Χρήστος – Αλέξανδρος Δαρδαμπούνης ΑΕΜ 10335

Εργασία 3: Αποκατάσταση Παραμορφωμένης Εικόνας με φίλτρα Wiener

Σκοπός της παρούσας εργασίας είναι η ανάκτηση της αρχικής εικόνας που έχει παραμορφωθεί από το φίλτρο καθώς και από λευκό Gaussian θόρυβο με την χρήση του φίλτρου Wiener καθώς επίσης και η σύγκριση αυτής της μεθόδου με την περίπτωση όπου για την ανάκτηση της εικόνας χρησιμοποιείται η αντίστροφη συνάρτηση μεταφοράς του συστήματος για την ανάκτηση της εικόνας.

Η υλοποίηση του φίλτρου **Wiener** έγινε με **δύο** τρόπους. Ο πρώτος βασίστηκε σε αυτά που έδινε η εργασία για την πραγματοποίηση της συνέλιξης του φίλτρου με την εικόνα εισόδου. Ο τρόπος με τον οποίο πραγματοποιείται η συνέλιξη έτσι όπως δίνεται στην εκφώνηση είναι σαν να κάνουμε την κυκλική συνέλιξη (διότι χρησιμοποιείται σε **mode = 'wrap'**). Επομένως για να μπορέσουμε να πάρουμε σωστά αποτελέσματα, με τον τρόπο που ορίζεται στην εκφώνηση, θα πρέπει να κάνουμε zero padding **μόνο** στον φίλτρο που πραγματοποιεί το **motion blur**. Αφού πραγματοποιήσουμε αυτό το zero padding στην συνέχεια βρίσκουμε τον **DFT** του **H** και με την βοήθεια αυτού φτιάχνουμε το φίλτρο **G** μέσω του οποίου θα ανακτήσουμε την αρχική εικόνα x που μας δίνεται στην εκφώνηση πραγματοποιώντας τον πολλαπλασιασμό του **G** με τον **DFT** του **Y**. Παρακάτω φαίνεται ο κώδικας του οποίου η διαδικασία περιεγράφηκε προηγουμένως καθώς επίσης και το demo μέσω του οποίου παράγω τις αντίστοιχες εικόνες που ζητούνται στην εκφώνηση καθώς επίσης και την καμπύλη του **J** για την εύρεση του βέλτιστου **SNR(K)**:

```python
import numpy as np


3 usages
def my_wiener_filter(y, h, K):

    M1, N1 = y.shape # getting the dimensions of y image
    M2, N2 = h.shape # getting the dimensions of h filter

    h1 = np.pad(h, ((0, M1 - M2), (0, N1 - N2)), mode = 'constant', constant_values = 0) # zero padding the H filter to match the input image dimensions
    # since the convolution that is performed is on mode='wrap'
    H = np.fft.fft2(h1) # finding the FFT of the padded h filter

    H_conj = np.conj(H) # finding the conjugate filter of the FFT H
    H_norm = np.abs(H)**2 # finding the squared norm of the FFT of the H

    G = H_conj / (H_norm + (1 / K)) # creating the G Wiener filter based on the theory

    Y = np.fft.fft2(y) # finding the FFT transform of the input image

    X_hat = G*Y # calculating the result of the Wiener filter on the input image
    x_hat = np.fft.ifft2(X_hat) # finding the inverse FFT of the output of the filter

    x_hat = np.real(x_hat[:y.shape[0], :y.shape[1]]) # keeping only the real part of the inverse FFT and forcing it to have dimensions
    # equal with the input image

    return x_hat
```

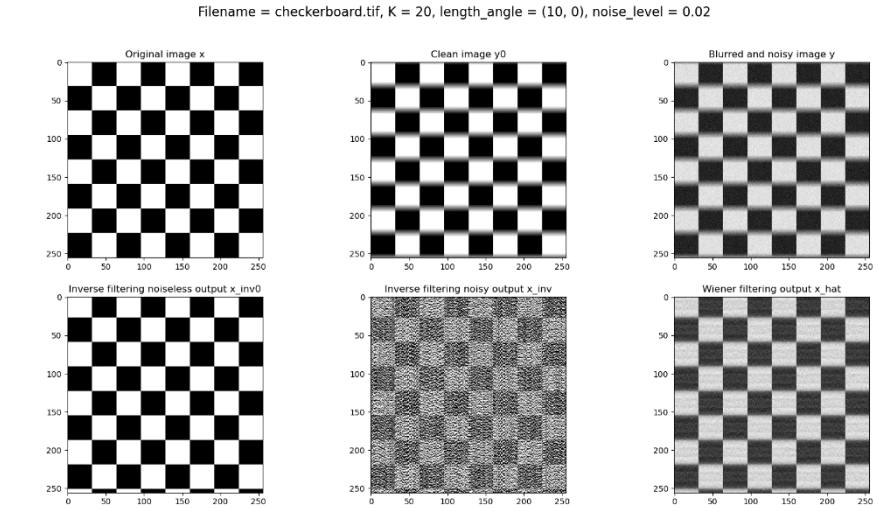
Χρήστος – Αλέξανδρος Δαρδαμπούνης ΑΕΜ 10335

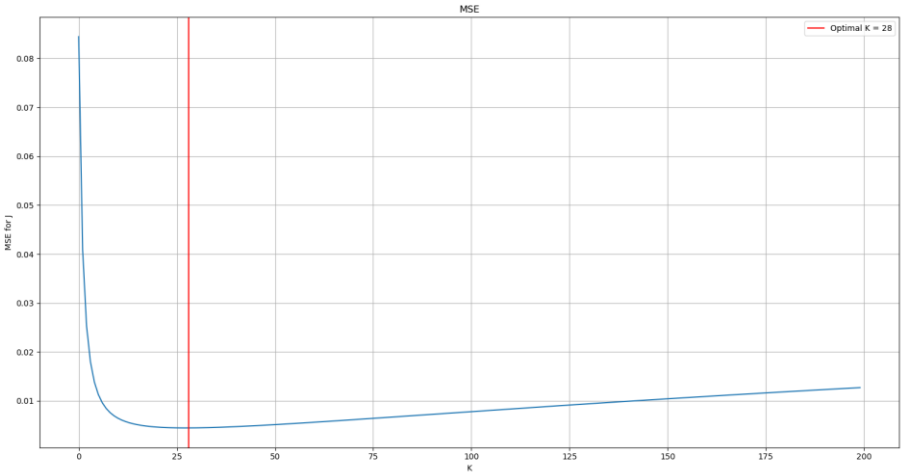
```python
import numpy as np
import matplotlib.pyplot as plt
import cv2
import hw3_helper_utils
from scipy.ndimage import convolve
from wiener_filtering import my_wiener_filter

images = ['checkerboard.tif', 'cameraman.tif']
K = 20
length_angle = [(10, 0), (20, 30)]
noise_levels = [0.02, 0.2]

for filename in images:
    for value in length_angle:
        for n in noise_levels:

            x = cv2.imread(filename, cv2.IMREAD_GRAYSCALE)  # reading the input image
            x = x.astype(np.float32) / 255.0

            v = n * np.random.randn(*x.shape)  # creating the noise
            h = hw3_helper_utils.create_motion_blur_filter(value[0], value[1])  # creating the motion blur filter

            y0 = convolve(x, h, mode="wrap")  # convolving the input image with the motion blur image
            y = y0 + v  # adding WGN to the result of the convolution

            M1, N1 = y.shape
            M2, N2 = h.shape

            x_hat = my_wiener_filter(y, h, K) # getting the result of the wiener filter

            h1 = np.pad(h, ((0, M1 - M2), (0, N1 - N2)), mode = 'constant', constant_values = 0)  # padding the h filter like I did in
            # my_wiener_filter() function

            H = np.fft.fft2(h1)  # performing the FFT transform on the padded filter
            Y = np.fft.fft2(y)  # performing the FFT transform on the result of the convolution with WGN
            mask = np.abs(H) < 1e-8  # creating a mask to find all the values of H which have a value close to zero
            H_inv = 1 / H  # creating the H_inv filter
            H_inv[mask] = 0  # frequencies that are zero fo

            X_inv = Y * H_inv  # applying the H_inv filter
            x_inv = np.fft.ifft2(X_inv) # finding the result of the filter

            x_inv = np.real(x_inv[:y.shape[0], :y.shape[1]])  # keeping only the real part with dimensions equal to y

            Y0 = np.fft.fft2(y0)  # finding the FFT of the y0 which is free of noise

            X_inv0 = Y0 * H_inv  # applying the filter to the Y0
            x_inv0 = np.fft.ifft2(X_inv0)

            x_inv0 = np.real(x_inv0[:y.shape[0], :y.shape[1]])  # keeping only the real part with dimensions equal to y
            # creating a list for J for K in range from 1 to 200
            J = []

            for k in range(1, 201):
                x_hat1 = my_wiener_filter(y, h, k)
                J.append(np.mean((x_inv0 - x_hat1) ** 2))

            # clipping the images so their values lie in the [0, 1]
            x_hat = np.clip(x_hat, 0, 1)
            x_inv = np.clip(x_inv, 0, 1)
            x_inv0 = np.clip(x_inv0, 0, 1)

            # plotting the desired images
            fig, axs = plt.subplots(nrows=2, ncols=3)
            fig.suptitle(f'Filename = {filename}, K = {K}, length_angle = {value}, noise_level = {n}', fontsize=16)
            axs[0][0].imshow(x, cmap='gray')
            axs[0][0].set_title("Original image x")
            axs[0][1].imshow(y0, cmap='gray')
            axs[0][1].set_title("Clean image y0")
            axs[0][2].imshow(y, cmap='gray')
            axs[0][2].set_title("Blurred and noisy image y")
            axs[1][0].imshow(x_inv0, cmap='gray')
            axs[1][0].set_title("Inverse filtering noiseless output x_inv0")
            axs[1][1].imshow(x_inv, cmap='gray')
            axs[1][1].set_title("Inverse filtering noisy output x_inv")
            axs[1][2].imshow(x_hat, cmap='gray')
            axs[1][2].set_title("Wiener filtering output x_hat")
            plt.show()

            #  plotting the J curve and finding the best K which minimises the MSE
            plt.plot(J)
            plt.ylabel('MSE for J')
            plt.xlabel('K')
            plt.title('MSE')
            minJ = np.argmin(J) + 1
            plt.axvline(minJ, color = 'r', label=f'Optimal K = {minJ}')
            plt.legend()
            plt.grid()
            plt.show()
```

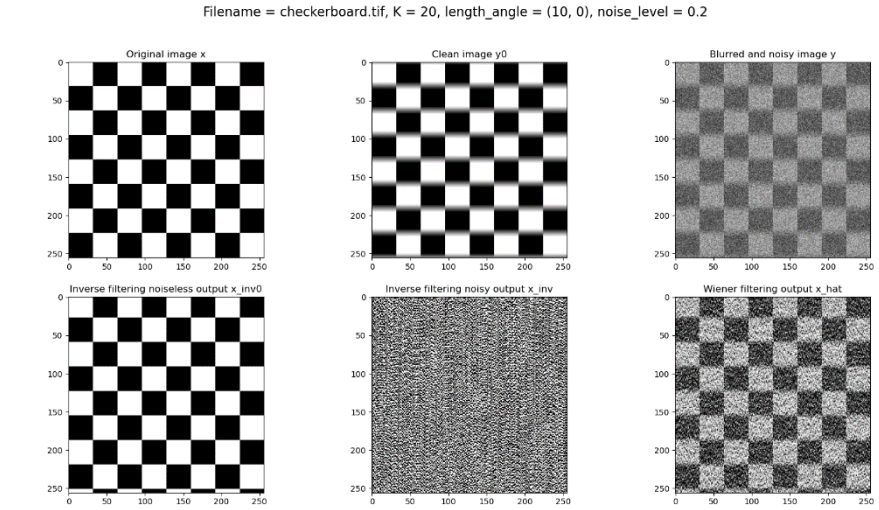
Επειδή για κάποιες συχνότητες βγαίνουν μηδενικά τα στοιχεία του **DFT** του **H,** θεωρώ πως αυτές οι συχνότητες χάνονται για πάντα οπότε τις βάζω μηδέν και στα αντίστοιχα στοιχεία του αντιστρόφου του **H.**

Χρήστος – Αλέξανδρος Δαρδαμπούνης ΑΕΜ 10335

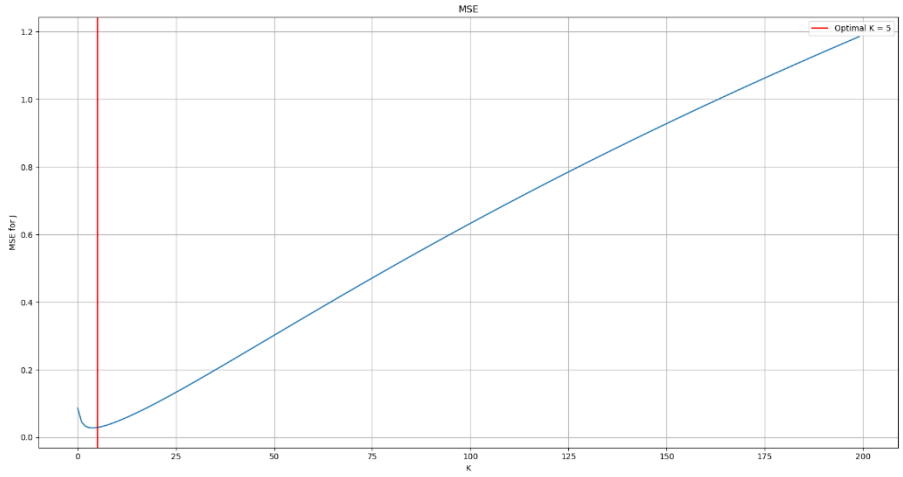
Τα αποτελέσματα και για τις δύο εικόνες με βάση αυτή την υλοποίηση του φίλτρου φαίνονται παρακάτω:



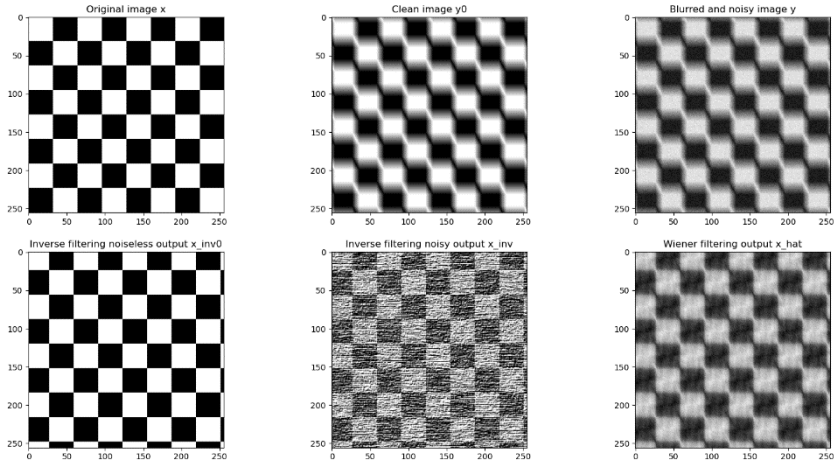


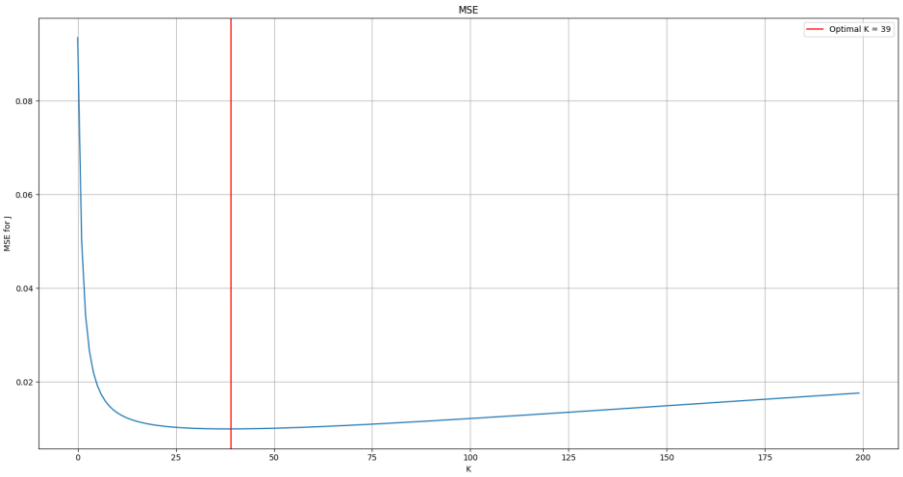


Χρήστος – Αλέξανδρος Δαρδαμπούνης ΑΕΜ 10335

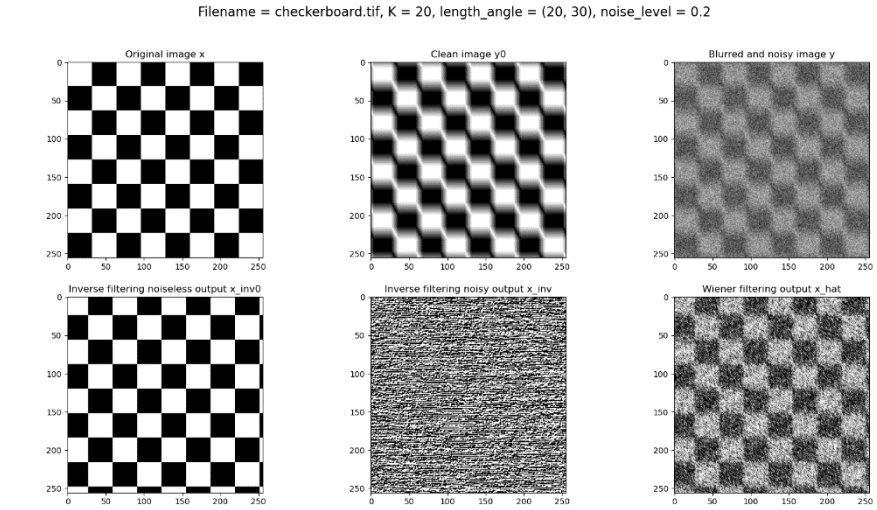


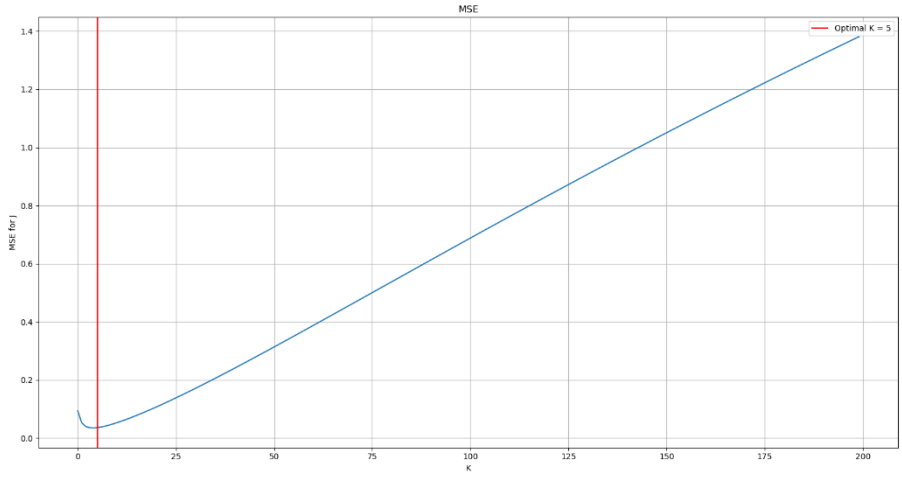
Filename = checkerboard.tif, K = 20, length_angle = (20, 30), noise_level = 0.02



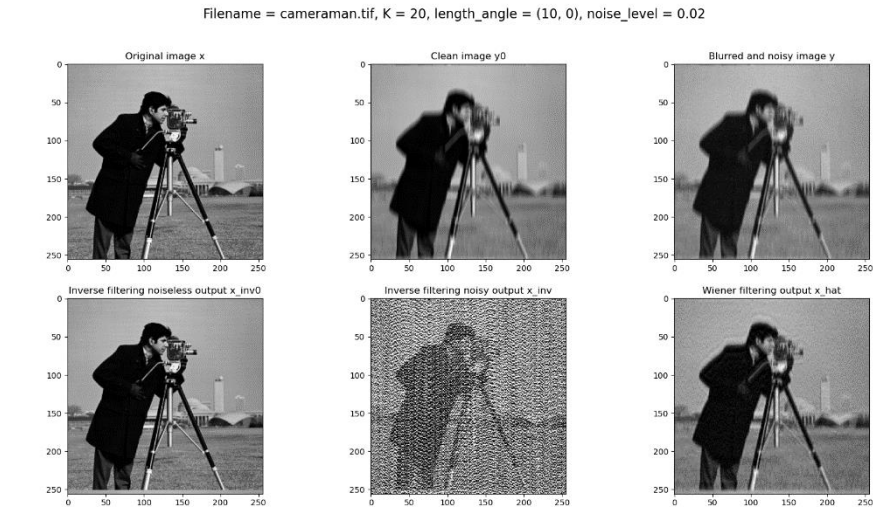


Χρήστος – Αλέξανδρος Δαρδαμπούνης ΑΕΜ 10335

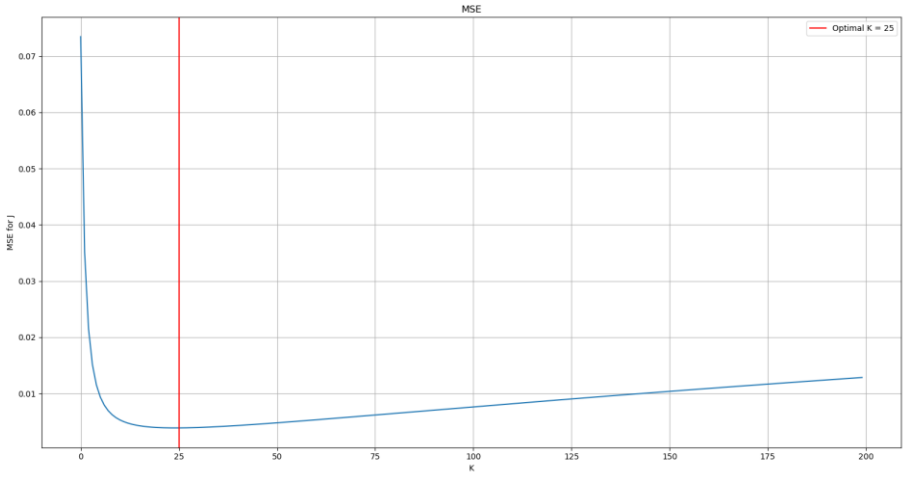




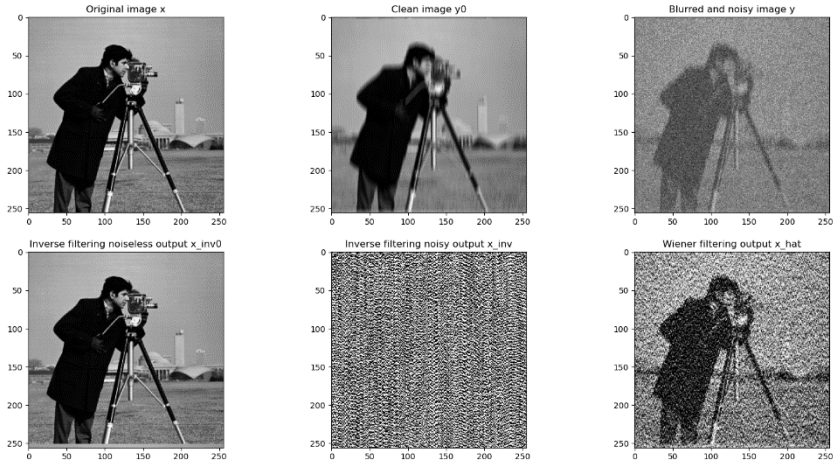
Παρακάτω ακολουθούν τα αντίστοιχα γραφήματα για την δεύτερη εικόνα που μας δόθηκε:

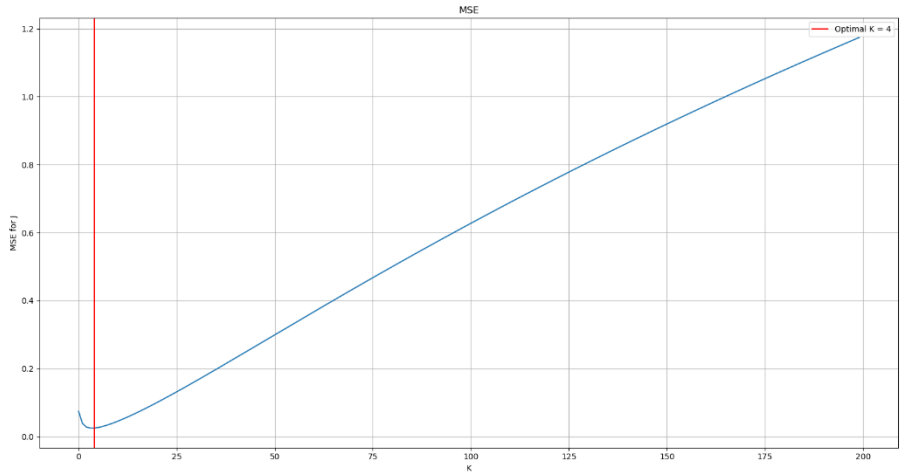


Χρήστος – Αλέξανδρος Δαρδαμπούνης ΑΕΜ 10335



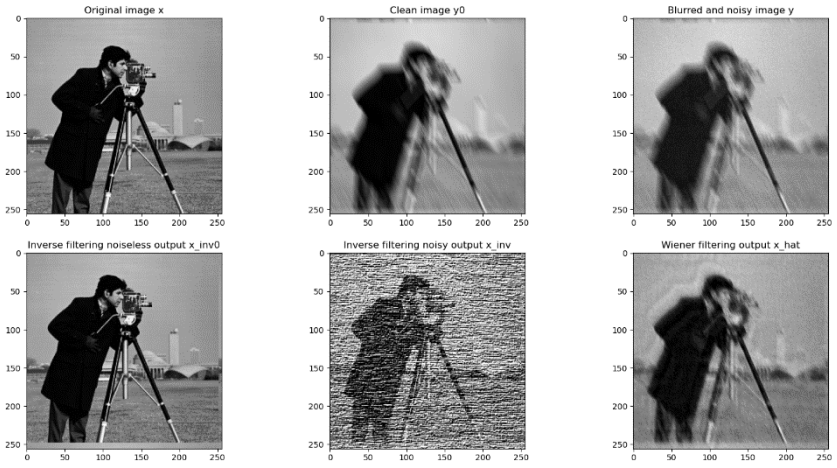
Filename = cameraman.tif, K = 20, length_angle = (10, 0), noise_level = 0.2

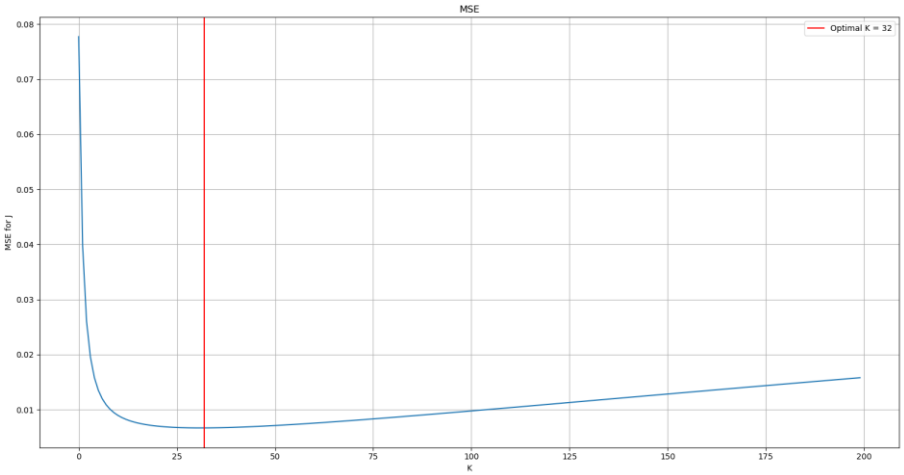




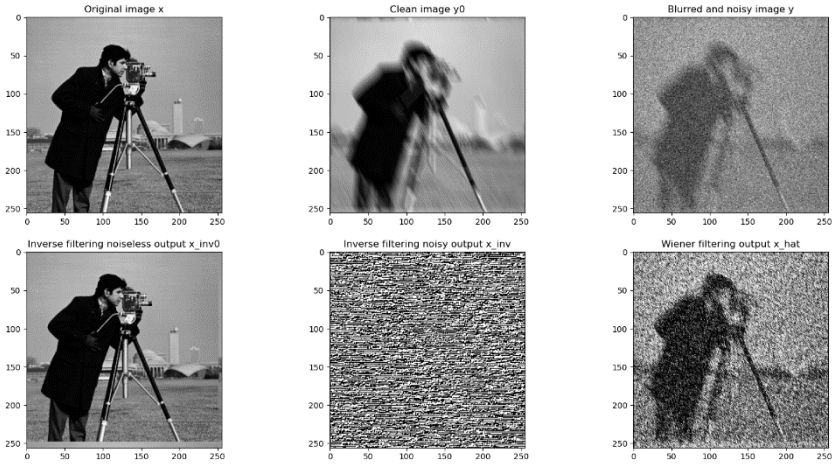
Χρήστος – Αλέξανδρος Δαρδαμπούνης ΑΕΜ 10335

Filename = cameraman.tif, K = 20, length_angle = (20, 30), noise_level = 0.02

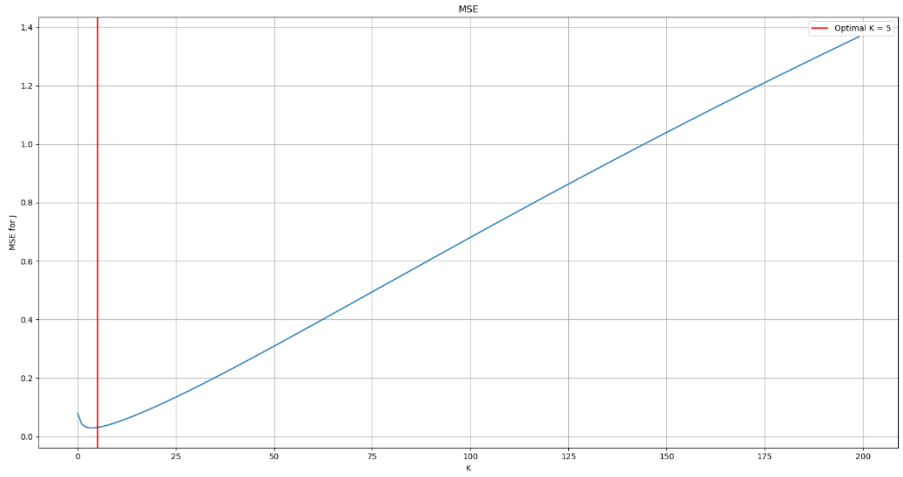




Filename = cameraman.tif, K = 20, length_angle = (20, 30), noise_level = 0.2



Χρήστος – Αλέξανδρος Δαρδαμπούνης ΑΕΜ 10335



Από τα παρακάτω γραφήματα είναι αρκετά φανερό πως με το φίλτρο Wiener μπορούμε να ανακτήσουμε σε αρκετά ικανοποιητικό βαθμό την αρχική μας εικόνα ακόμα και σε περιβάλλον υψηλού λευκού Gaussian θορύβου, ενώ επίσης φαίνεται πως διώχνουμε και σε αρκετά μεγάλο βαθμό την επίδραση του motion blur που εισάγει το φίλτρο **h** στην αρχική μας εικόνα. Επίσης, παρατηρούμε και αυτό που αναμέναμε για τις **x_inv0** και **x_inv,** δηλαδή όταν η εικόνα μας είναι απαλλαγμένη από θόρυβο τότε πολλαπλασιάζοντας με το αντίστροφο του φίλτρου έχουμε πλήρη ανάκτηση της εικόνας χωρίς καμία παραμόρφωση. Ωστόσο, με την παρουσία θορύβου άμα πολλαπλασιάσουμε με το αντίστροφο του φίλτρου ενδέχεται να ενισχύσουμε τον θόρυβο και έτσι να μην μπορούμε με τίποτα να ανακτήσουμε την αρχική μας εικόνα, γεγονός που παρατηρείται και στις παραπάνω εικόνες όπου σε περιβάλλον υψηλού θορύβου έχουμε χάσει πλήρως την εικόνα και έχει κυριαρχήσει ο θόρυβος. Για την παραγωγή των εικόνων επιλέχθηκε **K=20** το οποίο βρίσκεται πολύ κοντά στην γειτονιά όλων των βέλτιστων **K** που προκύπτουν από τα παραπάνω γραφήματα τα οποία ελαχιστοποιούν το **MSE.**

Για την παραγωγή της καμπύλης του **J** χρησιμοποιήθηκε όπως υποδεικνύεται το **x_inv0** λόγω της ολίσθησης που έχει σε σχέση με την αρχική εικόνα **x,** με την παραδοχή ότι είναι πολύ καλό αντίγραφο της **x.**

Χρήστος – Αλέξανδρος Δαρδαμπούνης ΑΕΜ 10335

Ο δεύτερος τρόπος με τον οποίο υλοποιώ το φίλτρο Wiener είναι χρησιμοποιώντας την **convolve2d** για την συνέλιξη της εικόνας εισόδου **x** και του φίλτρου **h** που δημιουργεί το motion blur. Με αυτό τον τρόπο πρέπει να κάνουμε zero padding και στο φίλτρο **h** αλλά και στην εικόνα εισόδου **y** έτσι ώστε οι διαστάσεις τους να γίνουν

$$(\boldsymbol{M_1 + M_2 - 1})\boldsymbol{x}(\boldsymbol{N_1 + N_2 - 1})$$

Αφού αποκτήσουν αυτές τις διαστάσεις, εφαρμόζουμε τον **DFT** και στα δύο σήματα και βρίσκουμε όπως και προηγουμένως το φίλτρο **G** που χρειάζεται για την αποκατάσταση της εικόνας από το **motion blur** και από τον θόρυβο. Παρακάτω φαίνεται ο κώδικας του οποίου η διαδικασία περιεγράφηκε προηγουμένως καθώς επίσης και το demo μέσω του οποίου παράγω τις αντίστοιχες εικόνες που ζητούνται στην εκφώνηση καθώς επίσης και την καμπύλη του **J** για την εύρεση του βέλτιστου **SNR(K)**:

```python
import numpy as np


def my_wiener_filter1(y, h, K):

    M1, N1 = y.shape# getting the dimensions of y image
    M2, N2 = h.shape# getting the dimensions of h filter
    Mx, Nx = (M1 - M2 + 1, N1 - N2 + 1) # finding the dimensions of the output of the Wiener filter

    # padding the h filter so it has dimensions equal to (M1 + M2 - 1)x(N1 + N2 - 1)
    h1 = np.pad(h, ((0, y.shape[0] - 1), (0, y.shape[1]- 1)), mode='constant', constant_values = 0)
    # padding the y filter so it has dimensions equal to (M1 + M2 - 1)x(N1 + N2 - 1)
    y1 = np.pad(y, ((0, h.shape[0] - 1), (0, h.shape[1] - 1)), mode='constant', constant_values  = 0)

    H = np.fft.fft2(h1)# finding the FFT of the padded h filter

    H_conj = np.conj(H) # finding the conjugate filter of the FFT H
    H_norm = np.abs(H)**2 # finding the squared norm of the FFT of the H

    G = H_conj / (H_norm + (1 / K))# creating the G Wiener filter based on the theory

    Y = np.fft.fft2(y1)# finding the FFT transform of the input image


    X_hat = G*Y # calculating the result of the Wiener filter on the input image
    x_hat = np.fft.ifft2(X_hat) # finding the inverse FFT of the output of the filter

    x_hat = np.real(x_hat[:Mx, :Nx]) #keeping only the real part of the inverse FFT and forcing it to have dimensions
    # equal with the input image

    return x_hat
```

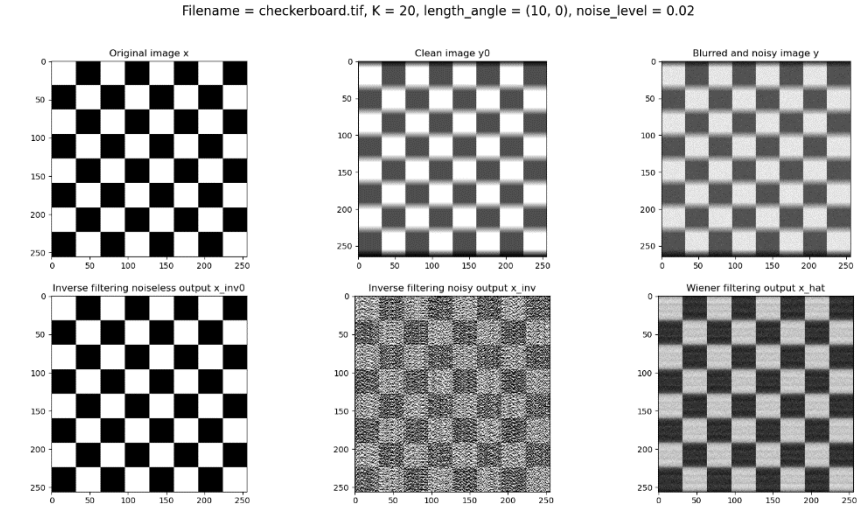
Χρήστος – Αλέξανδρος Δαρδαμπούνης ΑΕΜ 10335

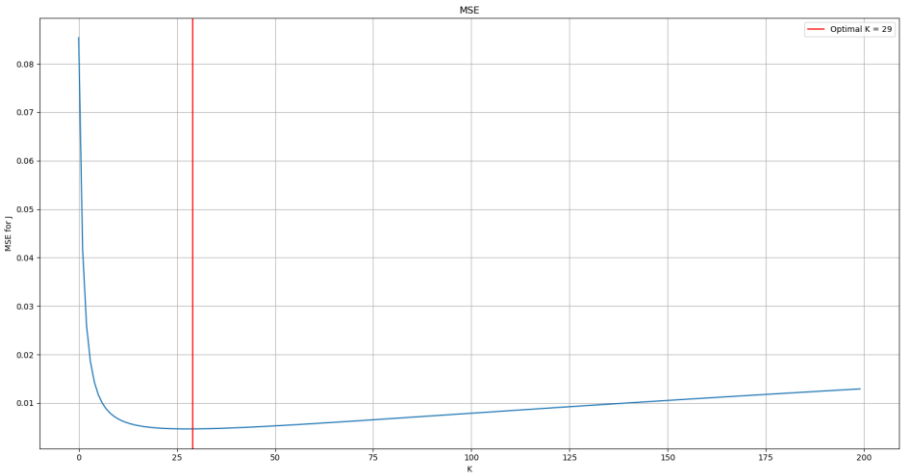
```python
import numpy as np
import matplotlib.pyplot as plt
import cv2
import hw3_helper_utils
from scipy.signal import convolve2d
from alt_wiener_filtering import my_wiener_filter1

K = 20
length_angle = [(10, 0), (20, 30)]
noise_levels = [0.02, 0.2]
images = ['checkerboard.tif', 'cameraman.tif']


for filename in images:
    for value in length_angle:
        for n in noise_levels:

            x = cv2.imread(filename, cv2.IMREAD_GRAYSCALE)  # reading the input image
            x = x.astype(np.float32) / 255.0
            h = hw3_helper_utils.create_motion_blur_filter(value[0], value[1])  # creating the motion blur filter
            y0 = convolve2d(x, h, mode="full")  # convolving the input image with the motion blur image
            v = n * np.random.randn(*y0.shape)  # creating the noise

            y = y0 + v # adding WGN to the result of the convolution

            M1, N1 = y.shape
            M2, N2 = h.shape
            (Mx, Nx) = (M1 - M2 + 1, N1 - N2 + 1)

            x_hat = my_wiener_filter1(y, h, K)  # getting the result of the wiener filter

            # correctly padding the filter and the y and y0 images so I can perform the DFT
            h1 = np.pad(h, ((0, y.shape[0] - 1), (0, y.shape[1] - 1)), mode='constant', constant_values = 0)
            y1 = np.pad(y, ((0, h.shape[0] - 1), (0, h.shape[1] - 1)), mode='constant', constant_values = 0)
            y2 = np.pad(y0, ((0, h.shape[0] - 1), (0, h.shape[1] - 1)), mode='constant', constant_values = 0)

            H = np.fft.fft2(h1)  # performing the FFT transform on the padded filter
            Y = np.fft.fft2(y1)  # performing the FFT transform on the result of the convolution with WGN

            mask = np.abs(H) < 1e-8  # creating a mask to find all the values of H which have a value close to zero
            H_inv = 1 / H  # creating the H_inv filter
            H_inv[mask] = 0  # frequencies that are zero fo

            X_inv = Y * H_inv
            x_inv = np.fft.ifft2(X_inv)

            x_inv = np.real(x_inv[:y.shape[0], :y.shape[1]])

            Y0 = np.fft.fft2(y2)

            X_inv0 = Y0 * H_inv
            x_inv0 = np.fft.ifft2(X_inv0)

            x_inv0 = np.real(x_inv0[:y.shape[0], :y.shape[1]])

            x_hat = x_hat[:Mx, :Nx]
            x_inv = x_inv[:Mx, :Nx]
            x_inv0 = x_inv0[:Mx, :Nx]

            # creating a list for J for K in range from 1 to 200
            J = []

            for k in range(1, 201):
                x_hat1 = my_wiener_filter1(y, h, k)
                J.append(np.mean((x_inv0 - x_hat1) ** 2))

            x_hat = np.clip(x_hat, 0, 1)
            x_inv = np.clip(x_inv, 0, 1)
            x_inv0 = np.clip(x_inv0, 0, 1)

            # plotting the desired images
            fig, axs = plt.subplots(nrows=2, ncols=3)
            fig.suptitle(f'Filename = {filename}, K = {K}, length_angle = {value}, noise_level = {n}', fontsize=16)
            axs[0][0].imshow(x, cmap='gray')
            axs[0][0].set_title("Original image x")
            axs[0][1].imshow(y0, cmap='gray')
            axs[0][1].set_title("Clean image y0")
            axs[0][2].imshow(y, cmap='gray')
            axs[0][2].set_title("Blurred and noisy image y")
            axs[1][0].imshow(x_inv0, cmap='gray')
            axs[1][0].set_title("Inverse filtering noiseless output x_inv0")
            axs[1][1].imshow(x_inv, cmap='gray')
            axs[1][1].set_title("Inverse filtering noisy output x_inv")
            axs[1][2].imshow(x_hat, cmap='gray')
            axs[1][2].set_title("Wiener filtering output x_hat")
            plt.show()

            #  plotting the J curve and finding the best K which minimises the MSE
            plt.plot(J)
            plt.ylabel('MSE for J')
            plt.xlabel('K')
            plt.title('MSE')
            minJ = np.argmin(J) + 1
            plt.axvline(minJ, color = 'r', label=f'Optimal K = {minJ}')
            plt.legend()
            plt.grid()
            plt.show()
```

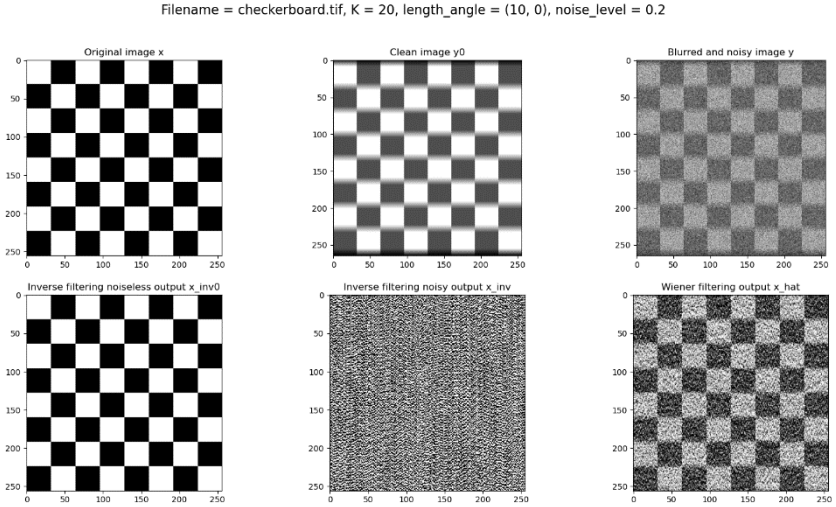
Χρήστος – Αλέξανδρος Δαρδαμπούνης ΑΕΜ 10335

Επειδή για κάποιες συχνότητες βγαίνουν μηδενικά τα στοιχεία του **DFT** του **H,** θεωρώ πως αυτές οι συχνότητες χάνονται για πάντα οπότε τις βάζω μηδέν και στα αντίστοιχα στοιχεία του αντιστρόφου του **H**

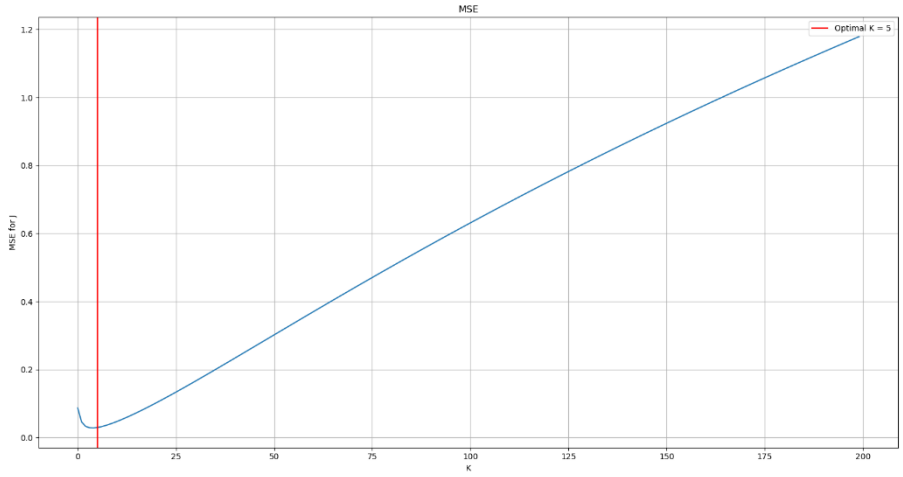
Τα αποτελέσματα και για τις δύο εικόνες με βάση αυτή την υλοποίηση του φίλτρου φαίνονται παρακάτω:



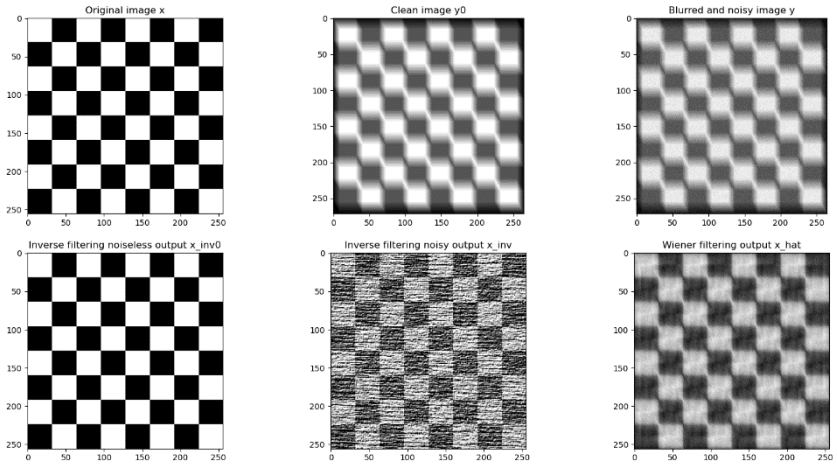


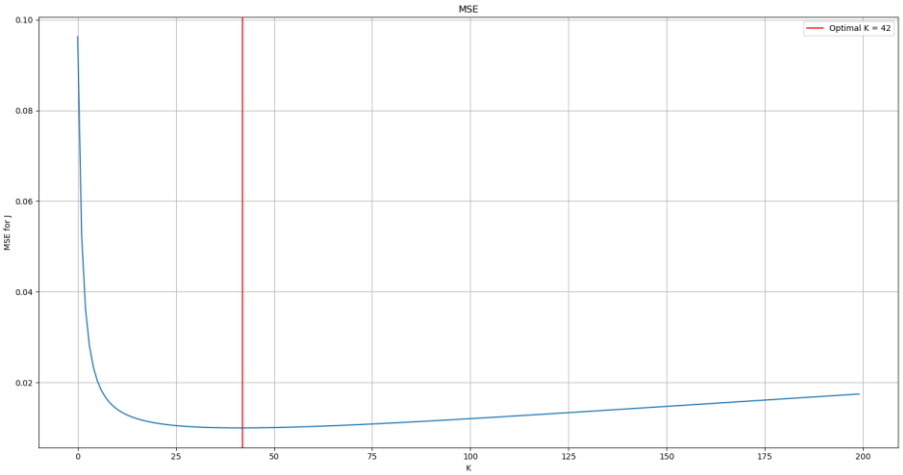


Χρήστος – Αλέξανδρος Δαρδαμπούνης ΑΕΜ 10335

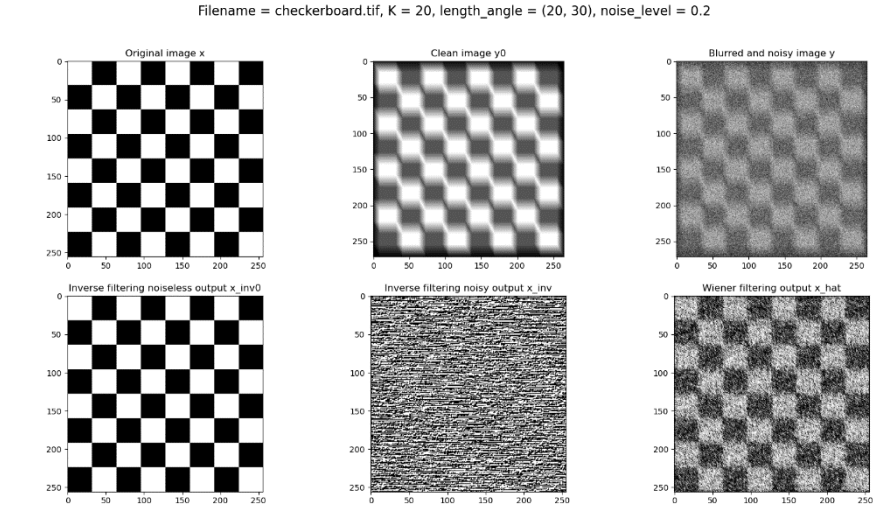


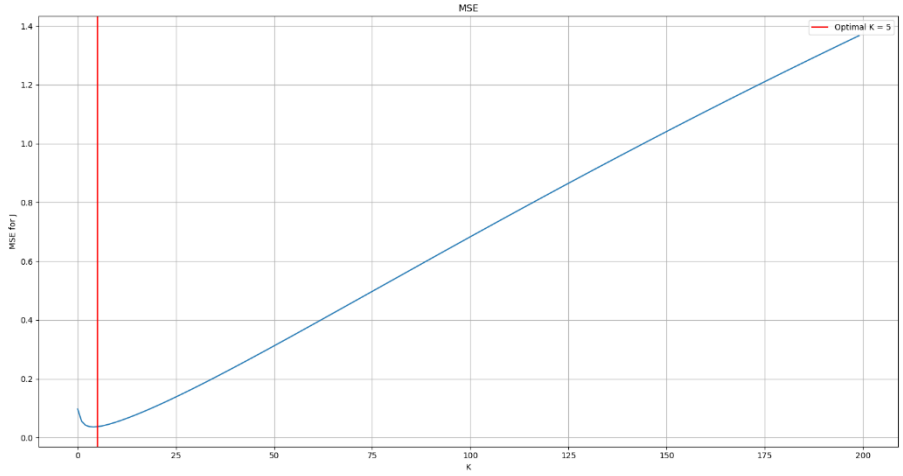
Filename = checkerboard.tif, K = 20, length_angle = (20, 30), noise_level = 0.02



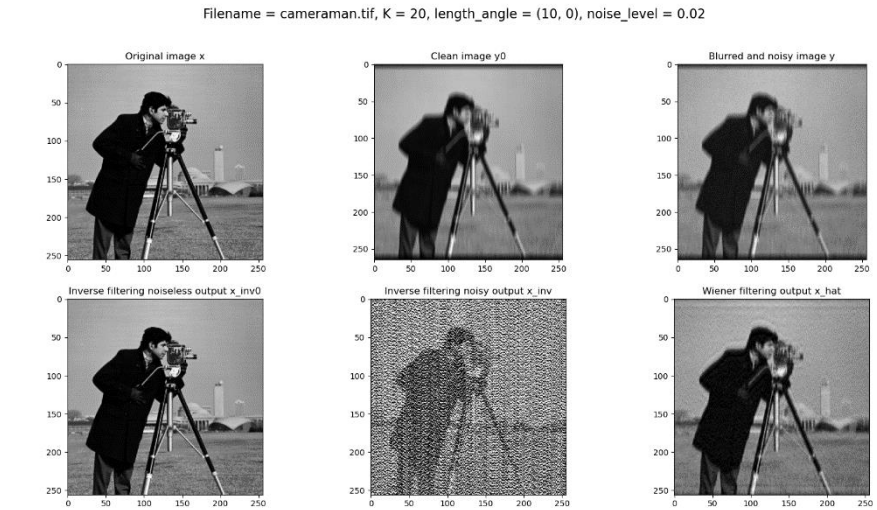


Χρήστος – Αλέξανδρος Δαρδαμπούνης ΑΕΜ 10335

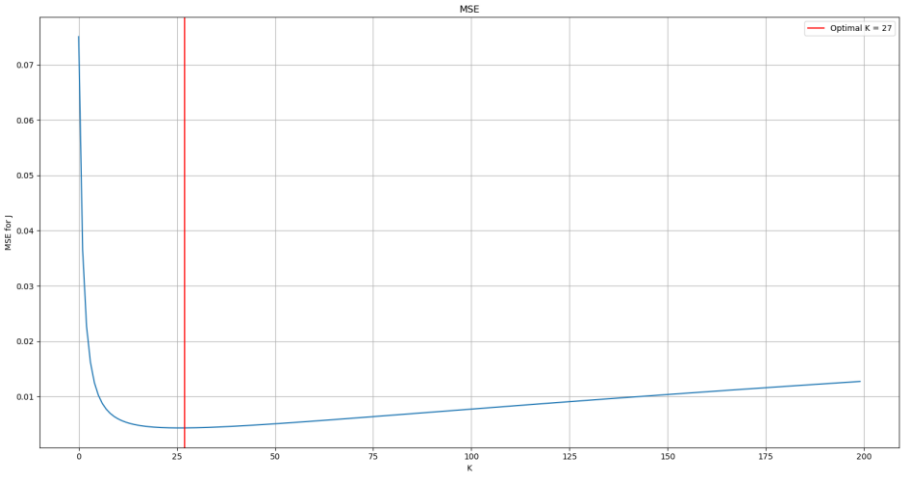




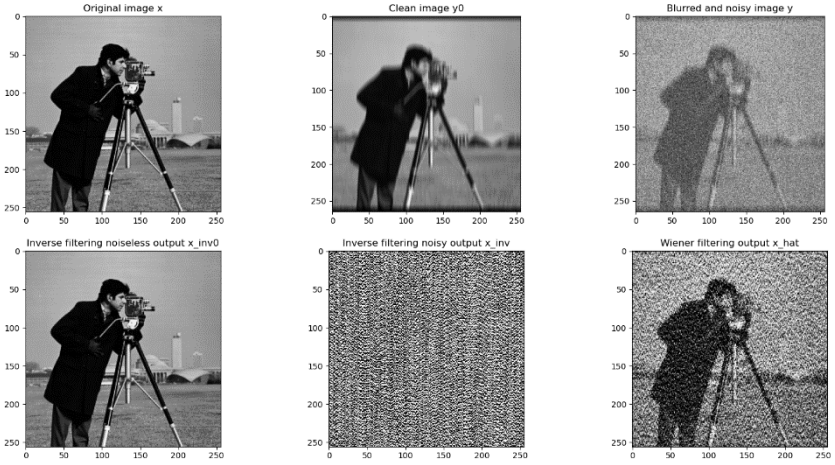
Παρακάτω ακολουθούν τα αντίστοιχα γραφήματα για την δεύτερη εικόνα που μας δόθηκε:

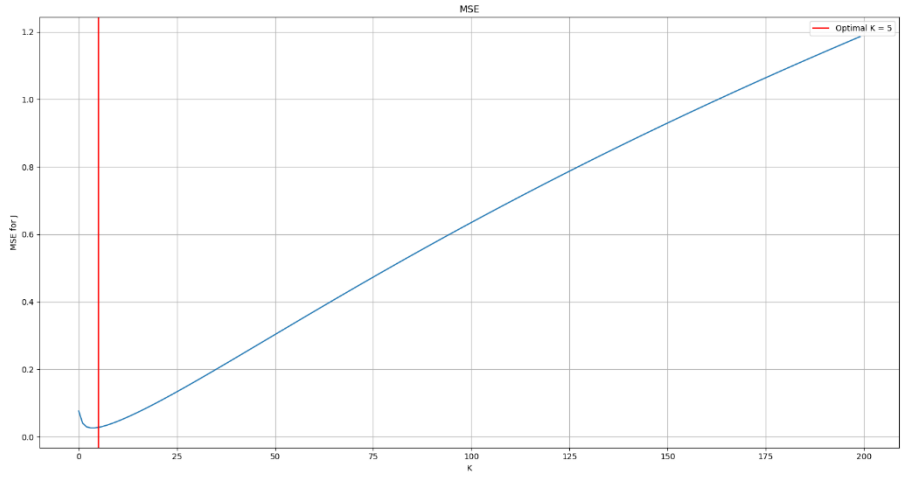


Χρήστος – Αλέξανδρος Δαρδαμπούνης ΑΕΜ 10335



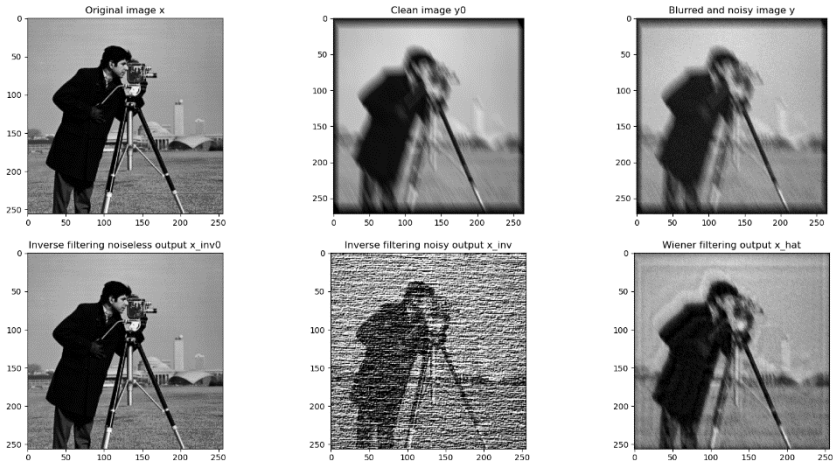
Filename = cameraman.tif, K = 20, length_angle = (10, 0), noise_level = 0.2

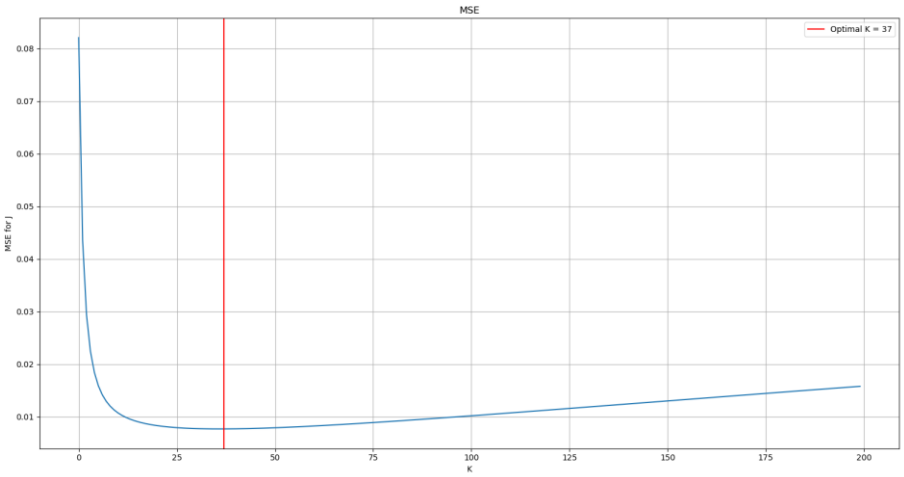




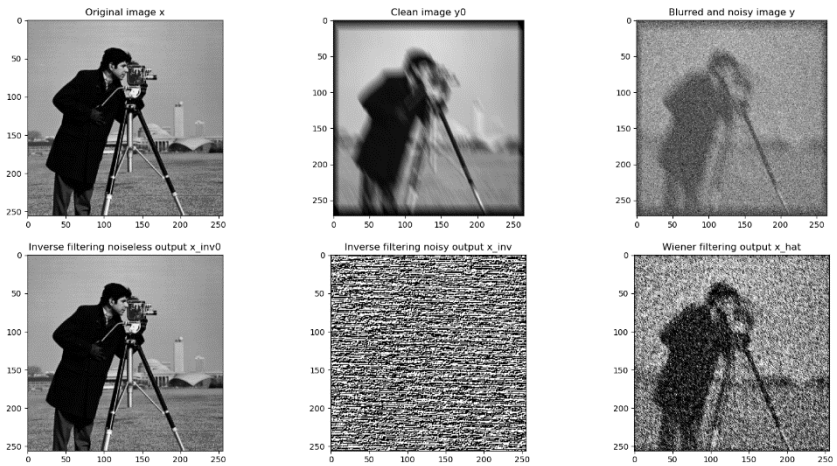
Χρήστος – Αλέξανδρος Δαρδαμπούνης ΑΕΜ 10335

Filename = cameraman.tif, K = 20, length_angle = (20, 30), noise_level = 0.02

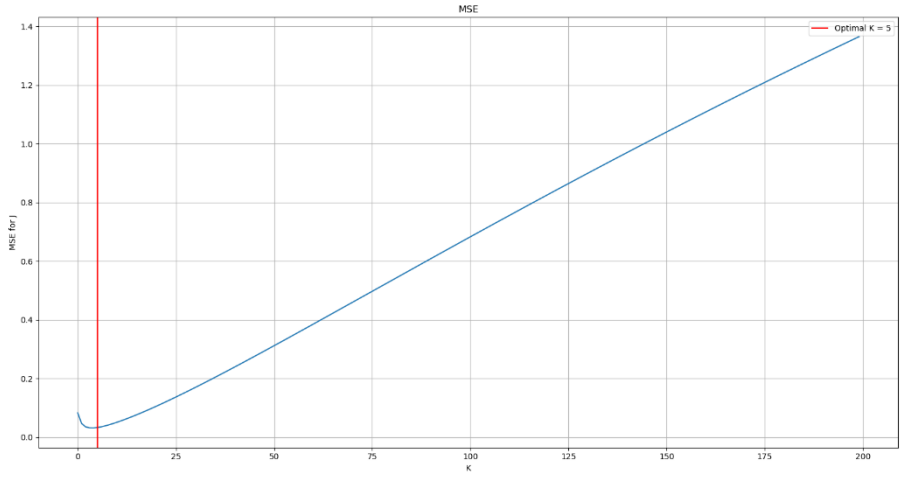




Filename = cameraman.tif, K = 20, length_angle = (20, 30), noise_level = 0.2



Χρήστος – Αλέξανδρος Δαρδαμπούνης ΑΕΜ 10335



Από τα παρακάτω γραφήματα είναι αρκετά φανερό πως με το φίλτρο **Wiener** μπορούμε όπως και πριν, να ανακτήσουμε σε αρκετά ικανοποιητικό βαθμό την αρχική μας εικόνα ακόμα και σε περιβάλλον υψηλού λευκού Gaussian θορύβου, ενώ επίσης φαίνεται πως διώχνουμε και σε αρκετά μεγάλο βαθμό την επίδραση του motion blur που εισάγει το φίλτρο **h** στην αρχική μας εικόνα. Επίσης, παρατηρούμε και αυτό που αναμέναμε για τις **x_inv0** και **x_inv,** δηλαδή όταν η εικόνα μας είναι απαλλαγμένη από θόρυβο τότε πολλαπλασιάζοντας με το αντίστροφο του φίλτρου έχουμε πλήρη ανάκτηση της εικόνας χωρίς καμία παραμόρφωση. Ωστόσο, με την παρουσία θορύβου άμα πολλαπλασιάσουμε με το αντίστροφο του φίλτρου ενδέχεται να ενισχύσουμε τον θόρυβο και έτσι να μην μπορούμε με τίποτα να ανακτήσουμε την αρχική μας εικόνα, γεγονός που παρατηρείται και στις παραπάνω εικόνες όπου σε περιβάλλον υψηλού θορύβου έχουμε χάσει πλήρως την εικόνα και έχει κυριαρχήσει ο θόρυβος. Για την παραγωγή των εικόνων επιλέχθηκε **K=20** το οποίο βρίσκεται πολύ κοντά στην γειτονιά όλων των βέλτιστων **K** που προκύπτουν από τα παραπάνω γραφήματα τα οποία ελαχιστοποιούν το **MSE.**

Για την παραγωγή της καμπύλης του **J** χρησιμοποιήθηκε όπως υποδεικνύεται το **x_inv0** λόγω της ολίσθησης που έχει σε σχέση με την αρχική εικόνα **x,** με την παραδοχή ότι είναι πολύ καλό αντίγραφο της **x.**

Με την εφαρμογή του δεύτερου τρόπου του φίλτρου Wiener παρατήρησα ότι υπήρχαν κάποιες μικρές αποκλίσεις ως προς την βέλτιστη επιλογή του **K**. Ωστόσο, αυτές οι αποκλίσεις είναι πολύ μικρές οπότε δεν θεωρώ πως είναι σημαντικές και θεωρώ πως και με τους δύο τρόπους υλοποίησης λαμβάνω ικανοποιητικά αποτελέσματα.

(**Σημείωση: Τα .py αρχεία για τον δεύτερο τρόπο υλοποίησης του φίλτρου Wiener καθώς και το αντίστοιχο demo βρίσκονται στα αρχεία alt_wiener_filtering.py και alt_demo.py)**

(Σημείωση: Επειδή για το άνοιγμα των εικόνων χρησιμοποιήθηκε η βιβλιοθήκη της opencv οι εικόνες θα πρέπει να βρίσκονται στο ίδιο path για να μπορέσουν να ανοιχτούν)