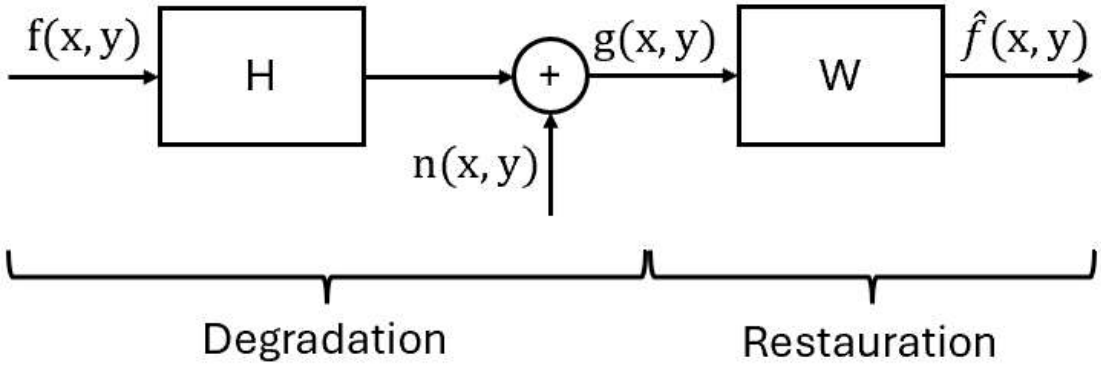
Modularbeit Wiener Filter

Herleitung des Wiener Filters:



Suche:

$$\underset{W}{argmin}\,\mathbb{E}\left[\|\hat{f}-f\|^2\right]$$

Annahmen:

$$g = h * f + n \implies G = H \cdot F + N \tag{1}$$
$$\hat{f} = g * w \implies \hat{F} = G \cdot W \tag{2}$$

$$S_{FF} = \mathbb{E}\left[\|F\|^2\right] \tag{3}$$
$$S_{NN} = \mathbb{E}\left[\|N\|^2\right] \tag{4}$$

Das Bild und das Rauschen sind unkorreliert:

$$S_{FN} = \mathbb{E}\left[F^*N\right] = \mathbb{E}\left[FN^*\right] = 0 \tag{5}$$

Herleitung:

Minimierung des quadratischen Fehlers vom Orginalbild und des rekonstruierten Bildes:

$$\mathbb{E}\left[\|\hat{F}-F\|^2\right] \overset{(2)}{=} E\left[\|GW-F\|^2\right]$$

$$= \mathbb{E}\left[\|GW\|^2 + \|F\|^2 - (GW)^*F - GWF^*\right]$$

$$= \|W\|^2\,\mathbb{E}\left[\|G\|^2\right] + \mathbb{E}\left[\|F\|^2\right] - W^*\,\mathbb{E}\left[G^*F\right] - W\,\mathbb{E}\left[GF^*\right]$$

$$\overset{(1,3)}{=} \|W\|^2\,\mathbb{E}\left[\|HF+N\|^2\right] + S_{FF} - W^*\,\mathbb{E}\left[(HF+N)^*F\right] - W\,\mathbb{E}\left[(HF+N)F^*\right]$$

$$= \|W\|^2 \mathbb{E}\left[\|HF\|^2 + \|N\|^2 + (HF)^* N + HFN^*\right] + S_{FF} - W^* \mathbb{E}\left[H^* \|F\|^2 + N^* F\right] -$$

$$= \|W\|^2 \left(\|H\|^2 \mathbb{E}\left[\|F\|^2\right] + \mathbb{E}\left[\|N\|^2\right] + H^* \mathbb{E}\left[F^* N\right] + H \mathbb{E}\left[FN^*\right]\right) + S_{FF}$$
$$- W^* \left(H^* \mathbb{E}\left[\|F\|^2\right] + \mathbb{E}\left[N^* F\right]\right) - W\left(H \mathbb{E}\left[\|F\|^2\right] + \mathbb{E}\left[NF^*\right]\right)$$

$$\overset{(3,4,5)}{=} \|W\|^2 \left(\|H\|^2 S_{FF} + S_{NN} + H^* 0 + H0\right) + S_{FF} - W^* \left(H^* S_{FF} + 0\right) - W\left(HS_{FF}\right.$$

$$= \|W\|^2 \left(\|H\|^2 S_{FF} + S_{NN}\right) + S_{FF} - W^* H^* S_{FF} - WHS_{FF}$$

Ableiten und Nullstellen:

$$\frac{\partial \mathbb{E}\left[\|\hat{F} - F\|^2\right]}{\partial W} = \frac{\partial \|W\|^2 \left(\|H\|^2 S_{FF} + S_{NN}\right) + S_{FF} - W^* H^* S_{FF} - WHS_{FF}}{\partial W}$$
$$= W^* \left(\|H\|^2 S_{FF} + S_{NN}\right) - HS_{FF} \overset{!}{=} 0$$

$$\iff W^* \left(\|H\|^2 S_{FF} + S_{NN}\right) = HS_{FF}$$
$$\iff W\left(\|H\|^2 S_{FF} + S_{NN}\right) = H^* S_{FF}$$
$$\iff W = \frac{H^* S_{FF}}{\|H\|^2 S_{FF} + S_{NN}} = \frac{H^*}{\|H\|^2 + \frac{S_{NN}}{S_{FF}}}$$

Implementation:

In [ ]:
```
from PIL import Image
import numpy as np
from matplotlib import pyplot as plt
from tqdm import tqdm
import math
```

In [ ]:
```
def read_image(image_path):
    image = np.array(Image.open(image_path).convert('L'))
    normalized_image = image / image.max()
    return normalized_image

def show_image(image):
    plt.imshow(image, cmap='gray')
    plt.axis('off')
    plt.show()
```

In [ ]:
```
def fft(image):
    return np.fft.fft2(image)

def ifft(image_fq):
    return np.abs(np.fft.ifft2(image_fq))
```

Mean Squared Error:

$$MSE = \mathbb{E}\left[\|\hat{f} - f\|^2\right]$$

Peak Signal To Noise Ratio:

$$PSNR = 10\log_{10}\left(\frac{MAX_f^2}{MSE}\right)$$

Accuracy Amoothness Error:

$$ASE = \mathbb{E}\left[\|\hat{g} - h * \hat{f}\|^2\right] + \lambda\mathbb{E}\left[\|\hat{f}'\|^2\right]$$

In [ ]:
```python
def mse(f, f_hat):
    return (np.square(f - f_hat)).mean()

def psnr(f, f_hat):
    mse_value = mse(f, f_hat)
    if mse_value == 0:
        return 0
    return 10 * np.log10(1 / mse_value)

def accuracy_smothness_error(g, f_hat, H, _lambda):
    data_accuracy = np.linalg.norm(g - ifft(fft(f_hat) * H))
    smothness = np.sum(np.square(np.gradient(f_hat)))
    return data_accuracy + _lambda * smothness
```

Wiener Filter:

$$\hat{f} = \mathcal{F}^{-1}\left\{\left[\frac{H^*}{\|H\|^2 + K}\right]G\right\}$$

In [ ]:
```python
def wiener_filter(G, H, K):
    W = np.conjugate(H) / (np.abs(np.square(H)) + K)
    F_hat = W * G
    f_hat = ifft(F_hat)
    return f_hat
```

Motion Blur Filter:

$$H(u,v) = Tsinc\Big(\pi\big(u\Delta x + v\Delta y\big)\Big)e^{i\pi\left(u\Delta x + v\Delta y\right)}$$

In [ ]:
```python
def get_motion_blur_filter_fq(shape, T, dx, dy):
    U_DIM ,V_DIM = shape
    H_u = np.arange(0, U_DIM, 1)
    H_v = np.arange(0, V_DIM, 1)

    H_U, H_V = np.meshgrid(H_v, H_u)
    H = np.dstack((H_U, H_V))
```

```
    dxy = np.array([dx, dy])

    def s(uv):
        return np.pi * uv.dot(dxy)

    H = np.apply_along_axis(s, axis=2, arr=H)
    H = T * np.sinc(H) * np.exp(-1j * H)

    return H
```

Gausian Noise Filter:

$$n(x,y) \sim \mathcal{N}(\mu, \sigma^2)$$
$$N(u,v) = \mathcal{F}\{n(x,y)\}$$

In [ ]:
```
def get_gausian_noise_filter_fq(shape, variance, mean=0):
    n = np.random.normal(mean, variance, shape)
    N = fft(n)
    return N
```

Plotting:

In [ ]:
```
def plot_images_restauration(original_image, disturbed_image, restored_images, r
    fig, ax = plt.subplots(2, math.ceil(len(restored_images) / 2) + 1)
    fig.set_size_inches((14, 12))

    ax[0,0].set_title(f'Original Image\n MSE: {mse(original_image, original_imag
    ax[0,0].imshow(original_image, cmap='gray')
    ax[0,0].axis('off')

    ax[1,0].set_title(f'Disturbed Image\n MSE: {mse(original_image, disturbed_im
    ax[1,0].imshow(disturbed_image, cmap='gray')
    ax[1,0].axis('off')

    for i in range(len(restored_images)):
        x = int(i / 2) + 1
        y = i % 2
        ax[y, x].set_title(f'{restored_titels[i]}\n MSE: {mse(original_image, re
        ax[y, x].imshow(restored_images[i], cmap='gray')
        ax[y, x].axis('off')

    plt.tight_layout()
    plt.show()
```

Blind Wiener Filtering

Wiener Filtering With Optimization:

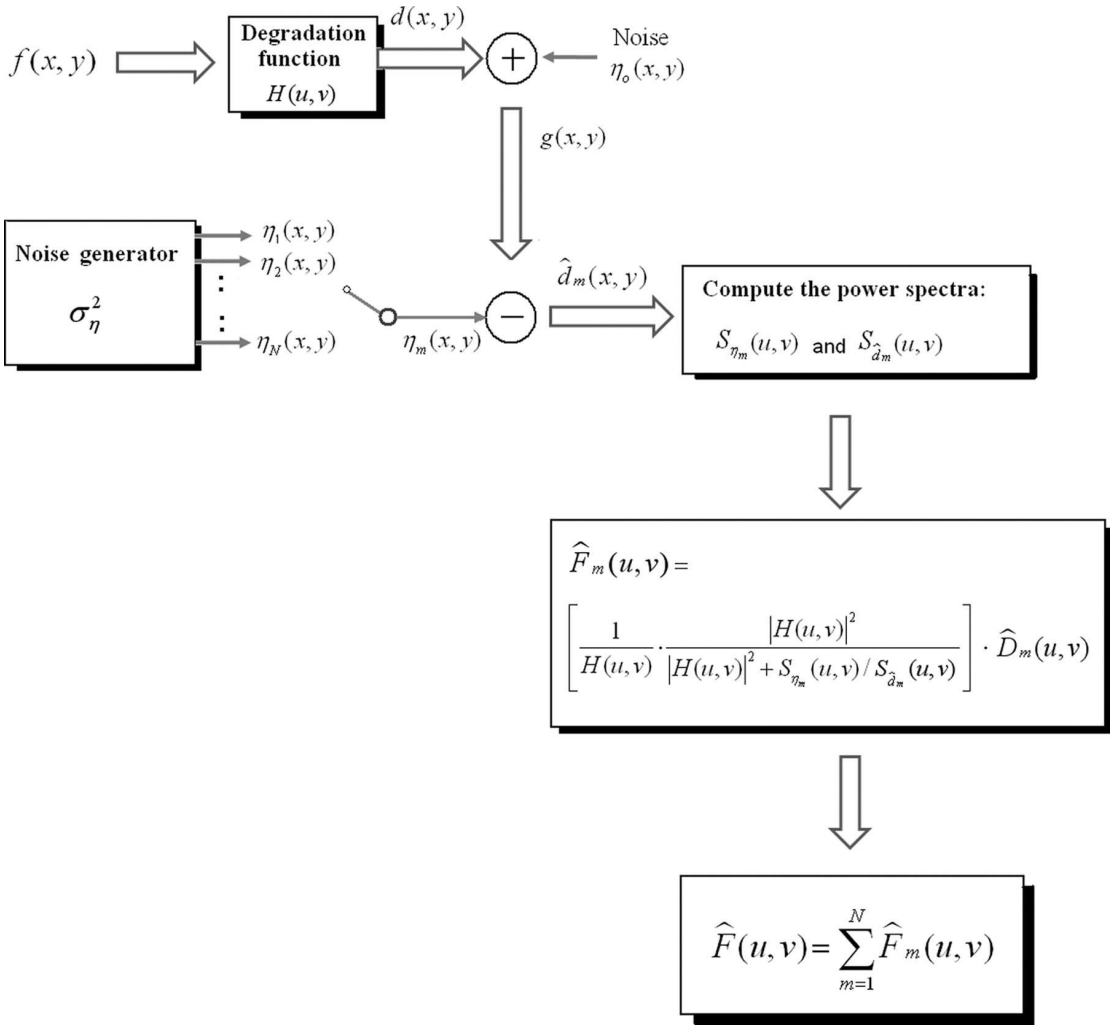
$$\underset{K}{argmin}\ \mathbb{E}\left[\|\hat{g} - h * \hat{f}\|^2\right] + \lambda\,\mathbb{E}\left[\|\hat{f}'\|^2\right]$$

In [ ]:
```
def optimize_k(disturbed_image_fq, motion_blur_filter_fq, _lambda=0, _dropout=Tr
```

```
    disturbed_image = ifft(disturbed_image_fq)

    best_error = np.inf
    best_k = 0.5
    step = 0.5

    for _ in tqdm(range(_max_iter), desc='Optimizing K'):

        last_iter_error = best_error
        K = np.linspace(best_k + step, best_k - step, 5, endpoint=False)[1:]

        for k in K:
            restored_image = wiener_filter(disturbed_image_fq, motion_blur_filte
            error = accuracy_smothness_error(disturbed_image, restored_image, mo
            if error < best_error:
                best_error = error
                best_k = k

        step /= 2
        if _dropout and last_iter_error - best_error < 0.01:
            break

    return best_k
```

Wiener filtering By Averaging Over Noise:

[Yoo, Jae-Chern, and Chang Wook Ahn. "Image restoration by blind-Wiener filter." IET Image Processing 8.12 (2014): 815-823.]

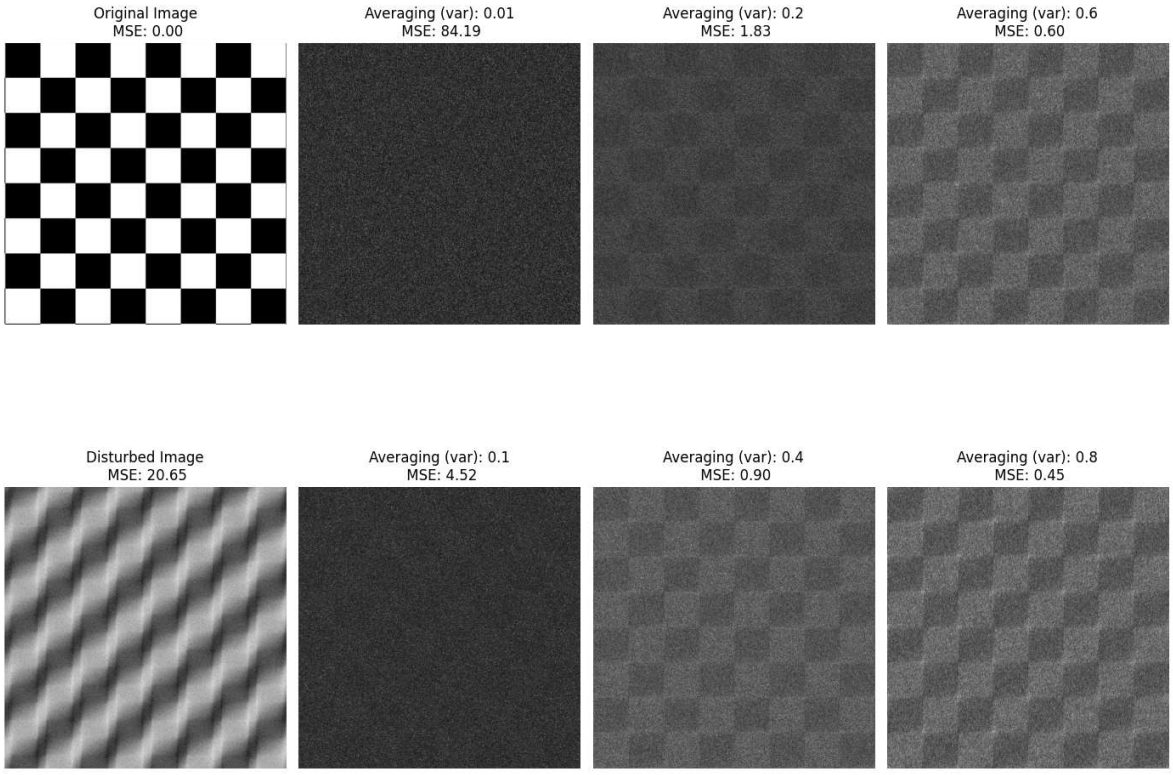


In [ ]:
```python
def wiener_filter_averaging(G, H, variance, n=10):
    f_hat = 0
    for _ in range(n):
        N_m = get_gausian_noise_filter_fq(G.shape, variance)
        D_hat_m = G - N_m
        S_D_hat_m= np.mean(np.abs(np.square(D_hat_m)))
        S_N_m = np.mean(np.abs(np.square(N_m)))

        f_hat_m = wiener_filter(D_hat_m, H, S_N_m/S_D_hat_m)
        f_hat += f_hat_m
    return f_hat / n
```

In [ ]:
```
def test_wiener_filter_averaging(image_path, _mb_T=1, _mb_dx=0.005, _mb_dy=0.001
    f = read_image(image_path)
    F = fft(f)
    H = get_motion_blur_filter_fq(F.shape, _mb_T, _mb_dx, _mb_dy)
    N = get_gausian_noise_filter_fq(F.shape, _g_var)
    G = H * F + N
    g = ifft(G)

    images = []
    titles = []
    variances = [0.01, 0.1, 0.2, 0.4, 0.6, 0.8]
    for var in tqdm(variances):
        images.append(wiener_filter_averaging(G, H, var))
        titles.append(f'Averaging (var): {var}')

    plot_images_restauration(f, g, images, titles)
```

In [ ]:
```
test_wiener_filter_averaging('images/schach.jpg', _mb_T=10, _mb_dx=0.01, _mb_dy=
```

```
100%|██████████| 6/6 [00:13<00:00,  2.18s/it]
```



Other Posibilities Of Blind Wiener Filtering:

- Parameter Estimation of Power Spectral Density of Noise and Image
- location depentently K-parametric wiener filtering: K as adaptively varying-coefficient
- ...

Testing:

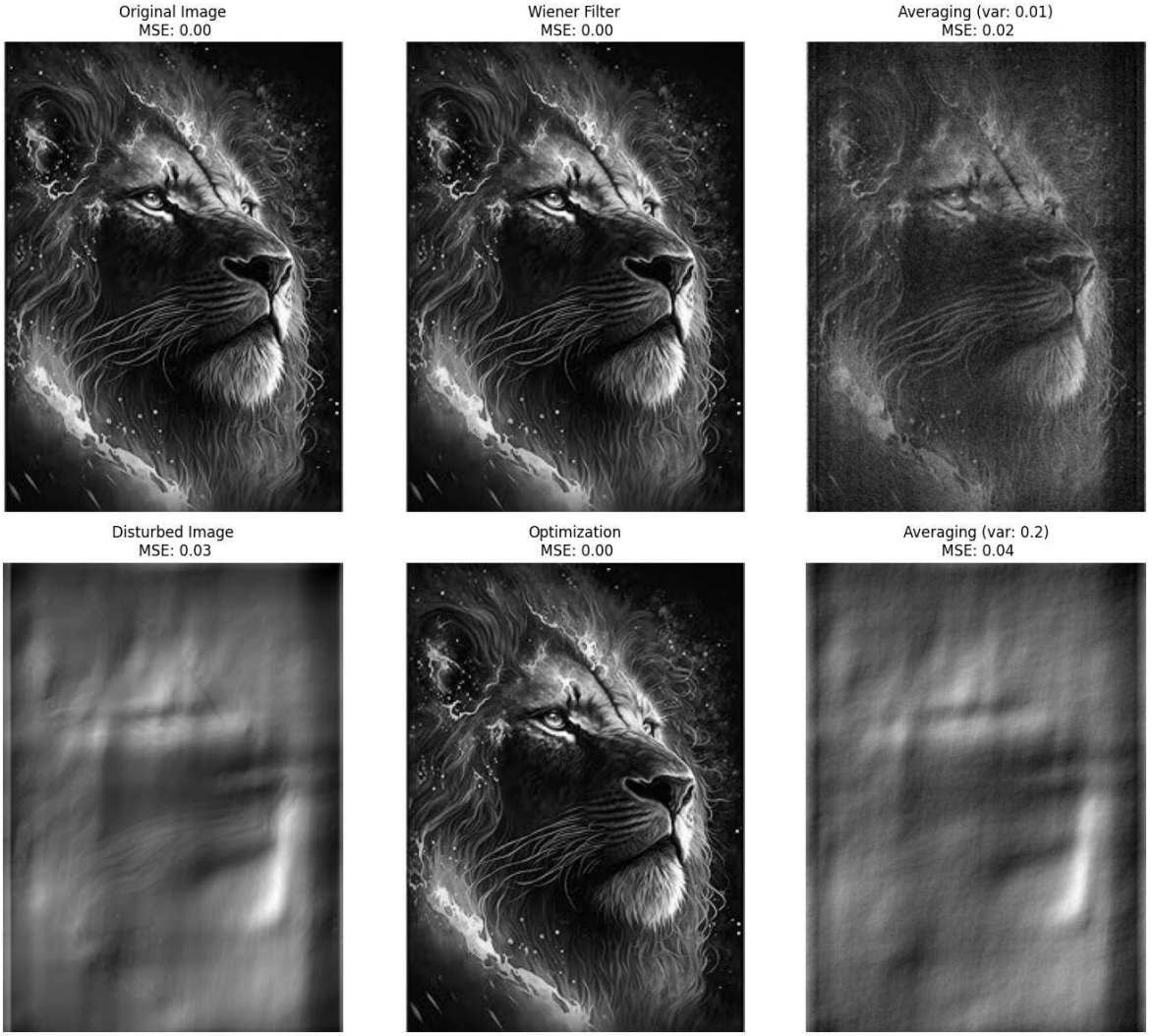
In [ ]:
```
def test_restaurations(image_path, _mb_T=1, _mb_dx=0.005, _mb_dy=0.001, _n_var=0
    f = read_image(image_path)
    F = fft(f)
```

```
    H = get_motion_blur_filter_fq(F.shape, _mb_T, _mb_dx, _mb_dy)
    N = get_gausian_noise_filter_fq(F.shape, _n_var)
    G = H * F + N
    g = ifft(G)

    # wiener filter with knowledge of powerspectrum of image and noise
    S_F = np.mean(np.abs(np.square(F)))
    S_N = np.mean(np.abs(np.square(N)))
    f_hat = wiener_filter(G, H, S_N/S_F)

    # wiener filter with optimization of K
    optimal_k = optimize_k(G, H, _lambda=0)
    f_hat_2 = wiener_filter(G, H, optimal_k)

    # wiener filter with averaging over noise
    f_hat_3 = wiener_filter_averaging(G, H, 0.01)

    f_hat_4 = wiener_filter_averaging(G, H, 0.2)

    plot_images_restauration(f, g, [f_hat, f_hat_2, f_hat_3, f_hat_4], ['Wiener
```

Only Motion Blur:

In [ ]:
```
test_restaurations('images/lion.jpg', _mb_T=1, _mb_dx=0.01, _mb_dy=0.01, _n_var=
```

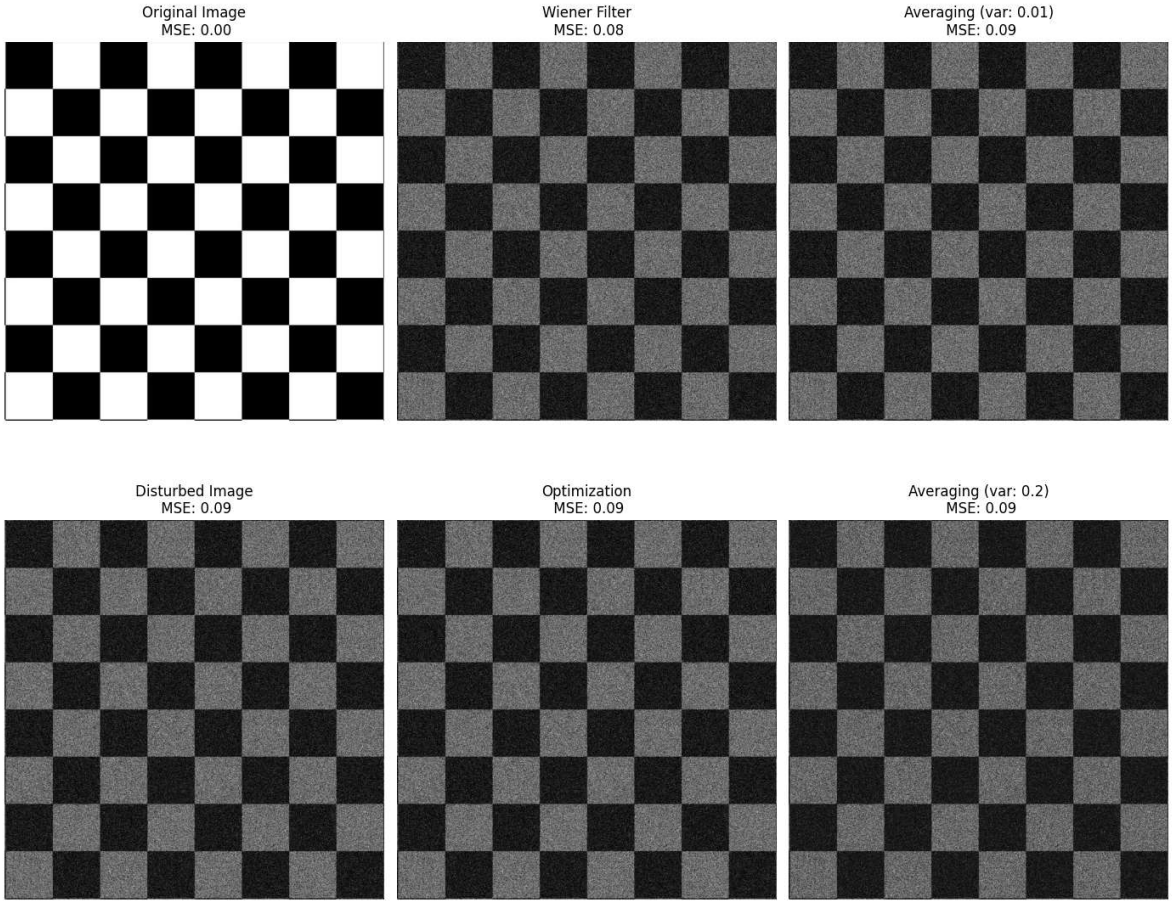
```
Optimizing K:  56%|█████     | 28/50 [00:03<00:02,  8.81it/s]
```



Only Noise:

```
In [ ]: test_restaurations('images/schach.jpg', _mb_T=1, _mb_dx=0, _mb_dy=0, _n_var=0.3)
```

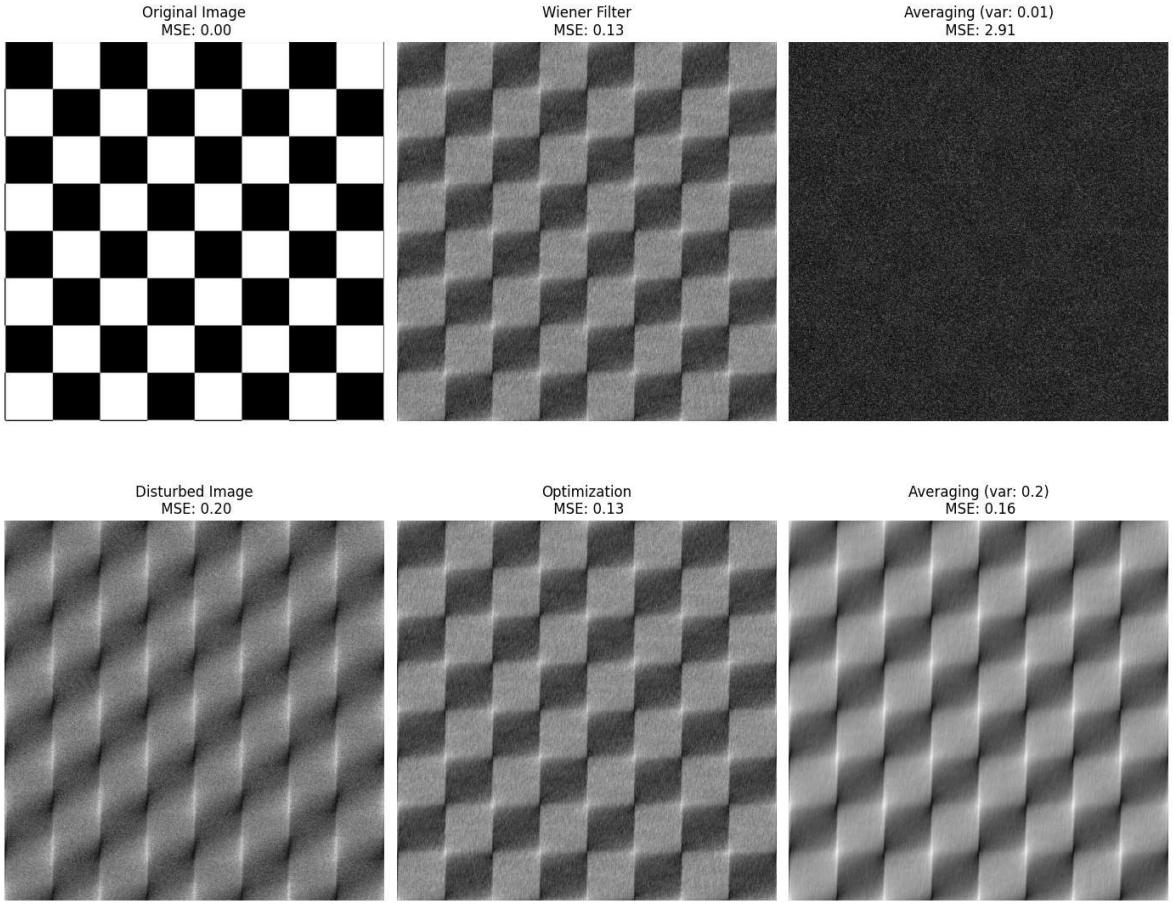
```
Optimizing K:  26%|██        | 13/50 [00:17<00:48,  1.32s/it]
```



Motion Blur and Noise:

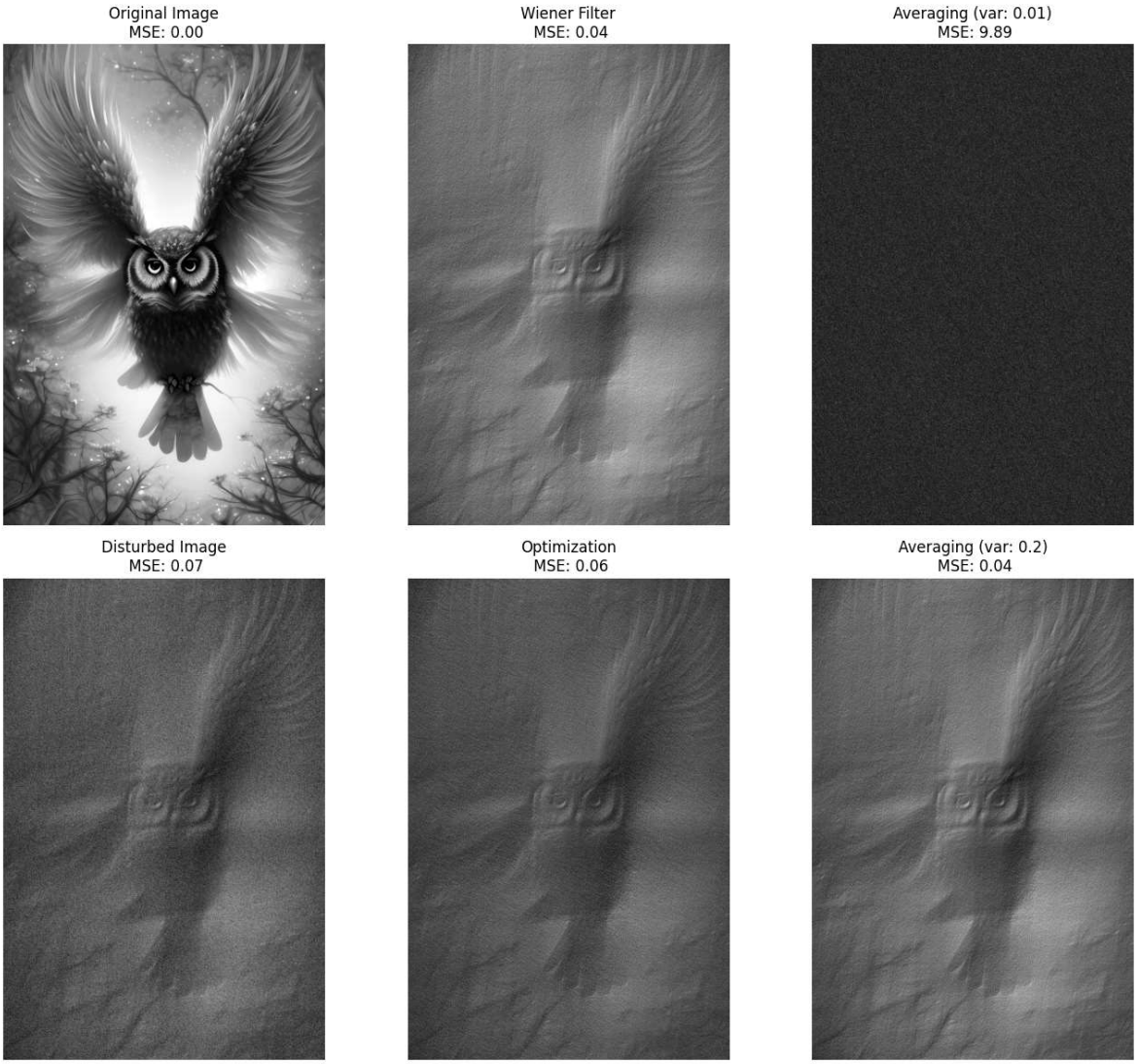
```
In [ ]: test_restaurations('images/schach.jpg', _mb_T=1, _mb_dx=-0.001, _mb_dy=-0.02, _n
```

```
Optimizing K:   6%|          | 3/50 [00:05<01:18,  1.67s/it]
```



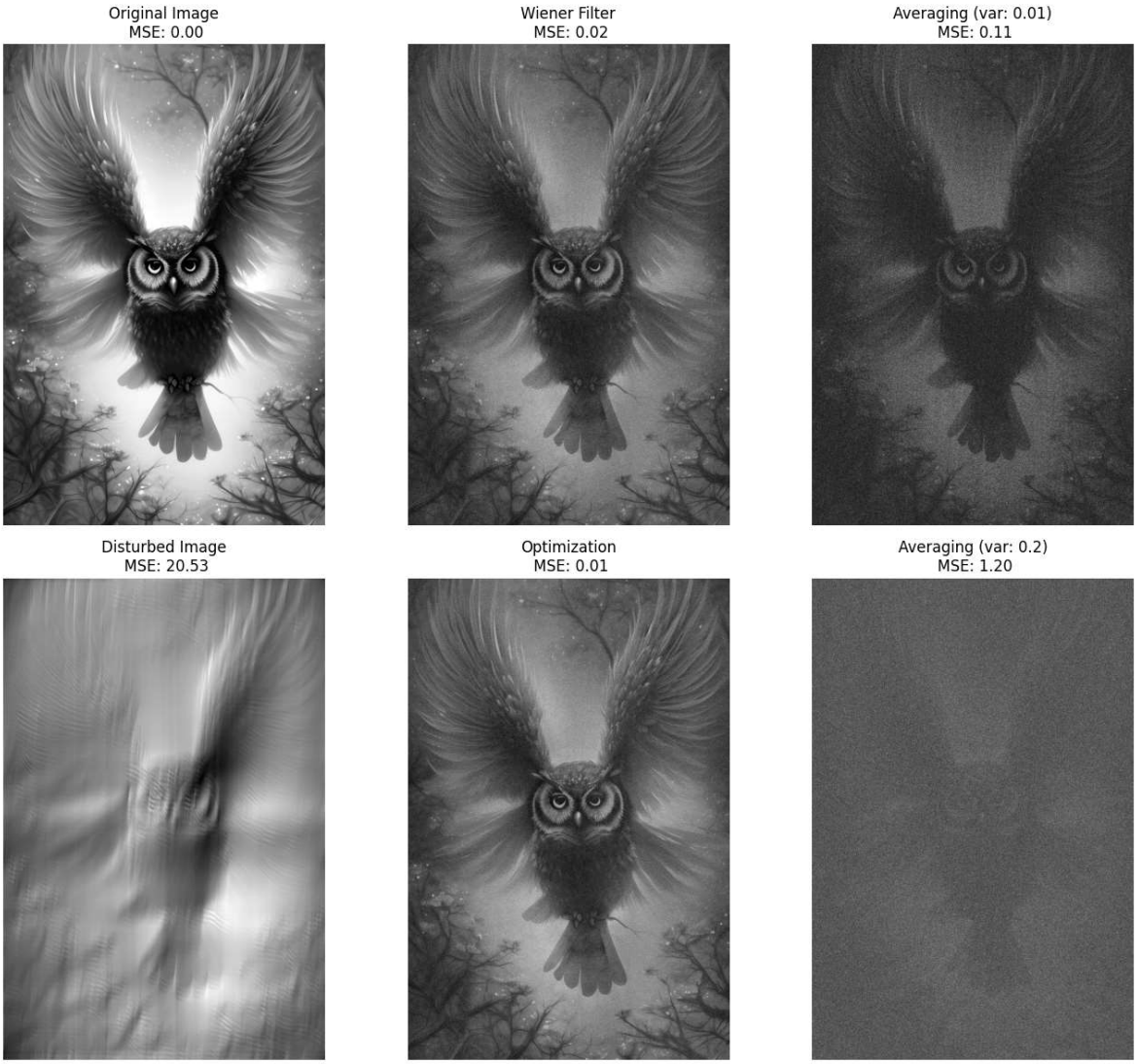
In [ ]:
```
test_restaurations('images/eule.png', _mb_T=1, _mb_dx=0.001, _mb_dy=0.001, _n_va
```

```
Optimizing K:   4%|▌          | 2/50 [00:07<02:56,  3.69s/it]
```



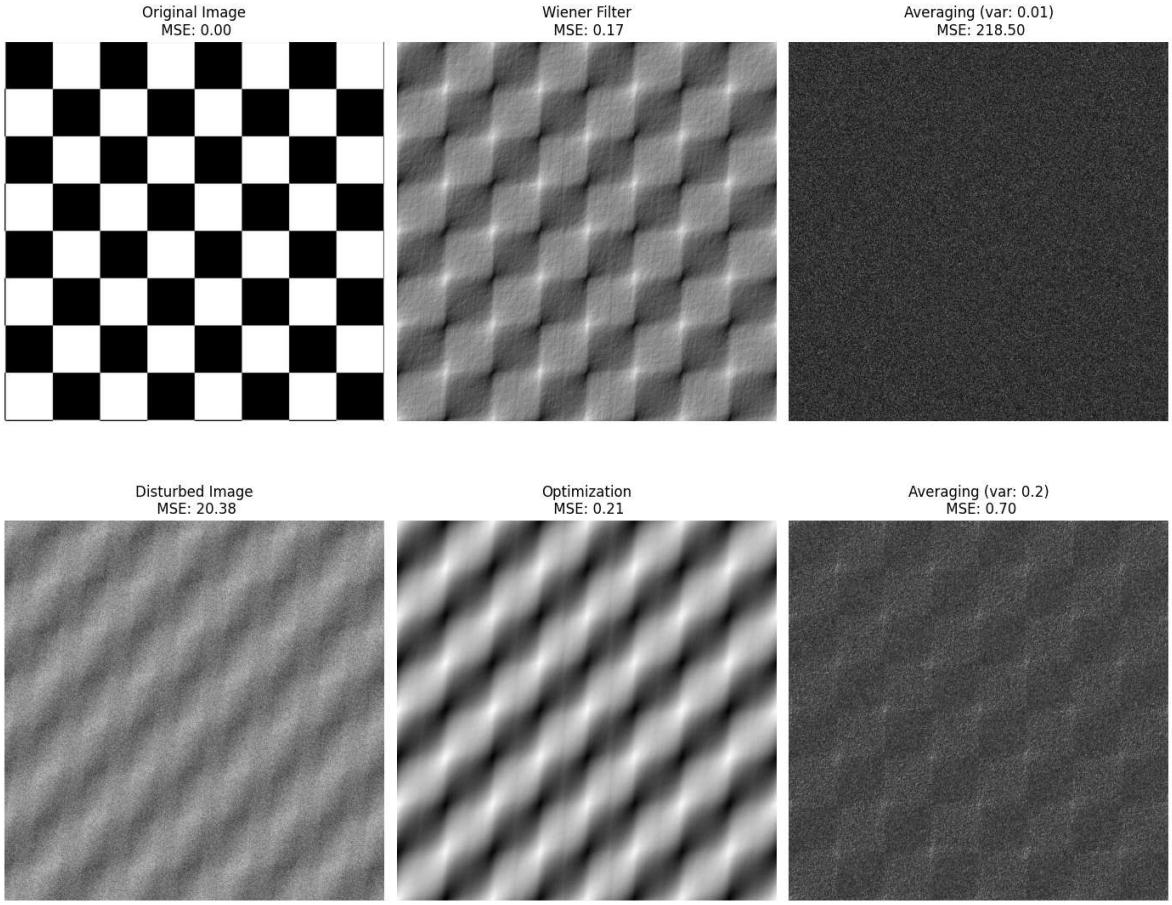
In [ ]:
```
test_restaurations('images/eule.png', _mb_T=10, _mb_dx=0.001, _mb_dy=-0.01, _n_v
```

```
Optimizing K:  16%|          | 8/50 [00:17<01:33,  2.23s/it]
```



In [ ]:
```
test_restaurations('images/schach.jpg', _mb_T=10, _mb_dx=0.1, _mb_dy=0.2, _n_var
```

```
Optimizing K:   6%|▌         | 3/50 [00:04<01:14,  1.59s/it]
```


Original Image
MSE: 0.00
Wiener Filter
MSE: 0.17
Averaging (var: 0.01)
MSE: 218.50
Disturbed Image
MSE: 20.38
Optimization
MSE: 0.21
Averaging (var: 0.2)
MSE: 0.70