Project 5

ID：　　　name：

(a) Source codes:

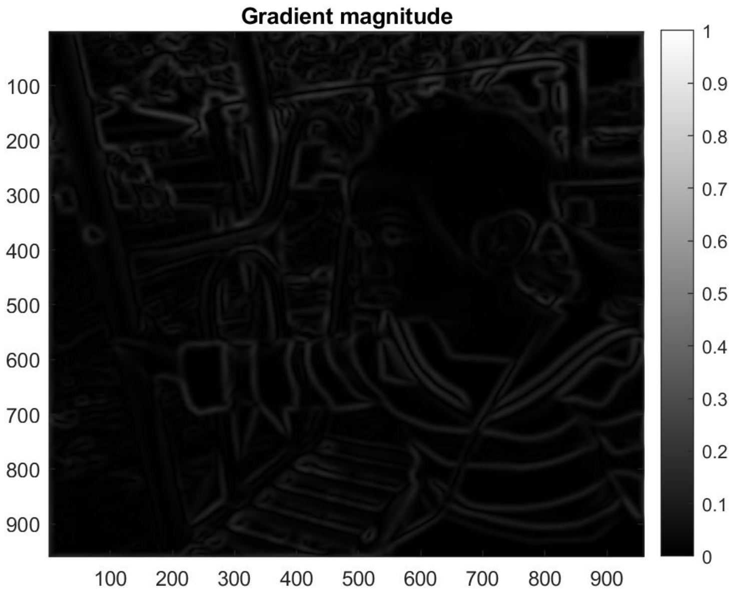
```
% Clear all command window, temporary variables and close all MATLAB window
close all;
clear;
clc;

% Read the image, data type: uint8 [0, 255]
F = imread('Kid at playground.tif');
% Change the image type to double and normalize to [0, 1]
F = im2double(F);

% sigma of Gaussian filter: 0.5% of the shortest dimension of the image
[M,N] = size(F);
sigma = 0.005*min(M,N);

% Defined Guanssian function. Size is an odd number greater than 6*sigma.
G=zeros(29,29);
for i=1:29
    for j=1:29
        G(i,j) = exp(-((i-15)^2+(j-15)^2)/(2*(sigma^2)));
    end
end
% get a smoothed image(FG) by convolving F and G.
FG=conv2(F,G, 'same');

% kernel for horizontal and vertical direction (Sobel operators)
KSx = [-1, 0, 1; -2, 0, 2; -1, 0, 1];
KSy = [-1, -2, -1; 0, 0, 0; 1, 2, 1];
% Convolving smoothed image by horizontal and vertical Sobel operators
Filtered_X = conv2(FG, KSx, 'same');
Filtered_Y = conv2(FG, KSy, 'same');

% Calculate gradient angles (directions)
angle = atan2 (Filtered_Y, Filtered_X);
angle = angle*180/pi;

% Adjusting directions to nearest 0, 45, 90, or 135 degree
angle_f=zeros(M, N);
for i=1:M
    for j=1:N
        if ((angle(i, j) >= -22.5 && angle(i, j) < 22.5) || (angle(i,j) >=
157.5) || angle(i, j) <= -157.5)
            angle_f(i, j) = 0;
        elseif ( ((angle(i, j) >= 22.5) && (angle(i, j) < 67.5)) || ((angle(i,
j) >= -157.5) && (angle(i, j) < -112.5)) )
            angle_f(i, j) = -45;
        elseif ( ((angle(i, j) >= 67.5 && angle(i, j) < 112.5)) || ((angle(i,
j) >= -112.5 && angle(i, j) < -67.5)) )
            angle_f(i, j) = 90;
        elseif ( ((angle(i, j) >= 112.5 && angle(i, j) < 157.5)) || ((angle(i,
j) >= -67.5 && angle(i, j) < -22.5)) )
            angle_f(i, j) = 45;
        end
    end
end

figure;
imagesc(angle);
colorbar;
colormap gray;
title('Gradient angle');
fig= gcf;
exportgraphics(fig,'Gradient angle.png','Resolution',200);

% Calculate gradient magnitude
magnitude = sqrt((Filtered_X.^2) + (Filtered_Y.^2));
magnitude = (magnitude-min(magnitude(:))) ./ (max(magnitude(:))-
min(magnitude(:))); % normalization

figure;
imagesc(magnitude);
colorbar;
colormap gray;
title('Gradient magnitude');
fig= gcf;
exportgraphics(fig,'Gradient magnitude.png','Resolution',200);

% Non-Maximum Suppression
g_N = zeros (M, N);
for i=2:M-1
    for j=2:N-1
        if (angle_f(i,j)==0)
            g_N(i,j) = (magnitude(i,j) == max([magnitude(i,j),magnitude(i,j+1),
magnitude(i,j-1)]));
        elseif (angle_f(i,j)==45)
            g_N(i,j) = (magnitude(i,j) == max([magnitude(i,j),magnitude(i+1,j-
1), magnitude(i-1,j+1)]));
        elseif (angle_f(i,j)==90)
```

```
            g_N(i,j) = (magnitude(i,j) == max([magnitude(i,j),magnitude(i+1,j),
magnitude(i-1,j)]));
        elseif (angle_f(i,j)==-45)
            g_N(i,j) = (magnitude(i,j) ==
max([magnitude(i,j),magnitude(i+1,j+1), magnitude(i-1,j-1)]));
        end
    end
end

g_N = g_N.*magnitude;
img_g_N = (g_N-min(g_N(:))) ./ (max(g_N(:))-min(g_N(:))); %normalization
figure;
imshow(img_g_N);
title('After Non-Maximum Supression');
fig= gcf;
exportgraphics(fig,'After Non-Maximum Supression.png','Resolution',200);

% Hysteresis thresholding: low threshold, T_low and high threshold, T_High
T_Low = 0.04*max(g_N(:));
T_High = 0.1*max(g_N(:));
g_NH = zeros(M, N);
g_NL = zeros(M, N);
for i =1:M
    for j =1:N
        % Strong edge pixels
        if g_N(i, j) >= T_High
            g_NH(i,j) = g_N(i,j);
        end
        % weak edge pixels
        if g_N(i, j) >= T_Low && g_N(i, j) <= T_High
            g_NL(i,j) = g_N(i,j);
        end
    end
end

img_g_NL = (g_NL-min(g_NL(:))) ./ (max(g_NL(:))-min(g_NL(:))); % normalization
figure;
imshow(img_g_NL);
title('gNL');
fig= gcf;
exportgraphics(fig,'gNL.png','Resolution',200);

img_g_NH = (g_NH-min(g_NH(:))) ./ (max(g_NH(:))-min(g_NH(:))); %normalization
figure;
imshow(img_g_NH);
```

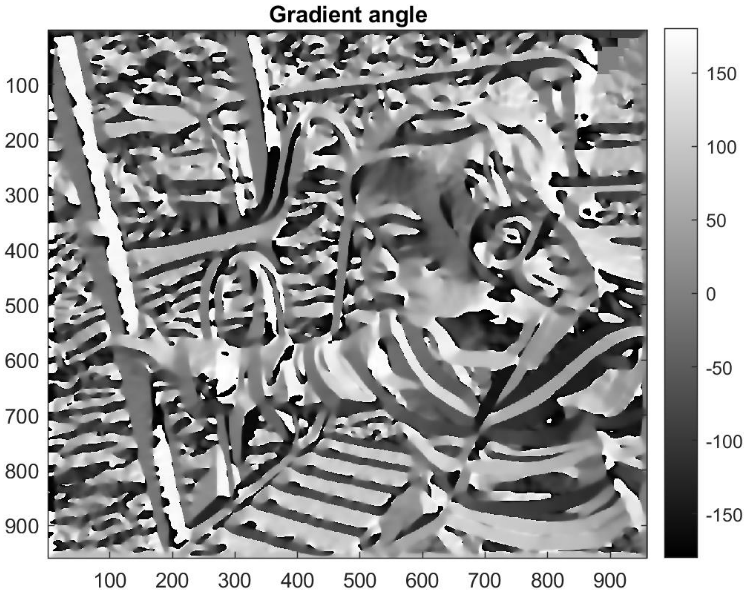
```
title('gNH');
fig= gcf;
exportgraphics(fig,'gNH.png','Resolution',200);

% Mark as valid edge pixels all the weak edge pixels in g_NL that are
connected to g_NH(i,j) using 8-connectivity.
g_NLn = zeros(M,N);
for i=1:M
    for j=1:N
        if g_NH(i,j) > 0
            if g_NL(i+1,j) > 0 || g_NL(i-1,j) > 0 || g_NL(i,j+1) > 0 ||
g_NL(i,j-1) > 0
                g_NLn(i+1,j)=1;
            end
            if g_NL(i+1,j+1) > 0 || g_NL(i-1,j-1) > 0 || g_NL(i-1,j+1) > 0 ||
g_NL(i+1,j-1) > 0
                g_NLn(i+1,j-1)=1;
            end
        end
    end
end

% Combine the g_NH and g_NLn of valid edge pixels into the final output image
g_NHn.
g_NHn = zeros(M,N);
for i=1:M
    for j=1:N
        if g_NH(i,j) > 0
            g_NHn(i,j) = 1;
        elseif g_NLn(i,j) > 0
            g_NHn(i,j) = 1;
        end
    end
end

figure;
imshow(g_NHn);
title('final edge');
fig= gcf;
exportgraphics (fig,'final edge.png','Resolution',200);
```

(b) Plot images of the gradient magnitude and gradient angle:

Magnitude

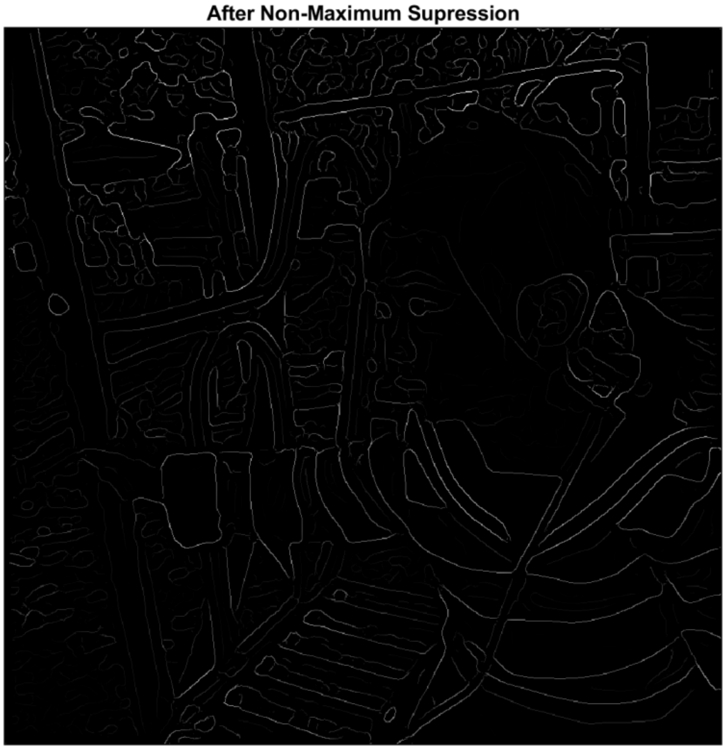


Angle

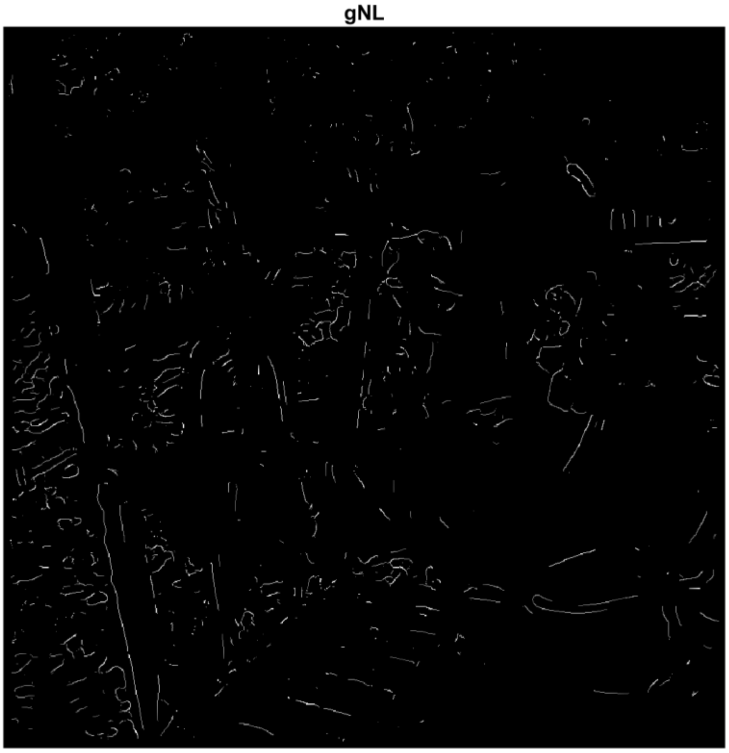


(c) Plot nonmaxima suppressed image g_N (x,y) as well as images of $g_{NL}(x,y)$ and $g_{NH}(x,y)$:

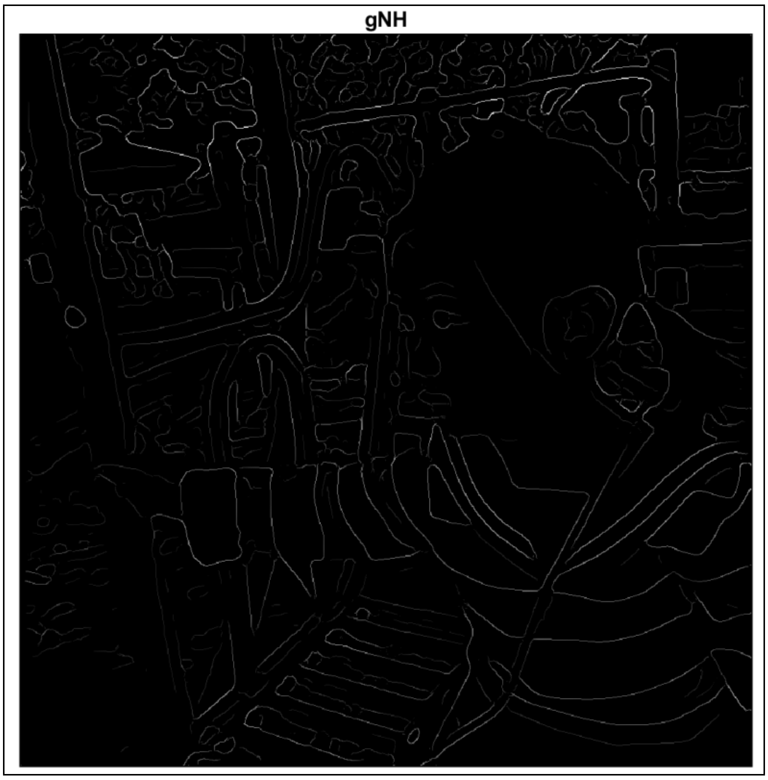
g_N (x,y)



$g_{NL}(x,y)$



$g_{NH}(x,y)$



(d) Plot final edge map e(x,y):