Homework 2:
Image Enhancement and Image Restoration

陳宥銓
313554013
Institute of Data Science and Engineering

Contents

1 Implementation

1.1 Image Enhancement

1.1.1 Gamma Correction

For each pixel value P_{in}, gamma correction is formulated as $P_{out} = AP_{in}^{\gamma}$, where A is the normalize factor.

```
"""
TODO Part 1: Gamma correction
"""
def gamma_correction(img, gamma):
    # Normalize the pixel values to [0, 1],
    # apply gamma correction, and scale back to [0, 255]
    gamma_img = ((img / 255) ** gamma) * 255
    return gamma_img.astype(np.uint8)
```

Figure 1: Implementation of gamma correction

1.1.2 Histogram Equalization

```
"""
TODO Part 2: Histogram equalization
"""
def histogram_equalization(img):
    h, w = img.shape[:2]
    pixel_num = h * w
    hist = np.zeros((256, 3), dtype=np.float64)
    hist_sum = np.zeros((256, 3), dtype=np.float64)
    # Construct the histogramof each channel
    for c in range(3):
        for i in range(h):
            for j in range(w):
                hist[img[i,j,c],c] += 1
    # Calculate the cumulative distribution function (CDF) of each channel
    hist_sum[0,:] = hist[0,:]
    for i in range(1, 256):
        hist_sum[i,:] = hist[i,:] + hist_sum[i-1,:]
    # Normalized the CDF to [0, 255]
    hist_sum = (np.round((hist_sum / pixel_num) * 255)).astype(np.uint8)
    # Assign new pixels of each channel
    he_img = np.zeros_like(img)
    for c in range(3):
        he_img[:,:,c] = hist_sum[img[:,:,c],c]
    return he_img
```

Figure 2: Implementation of histogram equalization

To make the output closer to the contrast enhancement effect of the human eye, we consider the minimum value of CDF when calculating normalization to avoid shifts in dark areas.

```
for c in range(3):
    cdf_min = hist_sum[:,c].min()
    hist_sum[:,c] = ((hist_sum[:,c] - cdf_min) / (pixel_num - cdf_min) * 255).clip(0, 255)
```

Figure 3: Implementation of histogram equalization

1.1.3 Homomorphic Filtering

An image can be decomposed into illumination and reflection i.e. $I(x,y) = L(x,y) \cdot R(x,y)$ and $log\ I(x,y) = log\ L(x,y) + log\ R(x,y)$. After Fourier transformation, since the relative change in illumination is small, it can be regarded as the low-frequency component of the image; while the relative change in reflection is large, it can be regarded as the high-frequency component. Eq. 1 shows the process of homomorphic filtering

$$I(u,v) \Rightarrow\ log \Rightarrow FFT \Rightarrow H(u,v) \Rightarrow FFT^{-1} \Rightarrow exp \Rightarrow I'(u,v) \tag{1}$$

$$H(u,v) = (\gamma_H - \gamma_L)H_{\text{high pass}}(u,v) + \gamma_L, \tag{2}$$

$$H_{\text{high pass}}(u,v) = 1 - exp[-c(D^2(u,v)/D_0^2)] \tag{3}$$

where $\gamma_H > 1$, $\gamma_L < 1$, D_0 controls the boundaries between high and low frequency, c controls the expansion coefficient of the filter.

```
def homomorpgic_filter(img, D0=80.0, gammaH=2.0, gammaL=0.5, c=2.0):
    img_normalized = img.astype(np.float32) / 255.0
    enhanced_img = np.zeros_like(img_normalized)
    # Construct homomorphic filter H(u, v)
    rows, cols = img.shape[:2]
    crow, ccol = rows // 2, cols // 2
    u = np.arange(-crow, crow)
    v = np.arange(-ccol, ccol)
    V, U = np.meshgrid(v, u)
    D = U**2 + V**2
    H = (gammaH - gammaL) * (1 - np.exp(-c * (D / D0**2))) + gammaL
    for channel in range(3):
        # Log transformation
        log_img = np.log1p(img_normalized[:,:,channel])
        # FFT
        img_fft = fftshift(fft2(log_img))
        # Apply homomorphic filter
        filtered_img = (gammaH - gammaL) * (img_fft * H) + gammaL
        filtered_img = ifft2(ifftshift(filtered_img))
        enhanced_img[:,:,channel] = np.exp(np.real(filtered_img)) - 1
    enhanced_img = np.clip(enhanced_img*255, 0, 255)
    return enhanced_img.astype(np.uint8)
```

Figure 4: Implementation of homomorphic Filtering

1.2 Image Restoration

1.2.1 Motion Blur PSF Generation

The PSF of motion blur distortion is a line segment with the length L of blur and the angle θ of motion.

```
"""
TODO Part 1: Motion blur PSF generation
"""
def generate_motion_blur_psf(size, length ,theta):
    psf = np.zeros(size, np.float64)
    center = (size[1]//2, size[0]//2)
    axes = (0, length//2)
    cv2.ellipse(psf, center, axes, theta, 0, 360, 1, -1)
    psf /= psf.sum()
    return psf
```

Figure 5: Implementation of PSF generation

1.2.2 Wiener Filtering

Consider a image $f(x, y)$ with motion blur $h(x, y)$, the blurred image $g(x, y) = f(x, y) * h(x, y) + \eta(x, y)$. Then the FFT of blurred image $G(u, v) = H(u, v)F(u, v) + N(u, v)$ and the FFT of recovered image $F'(x, y) = \frac{G(u,v)}{H(u,v)} \left[\frac{1}{1+\frac{NSR(u,v)}{|H(u,v)|^2}}\right]$, where $NSR(u, v) = \frac{|N(u,v)|^2}{|F(u,v)|^2}$. Then the Wiener Filter $W(u, v)$ is defined as follows

$$W(u, v) = \frac{1}{H(u, v)} \left[\frac{1}{1 + \frac{NSR(u,v)}{|H(u,v)|^2}}\right] = \frac{H^*(u, v)}{|H(u, v)|^2 + K} \tag{4}$$

1.2.3 Constrained Least Squares Restoration

Then the Constrained Least Squares Filter $CLSF(u, v)$ is defined as follows

$$CLSF(u, v) = \frac{H^*(u, v)}{|H(u, v)|^2 + \gamma|P(u, v)|^2} \tag{5}$$

where $P(u, v) = FFT(L)$, $L = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$

```
"""
TODO Part 2: Wiener filtering
"""
def wiener_filtering(blurred_img, psf, K):
    restored_img = np.zeros(blurred_img.shape[:])
    # Calculate Wiener filter (1 / H * |H|^2 / (|H|^2 + K))
    psf_fft = fft2(psf,s=blurred_img.shape[:2])
    psf_abs_squared = np.abs(psf_fft) ** 2
    wiener_filter = np.conj(psf_fft) / (psf_abs_squared + K)

    for c in range(3):
        # Apply the filter in the frequency domain
        blurred_fft = fft2(blurred_img[:,:,c])
        restored_fft = blurred_fft * wiener_filter
        restored_img[:,:,c] = np.real(fftshift(ifft2(restored_fft)))
    restored_img = np.clip(restored_img, 0, 255)
    return restored_img.astype(np.uint8)
```

Figure 6: Implementation of Wiener filter

```
"""
TODO Part 3: Constrained least squares filtering
"""
def constrained_least_square_filtering(blurred_img, psf, gamma):
    restored_img = np.zeros(blurred_img.shape[:])
    laplacian = [[0,-1,0],[-1,4,-1],[0,-1,0]]
    laplacian_fft = fft2(laplacian, s=blurred_img.shape[:2])
    laplacian_abs_squared = np.abs(laplacian_fft) ** 2

    # Compute the CLS filter in the frequency domain
    psf_fft = fft2(psf,s=blurred_img.shape[:2])
    psf_abs_squared = np.abs(psf_fft) ** 2
    cls_filter = np.conj(psf_fft) / (psf_abs_squared + gamma * laplacian_abs_squared)

    for c in range(3):
        blurred_fft = fft2(blurred_img[:,:,c])
        restored_fft = blurred_fft * cls_filter
        restored_img[:, :, c] = np.real(fftshift(ifft2(restored_fft)))
    restored_img = np.clip(restored_img, 0, 255)
    return restored_img.astype(np.uint8)
```

Figure 7: Implementation of CLSF

1.2.4 Lucy Richardson Restoration

Lucy-Richardson method is an image restoration algorithm based on maximum likelihood estimation iteration. Suppose the blurred image $g(x, y) = f(x, y) * h(x, y) + \eta(x, y)$ then the Lucy-Richardson method uses maximum likelihood estimation of the Poisson distribution

$$P(g|f) = \Pi \frac{(f * h)^{g(x,y)} e^{-(f*h)}}{g(x, y)!} \tag{6}$$

and use the following iteration to update the estimate of the image

$$f_{k+1} = f_k * (\frac{g}{f_k \otimes h} \otimes h^*) \tag{7}$$

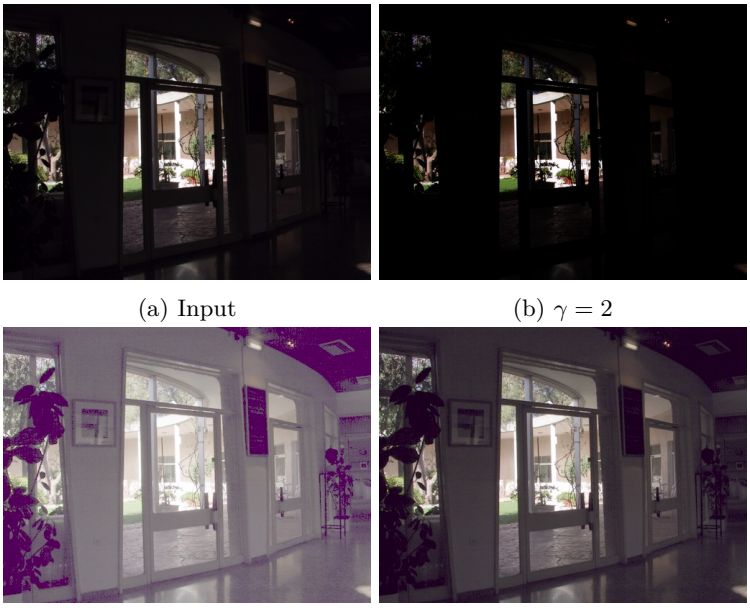
```python
def lucy_richardson(blurred_img, psf, iterations=200):
    estimate_img = blurred_img.copy()
    for _ in range(iterations):
        for c in range(3):
            # f_k * h
            conv_estimate = cv2.filter2D(estimate_img[:,:,c], -1, psf)
            # g/(f_k * h)
            relative_blur = blurred_img[:,:,c] / (conv_estimate + 1e-6)
            # (g/(f_k * h)) * h^*
            correction_factor = cv2.filter2D(relative_blur, -1, psf.T)
            # f_k * ((g/(f_k * h)) * h^*)
            estimate_img[:,:,c] *= correction_factor
    estimate_img = np.clip(estimate_img, 0, 255)
    return estimate_img.astype(np.uint8)
```

Figure 8: Implementation of Lucy-Richardson method

2 Results and Analysis

2.1 Image Enhancement

Gamma Correction adjusts the brightness of the image using a non-linear exponential function. According to the Fig. 9, the value of γ determines the effect on output image. If $\gamma > 1$, the exponential transformation will compresses the high brightness values in the image, darkening the highlight areas. Conversely, if $\gamma < 1$, it will enhance the brightness of low brightness areas, making dark areas brighter. Histogram Equalization (Fig. 10) enhances the contrast of the image, stretching the pixel distribution of the image to a wider brightness range to enhance the details of the image so that the details of both dark and bright areas can be displayed more clearly.



(a) Input (b) $\gamma = 2$

(c) $\gamma = 0.25$ (d) $\gamma = 0.5$

Figure 9: Gamma correction with different γ

(a) Original method (b) New method

Figure 10: Histogram equalization with different normalization method

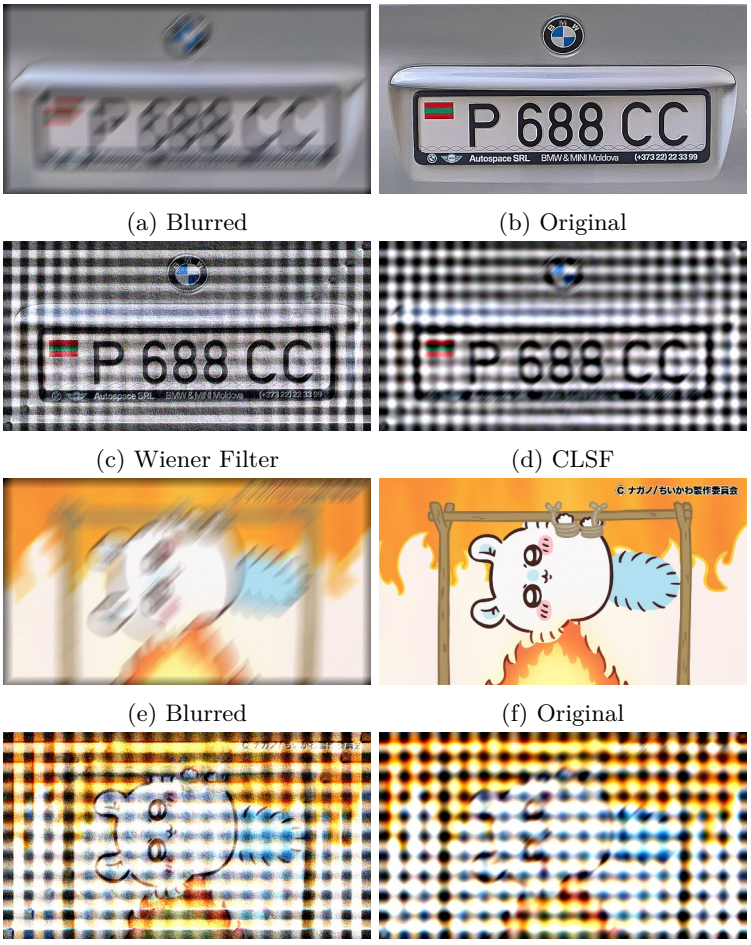
Fig. 11 shows the result of homomorphic filtering with $D_0 = 80, \gamma_H = 2, \gamma_L = 0.5, c = 2$. Compared with histogram equalization and gamma correction, homomorphic filtering can more accurately separate and adjust the illumination and reflection components of the image. Especially in the case of uneven illumination, it can simultaneously suppress low-frequency illumination components and enhance high-frequency details, thereby improving the image's clarity and contrast without changing the natural brightness of the image, and have higher flexibility.

(a) Input (b) Output

Figure 11: Homomorphic filtering

2.2 Image Restoration

We apply the Wiener filtering and constrained least squares filtering with the motion blur PSF (Fig. 5), using $L = 40$, $\theta = 45$ to restoration the images. Fig. 12 shows the resulting images for different filters. For Wiener filtering, we set $K = 0.001$ in Eq. 4 and obtain PSNR values of 11.06 and 9.77 in two cases. For CLSF we set $\gamma = 0.25$ in Eq. 5 and obtain PSNR values of 11.43 and 9.26 in two cases, respectively. However, in the two outputs, we found that the images, especially CLSF, contained a large number of black noises, which we suspect may be due to the boundary of the blurred image. We tried using different values of L in Eq. 5 and found that this issue improved in CLSF Fig. 13.
Fig. 14 shows the results using Lucy-Richardson method. Compared to Wiener wiltering and CLSF, the Lucy-Richardson method preserves more details in the deblurred image and reduces the large number of black noises in the output image. However, since Lucy-Richardson method is an iteration-based algorithm, convolution calculations are required for each iteration, so the calculation amount is large, especially with a large number of iterations, which will cause the calculation time to become longer.



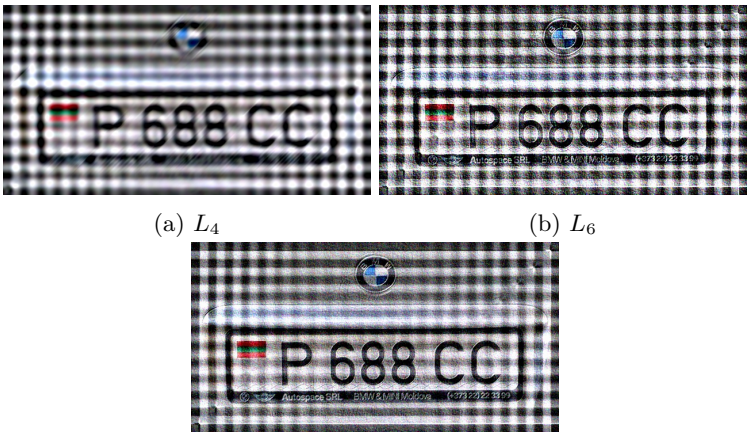
(a) Blurred (b) Original

(c) Wiener Filter (d) CLSF

(e) Blurred (f) Original

(g) Wiener Filter (h) CLSF

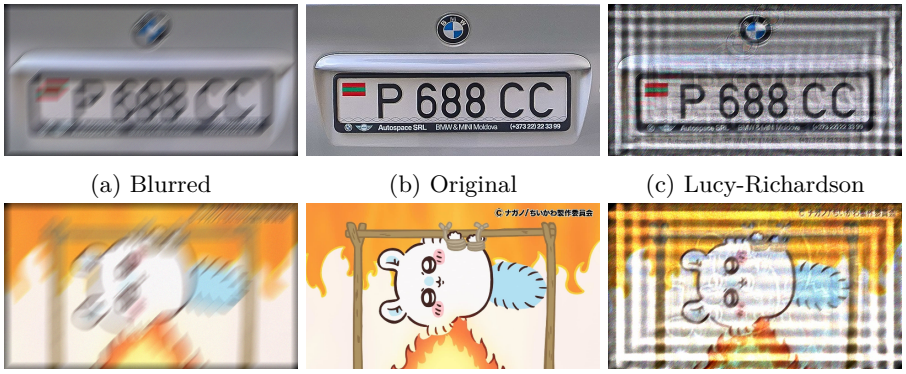
Figure 12: Image restoration results



(a) L_4 (b) L_6

(c) L_{10}

Figure 13: CLSF with $L_i = \begin{bmatrix} 0 & -1 & 0 \\ -1 & i & -1 \\ 0 & -1 & 0 \end{bmatrix}$



(a) Blurred (b) Original (c) Lucy-Richardson

(d) Blurred (e) Original (f) Lucy-Richardson

Figure 14: Lucy-Richardson method results

3 Answer the questions

3.1 Please describe a problem you encountered and how you solved it.

- When performing histogram equalization, we found that using traditional CDF normalization can lead to the shifts in dark region (Fig. 10a). To solve this problem, we can adjust the CDF of each channel. When normalizing a CDF, first find the minimum non-zero CDF value **cdf_min** in each channel and then normalize the CDF (Fig. 10b).

3.2 What potential limitations might arise when using Minimum Mean Square Error (Wiener) Filtering for image restorations? Please suggest possible solutions to address them.

- The value of K in Eq. 4 is often unknown, so the Wiener filter may not perform optimally when the noise is high. If K is underestimated, the filter may fail to sufficiently suppress noise, while overestimating K can lead to oversmoothing and loss of fine details in the image. Additionally, Wiener filtering requires an accurate degradation. However, in practical applications, the degradation process is usually unknown, and if the degraded model is estimated incorrectly, the restored image may have artifacts or distortions. In cases where the degradation model is unknown, we can use blind deconvolution method that do not require prior knowledge of the degradation model and iteratively estimate both the degradation parameters and the original image.

3.3 What potential limitations might arise when using Constrained Least Squares Restoration for image restorations? Please suggest possible solutions to address them.

- Same as wiener filter, Constrained Least Squares Restoration also depends on the accuracy of the degradation model and if the degradation model is inaccurate, it may also affect the recovery effect. In addition, if the parameters γ are set improperly, it may lead to excessive smoothing, resulting in the loss of details and edges in the image. We can through the iterative method to estimate γ.