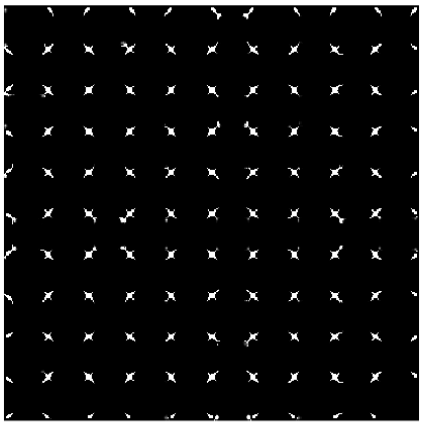
cse327 hw6

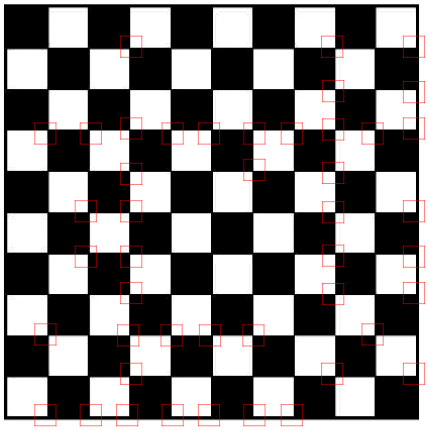
Zian Shang

Notes:

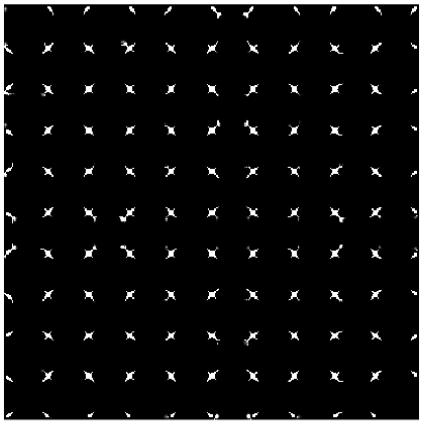
```
% Read both images and convert to greyscale
I_brickwall = double(rgb2gray(imread("brickwall.jpg")));
I_checkboard = double(rgb2gray(imread("checkboard.jpg")));

% Testing
% imshow(uint8(I_brickwall));
% imshow(uint8(I_checkboard));

% Set up input variables
Image = I_checkboard; % Change input image here
Sigma = 1;      % Gaussian sigma
N = 3;          % NxN neighborhood for accumulating sums
D = 10;          % radius of neiborhood for suppression multiple corner responses
M = 45;          % number of corners to detect

% Call detectHarrisCorners function here**
[corners, R] = detectHarrisCorners(Image, Sigma, N, D, M);
% disp(corners);
imshow(uint8(R));
```



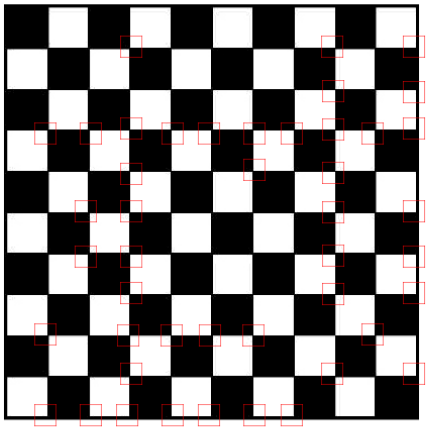
```
% Plot points onto the original image
figure;
imshow(uint8(Image));
hold on;

% Plot the points
plot(corners(:, 2), corners(:, 1), 'rsquare', 'MarkerSize', 15);

hold off;
```



```
% Call eigDecoCorners here
[corners1, R1] = eigDecoCorners(Image, Sigma, N, D, M);
% disp(corners1);
imshow(uint8(R1));
```



```
% Plot points onto the original image
figure;
imshow(uint8(Image));
hold on;

% Plot the points
plot(corners1(:, 2), corners1(:, 1), 'rsquare', 'MarkerSize', 15);

hold off;
```



```
function [corners, R] = detectHarrisCorners(Image, Sigma, N, D, M)
    % perform guassian filtering
    Ismoothed = gua_filtering(Image,Sigma);

    % Compute gradient images Gx, Gy
    kernelx = [-1 0 1];
    kernely = [-1; 0; 1];
    Gx = imfilter(Ismoothed, kernelx, "conv");
    Gy = imfilter(Ismoothed, kernely, "conv");

    % Compute products of derivatives
    Gx2 = Gx.^2;
    Gy2 = Gy.^2;
    Gxy = Gx.*Gy;

    % Compute sums of products over local N*N neighborhood
    box_filter = ones(N);

    Sx2 = imfilter(Gx2, box_filter); % M11
    Sy2 = imfilter(Gy2, box_filter); % M22
    Sxy = imfilter(Gxy, box_filter); % M12 or M21

    % Compute R value
    k=0.05;
    R = (Sx2.*Sy2-Sxy.^2)-k*(Sx2+Sy2).^2;

    Rcopy = R;

    % find the first corner position
    maxValue = max(max(Rcopy));
    [maxX, maxY] = find(Rcopy == maxValue);
    corners = [maxX maxY];

    % Loop M-1 times to find the rest corners
    for i = 2:M
        % mask neiboring positions of the previous corner
        Rcopy = mask(Rcopy, maxX, maxY, D);

        % look for next max position
        maxValue = max(max(Rcopy));
        [maxX, maxY] = find(Rcopy == maxValue);
        corners = [corners; maxX maxY];
    end
end
```

```
function [corners, R] = eigDecoCorners(Image, Sigma, N, D, M)
    % perform guassian filtering
    Ismoothed = gua_filtering(Image,Sigma);

    % Compute gradient images Gx, Gy
    kernelx = [-1 0 1];
    kernely = [-1; 0; 1];
    Gx = imfilter(Ismoothed, kernelx, "conv");
    Gy = imfilter(Ismoothed, kernely, "conv");
```

```
    % Compute products of derivatives
    Gx2 = Gx.^2;
    Gy2 = Gy.^2;
    Gxy = Gx.*Gy;

    % Compute sums of products over local N*N neighborhood
    box_filter = ones(N);

    % ---------------------------------------------------

    M11 = imfilter(Gx2, box_filter); % M11
    M22 = imfilter(Gy2, box_filter); % M22
    M12 = imfilter(Gxy, box_filter); % M12 or M21

    % Compute lambda values
    k=0.05;
    lambda1 = ( (M11+M22) + sqrt( 4*(M12.^2) + (M11-M22).^2) ) /2;
    lambda2 = ( (M11+M22) - sqrt( 4*(M12.^2) + (M11-M22).^2) ) /2;

    % Compute R value
    detM = lambda1.*lambda2;
    traceM = lambda1+lambda2;
    R = detM - k*(traceM).^2;

    % ---------------------------------------------------

    Rcopy = R;

    % find the first corner position
    maxValue = max(max(Rcopy));
    [maxX, maxY] = find(Rcopy == maxValue);
    corners = [maxX maxY];

    % Loop M-1 times to find the rest corners
    for i = 2:M
        % mask neiboring positions of the previous corner
        Rcopy = mask(Rcopy, maxX, maxY, D);

        % look for next max position
        maxValue = max(max(Rcopy));
        [maxX, maxY] = find(Rcopy == maxValue);
        corners = [corners; maxX maxY];
    end
end
```

```
% Guassian filtering function
% input:
% Image (double) : an image to apply filter to
% Sigma (double) : standard deviation of the Guassian kernel

% return: guaF (double) : filtered image
function guaF=gua_filtering(Image, Sigma)
    halfwid=3*Sigma;
    [xx, yy] = meshgrid(-halfwid:halfwid,-halfwid:halfwid);     % create meshgrid

    Gs=exp(-1/(2*Sigma^2) * (xx.^2 +yy.^2)) / (2*pi*Sigma^2);   % calculate Guassian space kernel

    guaF=imfilter(Image, Gs);                                    % apply Guassian kernel to input image, default zero-padding
end

% mask function
% input:
% R (double) : the matrix to be masked
% maxX, maxY (double) : center position of mask
% D (double) : radius of mask

% return: R (double) : original matrix with a neighborhood around the given position set to -inf
function R = mask(R, maxX, maxY, D)

    % Obtain start and end positions
    Xstart = max(1, maxX-D);
    Xend = min(size(R, 1), maxX+D);
    Ystart = max(1, maxY-D);
    Yend = min(size(R, 2), maxY+D);

    % Set the values in the neighborhood to -inf
    for i = Xstart : Xend
        for j = Ystart : Yend
```

```
            R(i, j) = -inf;
        end
    end
end
```