Homework 2: Image Enhancement & Image Restoration

Part I. Implementation:

Enhancement

Gamma Correction

```python
def gamma_correction(img, gamma=1.0):
    # normalize to [0~1]
    img_normalized = img.astype(np.float32) / 255.0
    # apply gamma correction
    img_gamma_corrected = np.power(img_normalized, 1.0 / gamma)
    # rearrnge
    img_gamma_corrected = img_gamma_corrected * 255.0

    img_gamma_corrected = img_gamma_corrected.astype(np.uint8)
    return img_gamma_corrected
```

$$I_{\text{out}} = I_{\text{in}}^{\frac{1}{\gamma}}$$

- Noramalize the image value into [0,1]
- Apply gamma correction
- Remapping back to [0,255]

histogram_equalization

```python
def histogram_equalization(img):
    # Convert image to YCrCb color space
    ycrcb_img = cv2.cvtColor(img, cv2.COLOR_BGR2YCrCb) #維度不會變 [H,W,3] -> [H,W,3] RGB->Y Cr Cb
    # Equalize the histogram of the Y channel
    ycrcb_img[:, :, 0] = cv2.equalizeHist(ycrcb_img[:, :, 0]) # 對Y channel進行直方圖均衡化 重新分配亮度
    # Convert back to BGR color space
    return cv2.cvtColor(ycrcb_img, cv2.COLOR_YCrCb2BGR)
```

- Convert to YCrCb color space
- Apply histogram_qualization to Y channel
- Convert back to BGR color space

Unsharp Masking

```python
def other_enhancement_algorithm(img, sigma=1.0, strength=10):
    """
    Unsharp Masking 實現
    :param img: 原始圖像
    :param sigma: 高斯模糊的標準差，用於控制模糊程度
    :param strength: 銳化強度，值越大，邊緣增強效果越強
    :return: 銳化後的圖像
    """
    # apply Gaussian blur to the image
    blurred_img = cv2.GaussianBlur(img, (0, 0), sigma)

    # calculate the enhanced image
    enhanced_img = cv2.addWeighted(img, 1 + strength, blurred_img, -strength, 0)

    return enhanced_img
```

$$I_{\text{enhanced}} = (1 + \text{strength}) \cdot I_{\text{original}} - \text{strength} \cdot I_{\text{blurred}}$$

- Apply gaussain blur to original image
- Calculate the enhanced image

Restoration

Generate PSF and decide angle and length

```python
def estimate_psf_angle_length(img_blurred):
    edges = cv2.Canny(img_blurred, 50, 150)
    lines = cv2.HoughLines(edges, 1, np.pi / 180, threshold=100)
    if lines is not None:
        angles = [line[0][1] for line in lines]
        angle = np.median(angles)  # 使用中位數避免極端值影響
        length = max([line[0][0] for line in lines])  # 取最大線段長度作為參考
        return np.degrees(angle), int(length)
    return None, None


def generate_motion_blur_psf(size=2,length=20,angle=45):
    '''
        描述導致圖像模糊的運動模糊點擴展函數 Point Spread Function (PSF)
        size : 15~45
        length : 10~30
        angle : 0~180
        psf 非0 模糊影響主要位置
    '''
    psf = np.zeros((size, size), dtype=np.float32)
    center = size // 2
    for i in range (length):
        x = int(center + i * np.cos(np.radians(angle)))
        y = int(center + i * np.sin(np.radians(angle)))
        if 0<=x<size and 0<=y<size:
            psf[y, x] = 1.0
    psf /= psf.sum() # normalization
    return psf
```

- Using Canny edge detection and Hough line transform to detect the image_blurred angle and length
- Given the length and angle to form a region
- If coordinate is in the region set to 1 , otherwise 0.

Wiener filtering

```python
def wiener_filtering(img_blurred, psf, k=0.1):
    """
    Apply Wiener filtering to an RGB image.
    """
    # Initialize the restored image
    img_restored = np.zeros_like(img_blurred)

    # Get the spatial dimensions
    H, W = img_blurred.shape[:2]


    # Compute the FFT of the padded PSF
    psf_fft = np.fft.fft2(psf , s= (H,W))
    psf_fft_conj = np.conj(psf_fft)
    psf_power = np.abs(psf_fft) ** 2
    wiener_filter = psf_fft_conj / (psf_power + k)

    # Apply the Wiener filter to each channel
    for c in range(3):
        # Transfer to frequency domain
        img_blurred_fft = np.fft.fft2(img_blurred[:, :, c])

        # Apply the Wiener filter
        img_restored_fft = img_blurred_fft * wiener_filter

        # Convert back to spatial domain and take the absolute value
        img_restored_channel = np.abs(np.fft.ifft2(img_restored_fft))

        # Clip the values and convert to uint8
        img_restored[:, :, c] = np.clip(img_restored_channel, 0, 255).astype(np.uint8)

    return img_restored
```

$$H_{wiener} = \frac{H^*}{|H|^2 + \frac{noise\_power}{signal\_power}}$$

- Transform the psf to frequency domain and padding it
- Caculate the filiter and apply it to each channel of image
    - Image also need to transform to frequency domain
- Transform back to spatial domain

Constrained_least_square_filtering

```python
def constrained_least_square_filtering(img_blurred, psf, reg_param=0.1):
    # Initialize the restored image
    img_restored = np.zeros_like(img_blurred)

    # Get the spatial dimensions
    H, W = img_blurred.shape[:2]


    # Compute the FFT of the padded PSF
    psf_fft = np.fft.fft2(psf , s= (H,W))
    psf_fft_conj = np.conj(psf_fft)
    psf_power = np.abs(psf_fft) ** 2
    laplacian = np.array([[0, -1, 0], [-1, 4, -1], [0, -1, 0]])
    laplacian_fft = np.fft.fft2(laplacian, s=(H, W))
    cls_filiter = psf_fft_conj / (psf_power + reg_param * laplacian_fft )

    # Apply the Wiener filter to each channel
    for c in range(3):
        # Transfer to frequency domain
        img_blurred_fft = np.fft.fft2(img_blurred[:, :, c])

        # Apply the cls filter
        img_restored_fft = img_blurred_fft * cls_filiter

        # Convert back to spatial domain and take the absolute value
        img_restored_channel = np.abs(np.fft.ifft2(img_restored_fft))

        # Clip the values and convert to uint8
        img_restored[:, :, c] = np.clip(img_restored_channel, 0, 255).astype(np.uint8)

    return img_restored
```

$$H_{\text{cls}} = \frac{H^*}{|H|^2 + \gamma |L|^2}$$

- Transform the psf to frequency domain and padding it
- Define the Laplacian filiter ,transform it to frequency domain and padding it
- Caculate the filiter and apply it to each channel of image
    - Image also need to transform to frequency domain
- Transform back to spatial domain

Inverse filitering

```python
def inverse_filtering(img_blurred, psf):
    # Initialize the restored image
    img_restored = np.zeros_like(img_blurred)

    # Get the spatial dimensions
    H, W = img_blurred.shape[:2]


    # Compute the FFT of the padded PSF
    psf_fft = np.fft.fft2(psf , s= (H,W))

    inverse_filiter = psf_fft+1e-8

    # Apply the Wiener filter to each channel
    for c in range(3):
        # Transfer to frequency domain
        img_blurred_fft = np.fft.fft2(img_blurred[:, :, c])

        # Apply the inverse filter
        img_restored_fft = img_blurred_fft * inverse_filiter

        # Convert back to spatial domain and take the absolute value
        img_restored_channel = np.abs(np.fft.ifft2(img_restored_fft))

        # Clip the values and convert to uint8
        img_restored[:, :, c] = np.clip(img_restored_channel, 0, 255).astype(np.uint8)

    return img_restored
```

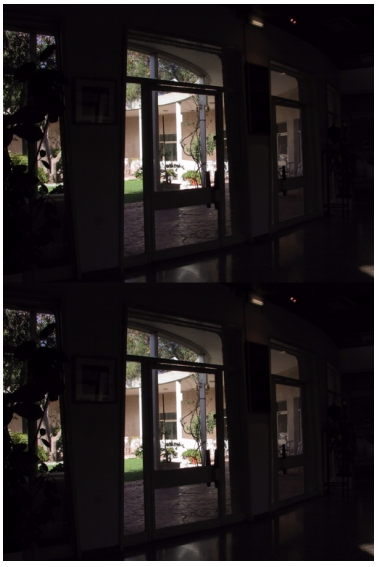
$$\hat{F}(u,v) = \frac{G(u,v)}{H(u,v)}$$

- Transform the psf to frequency domain and padding it
- Caculate the filiter and apply it to each channel of image
    - Image also need to transform to frequency domain
- Transform back to spatial domain

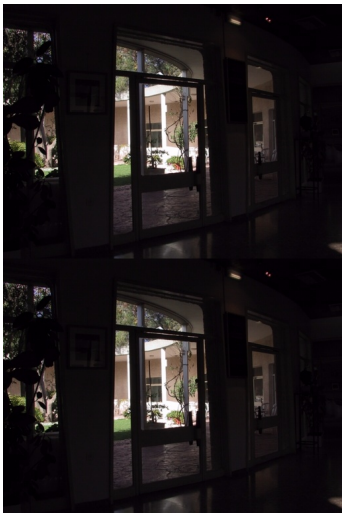
Part II. Results & Analysis:

Task 1: Image Enhancement

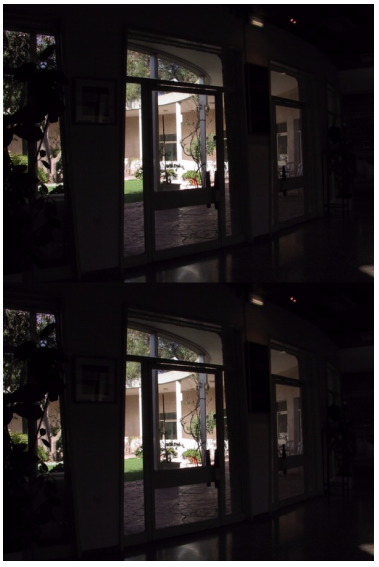
Gamma Correction



Gamma=0.5



Gamma=1



Gamma=1.5

- The above is the original image, and the bottom is the image after gamma correction.
- We can notice that as the gamma value increases, the resulting image becomes darker.

Histogram Equalization

Compare the above two methods

- Gamma correction adjusts the luminance of the image by applying a non-linear transformation to pixel values.
- Histogram equalization enhances the contrast by redistributing the intensities over the entire range of possible values.

Comparison	Gamma Correction	Histogram Equalization
Effect	Adjusts brightness with non-linear transformation, preserving details in highlights and shadows	Enhances contrast by redistributing intensity values, revealing hidden details
Use Cases	Suitable for images that are too dark or too bright, needing slight brightness adjustment	Suitable for images with low contrast, needing enhanced contrast and detail
Image Appearance	Improves brightness in dark areas while keeping highlights natural, resulting in a balanced image	Increases contrast, making details in dark and bright areas clearer, but can appear overly sharp or noisy
Advantages	Allows subtle brightness adjustments without affecting overall contrast, avoids over-sharpening	Suitable for revealing details hidden in dark and bright areas, enriching the image
Disadvantages	May not significantly improve image details if not adjusted properly	Can make the image look unnatural, adds noise, affecting the realism of the image

Bonus (Unsharpe Masking)

- Use **Unsharpe Masking** to enhance the image.The main goal is to sharpen the edge of the image.Below is the math equation.
- **Unsharp Masking** excels at sharpening and enhancing edges, making fine details more pronounced.
- **Unsharp Masking** is highly effective in making textures and small features pop, which is crucial for detailed images.

- Pros
  - Enhances sharpness and edges, making the image appear crisper.
  - Effective for making fine details more pronounced.
- Cons
  - May amplify noise and artifacts, especially in already noisy images.
  - Can make the image look overly sharpened or harsh if not applied subtly.

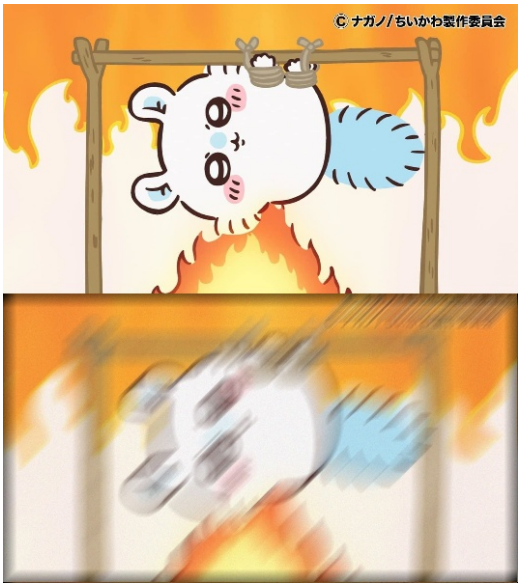
Conclusion:

Unsharp Masking is superior when you need precise sharpening and edge enhancement, especially in applications like photography or medical imaging. However, it requires careful tuning to avoid oversharpening or amplifying noise.

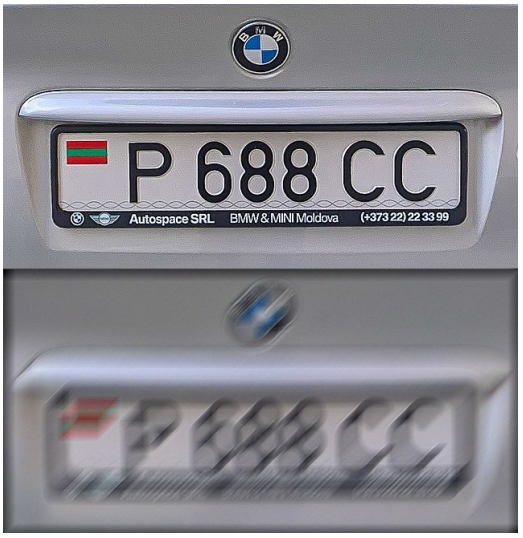
Task 2: Image Restoration

Minimum Mean Square Error (Wiener) Filtering

- degree : 46 length:3136

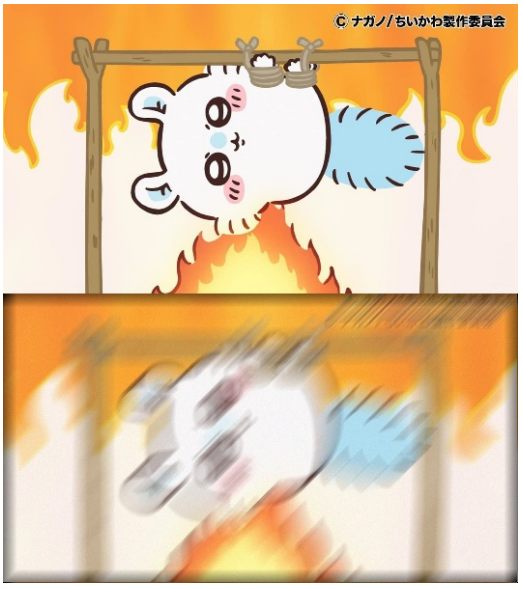


- degree : 92 length:486

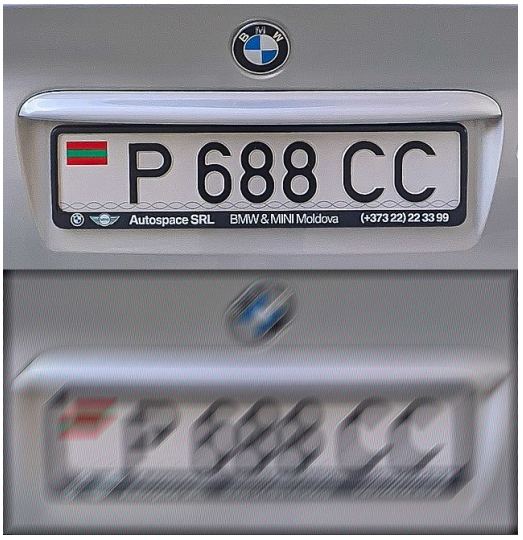


Constrained Least Squares Restoration

- degree : 46 length:3136



- degree : 92 length:486



Comparethe above two methods on different test cases

Constrained Least Squares (CLS): A restoration method that balances sharpness and noise reduction by minimizing reconstruction errors, preserving

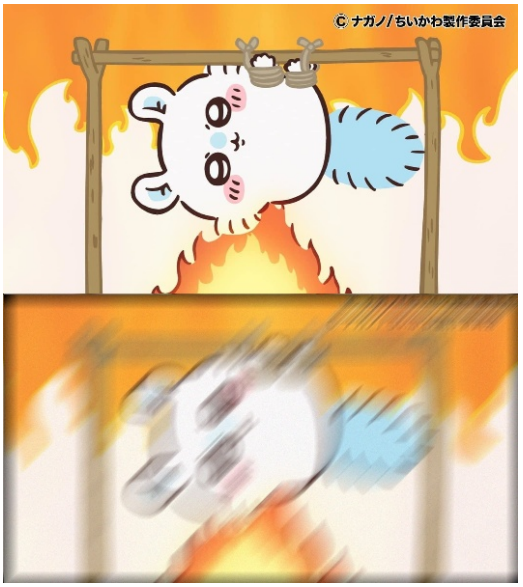
edge details effectively.

Wiener Filtering: A restoration technique that emphasizes noise reduction while smoothing the image, often at the expense of fine details.

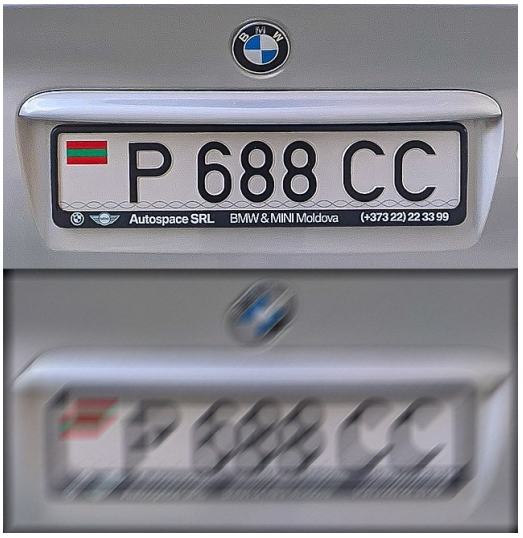
Comparison Aspect	Constrained Least Squares Restoration	Wiener Filtering
Sharpness	Sharper with better edge preservation	Slightly blurry, less detail
Noise Reduction	Balances noise reduction and detail retention, some noise remains	Stronger noise reduction, smoother appearance
Edge Details	Edges are sharp and well-defined	Edges are slightly blurred
Image Smoothness	Moderate smoothness	Smoother, effective at removing high-frequency noise
Best Use Case	Ideal for images requiring detail retention	Ideal for images needing noise smoothing

Bonus(Inverse filitering)

- degree : 46 length:3136



- degree : 92 length:486



- I use the inverse filitering to restore the image.

Methods Used:

The restoration for both images (testcase1 and testcase2) employs **Inverse Filtering**, targeting the degradation processes (Gaussian Blur and Motion Blur) to recover details.

Improvements in Image Quality:

1. **Testcase 1 (Animation Character)**:
    - The **inverse filtering** effectively restored the blurred edges of the character, making the details (like facial features and tail) more visible compared to the degraded version.
    - Compared to other methods like Wiener Filtering or Constrained Least Squares, it achieves sharper results but may amplify noise.
2. **Testcase 2 (Car Plate)**:
    - The car plate is now readable, with the text ("P 688 CC") and logo details significantly clearer than in the degraded version.
    - The improvement is prominent in high-frequency areas, such as text edges, which were smeared in the original blurred image.

Comparison with Previous Methods:

- **Inverse Filtering Advantages**:
    - Provides a sharp recovery when the blur kernel (Point Spread Function, PSF) is well-known.
    - Effective for images with moderate blur and low noise levels.
- **Inverse Filtering Limitations**:
    - Amplifies noise if present in the degraded image.
    - Requires precise knowledge of the degradation process for optimal results.

Part III. Answer the questions:

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

- **Problem** : Select the hyperparameter of psf length and angle.

- **Solution**:
  - **Edge detection** can help identify the primary blur direction in an image. For instance, using Canny edge detection or Sobel detection to locate edges and observe the direction and stretching of edge lines.
  - Using **Hough Transform**, the dominant line direction can be detected, allowing the estimation of the blur angle.
  - The extent of edge stretching provides clues about the PSF length. Longer blurred edges often indicate a longer PSF.

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

Problem:

The Wiener filter may amplify noise in high-frequency regions where the signal-to-noise ratio (SNR) is low. This results in a restored image with visible noise artifacts.

Solution:

1. **Regularization**: Incorporate a regularization term to suppress high-frequency noise during restoration. For example, Tikhonov regularization can help balance noise reduction and signal preservation.
2. **Thresholding**: Apply a frequency-based threshold to limit the amplification of noise in high-frequency components. Soft or hard thresholding techniques can be used to suppress insignificant coefficients.
3. **Post-processing**: Perform noise reduction after applying the Wiener filter using denoising methods such as Non-Local Means (NLM) or Total Variation (TV) denoising to further enhance the quality of the restored image.

What potential limitations might arise when using Constrained Least Squares Restoration for image restorations? Please suggest

possible
solutions to address them

- **Problem**: CLS requires accurate knowledge of the PSF. An incorrect or imprecise PSF leads to suboptimal restoration results.
- **Solution**:
    1. **Blind Deconvolution**: Incorporate blind deconvolution methods to jointly estimate the PSF and restore the image.
    2. **PSF Estimation**: Use edge detection and line analysis (e.g., via Hough transform) to estimate the PSF's properties, such as length and angle.