Digital Image Processing HW4

鄧奕辰 311605014

- **Image Restoration:** Gaussian Blur + Motion Blur

An algorithm for restoration motion-blurred images involves the generation of a **Point Spread Function (PSF)**, the creation of a **Wiener filter**, and the application of the **filter to a motion-blurred image in the frequency domain**. In this application, we handle each three channels separately. After the restoring, we will merge it into a colored image.

```
// Splitting the image into its component channels
vector<Mat> channels(3);
split(blurImg, channels);

// length of a motion
int LEN = 25;
// angle of a motion in degrees
double THETA = 45;
// signal to noise ratio
int snr = 25;

for(int i = 0; i < 3; ++i){
    // Ensure channel are even image
    Rect roi = Rect(0, 0, channels[i].cols & -2, channels[i].rows & -2);
    //Hw calculation (start)
    Mat Hw, h;
    calcPSF(h, roi.size(), LEN, THETA);
    calcWnrFilter(h, Hw, 1.0 / double(snr));
    //Hw calculation (stop)
    channels[i].convertTo(channels[i], CV_32F);
    // filtering (start)
    filter2DFreq(channels[i](roi), channels[i], Hw);
    // filtering (stop)
    channels[i].convertTo(channels[i], CV_8U);
    normalize(channels[i], channels[i], 0, 255, NORM_MINMAX);
}
```

A function calcPSF() forms a PSF according to input parameters LEN and THETA (in degrees):

```
void calcPSF(Mat& outputImg, Size filterSize, int len, double theta)
{
    Mat h(filterSize, CV_32F, Scalar(0));
    Point point(filterSize.width / 2, filterSize.height / 2);
    ellipse(h, point, Size(0, cvRound(float(len) / 2.0)), 90.0 -
 theta, 0, 360, Scalar(255), FILLED);
    Scalar summa = sum(h);
    outputImg = h / summa[0];
}
```

```
void calcWnrFilter(const Mat& input_h_PSF, Mat& output_G, double nsr)
{
    Mat h_PSF_shifted;
    fftshift(input_h_PSF, h_PSF_shifted);
    Mat planes[2] = { Mat_<float>(h_PSF_shifted.clone()), Mat::zeros(
h_PSF_shifted.size(), CV_32F) };
    Mat complexI;
    merge(planes, 2, complexI);
    dft(complexI, complexI);
    split(complexI, planes);
    Mat denom;
    pow(abs(planes[0]), 2, denom);
    denom += nsr;
    divide(planes[0], denom, output_G);
}
```

The calcWnrFilter() function is designed to compute the Wiener filter in the frequency domain based on the given Point Spread Function (PSF) input_h_PSF and a specified noise-to-signal ratio (nsr). Using fftshift to ensure that it is suitable for frequency domain operations. Apply the DFT to the complex image. This transforms the image from spatial domain to frequency domain. Compute the squared magnitude of the frequency component corresponding to the PSF. Divide the frequency component of the PSF (planes[0]) by the computed denominator (denom) to obtain the Wiener filter (output_G).

- **Sharpen image**:
  After restoration, I will apply a sharpen filter to the image, increasing the contrast between adjacent pixels to enhance the details in an image.

```
void sharpen(const Mat& inputImg, Mat& outputImg)
{
    Mat sharpening_kernel = (Mat_<double>(3, 3) << 0, -1, 0,
        -1, 5, -1,
        0, -1, 0);
    filter2D(inputImg, outputImg, -1, sharpening_kernel);
}
```

- **PSNR evaluation**:
  Use PSNR to evaluate your result based on the original image.

```
double calculatePSNR(const Mat& I1, const Mat& I2)
{
    Mat s1;
    absdiff(I1, I2, s1);       // |I1 - I2|
    s1.convertTo(s1, CV_32F);  // cannot make a square on 8 bits
    s1 = s1.mul(s1);           // |I1 - I2|^2
    Scalar s = sum(s1);        // sum elements per channel
    double sse = s.val[0] + s.val[1] + s.val[2]; // sum channels
    if( sse <= 1e-10) // for small values return zero
        return 0;
    else
    {
        double mse  = sse / (double)(I1.channels() * I1.total());
        double psnr = 10.0 * log10((255 * 255) / mse);
        return psnr;
    }
}
```

I look up the **PSNR definition**, it not the same as TA provided.
This is how I calculate the PSNR:

$$PSNR = 10 \cdot \log_{10}\left(\frac{MAX_I^2}{\frac{1}{3mn}\sum_{R,G,B}\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}[I_{color}(i,j) - K_{color}(i,j)]^2}\right)$$

- Result:
  - Image 1

  - Image 1 PSNR evaluation: PSNR: 20.6616
  - Image 2

  - Image 2 numbers:

WYS573	AAFM756	PHP2455	MKA532	405ZWU
MAY794	AFV2018	993KCM	YUT207	7121AN8
YMX644	MMG604	MKM239	378984K	JJS269
V67SFL	JJS131	552AOY	2AA4510	RCA3412
992KSM	9427A06	HPR476	YUT042	HLFV4
8SA231	4144AGN	YSE068	MHF686	342AE
YUT002	HHG352	JGN048	SAB3399	11H38