Dieu My Nguyen
Due: 02 March 2019

# Homework 3

CSCI 5722
Computer Vision

## Problem 1

Show that the following is True or False: If the zero vector is part of a set of vectors that set is dependent.

True.
*Proof:* Consider the finite set of vectors $V = \{v_1, v_2, ..., v_n\}$ in which $v_1$ is the zero vector. Let constants $c_0 = 1$ and $c_1 = c_2 = ... = c_n = 0$. Thus, we have

$$c_1v_1 + c_2v_2 + ... + c_nv_n = 1 \cdot 0 + 0 \cdot v_n + ... + 0 \cdot v_n = 0.$$

Therefore, the vector set $V$ is linearly dependent.

## Problem 2

Show that the following is True or False: If a set of vectors is dependent so is any larger set which contains it.

True.
*Proof:* Let $v_1, v_2, ..., v_n$ be a set of $V$. Suppose that the subset $v_1, v_2, ..., v_m$, where $m < n$, is linearly dependent. Thus, we have

$$b_1v_1 + b_2v_2 + ... + b_mv_m + 0v_{m+1} + 0v_{m+2} + ... + 0v_n = 0$$

where not all of $b_q, b_2, ..., b_m$ are zero. Therefore, $v_1, v_2, ..., v_n$ is linearly dependent.

## Problem 3

What is separability? How does separability affect computational complexity?

A filter is separable if it can be written as an outer product of two simple filters or vectors. For example, a 2D Sobel kernel can be written as a product of a column and a row vector. Separability reduces computational complexity and speeds up the filtering process. Given a $M \times N$ image and a $A \times B$ filter kernel, the computational complexity would be $A^2MN$ operations. If the filter kernel is separated into two 1D filters, we can filter in two steps for the rows and columns, with the first one's complexity being $MNA$ and the second $MNB$, for a total of $MN(A + B)$.

## Problem 4

Show that convolving an image with a discrete, separable 2D filter kernel is equivalent to convolving with two 1D filter kernels. Estimate the number of operations saved for an $N \times N$ image and a $(2k + 1) \times (2k + 1)$ kernel.

Let $y$ be the output image, $x$ the input image, and $h$ the 2D filter kernel. By the definition of 2D convolution,

$$y(m,n) = x(m,n) \times h(m,n)$$

A 2d filter kernel is separable if it can be expressed as the outer product of a row and a column vector

$$h(m,n) = h_1(m) \times h_2(n)$$

Then

$$y(m,n) = x(m,n) \times h(m,n) = x(m,n) \times h_1(m) \times h_2(n)$$

By the associative property of convolution

$$y(m,n) = [x(m,n) \times h_2(n)] \times h_1(m) = [x(m,n) \times h_1(m)] \times h_2(n)$$

Example of computing y(1,1):

Input: $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$ Separable kernel: $\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 2 & 1 \end{bmatrix}$

1) Convolving with the discrete, separable 2D filter kernel:

$$y(1,1) = 1 \cdot 1 + 2 \cdot 2 + 3 \cdot 1 + 4 \cdot 2 + 5 \cdot 4 + 6 \cdot 2 + 7 \cdot 1 + 8 \cdot 2 + 9 \cdot 1 = 80$$

2) Convolving with the two 1D filter kernels:

First, vertical 1D convolution:

$$\begin{aligned} y(m,1) &= \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \\ &= [1 \cdot 1 + 4 \cdot 2 + 7 \cdot 1 \quad 2 \cdot 1 + 5 \cdot 2 + 8 \cdot 1 \quad 3 \cdot 1 + 6 \cdot 2 + 9 \cdot 1] \\ &= [16 \quad 20 \quad 24] \end{aligned}$$

Then, horizontal convolution:

$$\begin{aligned} y(1,1) &= \begin{bmatrix} 16 & 20 & 24 \end{bmatrix} \times \begin{bmatrix} 1 & 2 & 1 \end{bmatrix} \\ &= [16 \cdot 1 + 20 \cdot 2 + 24 \cdot 1] \\ &= 80 \end{aligned}$$

Number of operations saved:

Filtering a $N \times N$ image with a $(2k+1) \times (2k+1)$ 2D kernel gives us $N^2(2k+1)^2$ operations total, while filtering with two 1D kernels will give us $N^2(2(2k+1))$ or $N^2(4k+2)$. The number of operations saved when we use the separable filter approach is

$$\begin{aligned} n &= N^2(2k+1)^2 - N^2(4k+2) \\ &= N^2\left(4k^2 + 4k + 1 - (4k+2)\right) \\ &= N^2(4k^2 - 1) \text{ operations} \end{aligned}$$

## Problem 5

What happens when we convolve a Gaussian with another Gaussian?

We'd get another (wider) Gaussian, whose variance is the sum of the two original variances, as shown below, supposing we have two Gaussians, $f$ and $g$.

$$f = \frac{1}{\sigma_1\sqrt{2\pi}} \cdot e^{-(t-\mu_1)^2 \ / \ 2\sigma_1^2}$$

$$g = \frac{1}{\sigma_2\sqrt{2\pi}} \cdot e^{-(t-\mu_2)^2 \ / \ 2\sigma_2^2}$$

$$f \cdot g = \frac{1}{\sqrt{2\pi(\sigma_1^2 + \sigma_2^2)}} \cdot e^{-[t-(\mu_1+\mu_2)]^2 \ / \ [2(\sigma_1^2+\sigma_2^2)]}$$

## Problem 6

What is Dimensionality Reduction and how is it important with respect to Image Processing? Elaborate on one advantage and one disadvantage.

A process for reducing the number of features to be used in an analysis or modeling exercise. Some of the techniques for image processing are: 1) principal component analysis (PCA), which transforms the high dimensional input space onto the feature space where the maximal variance is displayed, 2) image compression, where we keep the largest singular values that contain most of the info about the image, and 2) least squares, which is similar to PCA in that the directions along which there is greatest variance are the principal components.

- One advantage: It reduces storage space and computation time. For images, the features are the pixels. For a 32x32 image, we have 1024 features. We can reduce this number by compressing the image, keeping as much info as possible without the image losing its essential structure.

- One disadvantage: We might lose important information from the features we eliminate, such as the resolution of the image.

## Problem 7

Show that for a 2x2 matrix A, with eigenvalues $\lambda_1$, $\lambda_2$
a) $det(A) = \lambda_1 * \lambda_2$

Given the eigenvalues are roots of the characteristic polynomial $p(\lambda) = det(A - \lambda I_n)$, where $n = 2$, we have

$$det(A - \lambda I_2) = \begin{bmatrix} a_{11} - \lambda & a_{12} \\ a_{21} & a_{22} - \lambda \end{bmatrix} = \prod_{i=1}^{2} (\lambda_i - \lambda)$$

If we let $\lambda = 0$, then we observe that $det(A) = \prod_{i=1}^{2} \lambda_i = \lambda_1 \cdot \lambda_2$.

b) $trace(A) = \lambda_1 + \lambda_2$

We can compare the coefficients of $\lambda^{n-1}$ of both sides of the above $det(A - \lambda I_2)$ equation. For the determinant on the left side, the coefficient of $\lambda^{n-1}$ is

$$(-1)^{2-1}(a_{11} + a_{22}) = -1 trace(A)$$

For the determinant on the right side, the coefficient of $\lambda^{n-1}$ is

$$(-1)^{2-1} \sum_{i=1}^{2} \lambda_i = -1 \sum_{i=1}^{2} \lambda_i$$

Thus, $trace(A) = \sum_{i=1}^{2} \lambda_i = \lambda_1 + \lambda_2$.

## Problem 8. MATLAB Programming

Write a script that compares the performance of a 3x3 mean filter, a 3x3 median filter and Gaussian filters with different values of $\sigma$

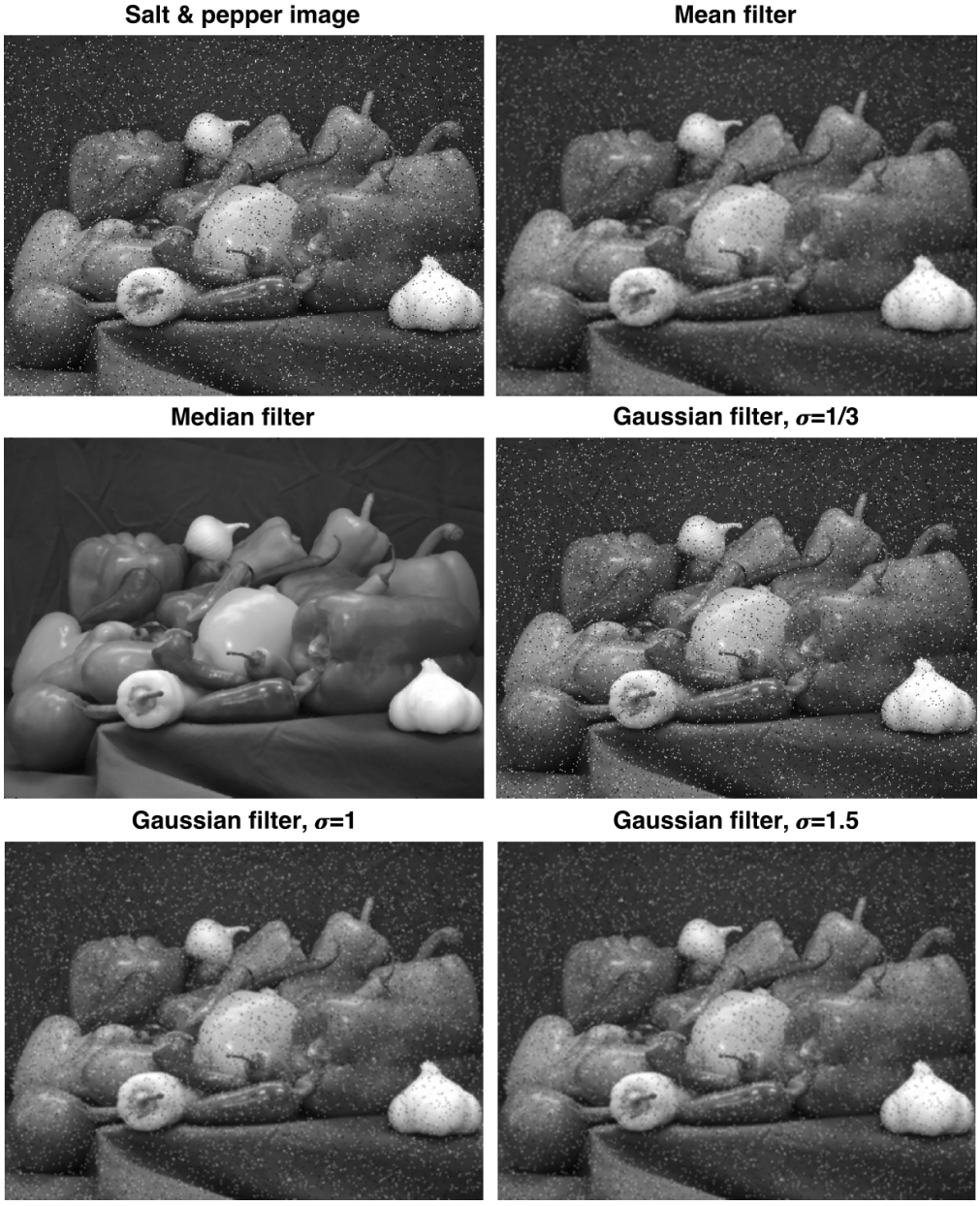
(A) Your script should load image peppers.png, convert it from RGB to grayscale (use function rgb2gray), and initially add salt and pepper noise to the image. Apply five filters to the image:

- a 3x3 mean filter
- a 3x3 median filter
- a Gaussian filter with $\sigma = 1/3$ pixel
- a Gaussian filter with $\sigma = 1$ pixel
- a Gaussian filter with $\sigma = 1.5$ pixels

Plot each one of the images and comment on what works best.

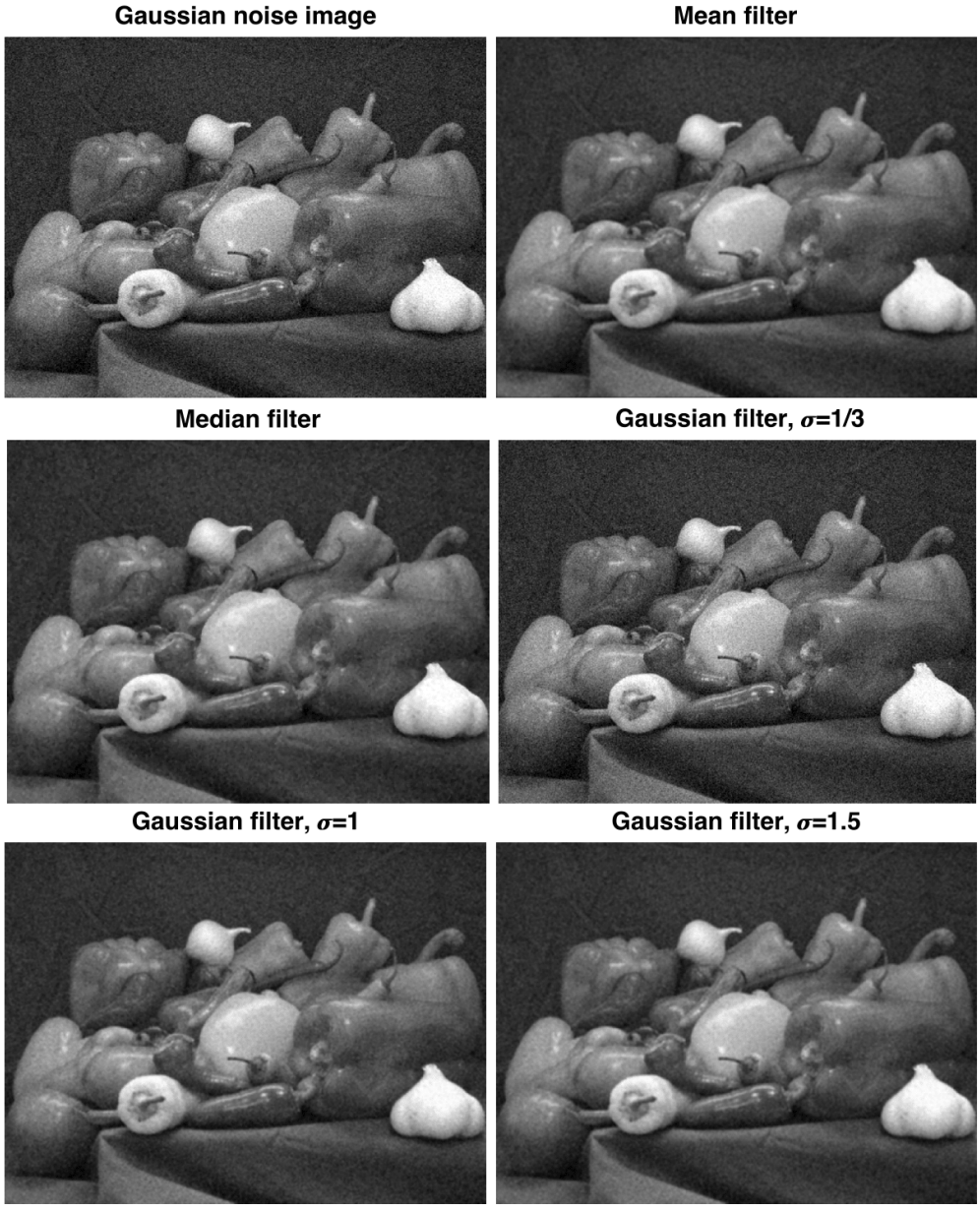
Shown below are salt and pepper noise image and the results of each filter. The median filter is the best in removing the salt and pepper noise. The mean and Gaussian filters at $\sigma$=1

and 1.5 show similar results with only somewhat blurring the noise. The worse performance is by the Gaussian filter with $\sigma = 1/3$.



(B) Repeat the same steps, but instead of salt & pepper noise, add Gaussian white noise with mean 0 and $\sigma = 1$ in the [0, 255] range (or $\sigma = 1/256$ in the [0, 1] range). Apply five filters to the image. Plot each one of the images and comment on what works best.

Shown below are the Gaussian noise image and the results of each filter. The filters show fairly similar (poor) result in removing the noise, so it might require a different value for the filter kernel size. Like for the salt and pepper noise, the worse performance is by the Gaussian filter with $\sigma = 1/3$.



## Problem 9. MATLAB Programming:

Write your own implementation of the Harris Corner Detector algorithm.

A. Write a function `harris` that implements the Harris Corner Detector: `[corner_coords, descriptors] = harris(I, w, threshold, suppression)`
Follow the steps outlined in lecture (Lecture 15, recording and slides on Moodle). Your function will accept four input arguments:

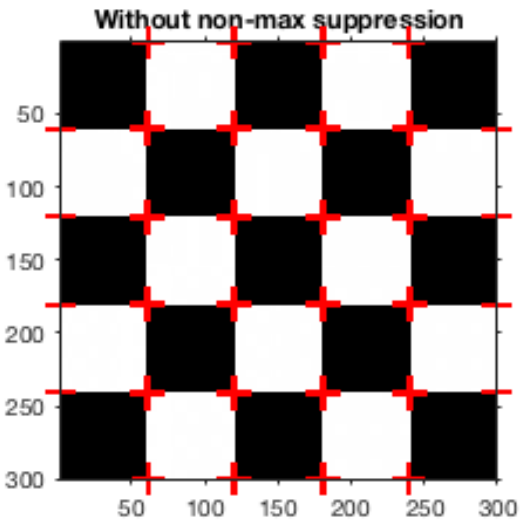
- `I` - an RGB image
- `w` - the width of the Gaussian filter window, used initially for de-noising. Please note that the algorithm you will implement uses a `w(u,v)` Gaussian window function, where `W(x,y)` is a `w x w` size neighborhood around pixel at coordinates `(x,y)` .
- `threshold` - the threshold for separating the values of the cornerness function R
- `suppression` - a boolean which determines if non-maximum suppression is applied

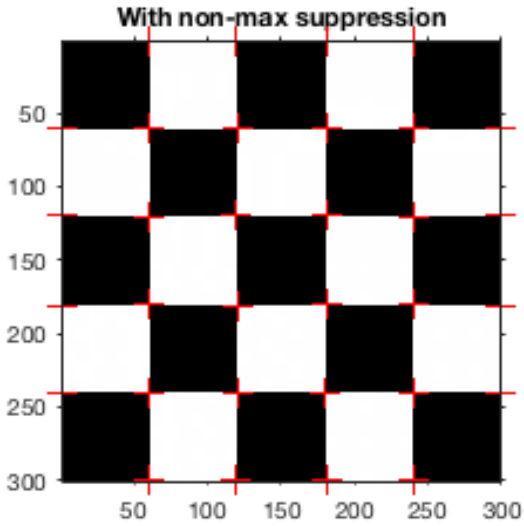
The function should return two variables:

- `corner_coords` - a matrix with 2 columns, representing the `(x,y)` coordinates of the feature points selected.
- `descriptors` - a matrix with 9 columns, representing the pixel intensities (in grayscale) of a `3 x 3` patch around each feature point.

Plot the results of the corner detection, with and without non-maximum suppression. You can use any image of your choosing, for example one of the images used for the image stitching assignment.

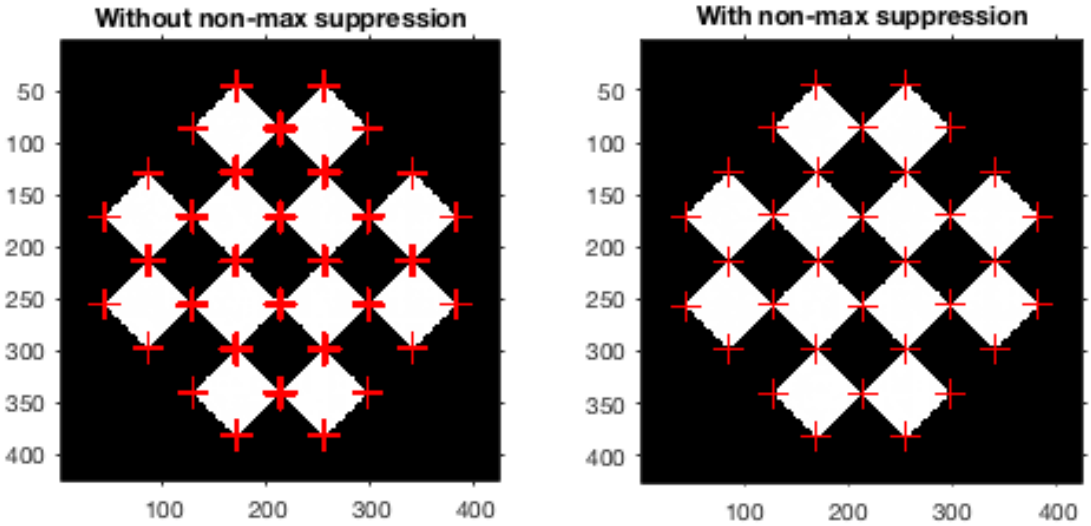
I'm using a very simple test image of a black and white checkerboard with obvious corner points. Detected corner points are shown as plus symbols. As shown, without the non-max suppression, multiple points aggregate at roughly the same corner of a given square in the board. Note that this and everything below use a threshold value of 0.1 and window width of 3.



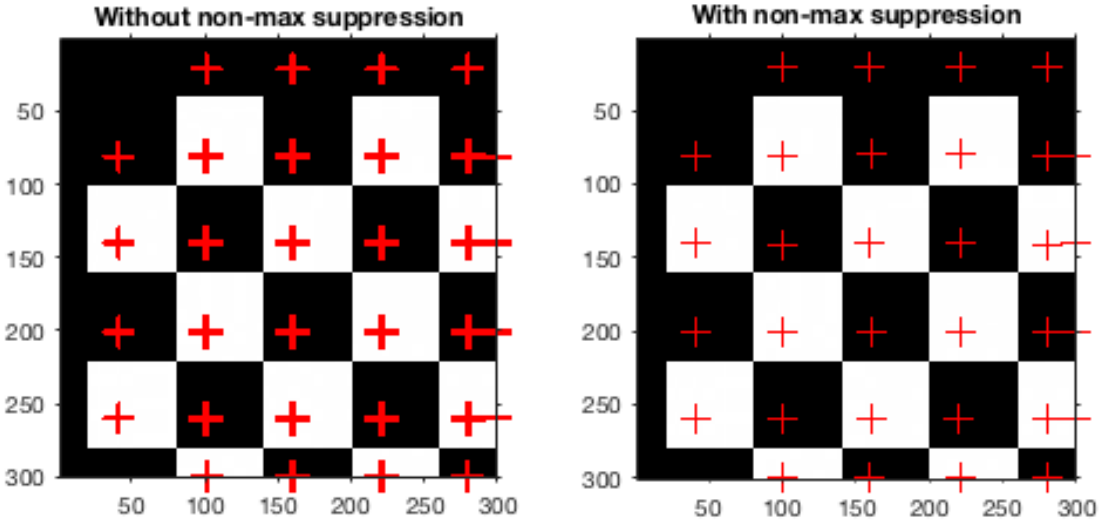



B. Now test how well your corner detector behaves under rotation, translation, and scale of the image. You can do this by a simple exercise in matching. Take one test image (you can use any image of your choosing) and prepare a rotated, translated, and scaled version of that image. Plot the results of the corner detection in each image and comment on how well the features in the original image have matching features in the rotated, translated and scaled versions.

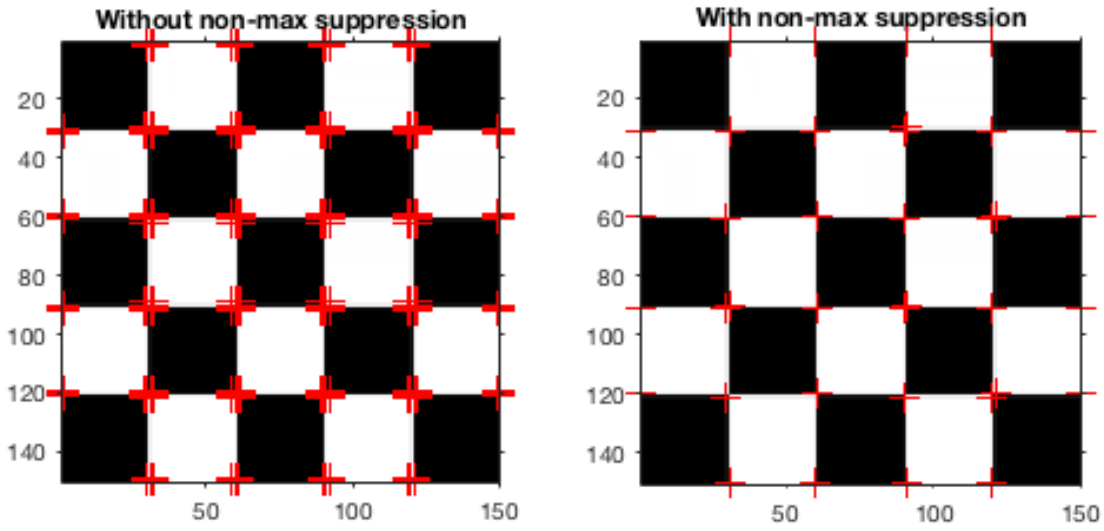
Rotation: The detector does pretty well with a 45° rotated image. Again, it is better with non-max suppression.




Translation: Translating 20 pixels to the right and 40 down. The Harris corner detection algorithm should be invariant to translational changes, but this is the worst result among all the different transformations for my implementation. This might be due to some error in the algorithm function or due to a translation bug.



Scaled: The original image was scaled down 50% (please note the change in the x and y axes). The detector detects the corners very well, but with this smaller image, the non-max suppression still yields some overlapped points compared to the original result. For the version without non-max suppression, more points are detected for a given corner with higher variability in spatial location.

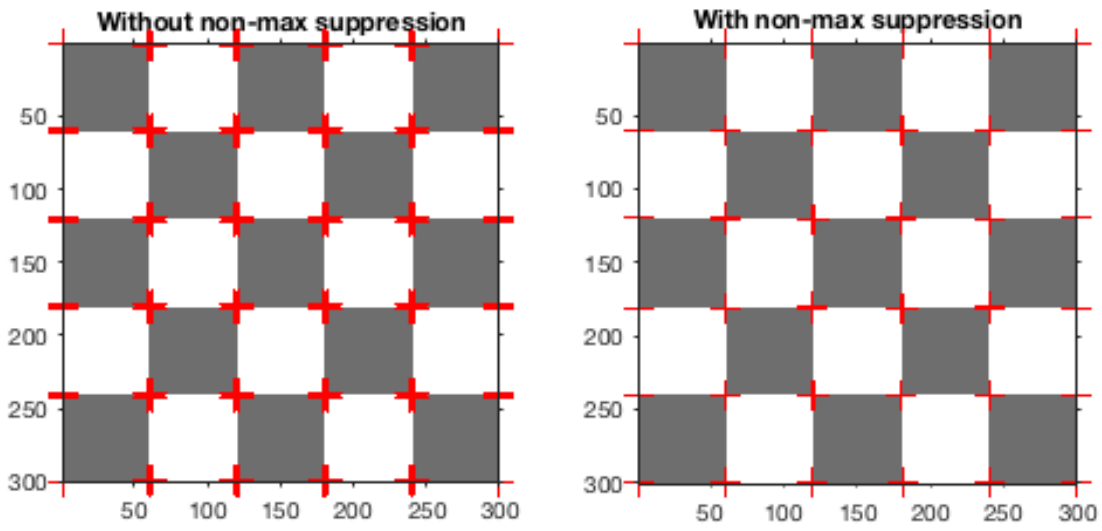



C. Repeat the test at point B. This time, compare the results of the corner detector in the original image with four new versions of it:

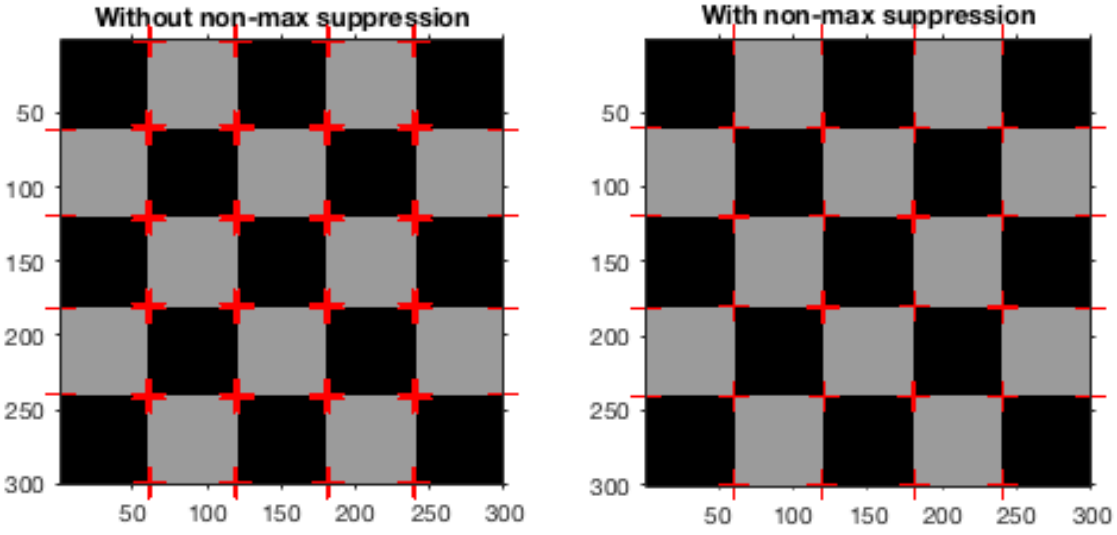
- Prepare a brighter version of the original image by adding a constant positive offset to all pixel values
- Prepare a darker version of the original image by adding a constant negative offset to all pixel values
- Prepare a brighter version of the original image by multiplying a constant positive offset to all pixel values
- Prepare a darker version of the original image by multiplying a constant negative offset to all pixel values

Plot the results of the corner detection in each image and comment on how well the features in the original image have matching features in the four new versions.

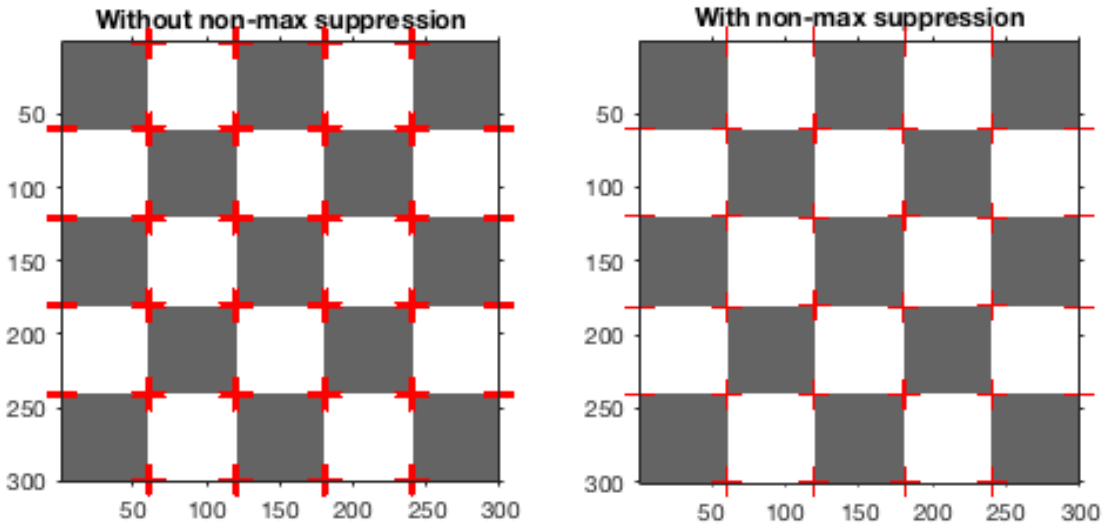
Brighter - Addition: The constant here is 100. Corners are still detected, but multiple are detected around the same point even with non-max suppression.




Darker - Addition: Similar to above: The constant here is 100. Corners are still detected, but multiple are detected around the same point even with non-max suppression.




Brighter - Multiplication: The constant here is 10. Similar to above: Corners are still detected, but multiple are detected around the same point even with non-max suppression.




Darker - Multiplication: The constant here is 0.5. Similar to above: Corners are still detected, but multiple are detected around the same point even with non-max suppression.