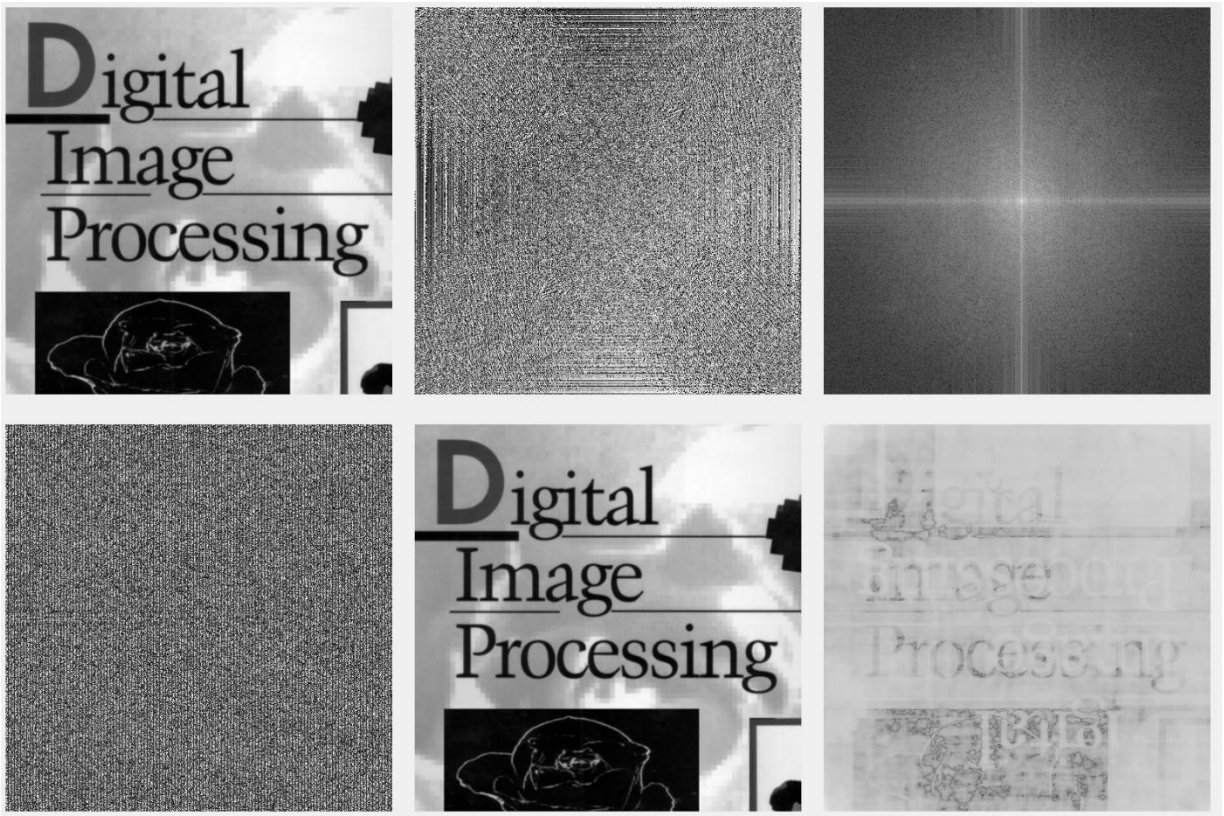
Principles and Applications of Digital Image Processing

HW4 R12631055 林東甫

Part1

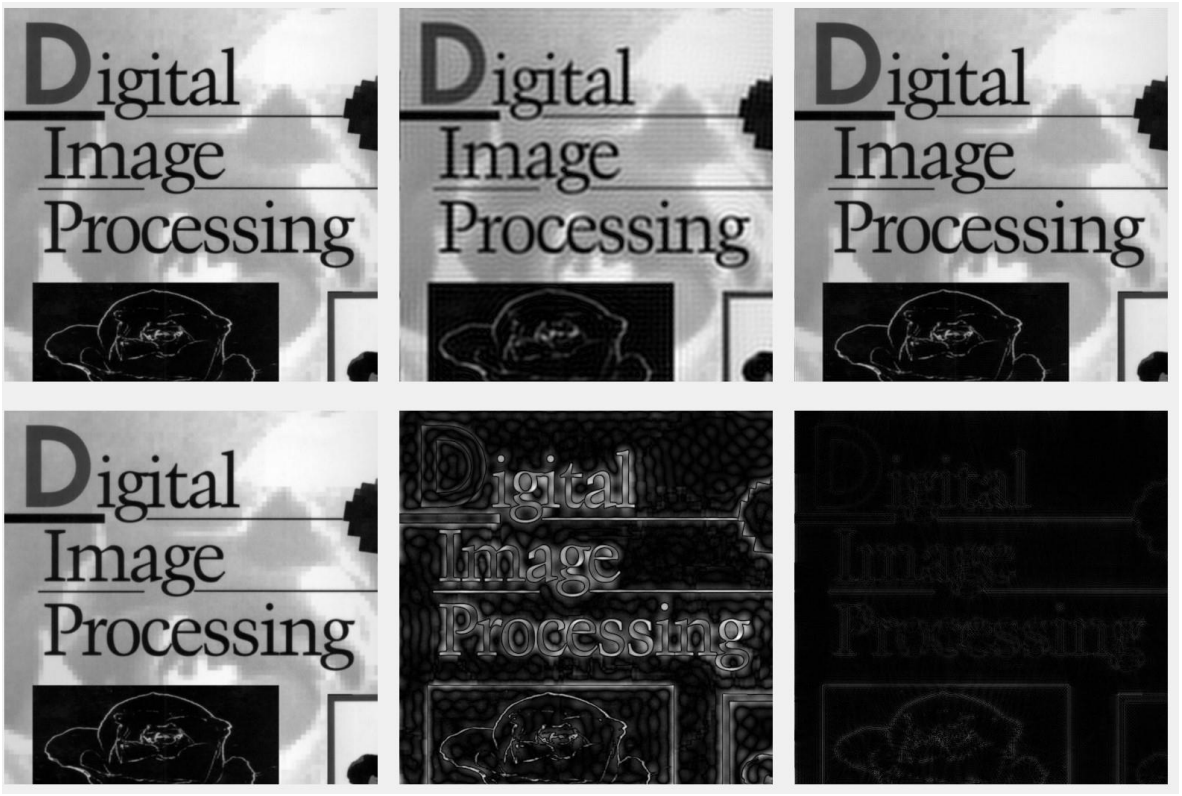


左上:原始圖片 中上: image phase angle 右上: image spectrum
左下: spectrum src 中下:FFT->iFFT 後 右下:sub

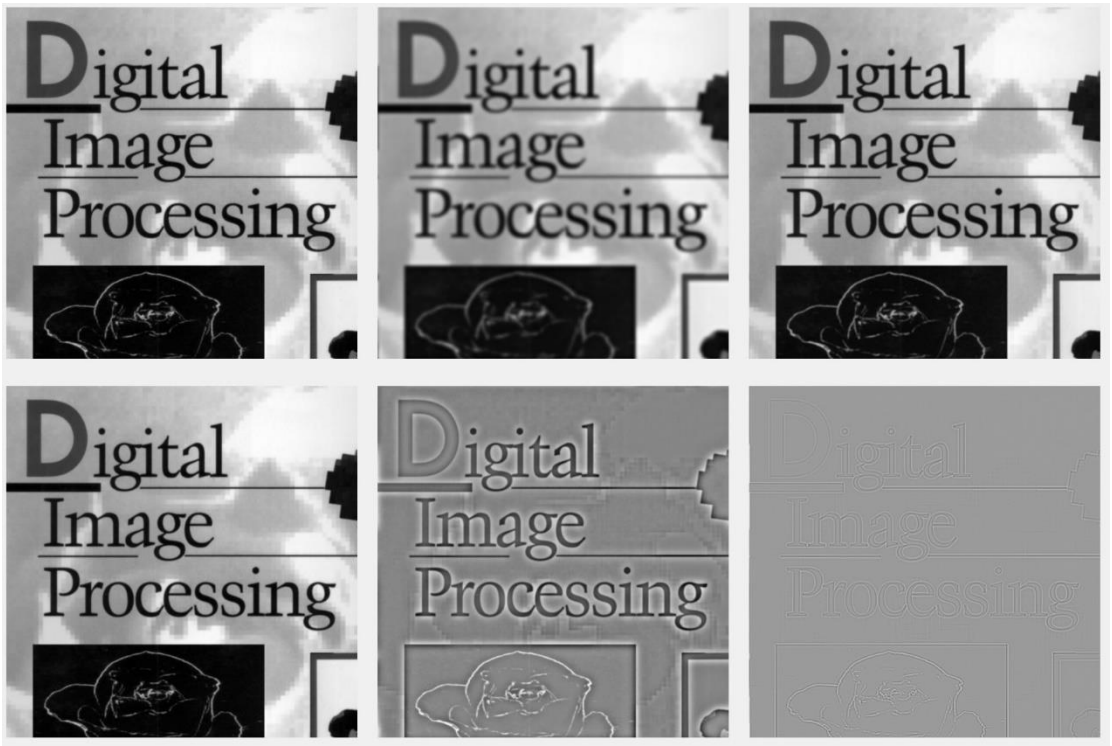
```
Mat padded; //expand input image to optimal size
int m = getOptimalDFTSize( img.rows );
int n = getOptimalDFTSize( img.cols ); // on the border add zero values
copyMakeBorder(img, padded, 0, m - img.rows, 0, n - img.cols, BORDER_CONSTANT, Scalar::all(0));

Mat planes[] = {Mat_<float>(padded), Mat::zeros(padded.size(), CV_32F)};
Mat complexI;
merge(planes, 2, complexI); // Add to the expanded another plane with zeros

dft(complexI, complexI); // this way the result may fit in the source matrix
// compute the magnitude and switch to logarithmic scale
// => log(1 + sqrt(Re(DFT(I))^2 + Im(DFT(I))^2))
split(complexI, planes); // planes[0] = Re(DFT(I), planes[1] = Im(DFT(I))
//fftshift(planes[0], planes[1]);
Mat amplitude, angle;
magnitude(planes[0], planes[1], amplitude);// planes[0] = magnitude


phase(planes[0], planes[1], angle);
angle.convertTo(angle, CV_8U);
magnitude(planes[0], planes[1], planes[0]);
```

基本上與上一次作業相似,將影像先做 FFT 以後,在進行 iFFT 後得回相似影像,然而可能因為浮點數計算上因此與原圖會有若干差異,諸如:離散取樣:影通常以 pixel 離散取樣,能在取樣和反變換過程中失真,也可能因為計算或截斷某些部分而與原始影像不完全相同.以我自己的程式來看,影像尺寸越大,處理時間越久,實感蠻接近 O(N^2)

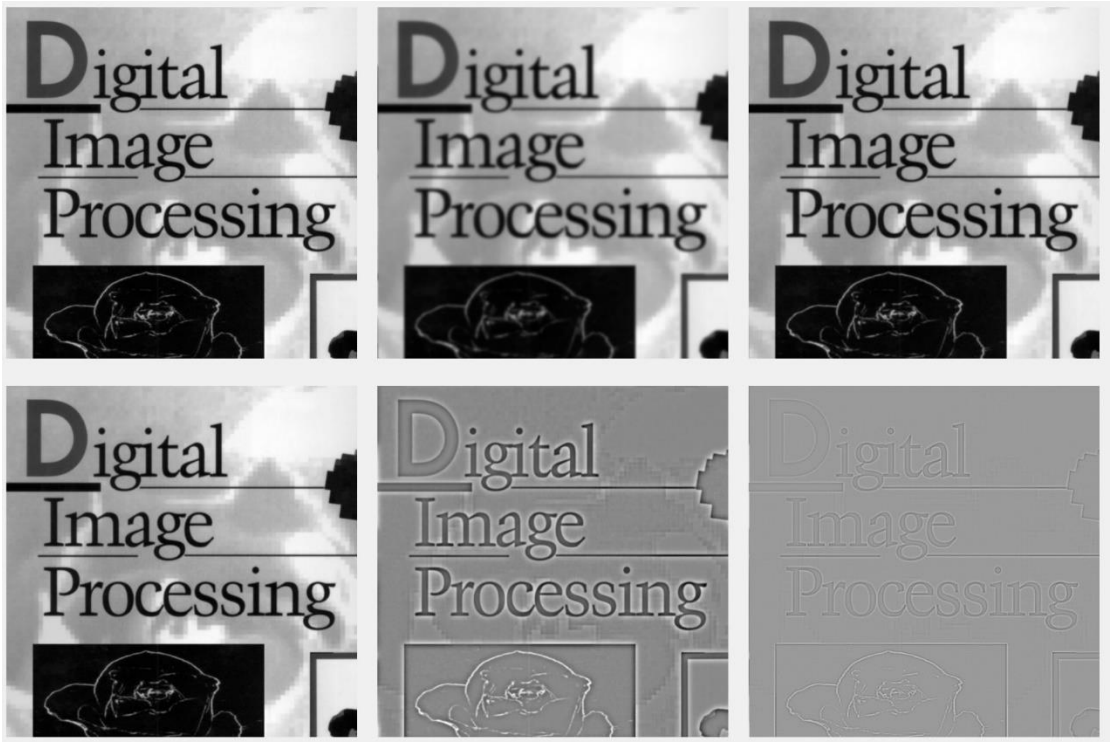
Part2



左上:原始圖片　中上: ILPF d0=60　右上: ILPF d0=160
左下:　原始圖片 中下:IHPF d0=20　右下: IHPF d0=120



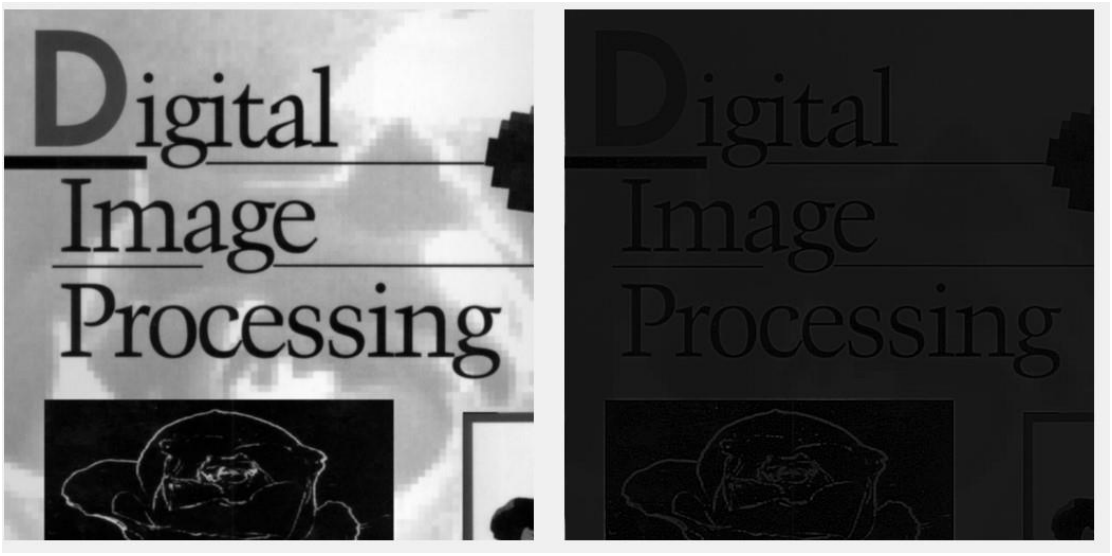
左上:原始圖片　中上: BLPF d0=60　右上: BLPF d0=160
左下:　原始圖片 中下:BHPF d0=20　右下: BHPF d0=120



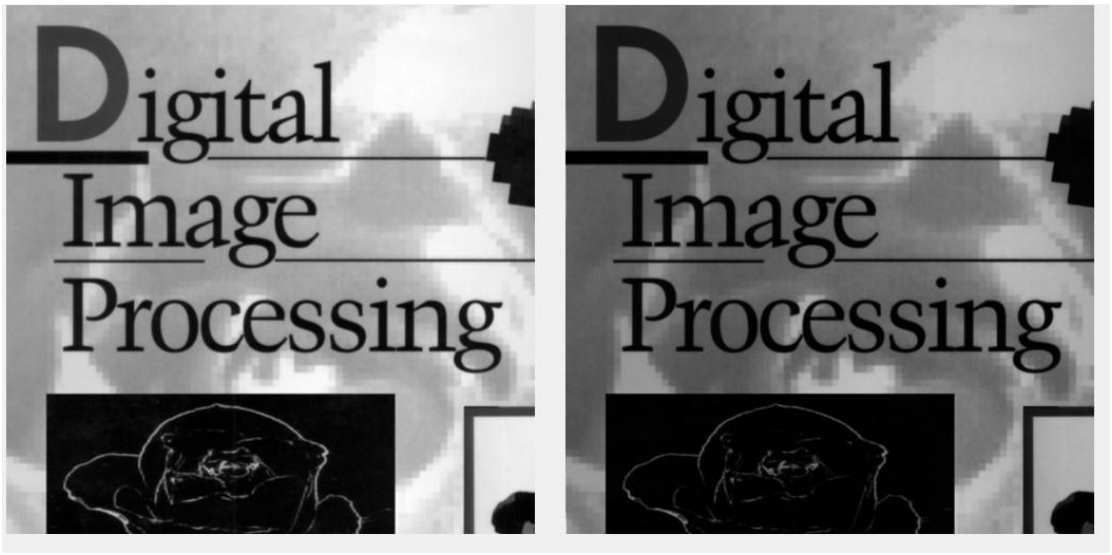
左上:原始圖片　中上: GLPF d0=60　右上: GLPF d0=160
左下:　原始圖片 中下:GHPF d0=20　右下: GHPF d0=120

cut-off frequency 在 lowpass 的時候,通常只保留低頻而過濾掉高頻,因此會用來去除高頻 noise,也因此整體比較平滑而且模糊一點,可以應用在去除 noise 或是模糊化.而在 highpass 的時候,會保留高頻而過濾掉低頻.可用於增強細節和邊緣特徵.通常用於銳化處理,因為它有助於突出細節.

Part3



H=2.00 L=0.25 D=160



H=2.00 L=1.5 D=160

H=3.00 L=1.5 D=160

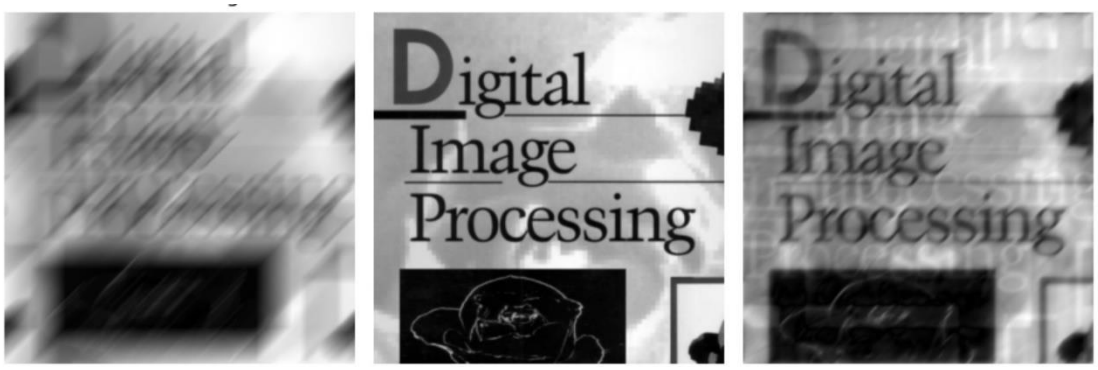
Cut-off Frequency 較高會使高頻的變化更為敏感，而較低會導致對較低頻率成分的過濾。調整 Cut-off Frequency 可以控制處理後圖像的細節程度。

同時也可以看出 L 越高整體會越明亮 H 越高整體越暗,但對比效果越好.

Part4

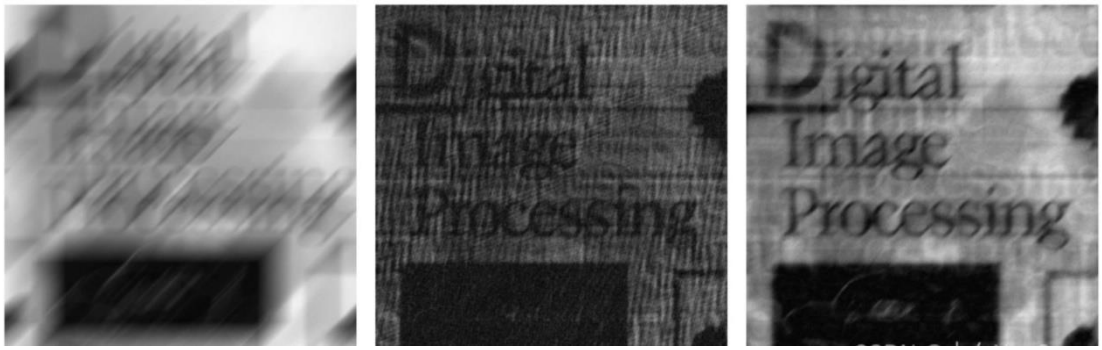
```
cv::Mat makeBlurred(const cv::Mat& image, const cv::Mat& PSF, double eps) {
    cv::Mat fftImg, fftPSF;
    cv::dft(image, fftImg, cv::DFT_COMPLEX_OUTPUT);
    cv::dft(PSF, fftPSF, cv::DFT_COMPLEX_OUTPUT);
    fftPSF += eps;

    cv::Mat fftBlur = fftImg.mul(fftPSF);
    cv::idft(fftBlur, fftBlur, cv::DFT_SCALE | cv::DFT_REAL_OUTPUT);
    cv::Mat result;
    cv::normalize(fftBlur, result, 0, 255, cv::NORM_MINMAX);
    return result;
}

cv::Mat inverseFilter(const cv::Mat& image, const cv::Mat& PSF, double eps) {
    cv::Mat fftImg, fftPSF;
    cv::dft(image, fftImg, cv::DFT_COMPLEX_OUTPUT);
    cv::dft(PSF, fftPSF, cv::DFT_COMPLEX_OUTPUT);
    fftPSF += eps;

    cv::Mat imgInvFilter = fftImg / fftPSF;
    cv::idft(imgInvFilter, imgInvFilter, cv::DFT_SCALE | cv::DFT_REAL_OUTPUT);
    cv::Mat result;
    cv::normalize(imgInvFilter, result, 0, 255, cv::NORM_MINMAX);
    return result;
}

cv::Mat wienerFilter(const cv::Mat& input, const cv::Mat& PSF, double eps, double K = 0.01) {
    cv::Mat fftImg, fftPSF, fftWiener;
    cv::dft(input, fftImg, cv::DFT_COMPLEX_OUTPUT);
    cv::dft(PSF, fftPSF, cv::DFT_COMPLEX_OUTPUT);
    fftPSF += eps;


    fftWiener = std::conj(fftPSF) / (cv::abs(fftPSF.mul(fftPSF)) + K);
    cv::Mat imgWienerFilter = fftImg.mul(fftWiener);
    cv::idft(imgWienerFilter, imgWienerFilter, cv::DFT_SCALE | cv::DFT_REAL_OUTPUT);
    cv::Mat result;
    cv::normalize(imgWienerFilter, result, 0, 255, cv::NORM_MINMAX);
    return result;
}
```

從上至下分別是 motion blurred, inverse filter and Wiener filter.



左邊是 motion blurred 後,中間是 inverse filter,右邊則是 Wiener filter.的效果

接著加入高斯雜訊



左邊是 motion blurred 加入高斯 中間是 inverse filter,右邊則是 Wiener filter.效果可以明顯看得出來一般情況下, inverse filter 對於動態模糊地處裡效果較好,但對抗雜訊能力很差,而 Wiener filter 則是在一般和有雜訊的情況下都差不多都能取得不錯的效果.整體來說 noise 使得影像變得更加模糊,也更難做到還原 motion blurred.