
%Calculate directions/orientations
arah = atan2 (Filtered_Y, Filtered_X);
arah = arah*180/pi;
pan=size(A,1);
leb=size(A,2);
%Adjustment for negative directions, making all
directions positive
for i=1:pan
    for j=1:leb
        if (arah(i,j)<0)
            arah(i,j)=360+arah(i,j);
        end;
    end;
end;
arah2=zeros(pan, leb);
%Adjusting directions to nearest 0, 45, 90, or 135 degree
for i = 1  : pan
    for j = 1 : leb
        if ((arah(i, j) >= 0 ) && (arah(i, j) < 22.5) ||
(arah(i, j) >= 157.5) && (arah(i, j) < 202.5) || (arah(i,
j) >= 337.5) && (arah(i, j) <= 360))
            arah2(i, j) = 0;
        elseif ((arah(i, j) >= 22.5) && (arah(i, j) <
67.5) || (arah(i, j) >= 202.5) && (arah(i, j) < 247.5))
            arah2(i, j) = 45;
        elseif ((arah(i, j) >= 67.5 && arah(i, j) <
112.5) || (arah(i, j) >= 247.5 && arah(i, j) < 292.5))
            arah2(i, j) = 90;
        elseif ((arah(i, j) >= 112.5 && arah(i, j) <
157.5) || (arah(i, j) >= 292.5 && arah(i, j) < 337.5))
            arah2(i, j) = 135;
        end;
    end;
end;
figure, imagesc(arah2); colorbar;
%Calculate magnitude
magnitude = (Filtered_X.^2) + (Filtered_Y.^2);
magnitude2 = sqrt(magnitude);
BW = zeros (pan, leb);
%Non-Maximum Supression
for i=2:pan-1
    for j=2:leb-1
```

```
        if (arah2(i,j)==0)
            BW(i,j) = (magnitude2(i,j) ==
max([magnitude2(i,j), magnitude2(i,j+1), magnitude2(i,j-
1)]));
        elseif (arah2(i,j)==45)
            BW(i,j) = (magnitude2(i,j) ==
max([magnitude2(i,j), magnitude2(i+1,j-1), magnitude2(i-
1,j+1)]));
        elseif (arah2(i,j)==90)
            BW(i,j) = (magnitude2(i,j) ==
max([magnitude2(i,j), magnitude2(i+1,j), magnitude2(i-
1,j)]));
        elseif (arah2(i,j)==135)
            BW(i,j) = (magnitude2(i,j) ==
max([magnitude2(i,j), magnitude2(i+1,j+1), magnitude2(i-
1,j-1)]));
        end;
    end;
end;
BW = BW.*magnitude2;
figure, imshow(BW);
%Hysteresis Thresholding
T_Low = T_Low * max(max(BW));
T_High = T_High * max(max(BW));
T_res = zeros (pan, leb);
for i = 1  : pan
    for j = 1 : leb
        if (BW(i, j) < T_Low)
            T_res(i, j) = 0;
        elseif (BW(i, j) > T_High)
            T_res(i, j) = 1;
        %Using 8-connected components
        elseif ( BW(i+1,j)>T_High || BW(i-1,j)>T_High ||
BW(i,j+1)>T_High || BW(i,j-1)>T_High || BW(i-1, j-
1)>T_High || BW(i-1, j+1)>T_High || BW(i+1, j+1)>T_High
|| BW(i+1, j-1)>T_High)
            T_res(i,j) = 1;
        end;
    end;
end;
edge_final = uint8(T_res.*255);
%Show final edge detection result
figure, imshow(edge_final);
```

OUTPUT:

ALGORITHM:



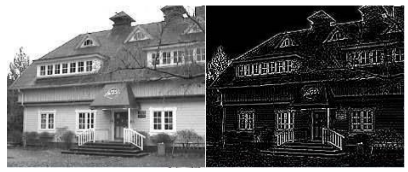
LAPLACIAN OF GAUSSIAN:

CODE:

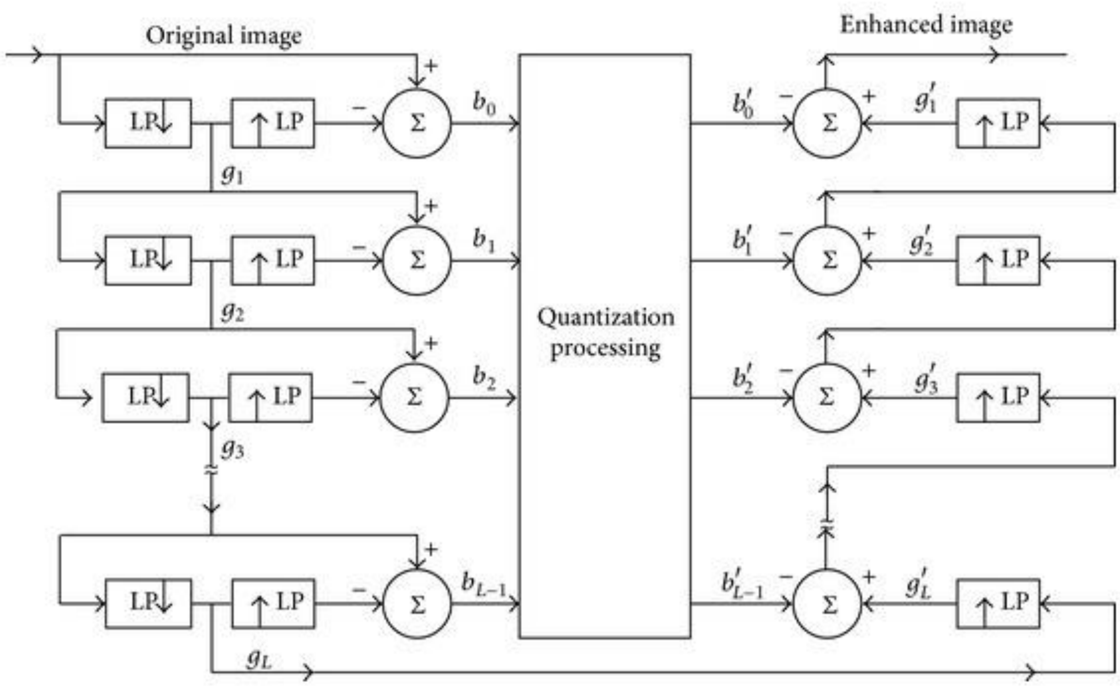
```
x=imread('canny.jpg');
Iblur = imgaussfilt(x,2);
i=rgb2gray(x);
[r,c,d]=size(i);
i=im2double(i);
for m=2:r-1
    for n=2:c-1
        for o=1:d
            a(m,n,o)=(0)*i(m-1,n-1,o)+(-1)*i(m-
1,n,o)+(0)*i(m-1,n+1,o)+(-1)*i(m,n-1,o)+(4)*i(m,n,o)+(-
1)*i(m,n+1,o)+(0)*i(m+1,n-1,o)+(-
1)*i(m+1,n,o)+(0)*i(m,n+1,o);

        end
    end
```

```
end
imshow(a);
```

OUTPUT:



ALGORITHM:



DIFFERENCE OF GAUSSIAN

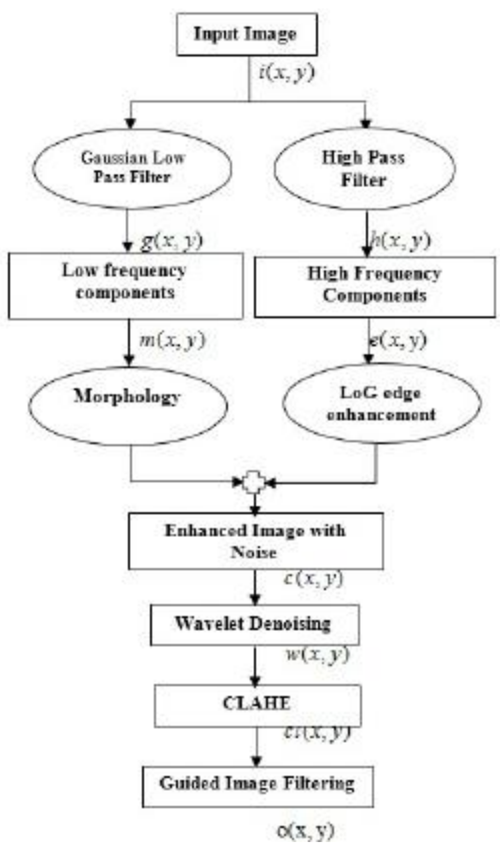
CODE:

```
i=imread('canny.jpg');

grayImage=rgb2gray(i);

gaussian1 = fspecial('Gaussian', 21, 15);

gaussian2 = fspecial('Gaussian', 21, 20);

dog = gaussian1 - gaussian2;

dogFilterImage = conv2(double(grayImage), dog, 'same');
```

```
figure;

imshow(dogFilterImage);
```

OUTPUT:

ALGORITHM:

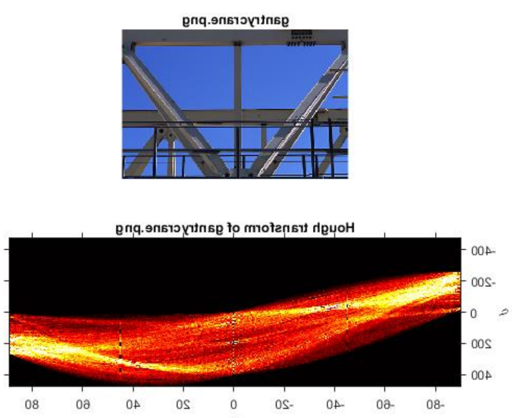


HOUGH TRANSFORM:

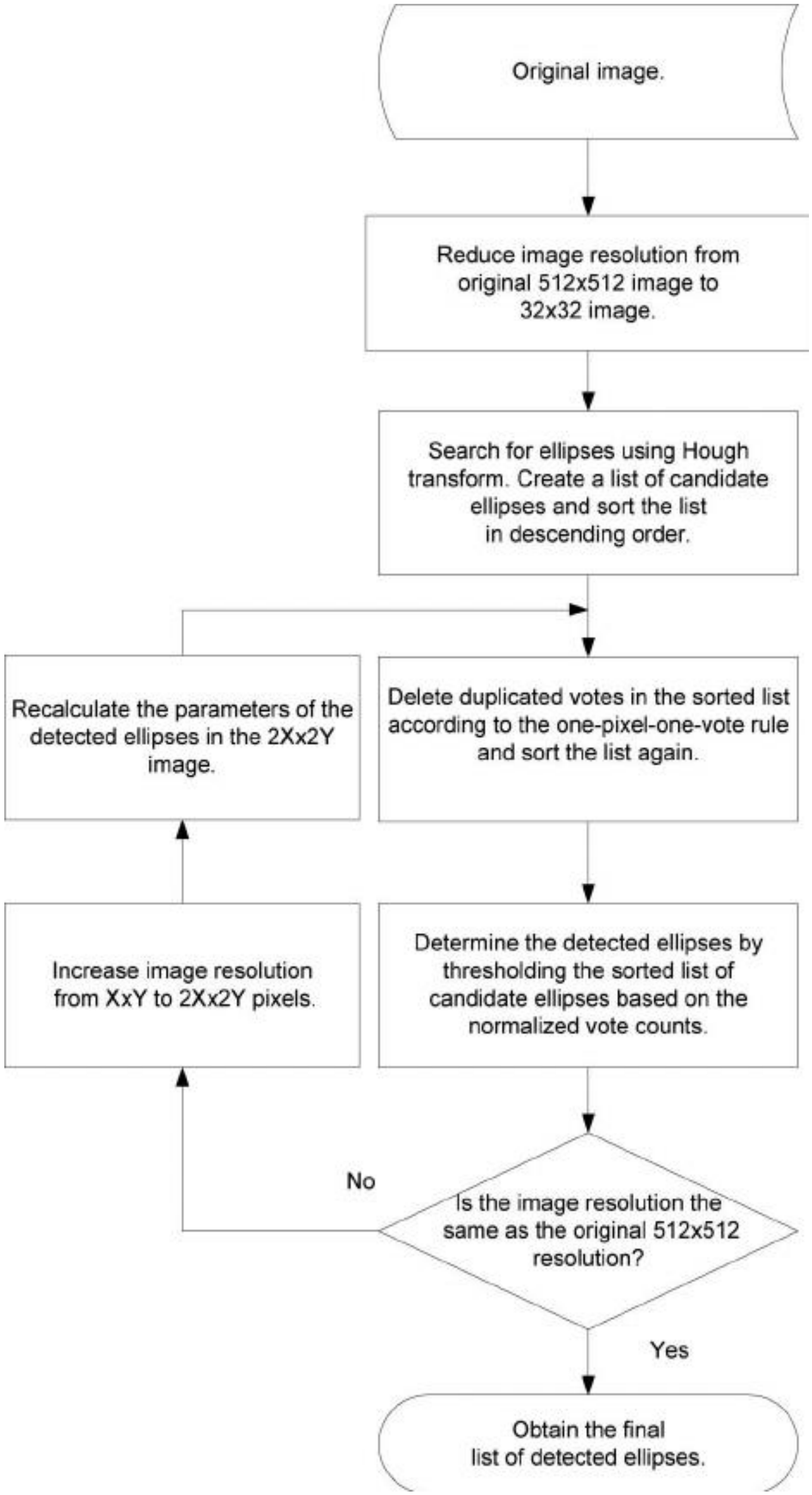
CODE:

```
RGB = imread('canny.jpg');
I  = rgb2gray(RGB);
BW = edge(I,'canny');
[H,T,R] = hough(BW,'RhoResolution',0.5,'Theta',-
90:0.5:89);
subplot(2,1,1);
imshow(RGB);
title('canny.jpg
');
subplot(2,1,2);
imshow(imadjust(rescale(H)),'XData',T,'YData',R,...
      'InitialMagnification','fit');
title('Hough transform of gantrycrane.png');
xlabel('\theta'), ylabel('\rho');
axis on, axis normal, hold on;
colormap(gca,hot);
```

OUTPUT:



ALGORITHM:


Original image.
Reduce image resolution from original 512x512 image to 32x32 image.
Search for ellipses using Hough transform. Create a list of candidate ellipses and sort the list in descending order.
Delete duplicated votes in the sorted list according to the one-pixel-one-vote rule and sort the list again.
Recalculate the parameters of the detected ellipses in the 2Xx2Y image.
Determine the detected ellipses by thresholding the sorted list of candidate ellipses based on the normalized vote counts.
Increase image resolution from XxY to 2Xx2Y pixels.
No
Is the image resolution the same as the original 512x512 resolution?
Yes
Obtain the final list of detected ellipses.

DISCRETE FOURIER TRANSFORM

CODE:

```
x=[4,4;5,7];
[m,n]=size(x);
y=zeros(m,n);
for k=0:m-1
    for l=0:n-1
        for p=0:m-1
            for q=0:n-1
                y(k+1,l+1)=y(k+1,l+1)+x(p+1,q+1)*exp((((-
i)*2*pi*k*p)/m)+(((-i)*2*pi*l*q)/n));
            end
        end
    end
end
disp(y)
```

ALGOTRITHM:

The discrete Fourier transform, or DFT, is the primary tool of digital signal processing. The foundation of the product is the fast Fourier transform (FFT), a method for computing the DFT with reduced execution time. Many of the toolbox functions (including *Z*-domain frequency response, spectrum and cepstrum analysis, and some filter design and implementation functions) incorporate the FFT.

The MATLAB® environment provides the functions `fft` and `ifft` to compute the discrete Fourier transform and its inverse, respectively. For the input sequence *x* and its transformed version *X* (the discrete-time Fourier transform at equally spaced frequencies around the unit circle), the two functions implement the relationships

DISCRETE COSINE TRANSFORM:

CODE:

```
x=[10,-1;-2,1];
[m,n]=size(x);
y=zeros(m,n);
for k=0:m-1
    for l=0:n-1
        for p=0:m-1
            for q=0:n-1

y(k+1,l+1)=y(k+1,l+1)+x(p+1,q+1)*cos((p+0.5)*k*(pi/m))*co
s((q+0.5)*l*(pi/n));
            end
        end
```

```
    end
end
for p=0:m-1
    for q=0:n-1
        if p==0 &&q==0
            y(p+1,q+1)=y(p+1,q+1)*((1/m)^0.5)*((1/n)^0.5)
        elseif p~=0 && q==0

y(p+1,q+1)=y(p+1,q+1)*((2/m)^0.5)*((1/n)^0.5)
        elseif p==0 &&q~=0
            y(p+1,q+1)=y(p+1,q+1)*((1/m)^0.5)*((2/n)^0.5)
        else
            y(p+1,q+1)=y(p+1,q+1)*((2/m)^0.5)*((2/n)^0.5)
        end
    end
end
disp(y)
```

ALGORITHM:

The discrete cosine transform (DCT) is the most popularly used signal processing tool for compressing images and sounds, found in standards such as JPEG and MP3. (Less often used methods include wavelet transforms, polyphase filters, Hadamard transforms, etc.) However, algorithms for computing the DCT quickly are not well-known. The formulas for the naive $\Theta(n^2)$ algorithms are often cited, but desirably fast $\Theta(n \log n)$ algorithms are rarely described in detail. Figuring out how to compute the DCT quickly and efficiently is not obvious.

By contrast, the discrete Fourier transform (DFT) is popular for frequency analysis and visualization (e.g. spectrograms), and many kinds of image/audio processing, but is rarely used for compression.

OTSU SEGMENTATION:

CODE:

```
close all;clear all;clc
%%==========================================================
==========================================
I=imread('images.png'); % Read the Image
figure(1),imshow(I); % display the  Original Image
figure(2),imhist(I); % display the Histogram
%%==========================================================
===========================================
```

```
n=imhist(I); % Compute the histogram
N=sum(n); % sum the values of all the histogram values
max=0; %initialize maximum to zero
%%================================================================
=========================================
for i=1:256
    P(i)=n(i)/N; %Computing the probability of each
intensity level
end
%%================================================================
=========================================
for T=2:255      % step through all thresholds from 2 to
255
    w0=sum(P(1:T)); % Probability of class 1 (separated
by threshold)
    w1=sum(P(T+1:256)); %probability of class2 (separated
by threshold)
    u0=dot([0:T-1],P(1:T))/w0; % class mean u0
    u1=dot([T:255],P(T+1:256))/w1; % class mean u1
    sigma=w0*w1*((u1-u0)^2); % compute sigma i.e
variance(between class)
    if sigma>max % compare sigma with maximum
        max=sigma; % update the value of max i.e
max=sigma
        threshold=T-1; % desired threshold corresponds to
maximum variance of between class
    end
end
%%================================================================
=============================================
bw=im2bw(I,threshold/255); % Convert to Binary Image
figure(3),imshow(bw); % Display the Binary Image
```

OUTPUT:

ALGORITHM:

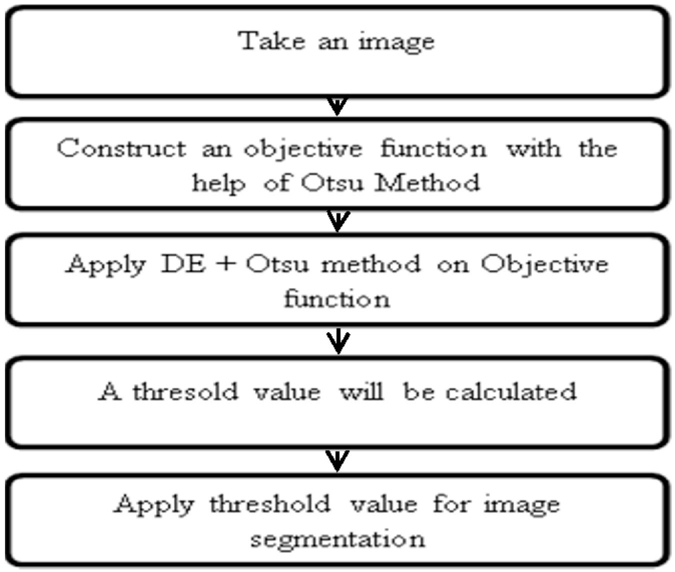


Figure 2: Step by step procedure of the proposed Otsu+DE method

REGION SPLITTING:

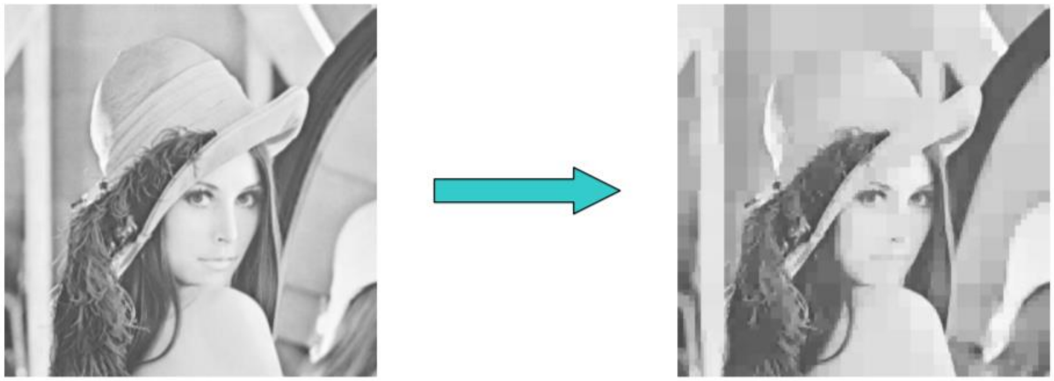
CODE:

```
I = im2double(imread('canny.jpg'));
 figure; imshow(I);
 J = regionspilitting(I);
 figure, imshow(I+J);



function J=regionsplitting(I,x,y,reg_maxdist)
% This function performs "region growing" in an image
from a specified
% seedpoint (x,y)
%
% J = regiongrowing(I,x,y,t)
%
% I : input image
% J : logical output image of region
% x,y : the position of the seedpoint (if not given uses
function getpts)
% t : maximum intensity distance (defaults to 0.2)
%
```

```
% The region is iteratively grown by comparing all
unallocated neighbouring pixels to the region.
% The difference between a pixel's intensity value and
the region's mean,
% is used as a measure of similarity. The pixel with the
smallest difference
% measured this way is allocated to the respective
region.
% This process stops when the intensity difference
between region mean and
% new pixel become larger than a certain treshold (t)
%
% Example:
%

%
% Author: D. Kroon, University of Twente
if(exist('reg_maxdist','var')==0), reg_maxdist=0.2; end
if(exist('y','var')==0), figure, imshow(I,[]);
[y,x]=getpts; y=round(y(1)); x=round(x(1)); end
J = zeros(size(I)); % Output
Isizes = size(I); % Dimensions of input image
reg_mean = I(x,y); % The mean of the segmented region
reg_size = 1; % Number of pixels in region
% Free memory to store neighbours of the (segmented)
region
neg_free = 10000; neg_pos=0;
neg_list = zeros(neg_free,3);
pixdist=0; % Distance of the region newest pixel to the
regio mean
% Neighbor locations (footprint)
neigb=[-1 0; 1 0; 0 -1;0 1];
% Start regiogrowing until distance between regio and
posible new pixels become
% higher than a certain treshold
while(pixdist<reg_maxdist&&reg_size<numel(I))
    % Add new neighbors pixels
    for j=1:4
        % Calculate the neighbour coordinate
        xn = x +neigb(j,1); yn = y +neigb(j,2);

        % Check if neighbour is inside or outside the
image

ins=(xn>=1)&&(yn>=1)&&(xn<=Isizes(1))&&(yn<=Isizes(2));
```

```
        % Add neighbor if inside and not already part of
the segmented area
        if(ins&&(J(xn,yn)==0))
                neg_pos = neg_pos+1;
                neg_list(neg_pos,:) = [xn yn I(xn,yn)];
J(xn,yn)=1;
        end
    end
    % Add a new block of free memory
    if(neg_pos+10>neg_free), neg_free=neg_free+10000;
neg_list((neg_pos+1):neg_free,:)=0; end

    % Add pixel with intensity nearest to the mean of the
region, to the region
    dist = abs(neg_list(1:neg_pos,3)-reg_mean);
    [pixdist, index] = min(dist);
    J(x,y)=2; reg_size=reg_size+1;

    % Calculate the new mean of the region
    reg_mean= (reg_mean*reg_size +
neg_list(index,3))/(reg_size+1);

    % Save the x and y coordinates of the pixel (for the
neighbour add proccess)
    x = neg_list(index,1); y = neg_list(index,2);

    % Remove the pixel from the neighbour (check) list
    neg_list(index,:)=neg_list(neg_pos,:);
neg_pos=neg_pos-1;
end
% Return the segmented area as logical matrix
J=J>1;
End
```

OUTPUT:



ALGORITHM:

- If a region R is inhomogeneous (P(R)= False) then is split into four sub regions
- If two adjacent regions Ri,Rj are homogeneous (P(Ri U Rj) = TRUE), they are merged

The algorithm stops when no further splitting or merging is possible

REGION MERGING:

CODE:

```
clc
addpath('docde');


filename = 'canny.jpg';
image = imread(filename);

%% Region Growing method
[Region1, Region2, Region3,
Region4]=RegionGrowing(image);

figure; imshow(uint8(image));
hold on;
DrawLine(Region1);
hold off;
title('Scale 1');
```

```
figure; imshow(uint8(image));
hold on;
DrawLine(Region2);
hold off;
title('Scale 2');

figure; imshow(uint8(image));
hold on;
DrawLine(Region3);
hold off;
title('Scale 3');

figure; imshow(uint8(image));
hold on;
DrawLine(Region4);
hold off;
title('Scale 4');


%% Region merging method
% method has two parameters minimum adjacent pixel which
means connected
% regions has at least this number of pixel connected
mnadj=10;
% threshold value to make decision to merge two region or
nor
RMThresh=3.5;

RegionResult=RegionMerging(image,Region1,mnadj,RMThresh);

% figure results
figure; imshow(uint8(image));
hold on;
DrawLine(RegionResult);
hold off;
title('Final segmentation');




function DrawLine(SegI)
% input: SegI    --- Segments with segment labels
% output: ImgS   --- Segments with line boundaries

[m, n] = size(SegI);
```

```
ImgS = zeros(m, n);
% RGB = label2rgb(SegI);
% figure; imshow(RGB);
% hold on;
BRegion = imdilate(SegI, [0 1 0; 1 1 1; 0 1 0]);
Boundary = BRegion & ~SegI;


for i = 1:max(SegI(:))
    S = zeros(m, n);
    [x, y] = find(SegI == i);
    for j = 1:length(x)
        S(x(j), y(j)) = 1;
    end
    [B,L] = bwboundaries(S,'noholes');
    for k = 1:length(B)
        boundary = B{k};
        plot(boundary(:,2), boundary(:,1), 'w',
'LineWidth', 1);
    end
    ImgS = ImgS + S;
end
end
function [Region1, Region2, Region3,
Region4]=RegionGrowing(image_org)

%% Written by Muhammet Balcilar, France
% all rights reverved


[m,n,d] = size(image_org);


X = reshape(double(image_org), m*n,d);
[tmp,M,tmp2,P] = kmeansO(X,[],16,0,0,0,0);
map = reshape(P, m, n);

for w = 1:4
    W = GenerateWindow(w);
    JI{w} = JImage(map, W);
end


ImgQ = class2Img(map, image_org);
Region = zeros(m, n);

u = mean(JI{4}(:));
s = std(JI{4}(:));
```

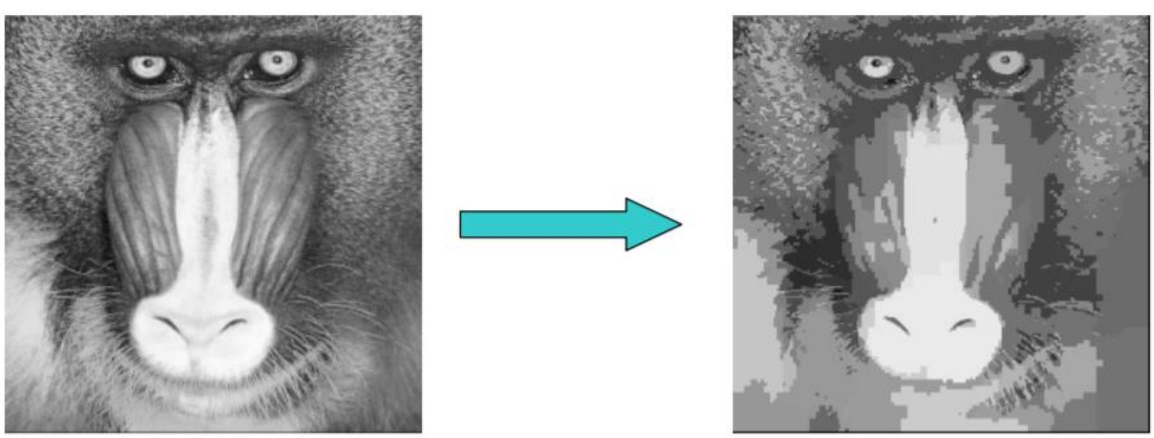
```
Region = ValleyD(JI{4},  4, u, s);
Region = ValleyG1(JI{4}, Region);
Region = ValleyG1(JI{3}, Region);
Region = ValleyG2(JI{1}, Region);
Region4 = Region;


w = 3;
Region = SpatialSeg(JI{3}, Region, w);
Region = ValleyG1(JI{2}, Region);
Region = ValleyG2(JI{1}, Region);
Region3 = Region;

w = 2;
Region = SpatialSeg(JI{2}, Region, w);
Region = ValleyG1(JI{1}, Region);
Region = ValleyG2(JI{1}, Region);
Region2 = Region;

w = 1;
Region = SpatialSeg(JI{1}, Region, w);
Region = ValleyG2(JI{1}, Region);
Region1 = Region;

end
function
RegionResult=RegionMerging(image,Region,mnadj,RMThresh)
%% Written by Muhammet Balcilar, France
% all rights reverved



%% main bloc of algorithm

% find which rregions are connected to each other
[N, Con]=findNeighbour(Region,mnadj);

% find every regions sttistci which means means and
covariances
Stat=findStatistic(Region,image);

% calculate initial regions S similarity values
Sval=calcSval(Stat,Con);

% separate R,G, and B channel of image
R=image(:,:,1);
G=image(:,:,2);
B=image(:,:,3);
```

```
R=double(R(:));
G=double(G(:));
B=double(B(:));

% while minimum similarity of connected regions are still
less than given
% certain threshold continue the process
while min(Sval)<RMThresh
    % find two region which are connected and has minimum
S value
    [a, b]=min(Sval);
    % take the region which will be alive
    i=Con(b,1);
    % take the region number which will be dead
    j=Con(b,2);
    % remove this connection and s value since this two
region are no
    % longer separate regions they will merge to each
other
    Con(b,:)=[];
    Sval(b)=[];
    % make all j th region pixel as ith region
    Region(Region==j)=i;
    % find all pixel which assigned new merged ith region
    I=find(Region==i);

    % calculate new merged ith regions statistics
    tmp=[R(I) G(I) B(I)];
    Stat{i}.mean=mean(tmp);
    Stat{i}.cov=cov(tmp);

    % make all jth class name as ith since jth class no
longer alive
    Con(Con==j)=i;
    % find new merged ith region connections
    I=find(Con(:,1)==i | Con(:,2)==i);
    % calculate new similartiy between the new merged ith
class and its
    % neighbourhood
    for itr=1:length(I)
        muA =Stat{Con(I(itr),1)}.mean';
        muB =Stat{Con(I(itr),2)}.mean';
        covA=Stat{Con(I(itr),1)}.cov;
        covB=Stat{Con(I(itr),2)}.cov;
        Sval(I(itr))=(muA-muB)'*inv(covA+covB)*(muA-muB);
    end

end
```

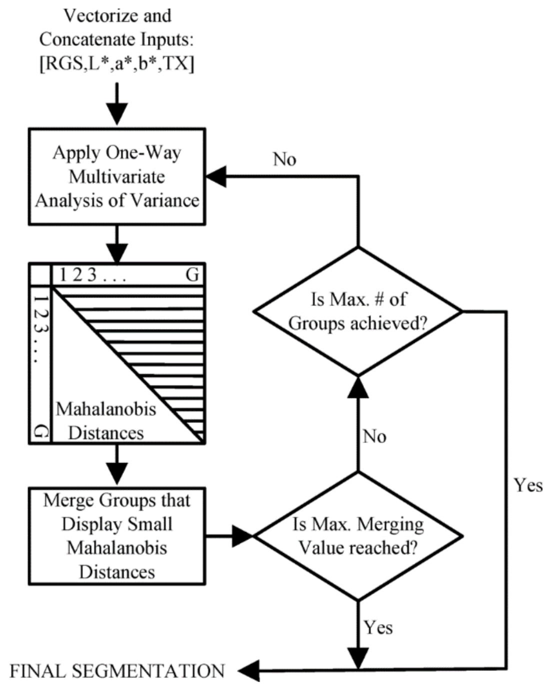
```
% rename all region name from 1 to the end ascendend
I=unique(Region(:));
tmp=Region;
for i=1:length(I)
    tmp(Region==I(i))=i;
end
RegionResult=tmp;



end
function Sval=calcSval(Stat,Con)

%% Written by Muhammet Balcilar, France
% all rights reverved


Sval=zeros(size(Con,1),1);

for itr=1:size(Con,1)
    muA =Stat{Con(itr,1)}.mean';
    muB =Stat{Con(itr,2)}.mean';
    covA=Stat{Con(itr,1)}.cov;
    covB=Stat{Con(itr,2)}.cov;

    Sval(itr)=(muA-muB)'*inv(covA+covB)*(muA-muB);
end
end
function [N, Con]=findNeighbour(Region,mnadj)

%% Written by Muhammet Balcilar, France
% all rights reverved

n=length(unique(Region(:)));
N=zeros(n,n);
Con=[];

for i=1:size(Region,1)-1
    for j=1:size(Region,2)-1
        tmp=Region(i:i+1,j:j+1);
        I=unique(tmp(:));
        if length(I)>1

            if N(I(1),I(2))==mnadj-1
                Con=[Con;min(I(1),I(2)) max(I(1),I(2)) ];
            end
            N(I(1),I(2))=N(I(1),I(2))+1;
            N(I(2),I(1))=N(I(2),I(1))+1;
```

```
        end
    end
end
end
function Stat=findStatistic(Region,image)
%% Written by Muhammet Balcilar, France
% all rights reverved
n=length(unique(Region(:)));

R=image(:,:,1);
G=image(:,:,2);
B=image(:,:,3);

R=double(R(:));
G=double(G(:));
B=double(B(:));

region=Region(:);

for i=1:n
    I=find(region==i);
    tmp=[R(I) G(I) B(I)];
    Stat{i}.mean=mean(tmp);
    Stat{i}.cov=cov(tmp);
end
end
```

OUTPUT:



ALGORITHM:



REGION GROWING:

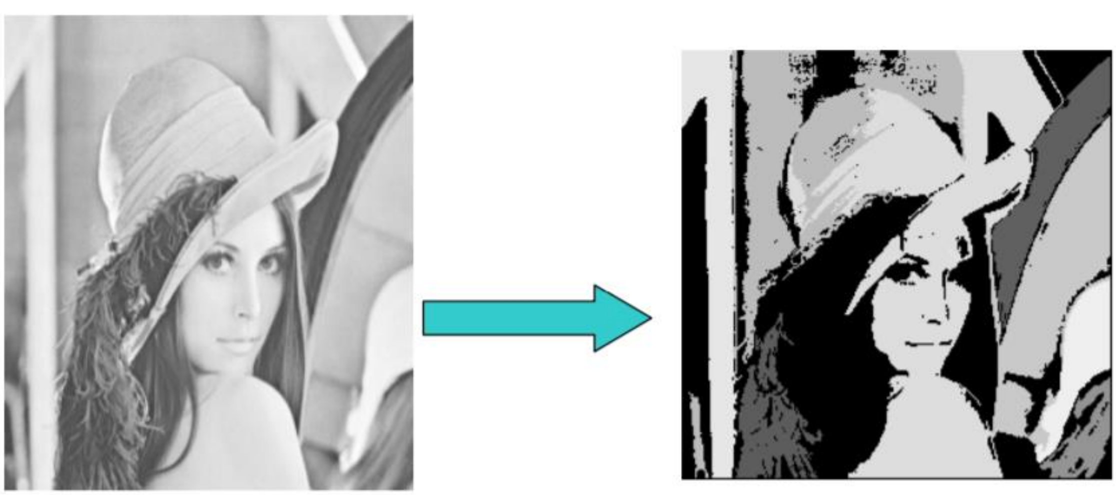
CODE:

```
I = im2double(imread('canny.jpg'));
 figure; imshow(I);
 J = regiongrowing(I);
 figure, imshow(I+J);



function J=regiongrowing(I,x,y,reg_maxdist)
% This function performs "region growing" in an image
from a specified
% seedpoint (x,y)
%
% J = regiongrowing(I,x,y,t)
%
% I : input image
% J : logical output image of region
% x,y : the position of the seedpoint (if not given uses
function getpts)
% t : maximum intensity distance (defaults to 0.2)
%
% The region is iteratively grown by comparing all
unallocated neighbouring pixels to the region.
% The difference between a pixel's intensity value and
the region's mean,
% is used as a measure of similarity. The pixel with the
smallest difference
```

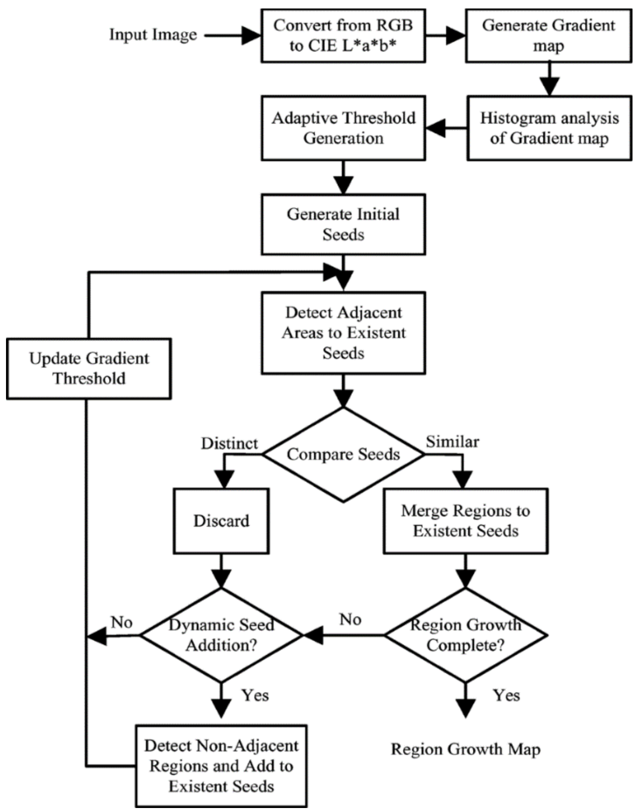
```
% measured this way is allocated to the respective
region.
% This process stops when the intensity difference
between region mean and
% new pixel become larger than a certain treshold (t)
%
% Example:
%

%
% Author: D. Kroon, University of Twente
if(exist('reg_maxdist','var')==0), reg_maxdist=0.2; end
if(exist('y','var')==0), figure, imshow(I,[]);
[y,x]=getpts; y=round(y(1)); x=round(x(1)); end
J = zeros(size(I)); % Output
Isizes = size(I); % Dimensions of input image
reg_mean = I(x,y); % The mean of the segmented region
reg_size = 1; % Number of pixels in region
% Free memory to store neighbours of the (segmented)
region
neg_free = 10000; neg_pos=0;
neg_list = zeros(neg_free,3);
pixdist=0; % Distance of the region newest pixel to the
regio mean
% Neighbor locations (footprint)
neigb=[-1 0; 1 0; 0 -1;0 1];
% Start regiogrowing until distance between regio and
posible new pixels become
% higher than a certain treshold
while(pixdist<reg_maxdist&&reg_size<numel(I))
    % Add new neighbors pixels
    for j=1:4
        % Calculate the neighbour coordinate
        xn = x +neigb(j,1); yn = y +neigb(j,2);

        % Check if neighbour is inside or outside the
image

ins=(xn>=1)&&(yn>=1)&&(xn<=Isizes(1))&&(yn<=Isizes(2));

        % Add neighbor if inside and not already part of
the segmented area
        if(ins&&(J(xn,yn)==0))
                neg_pos = neg_pos+1;
                neg_list(neg_pos,:) = [xn yn I(xn,yn)];
J(xn,yn)=1;
        end
```

```
    end
    % Add a new block of free memory
    if(neg_pos+10>neg_free), neg_free=neg_free+10000;
neg_list((neg_pos+1):neg_free,:)=0; end

    % Add pixel with intensity nearest to the mean of the
region, to the region
    dist = abs(neg_list(1:neg_pos,3)-reg_mean);
    [pixdist, index] = min(dist);
    J(x,y)=2; reg_size=reg_size+1;

    % Calculate the new mean of the region
    reg_mean= (reg_mean*reg_size +
neg_list(index,3))/(reg_size+1);

    % Save the x and y coordinates of the pixel (for the
neighbour add proccess)
    x = neg_list(index,1); y = neg_list(index,2);

    % Remove the pixel from the neighbour (check) list
    neg_list(index,:)=neg_list(neg_pos,:);
neg_pos=neg_pos-1;
end
% Return the segmented area as logical matrix
J=J>1;
End
```

OUTPUT:



ALGORITHM:



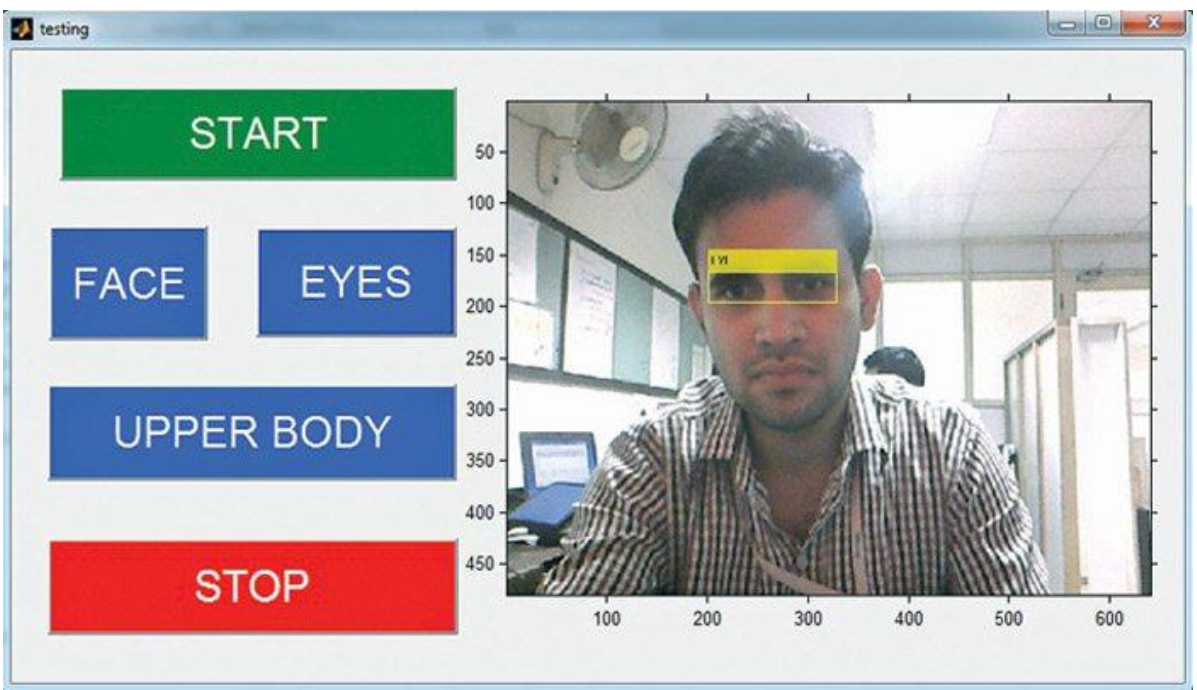
VOILA-JONES OBJECT/FACE DETECTION

CODE:

```
faceDetector=vision.CascadeObjectDetector('FrontalFaceCAR
T'); %Create a detector object
img=imread('images.png'); %Read input image
img=rgb2gray(img); % convert to gray
BB=step(faceDetector,img); % Detect faces
iimg = insertObjectAnnotation(img, 'rectangle', BB,
'Face'); %Annotate detected faces.
figure(1);
imshow(iimg);
title('Detected face');
%htextinsface = vision.TextInserter('Text', 'face   :
%2d', 'Location',  [5 2],'Font', 'Courier
New','FontSize', 14);
hold on
for i=1:size(BB,1)

rectangle('position',BB(i,:),'Linewidth',2,'Linestyle','-
','Edgecolor','y');
end
hold on
N=size(BB,1);
handles.N=N;
```

```
counter=1;
for i=1:N
    face=imcrop(img,BB(i,:));
   %savenam = strcat('D:\Detect face\' ,num2str(counter),
'.jpg'); %this is where and what your image will be saved
    %baseDir  = 'D:\Detect face\TestDatabase\';
    %      baseName = 'image_';
   % newName  = [baseDir num2str(counter) '.jpg'];
   % handles.face=face;
    %while exist(newName,'file')
     %    counter = counter + 1;
      %   newName = [baseDir num2str(counter) '.jpg'];
    end
    fac=imresize(face,[112,92]);
   % imwrite(fac,newName);
figure(2);
imshow(face);
title('crop pic');

    pause(.5);
```

OUTPUT:



ALGORITHM:

This real-time face detection program is developed using MATLAB version R2012a. A graphic user interface (GUI) allows users to perform tasks interactively through controls like switches and sliders.

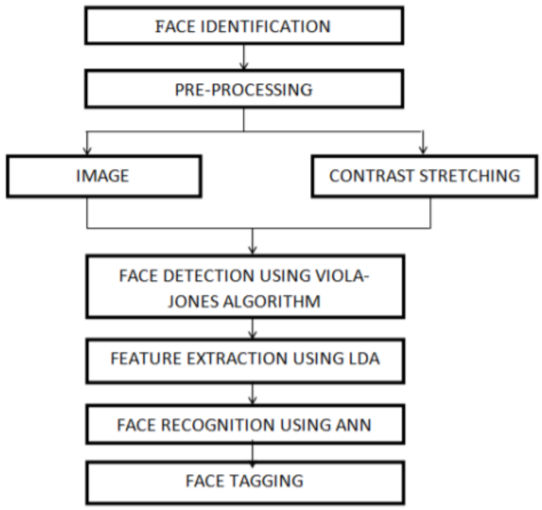


Fig.1: Flow chart for proposed methodology