CSCI 5722 Computer Vision, Spring 2020
Student: Yinbo Chen
Instructor: Fleming
Homework 3

Q1. True or False? Explain your reasoning. If the zero vector is part of a set of vectors, then that set is linearly dependent.

It's true. To assume that v is a vector space and consider **{0, v_1,v_2 , … ,v_n}** $v_i \in$ **v.**To examinec_0***0 +** c_1*****v_1 **+** c_2*****v_2 **+…+** c_n*****v_n **= 0**.Letc_0 **= 1** andc_1 **=** c_2 **=** c_3 **…** c_n **= 0**.So the equation can be expressed as**1*0 + 0 *** v_1 **+ 0 ***v_2 **+ 0 *** v_3 **+ … +0 *** v_n **= 0**. It shows that there existsc_i**'**s not all zero and still satisfy the equation.

Q2. True or False? Explain your reasoning. If a set of vectors is linearly dependent so is any larger set which contains it.

It's true. To assume that **p** is a vector space and consider**{**p_1,p_2,p_3,…,p_n**}** $p_i \in$**p**is dependent. And there are coefficientsa_1**,** a_2**, …** a_n,are not all zero. The equation can be expressed as

$$a_1p_1 + a_2p_2 + \dots + a_np_n = 0$$

Now let **q** is **p's** larger set and **m>= n**, and consider {p_1, p_2, p_3, …, p_n ,p_{n+1}, p_{n+2}, … , p_m}. The equation of **q** can be expressed as

$$b_1p_1 + b_2p_2 + \dots + b_np_n + b_{n+1}p_{n+1} + b_{n+2}p_{n+2} + \dots + b_mp_m = 0$$

Take $b_1 = a_1$, $b_2 = a_2$, … ,$b_n = a_n$, and $b_{n+1} = b_{n+2} = \dots = b_m = 0$. Because$a_i$'s are not all zero, b_i'sare not all zero either. We got the equations

$$b_1p_1 + b_2p_2 + \dots + b_mp_m = a_1p_1 + a_2p_2 + \dots + a_np_n = 0$$

So {p_1, p_2, …, p_m} is linearly dependent as well.

Q3. 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*N image and a (2k + 1) * (2k + 1) kernel.

Let y[m,n] as the output image and x[m,n] is an input image. h[m,n] is a 2D filter. The 2D convolution is expressed below,

$$Y[m, n] = x[m, n] * h[m, n]$$

The 2D filter can be decomposed into two 1D filters. The equation is:

$$h[m, n] = h_1[m] * h_2[n]$$

So the final equation can be expressed as:

$$Y[m, n] = x[m, n] * h_1[m] * h_2[n]$$

As well as

$$Y[m, n] = x[m, n] * h_2[n] * h_1[m]$$

For example, in convolution a 2D filter with M*N kernel. To assume if the kernel size is 3 * 3 and it can be decomposed into (M *1) and (1 * N) matrices as below:

$$\begin{bmatrix} A*a & A*b & A*c \\ B*a & B*b & B*c \\ C*a & C*b & C*c \end{bmatrix} = \begin{bmatrix} A \\ B \\ C \end{bmatrix} * \begin{bmatrix} a & b & c \end{bmatrix}$$

So convolution with this separable kernel is equivalent to:

$$x[m, n] * \begin{bmatrix} A*a & A*b & A*c \\ B*a & B*b & B*c \\ C*a & C*b & C*c \end{bmatrix} = x[m, n] * \begin{bmatrix} A \\ B \\ C \end{bmatrix} * \begin{bmatrix} a & b & c \end{bmatrix}$$

Number of operations saved:

Using a 2D filter convolves an N x N image with a (2k+1) x (2k+1) kernel. It totally needs $N^2(2k+1)^2$ operations. But performing two 1D filters only needs $N^2(2(2k+1))$. Let **i** equal to the saved operations, then:

$$i= \quad N^2 (2k+1)^2 - N^2(2(2k+1)) = N^2(4k^2 + 4k +1 -4K -2) =N^2(4k^2 - 1)$$

Q4. What happens when we convolve a Gaussian with another Gaussian? Explain.

We will get a wider Gaussian. The variance equals to the sum of two original variances. Consider that we have two Gaussians f_1 and f_2:

$$f1 = \frac{1}{\sigma_1\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu_1}{\sigma_1})^2}$$

$$f2 = \frac{1}{\sigma_2\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu_2}{\sigma_2})^2}$$

$$f1 \cdot f2 = \frac{1}{\sigma_1\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu_1}{\sigma_1})^2} \cdot \frac{1}{\sigma_2\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu_2}{\sigma_2})^2} = \frac{1}{\sqrt{(\sigma_1^2+\sigma_2^2)}\sqrt{2\pi}} e^{-\frac{[x-(\mu_1+\mu_2)]^2}{2(\sigma_1^2+\sigma_2^2)}}$$

Q5. What is Dimensionality Reduction and how is it important with respect to Image Processing? Elaborate on at least one advantage and one disadvantage.

Dimensionality Reduction is the process of reducing the number of features which is the way to reduce the complexity of a model. It can be used in data analysis and it's very important in image processing. For example, the most popular linear dimension reduction is Principal Component Analysis (PCA) which transfers the data from the high dimension space coordinate system to a low space coordinate system. Using this method can significantly retrieve the main features or key points from a new system, such as using Scale-Invariant Features Transform (SIFT) which shows scaling the space to several lower dimension spaces and find the invariant points.

The advantage of using Dimensionality Reduction: it reduces the consumption of resources such as memory space and computational time.

The disadvantage of using Dimensionality Reduction: it might lose some crucial details and sometimes, it's fatal. For example, in an auto-pilot field, real-time computation with a small time lag is necessary. Most of the time, Dimensionality Reduction fits this position very well, but in some situations, some critical features are reduced or be recognized as different objects that can cause serious results due to some special weather conditions and it's fearful.

Q6. Show that for a 2*2 matrix A , with eigenvalues λ_1, λ_2

(a) det(A) = $\lambda_1 * \lambda_2$

Because eigenvalues are roots of the characteristic polynomial $p(\lambda) = \det(A - \lambda \mathbf{I}_n)$. We have:

$$\det(A - \lambda \mathbf{I_n}) = \begin{vmatrix} a_{11} - \boldsymbol{\lambda} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} - \boldsymbol{\lambda} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} - \boldsymbol{\lambda} \end{vmatrix} = \prod_{i=1}^{n} (\boldsymbol{\lambda_i} - \boldsymbol{\lambda})$$

In our situation, A is a 2 x 2 matrix, so n =2 ,$\det(A - \lambda \mathbf{I_2}) = \prod_{i=1}^{2} (\boldsymbol{\lambda_i} - \boldsymbol{\lambda})$. Let $\boldsymbol{\lambda} = 0$, we have:
$\det(A) = \prod_{i=1}^{2} \boldsymbol{\lambda_i} = \boldsymbol{\lambda_1} * \boldsymbol{\lambda_2}$.

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

According to the above equation, we compare the coefficients of $\boldsymbol{\lambda^{n-1}}$ of both sides.
The coefficient of $\boldsymbol{\lambda^{n-1}}$ of the determinant on the left side of the equation, we have:

$$(-1)^{n-1}(a_{11} + a_{22} + \cdots + a_{nn}) = (-1)^{n-1} trace(A)$$

The coefficient of $\boldsymbol{\lambda^{n-1}}$ of the determinant on the right side of the equation, we have:

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

In our situation, n =2. The equation is:

$$-(trace(A)) = (-1)^{n-1} \sum_{i=1}^{n} \boldsymbol{\lambda_i} = -(\boldsymbol{\lambda_1} + \boldsymbol{\lambda_2})$$
$$trace(A) = \boldsymbol{\lambda_1} + \boldsymbol{\lambda_2}$$

Q7.A rigid – body motion is a family of transformations that preserve the shape and size of objects. In general, any proper rigid- body transformation can be decomposed as a rotation followed by a transformation. Show that, for any two vectors u, v $\in$ R^3 :

(a) A rigid- body transformation g: $R^3 \rightarrow R^3$ preserves the norm of the vectors: $\|g * v\| = \|v\|$

Let $\boldsymbol{v} = (x_1, y_1, 1) - (x_2, y_2, 1) = (x_1 - x_2, y_1 - y_2, 0)$

$\boldsymbol{so}\ \|\boldsymbol{v}\| = (x_1 - x_2)^2 + (y_1 - y_2)^2$

Let $\mu = \boldsymbol{g} * \boldsymbol{v}$

The right side of the equation $\mu = \boldsymbol{g} * \boldsymbol{v}$ =

$$\begin{bmatrix} cos\theta & -sin\theta & tx \\ sin\theta & cos\theta & ty \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x_1 - x_2 \\ y_1 - y_2 \\ 0 \end{bmatrix} = \begin{bmatrix} cos\theta(x_1 - x_2) - sin\theta(y_1 - y_2) \\ sin\theta(x_1 - x_2) + cos\theta(y_1 - y_2) \\ 0 \end{bmatrix}$$

$$\|\boldsymbol{g} * \boldsymbol{v}\| = [cos\theta(x_1 - x_2) - sin\theta(y_1 - y_2)]^2 + [sin\theta(x_1 - x_2) + cos\theta(y_1 - y_2)]^2$$
$$= (x_1 - x_2)^2 + (y_1 - y_2)^2 = \|\boldsymbol{v}\|$$

(b) A rigid – body transformation g: $R^3 \rightarrow R^3$ preserves the cross product of two vectors:
(g * u)*(g * v) = g* (u * v)

Let $g = \begin{bmatrix} cos\theta & -sin\theta & tx \\ sin\theta & cos\theta & ty \\ 0 & 0 & 1 \end{bmatrix}$, let $u=(x_1, y_1, 1)$, $v = (x_2, y_2, 1)$.

Consider g * u and g * v equal to:

$$g * u = \left(\begin{bmatrix} cos\theta & -sin\theta & tx \\ sin\theta & cos\theta & ty \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} \right) = \begin{bmatrix} cos\theta x_1 - sin\theta y_1 + tx \\ sin\theta x_1 + cos\theta y_1 + ty \\ 1 \end{bmatrix} \text{ and}$$

$$g * v = \left(\begin{bmatrix} cos\theta & -sin\theta & tx \\ sin\theta & cos\theta & ty \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix}\right) = \begin{bmatrix} cos\theta x_2 - sin\theta y_2 + tx \\ sin\theta x_2 + cos\theta y_2 + ty \\ 1 \end{bmatrix}$$

The left side of the equation is **(g * u)*(g * v)**

$$\textbf{(g * u)*(g * v)} = \begin{vmatrix} \boldsymbol{i} & \boldsymbol{j} & \boldsymbol{k} \\ cos\theta x_1 - sin\theta y_1 + tx & sin\theta x_1 + cos\theta y_1 + ty & \mathbf{1} \\ cos\theta x_2 - sin\theta y_2 + tx & sin\theta x_2 + cos\theta y_2 + ty & \mathbf{1} \end{vmatrix} =$$

$$(sin\theta x_1 + cos\theta y_1 - sin\theta x_2 - cos\theta y_2)\ \boldsymbol{i} + (sin\theta x_1 cos\theta x_2 - sin\theta x_1 sin\theta y_2 + sin\theta x_1 tx + cos\theta x_2 cos\theta y_1 - sin\theta y_2 cos\theta y_1 + cos\theta y_1 tx + cos\theta x_2 ty - sin\theta y_2 ty + txty - cos\theta x_1 + sin\theta y_1 - tx)\ \boldsymbol{j} + (cos\theta x_1\, sin\theta x_2 + cos\theta x_1\, cos\theta y_2 + cos\theta x_1\, ty - sin\theta y_1 sin\theta x_2 - sin\theta y_1 cos\theta y_2 - sin\theta y_1 ty + sin\theta x_2 tx + cos\theta y_2 tx - sin\theta x_1 cos\theta x_2 + sin\theta x_1 sin\theta y_2 - sin\theta x_1 tx - cos\theta y_1 cos\theta x_2 + sin\theta y_2 cos\theta y_1 - cos\theta y_1 tx - cos\theta x_2 ty + sin\theta y_2 ty)\ \boldsymbol{k}$$

The right side of the equation is g* (u * v)

$$u * v = \begin{vmatrix} \boldsymbol{i} & \boldsymbol{j} & \boldsymbol{k} \\ x_1 & y_1 & \mathbf{1} \\ x_2 & y_2 & \mathbf{1} \end{vmatrix} = (y_1 - y_2)\boldsymbol{i} + (x_1 y_1 - x_1)\boldsymbol{j} + (x_1 y_2 - x_2 y_1)\boldsymbol{k}$$

$$g* (u * v) = \begin{bmatrix} cos\theta & -sin\theta & tx \\ sin\theta & cos\theta & ty \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} (y_1 - y_2)\boldsymbol{i} \\ (x_1 y_1 - x_1)\boldsymbol{j} \\ (x_1 y_2 - x_2 y_1)\boldsymbol{k} \end{bmatrix} =$$

$$(sin\theta x_1 + cos\theta y_1 - sin\theta x_2 - cos\theta y_2)i + (sin\theta x_1 cos\theta x_2 - sin\theta x_1 sin\theta y_2 + sin\theta x_1 tx + cos\theta x_2 cos\theta y_1 - sin\theta y_2 cos\theta y_1 + cos\theta y_1 tx + cos\theta x_2 ty - sin\theta y_2 ty + txty - cos\theta x_1 + sin\theta y_1 - tx)\ j + (cos\theta x_1\, sin\theta x_2 + cos\theta x_1\, cos\theta y_2 + cos\theta x_1\, ty - sin\theta y_1 sin\theta x_2 - sin\theta y_1 cos\theta y_2 - sin\theta y_1 ty + sin\theta x_2 tx + cos\theta y_2 tx - sin\theta x_1 cos\theta x_2 + sin\theta x_1 sin\theta y_2 - sin\theta x_1 tx - cos\theta y_1 cos\theta x_2 + sin\theta y_2 cos\theta y_1 - cos\theta y_1 tx - cos\theta x_2 ty + sin\theta y_2 ty)\ k$$

Q8. Design and implement a way of verifying experimentally that repeatedly applying an averaging filter approximates Gaussian smoothing. Note: this problem will require you to explain your design in writing, then implement it and test it via Matlab.

The original image

The approximated Gaussian smoothing

Gaussian smooting

These three images are the result of applying an averaging filter multiple times to approximate Gaussian smoothing. The first image is the original picture, the second one is the approximated one with 4 times convolution and the third figure is an comparasion that is generated by a Gaussian smooting filter.

I used a box filter to perform this simulation. The filter is a matrix computed by two 1D vectors. The column vector has fluctuant values from larger to smaller then to larger again. The row vector is an averaging vector. After multiplying these two vectors, I normalized it and let the sum equal to 1. This's the final filter which can be used for convolution. I applied this filter 4 times, the results as below:

The 1st time

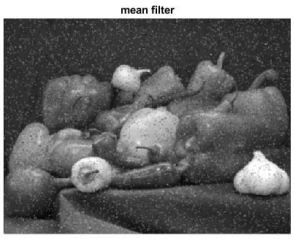
The 2nd time

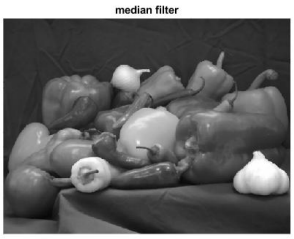
The 3rd time

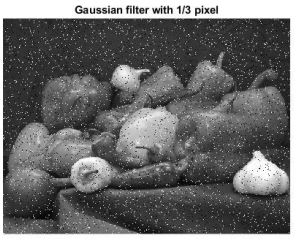
The 4th time

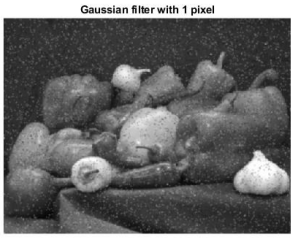
Q9.Matlab programming:

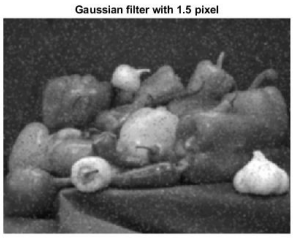
Part A:

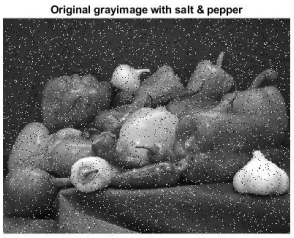






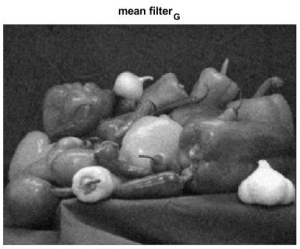


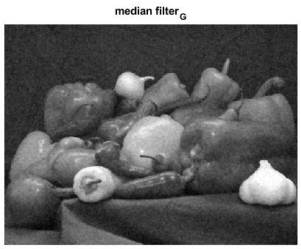


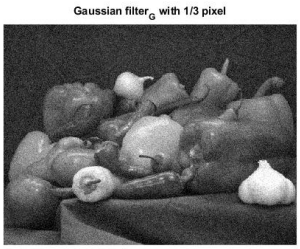


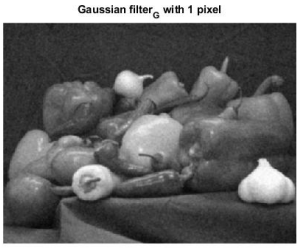
These images were performed by adding “salt & pepper” noise then applied five different filters on them. The image in the bottom right corner is the original picture without filtering, only added noise. To compare the results of these images. I found that the image with a median filter has the best result which hardly to see any noise at all. The worst result comes from a Gaussian filter with 1/3 pixel. Instead of disappearing, the noise tends to sharpen. The results from the mean filter and the Gaussian filter with 1 pixel are similar. The image was operated by the Gaussian filter with 1.5 pixels which can see the result as same as the mean filter’s and the Gaussian filter with 1 pixel’s, but the noise is blurred a little bit.

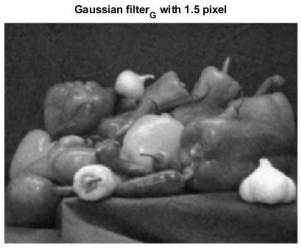
Part B:

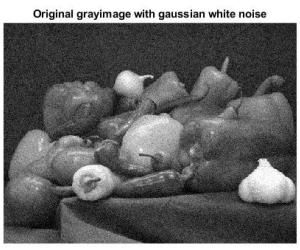










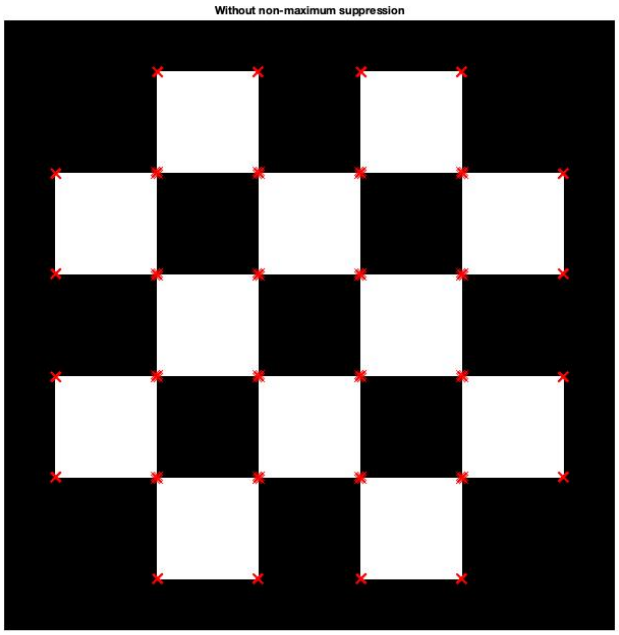


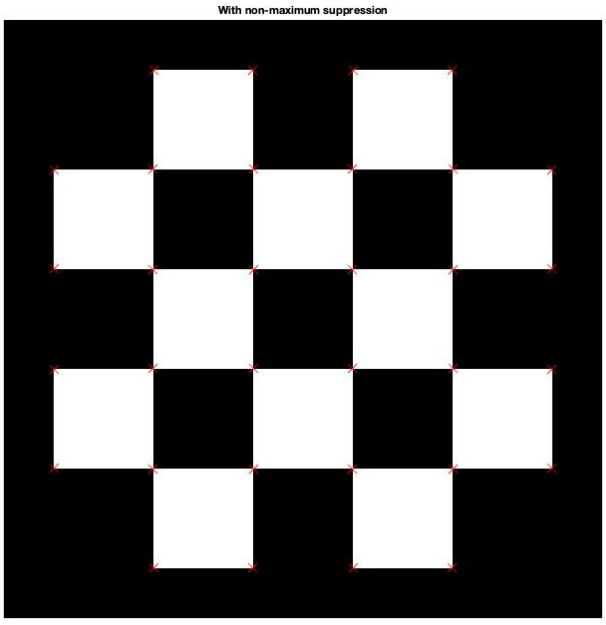
These images were performed by adding Gaussian white noise with mean 0 and $\delta = 1/256$ in the [0,1] range then applied five different filters on them. The image in the bottom right corner is

the original picture without filtering, only added noise. To compare the results of these images. I found that all results are similar that we can see the noise clearly. The worse result comes from the Gaussian filter with 1/3 pixel which is similar to the original image without filtering. The image performed by a Gaussian filter with 1.5 pixels is blurred, and the three remaining images are the same.

Q10.Matlab programming: write your own implements of the Harris Corner Detector algorithm.

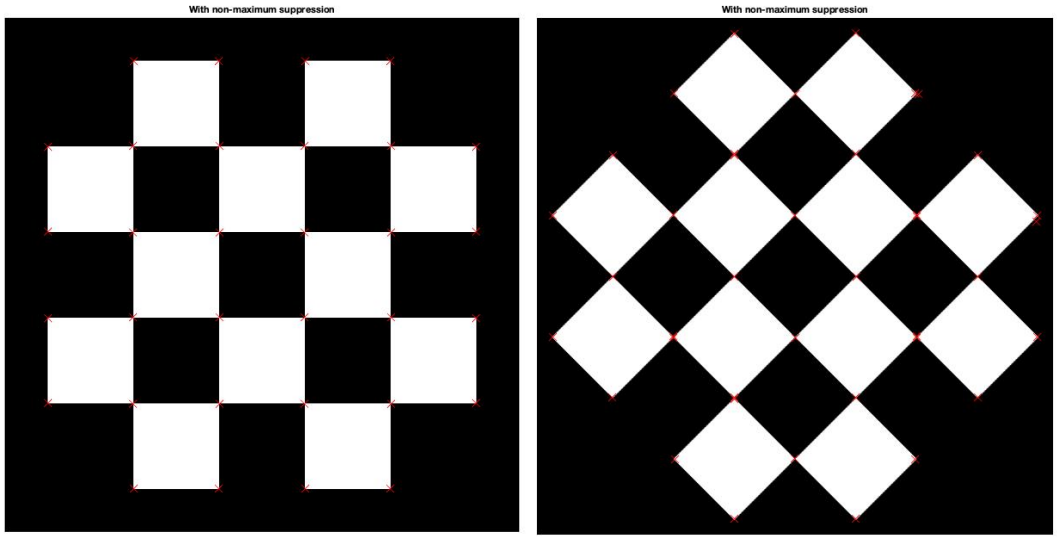
Part A:



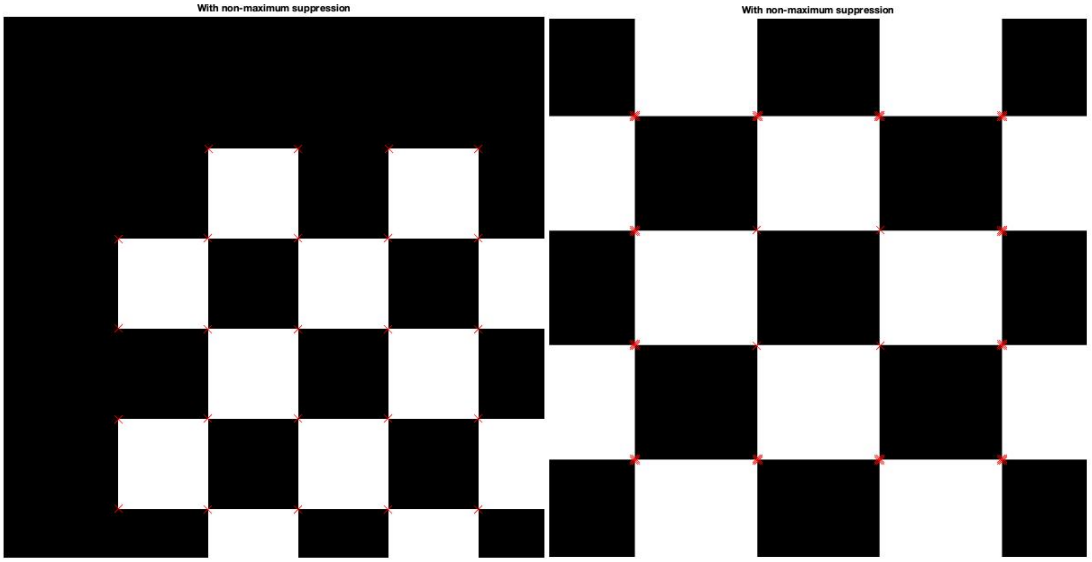


Part B:

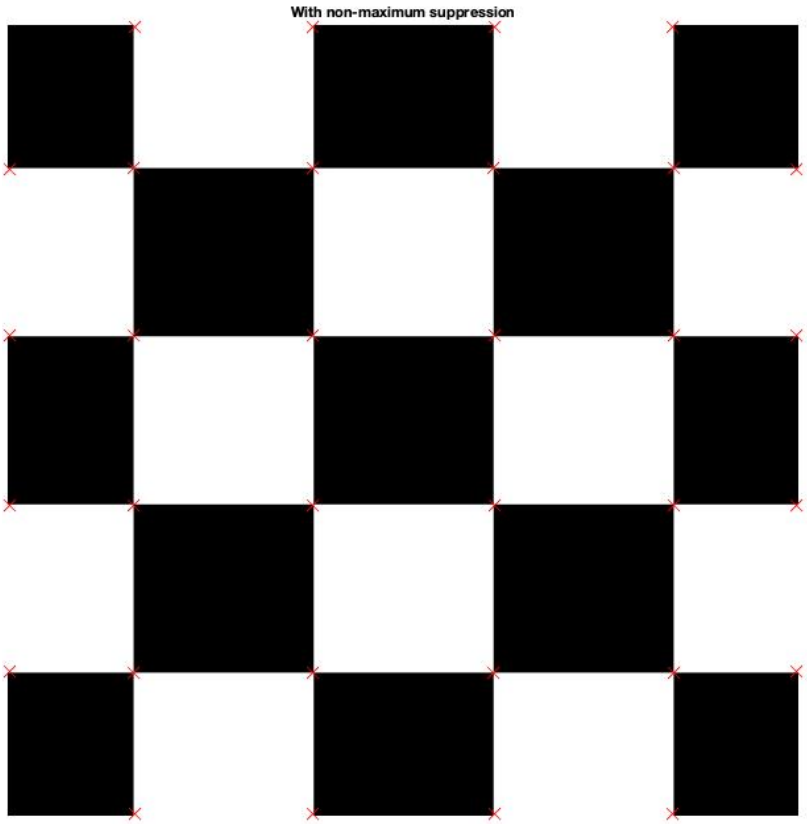
All results in this section under the same parameters' condition. The variables values are: Threshold = 100; w = 3 and suppression = true.



The first image is the original image. The corner detector detected all corners. The second image is the original image that made a 45-degree clockwise rotation. The results are similar. The corner detector detected all corners in both images.

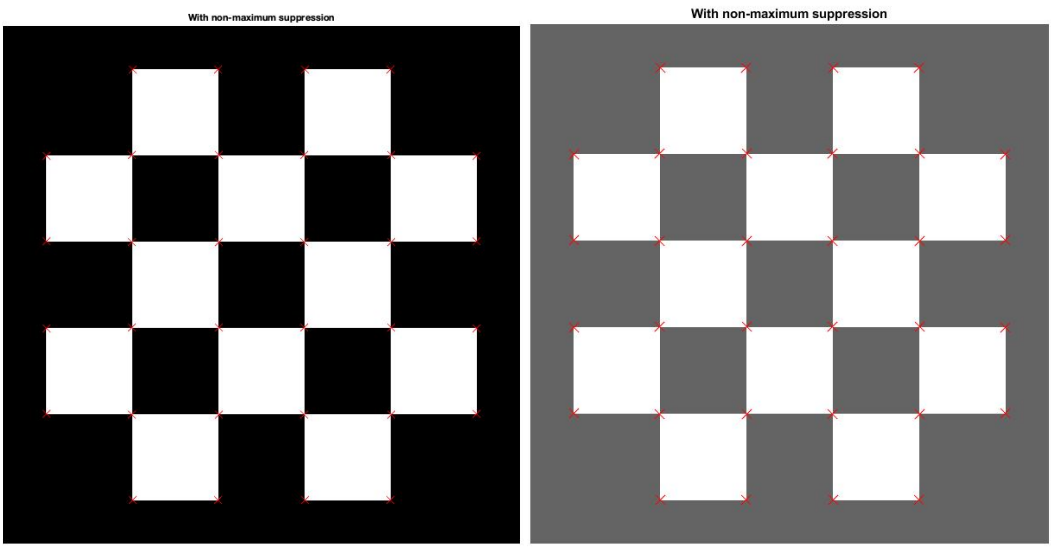


These two images are translated and scaled versions. In the translated image, all corners are detected except the corners at the edge of the picture. This phenomenon also exists in the scaled version. In the scaled image, the corner detector missed the corners at the edge and marked multiple detections of all corners. It might be caused by the threshold's value and w's value. I experimented to increase w's value from 3 to 7 and keep the threshold's value the same. It succeeded in detection. The result shows below:



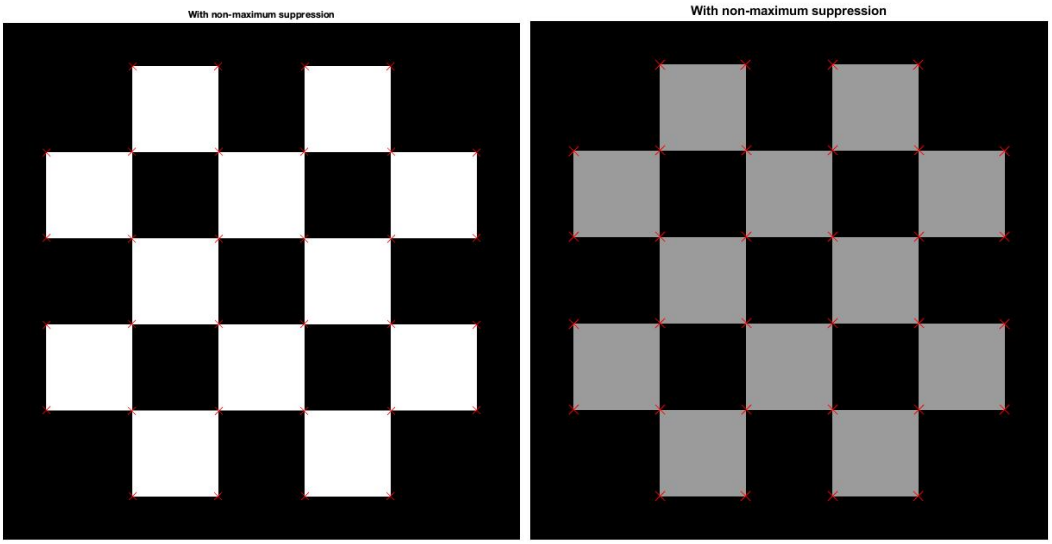
Part C

1. **Prepare a brighter version of the original image by adding a constant positive offset 100 to all pixel values.**



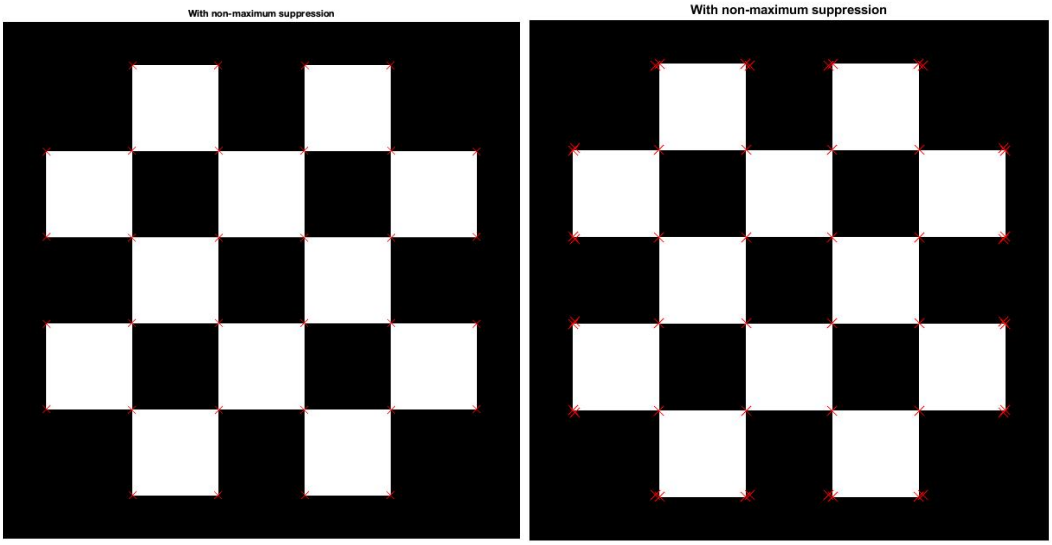
The left side image is the original image. The right side image is the fixed version. The corner detector works well on both images. All corners are detected correctly.

2. **Prepare a darker version of the original image by adding a constant negative offset -100 to all pixel values.**



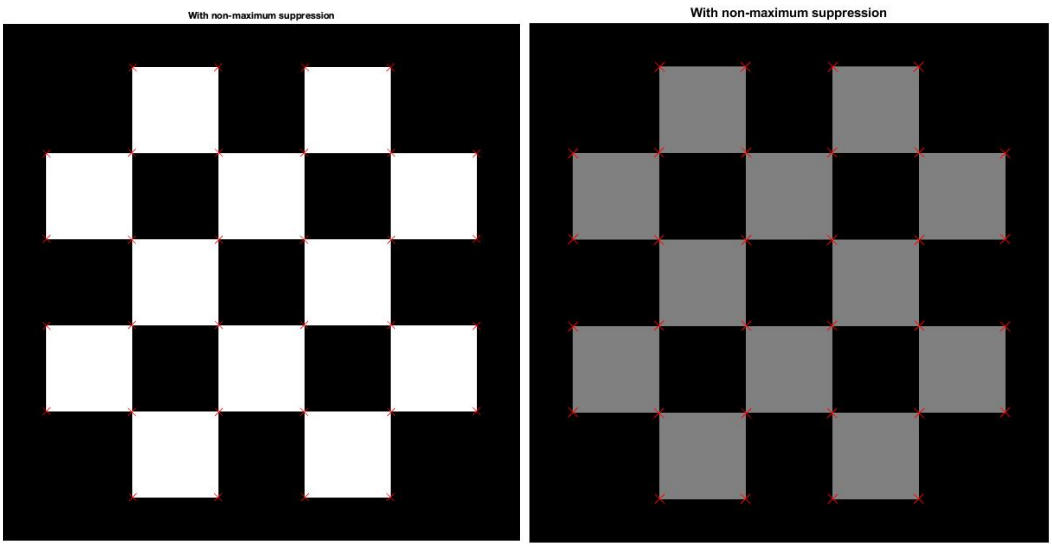
The left side image is the original image. The right side image is the fixed version. The corner detector works well on both images. All corners are detected correctly.

3. **Prepare a brighter version of the original image by multiplying a constant positive offset 10 to all pixel values.**



The left side image is the original image. The right side image is the fixed version. The corner detector correctly detected some corners which are located around the center of the image. The corners near the edge are multiple detected. In the original picture, it has 32 corners which are detected correctly, but in the fixed picture, the correct detected corners are 16. The percentage of matching features is 50%.

4. **Prepare a darker version of the original image by multiplying a constant negative offset 0.5 to all pixel values.**

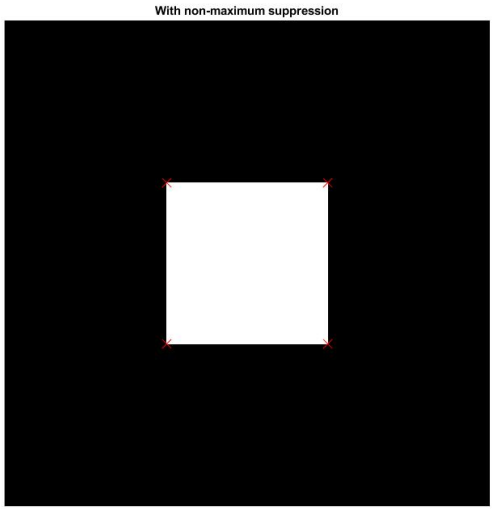


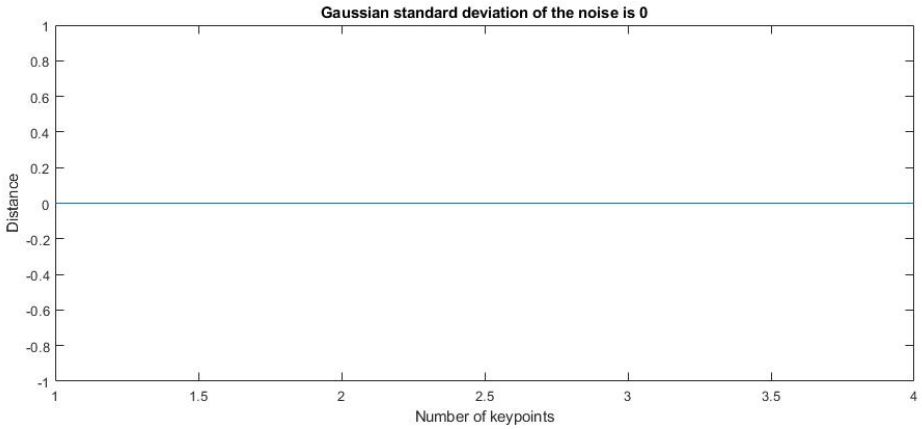
The left side image is the original image. The right side image is the fixed version. The corner detector works well on both images. All corners are detected correctly.

Part D

All results in this section under the same parameters' condition. The variables values are: Threshold = 0.6*10^10; w = 5 and suppression = true. Apply Gaussian noise to the same synthetic image.

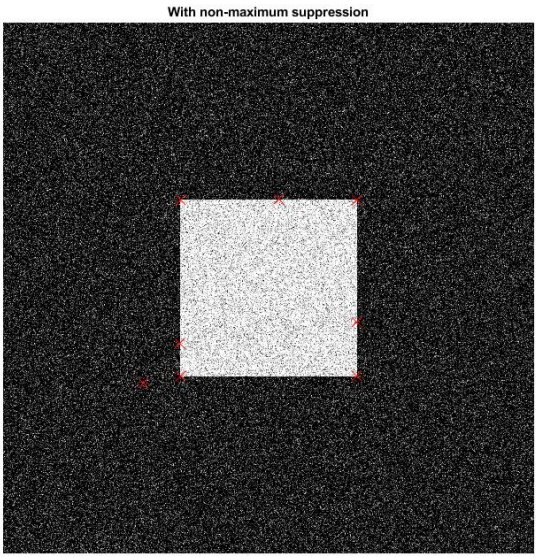
1. Standard (without the Gaussian noise)

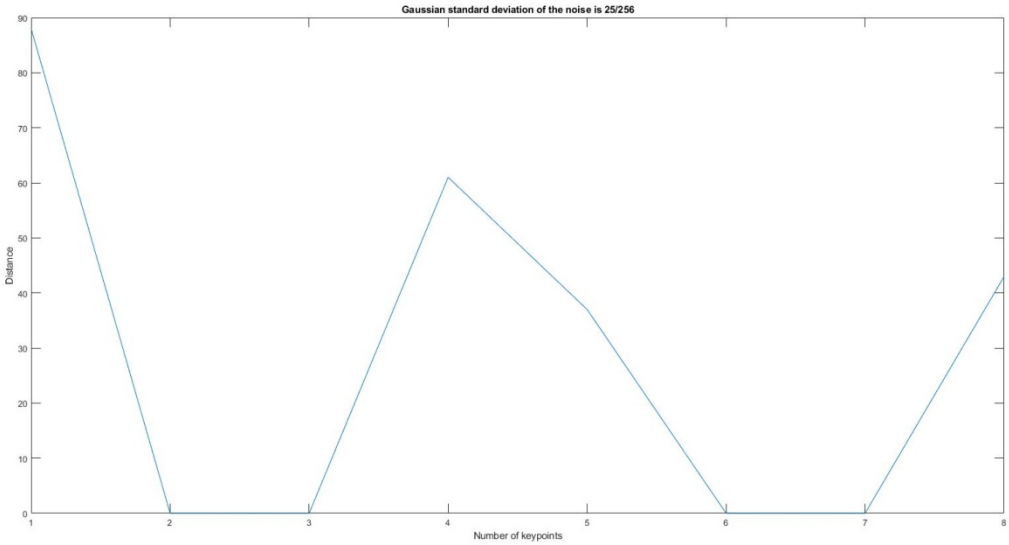




0 corners are missing and 0 corners are wrongly detected.

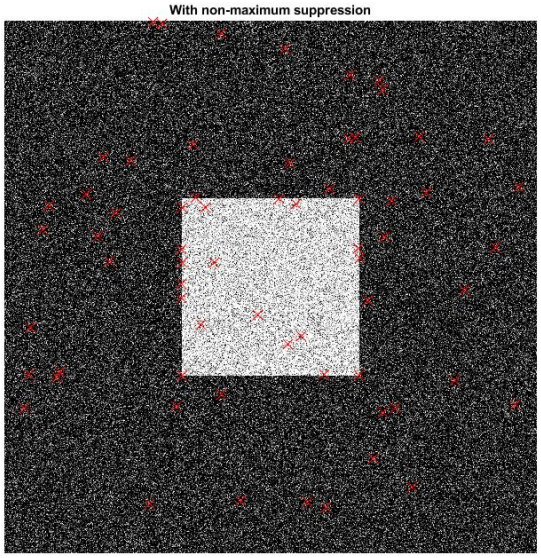
2. With Gaussian standard deviation of the noise is 25/256

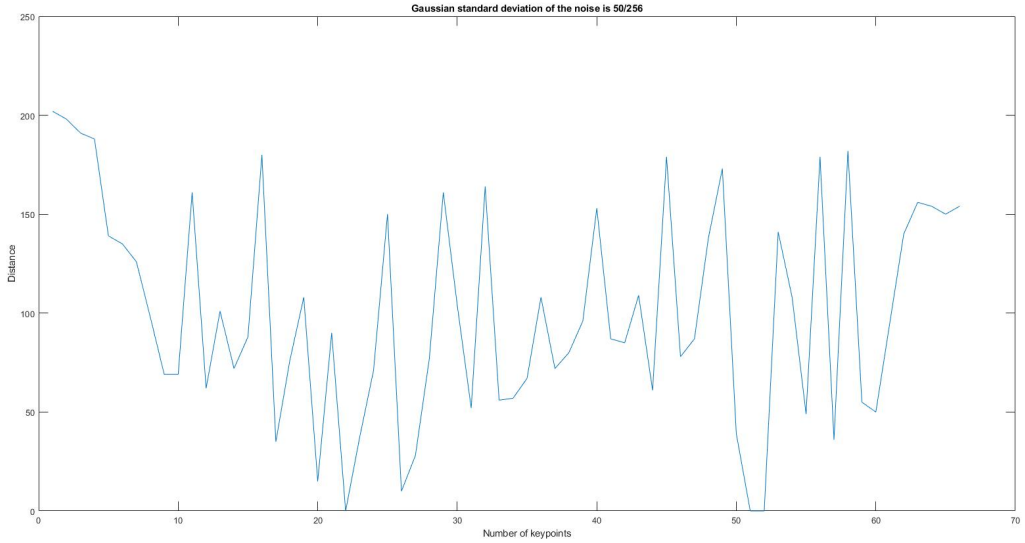




0 corners are missing and 4 corners are wrongly detected.

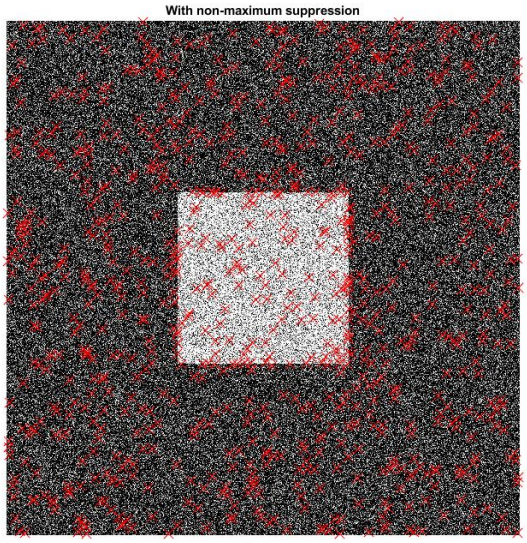
3. With Gaussian standard deviation of the noise is 50/256

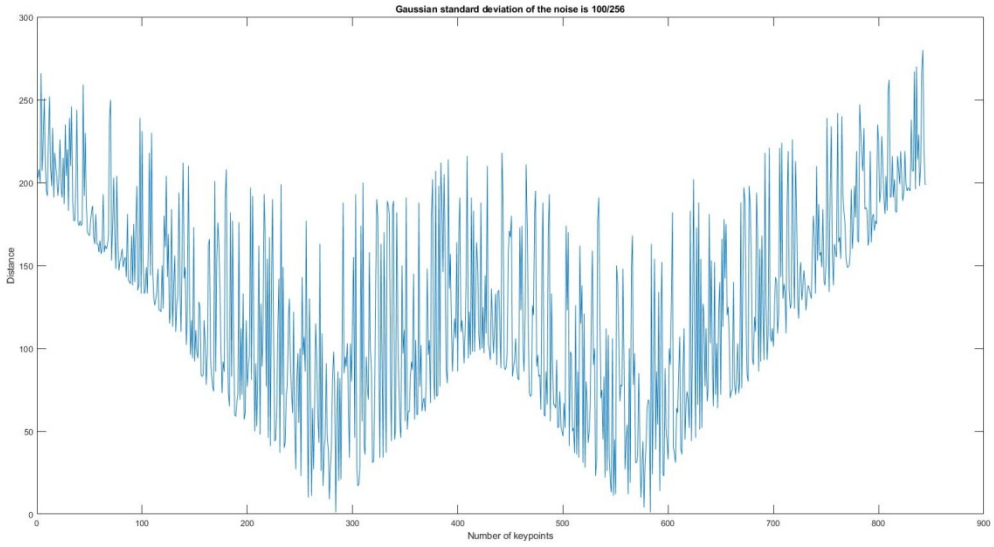




1 corner is missing and 63 corners are wrongly detected.

5. With Gaussian standard deviation of the noise is 100/256





2 corners are missing and 843 corners are wrongly detected