1 Image Filtering

1.1 Quá trình thực hiện

1.1.1 Padding

Đây là đoạn code:

```
def padding_img(img, filter_size=3):
    pad_width = filter_size // 2
    padded_img = cv2.copyMakeBorder(img, pad_width, pad_width,
    pad_width, pad_width, cv2.BORDER_REPLICATE)
    return padded_img
```

1.1.2 Mean Filter

Đây là đoạn code:

```
padded_img = padding_img(img, filter_size)
    kernel = (np.ones((filter_size, filter_size),
    dtype=np.float32) / (filter_size ** 2))
    smoothed_img = cv2.filter2D(padded_img, -1, kernel)
    pad_width = filter_size // 2
    smoothed_img = smoothed_img[pad_width:-pad_width, pad_width:-pad_width]
    return smoothed_img
```

Hàm trên áp dụng bộ lọc trung bình vào 'img' sau khi đã lấp đầy ảnh bằng Padding. Sau khi chuyển hóa xong hình ảnh, hàm quay lại kích thước vốn có.

Hình 1.1: Ảnh sau khi chạy hàm

1.1.3 Median Filter

Đây là đoạn code:

```
def median_filter(img, filter_size=3):
    padded_img = padding_img(img, filter_size)
    smoothed_img = np.zeros_like(img)
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            smoothed_img[i, j] = np.median(padded_img[i:i
            + filter_size, j:j + filter_size])
    return smoothed_img
```

Hàm trên áp dụng bộ lọc trung vị vào 'img' sau khi đã lấp đầy ảnh bằng Padding. Sau khi chuyển hóa xong hình ảnh, hàm quay lại kích thước vốn có.

Hình 1.2: Ảnh sau khi chạy hàm

1.1.4 Peak Signal-to-Noise Ratio

Đây là đoạn code:

```
def psnr(gt_img, smooth_img):
    mse = np.mean((gt_img - smooth_img) ** 2)
    if mse == 0:
        return float('inf')
    max_pixel = 255.0
    psnr_score = 20 * math.log10(max_pixel / math.sqrt(mse))
    return psnr_score
```

Hàm trên trả về chỉ số PSNR giữa hình ảnh gốc với hình ảnh sau khi được làm mịn

1.2 Kết quả cuối cùng

Hình 1.3: Ảnh sau khi chạy chương trình

2 Fourier Transform

2.1 Biến đổi Fourier rời rạc (DFT)

2.1.1 1 chiều (1D)

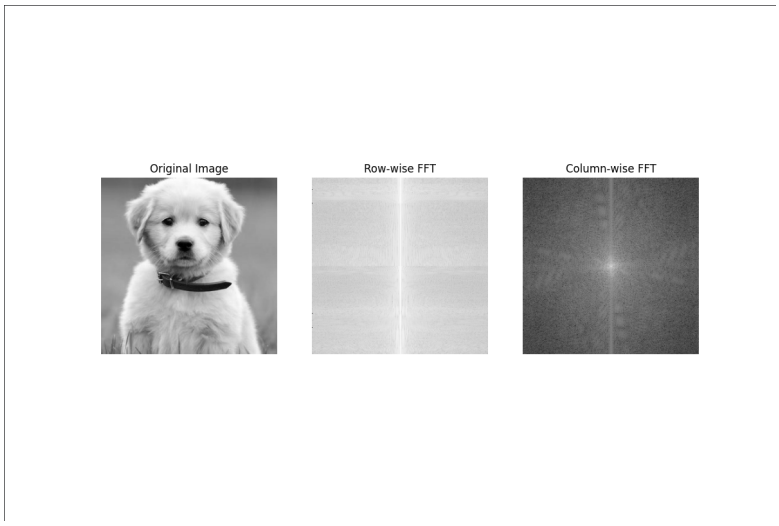
```
def DFT_slow(data):
    N = len(data)
    n = np.arange(N)
    k = n.reshape((N, 1))
    M = np.exp(-2j * np.pi * k * n / N)
    return np.dot(M, data)
```

Hàm này thực hiện DFT cho một tín hiệu một chiều được đưa vào dưới dạng một mảng numpy 1D, sử dụng vòng lặp để tính toán từng phần tử

2.1.2 2 chiều (2D)

```
def DFT_2D(gray_img):
    row_fft = np.fft.fft(gray_img, axis=1)
    row_col_fft = np.fft.fft(row_fft, axis=0)
    return row_fft, row_col_fft
```

Hàm này thực hiện DFT hai chiều cho ảnh grayscale đầu vào, sử dụng hàm fft của thư viện numpy để tính toán DFT theo hàng và theo cột.



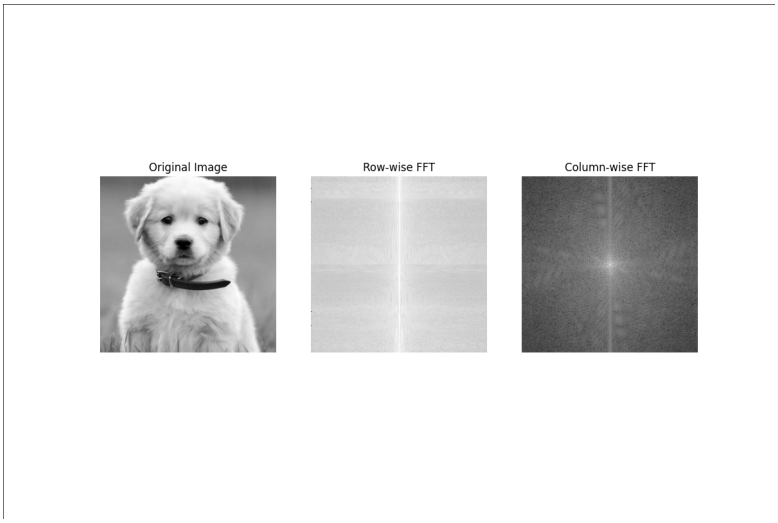
Hình 2.1: Ảnh sau khi chạy chương trình

2.2 Frequency removal

2.2.1 Quy trình lọc tần

Đây là đoạn code:

```
def filter_frequency(orig_img, mask):
    f_img = np.fft.fft2(orig_img)
    f_img = np.fft.fftshift(f_img)
    f_img = f_img * mask
    f_img = np.fft.ifftshift(f_img)
    img = np.fft.ifft2(f_img)
    f_img = np.fft.ifftshift(f_img)
    img = np.abs(img)
    f_img = np.abs(f_img)
    return f_img, img
```



Hình 2.2: Ảnh sau khi chạy hàm

Hàm này thực hiện lọc tần số trên ảnh grayscale dựa mask cho trước

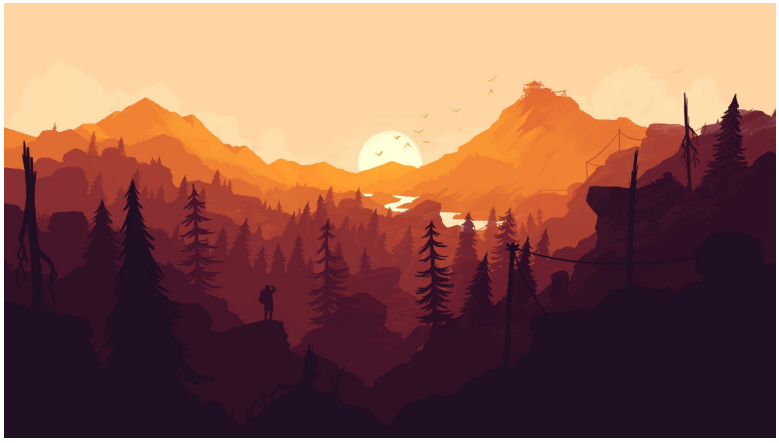
2.2.2 Tạo ảnh Hybrid

Đây là đoạn code:

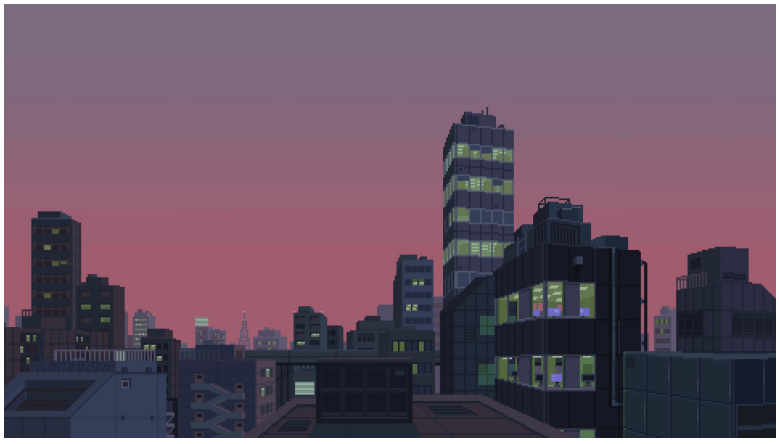
```
def create_hybrid_img(img1, img2, r):
    img1_blur = cv2.GaussianBlur(img1, (5, 5), 0)
    img2_blur = cv2.GaussianBlur(img2, (5, 5), 0)
```

```
    mask = np.zeros_like(img1, dtype=np.float32)
    mask[:r, :] = 1.0
    hybrid_img = img1_blur * mask + img2_blur * (1.0 - mask)
    return hybrid_img
```

Hàm áp dụng làm mờ Gaussian vào 2 ảnh đầu vào và khởi tạo 1 lớp mask. Sau khi kết hợp tần thấp ở ảnh 1 và tần cao ở ảnh 2 qua việc nhân với mask, ta tạo ra được hình ảnh Hybrid



Hình 2.3: Ảnh 1



Hình 2.4: Ảnh 2

Hình 2.5: Kết quả ảnh Hybrid