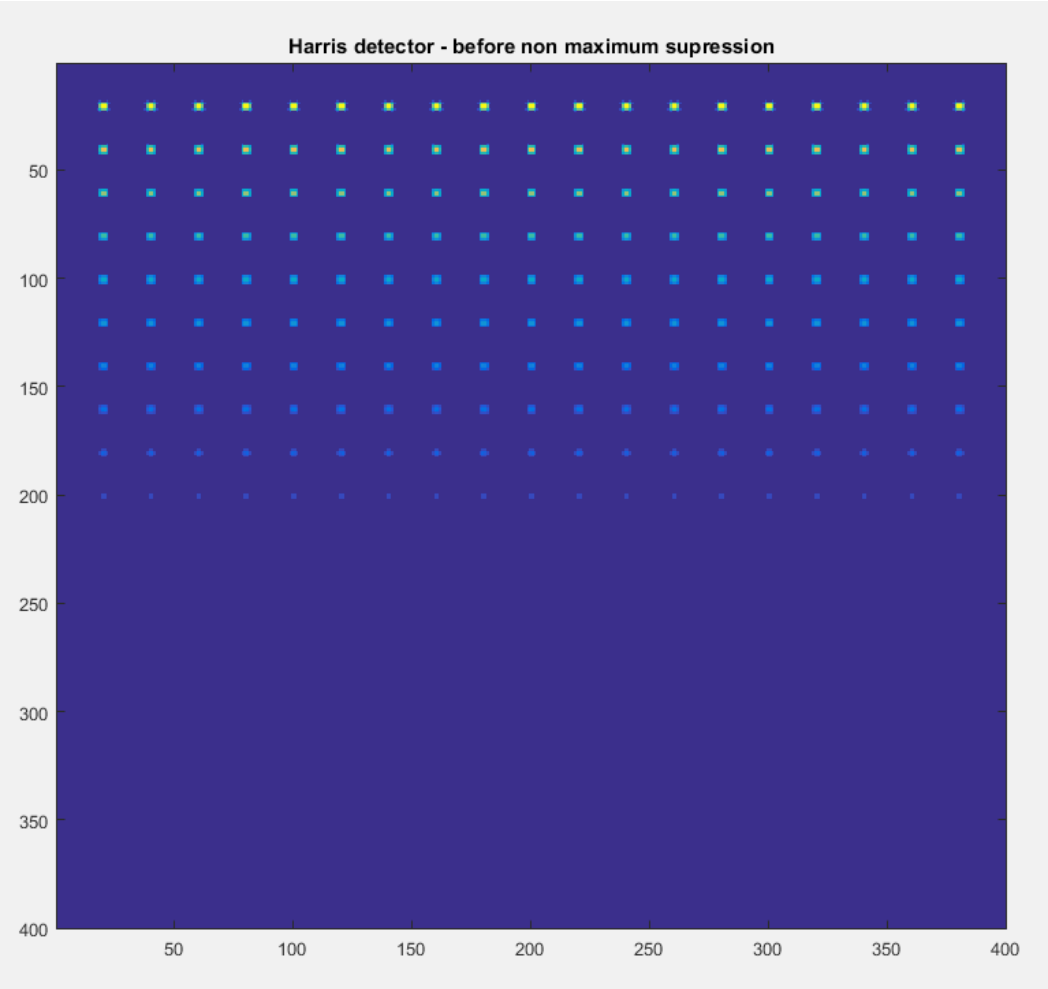
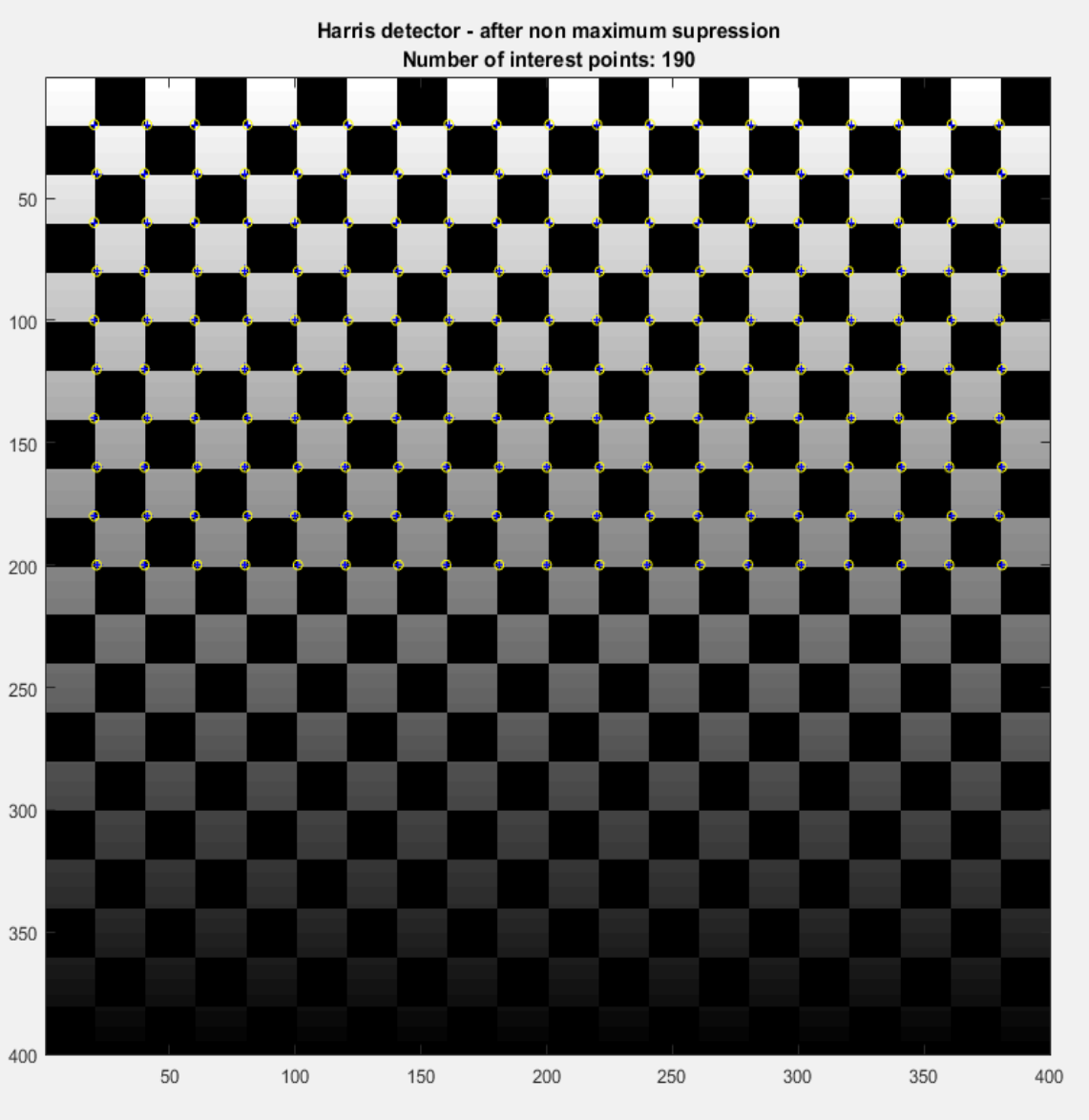
CSE 5524 - Homework #7 **10/12/2016**
Chandrasekar Swaminathan (swaminathan.42)

1) Compute and display the Harris pixel-wise cornerness function values for the image checker.jpg using a) Gaussian window/weighting function with a standard deviation of σI = 1 (use 3σ mask size), b) Gaussian Gx,Gy gradients with a standard deviation of σD = 0.7 (use 3σ mask size), and c) trace weighting factor of α = 0.05. Remove small and negative values in R (< 1e6). Display the thresholded R using imagesc. Lastly, do non-maximum suppression on R to identify the actual corner points and display them on the original image.



After running the Harris detector on the checkerboard image, we can see that the corners in each square of the checkerboard is flagged as a point of interest. The corners of the squares in the bottom half of the checkerboard are not flagged because they were filtered out when we applied the threshold of R<1e6 on the output of the Harris detector. In the above image, there is more than one pixel being flagged as an interest point for each corner. This is rectified by applying a non-maximum suppression filter on the output. A pixel is retained only if it has the maximum value in its 8-connected neighborhood.



The output of the non-maximum suppression filter is taken and then plotted over the original image, as shown above. Each point of interest is highlighted with a yellow circle.

2) Implement the FAST feature point detector using a radius of r = 3, intensity threshold of T = 10, and a consecutive number of points threshold of n* = 9. Run the detector on the image tower.jpg. Display the image and overlay the FAST feature points. Repeat with T = {20, 30, 50} and compare all four results.

```
figure;
imshow(tower);
hold on;
plot(fastX,fastY,'rx');
hold off;
```

FAST detector
Threshold: 10
Number of points: 17659

FAST detector
Threshold: 20
Number of points: 9893

FAST detector
Threshold: 30
Number of points: 6333

FAST detector
Threshold: 50
Number of points: 2969

The number of points detected by the FAST detector drops drastically as the threshold is increased. At threshold=50, we can see that the detector performs reasonably well as it detects sufficient number of points in the foreground objects while ignoring a lot of unwanted feature points detected in the background objects like trees and shadows.

Code

HW7.m

```
mkdir 'out';
% Problem 1
clc;clear;close all;
img = double(imread('input\checker.png'));
[height, width] = size(img);
sigma_d = 0.7;
sigma_i = 1;
trace_weight = 0.05;
[points, R] = harris_detector(img, sigma_d, sigma_i, trace_weight);
imagesc(R);
title('Harris detector - before non maximum supression');
save_current_frame(fullfile('out','before_max_supression.png'));
pause;
imagesc(img);
colormap('gray');
hold on;
plot(points(:, 1), points(:, 2), 'b+', 'MarkerSize', 5);
plot(points(:, 1), points(:, 2), 'yo', 'MarkerSize', 5);
hold off;
title(sprintf('Harris detector - after non maximum supression\nNumber of interest points: %d',
size(points, 1)));
save_current_frame(fullfile('out','after_max_supression.png'));
pause;

% Problem 2
clc;clear;close all;
img = double(imread('input\tower.png'));
thresholds = [10 20 30 50];
n_star = 9;
for threshold=thresholds
    points = fast_detector(img, threshold, n_star);
    imagesc(img);
    colormap('gray');
    hold on;
    plot(points(:, 1), points(:, 2), 'rx');
    hold off;
    title(sprintf('FAST detector - threshold %d\nNumber of interest points: %d', threshold,
size(points, 1)));
    pause;
    save_current_frame(fullfile('out',sprintf('fast-detector-%d.png', threshold)));
end
```

fast_detector.m

```
function [points] = fast_detector(img, threshold, n_star)
   points_on_circle = [0    -3; 1    -3;    2    -2;    3    -1;    3     0;    3     1;
                        2     2; 1     3;    0     3;    -1     3;    -2     2;    -3     1;
                       -3     0;    -3    -1;    -2    -2;    -1    -3];
   points = [];
   [height, width] = size(img);
   radius = 3;
   for x=radius+1:width-radius-1
      for y=radius+1:height-radius-1
         color_values = zeros(1, size(points_on_circle, 1));
         current_pixel_value = img(y,x);
         for n=1:size(points_on_circle,1)
            color_values(1, n) = img(y+points_on_circle(n, 2),x+points_on_circle(n, 1));
         end
         above_values = color_values > current_pixel_value + threshold;
         longest_seq_of_up_values = longest_non_zero_seq(above_values);
         below_values = color_values < current_pixel_value - threshold;
         longest_seq_of_down_values = longest_non_zero_seq(below_values);
         if longest_seq_of_up_values >= n_star || longest_seq_of_down_values >= n_star
            points = cat (1, points, [x y]);
         end
      end
   end
end
```

gauss.m

```
function [out] = gauss(img, sigma)
   gauss_filter = fspecial('gaussian', 2*ceil(3*sigma)+1, sigma);
   out = imfilter(img, gauss_filter, 'replicate');
end
```

gaussDeriv2D.m

```
function [Gx, Gy] = gaussDeriv2D (sigma)
   maskSize = 2*ceil(3*sigma) +1;
   twoSigmaSquared = 2 * sigma^2;
   twoPiSigmaQuad = 2 * pi * sigma^4 ;
   Gx = zeros (maskSize,maskSize);
   Gy = zeros (maskSize,maskSize);
   x0 = floor(maskSize/2) + 1;
   y0 = floor(maskSize/2) + 1;
   for n=1:maskSize
      for m=1:maskSize
         exponentTerm = exp(-((n-x0)*(n-x0) + (m-y0)*(m-y0))/twoSigmaSquared);
         Gx(n,m)= (m-y0) * exponentTerm/twoPiSigmaQuad;
```

```
            Gy(n,m)= (n-x0) * exponentTerm/twoPiSigmaQuad;
        end
    end
end
```

harris_detector.m

```
function [points, R] = harris_detector(img, sigma_d, sigma_i, trace_weight)
    [Gx, Gy] = gaussDeriv2D(sigma_d);
    Ix = imfilter(img, Gx, 'replicate');
    Iy = imfilter(img, Gy, 'replicate');
    g_Ix2 = gauss(Ix.^2, sigma_i);
    g_Iy2 = gauss(Iy.^2, sigma_i);
    g_IxIy = gauss(Ix.*Iy, sigma_i);
    R = g_Ix2 .* g_Iy2 - g_IxIy.^2 - trace_weight .* ((g_Ix2 + g_Iy2).^2);
    R(R < 1e6) = 0;
    new_image = non_maximal_supression(R);
    [y, x] = find(new_image > 0);
    points = [x y];
end
```

longest_non_zero_seq.m

```
function [longest_seq_length] = longest_non_zero_seq(row_vector)
    number_of_items = size(row_vector, 2);
    seq_length = 0;
    longest_seq_length = 0;
    longest_seq_end_index = 0;
    for n=1:number_of_items
        if row_vector(1, n) == 1
            seq_length = seq_length + 1;
        else
            if seq_length > longest_seq_length
                longest_seq_length = seq_length;
                longest_seq_end_index = n - 1;
            end
            seq_length = 0;
        end
    end
    if seq_length ~= 0
        longest_seq_start_index = number_of_items-seq_length+1;
        longest_seq_end_index = number_of_items;
        longest_seq_length = longest_seq_end_index - longest_seq_start_index + 1;
    end
    if longest_seq_end_index == number_of_items && longest_seq_length ~=
number_of_items && row_vector(1,1) == 1
        for n=1:number_of_items
```

```
        if row_vector(1, n) == 0
            break;
        end
        longest_seq_length=longest_seq_length+1;
      end
    end
end
```

non_maximal_suppression.m

```
function [out] = non_maximal_suppression(img)
    [height, width] = size(img);
    out = zeros(height, width);
    for x=2:width-1
        for y=2:height-1
            max_in_neighborhood = max(max(img(y-1:y+1, x-1:x+1)));
            if img(y, x) == max_in_neighborhood
                out(y,x) = max_in_neighborhood;
            end
        end
    end
end
```

save_current_frame.m

```
function [] = save_current_frame (outputFileName)
% saves whatever is displayed in the window to the given file. useful when
% there is dynamic text rendered on top of the image and that should also
% be saved to the file
  frame = getframe;
  imwrite(frame.cdata, outputFileName);
end
```