Image Processing HW4

tags: 2019imageprocess

github url: https://github.com/linnil1/2019ImageProcessing/tree/master/hw4 (https://github.com/linnil1/2019ImageProcessing/tree/master/hw4)

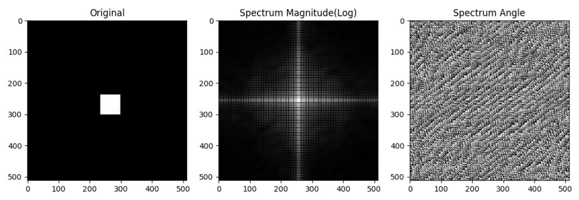
hackmd url: https://hackmd.io/EoTYaXyRRG-zJN2hNLn4Fg (https://hackmd.io/EoTYaXyRRG-zJN2hNLn4Fg)

Part1

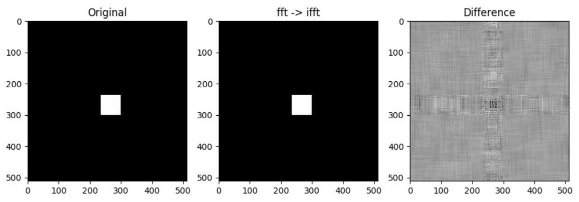
Duplicated in hw3. I would like to use recursion method for convenience.

It will be easy if you wrap fft function well.

```
gray_image[0::2, 1::2] *= -1
gray_image[1::2, 0::2] *= -1
fft_image  = np.fft.fft2(gray_image)
mag = np.log(np.abs(fft_image) + 1)
mag = (mag - mag.min()) / (mag.max() - mag.min())
ang = (np.arctan2(np.imag(fft_image), np.real(fft_image)) + np.pi) / 2 / np.pi * 255
```



The result of inversing back to the original image is not exactly same due to floating point error.



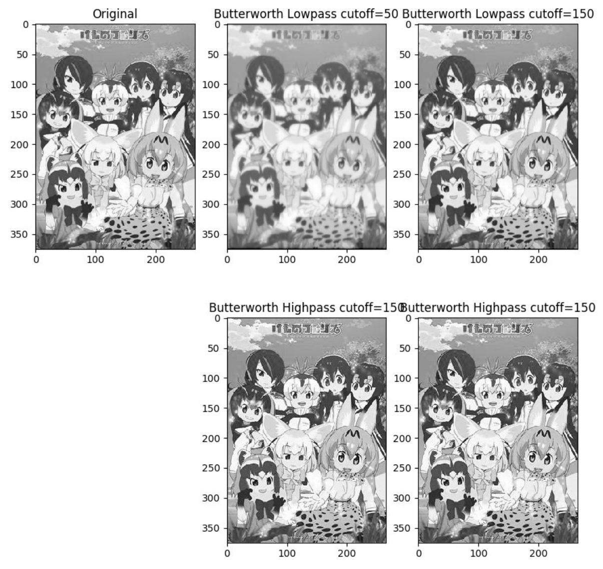
Part2

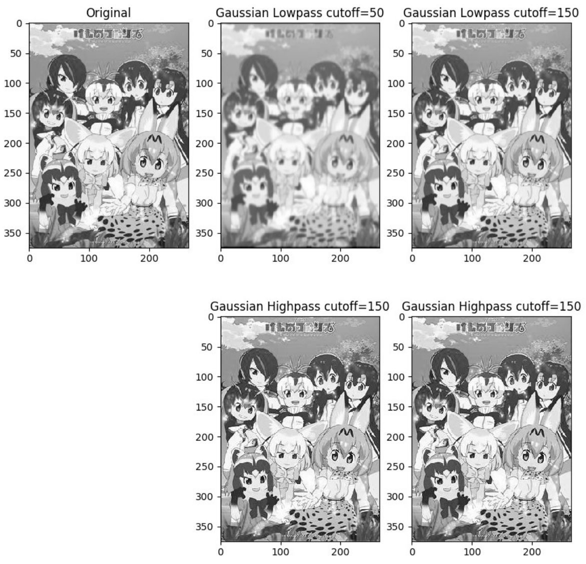
Duplicated in hw3. The method to reuse the function is passing a function with distance as it's arguments.

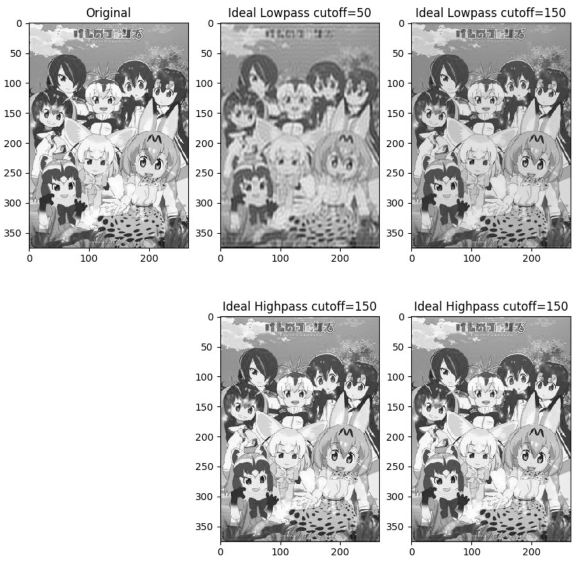
```
def idealLowpass(img, cutoff):
    """ Low Pass: Ideal """
    def idealFilter(dist):
        return dist < cutoff ** 2
    return feqOperation(img, idealFilter)
```

Only Ideal filter has the ripple effect(Not Good).

Note: Highpass filter and highboost are used together.


Original
Butterworth Lowpass cutoff=50
Butterworth Lowpass cutoff=150
Butterworth Highpass cutoff=150
Butterworth Highpass cutoff=150

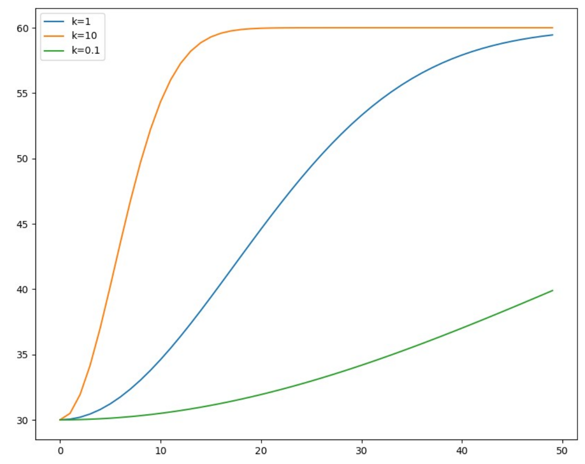

Original
Gaussian Lowpass cutoff=50
Gaussian Lowpass cutoff=150
Gaussian Highpass cutoff=150
Gaussian Highpass cutoff=150



Part3

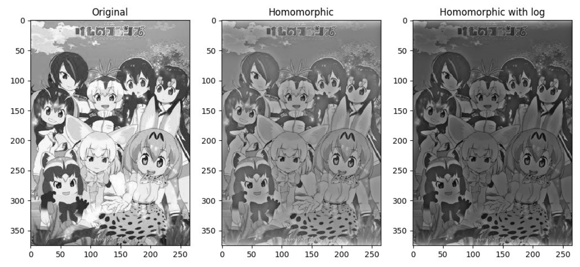
> Homomorphic filter

First plot the filter in one dimension. We can found it's good to set `c=1` then the cutoff frequency will almost be on the middle of γ_H and γ_L.

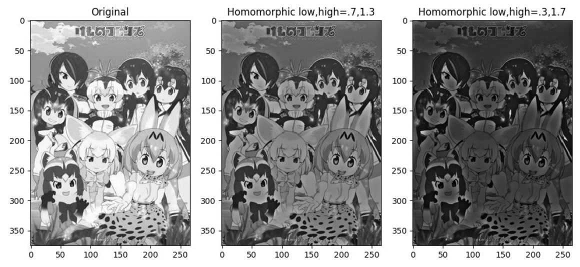


Discuss the effect:

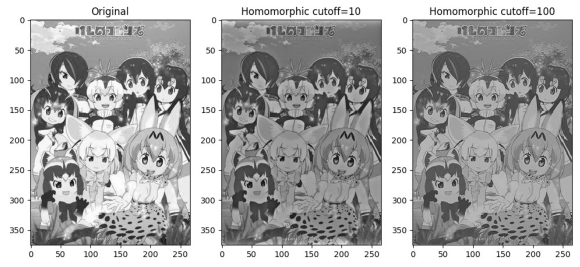
I try homomophric without using log, the effect show that the brightness still exist.



I try changing γ_H and γ_L to larger difference. We can see the homomophric suppress low frequency data and emphasize high frequency details together.

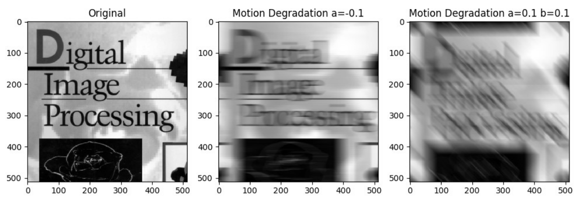


If I set cutoff value larger, the effect of emphasizing the image detail become not more significant.

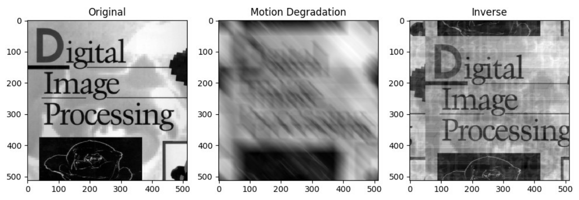


Part4

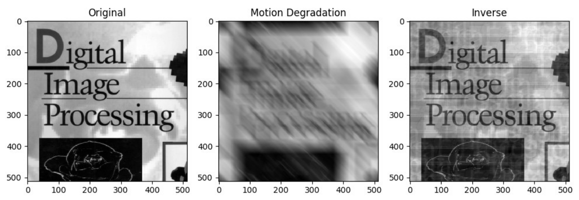
Let’s start with motion blur. After some experiments, I found the spectrum should be centered before applying motion blur.



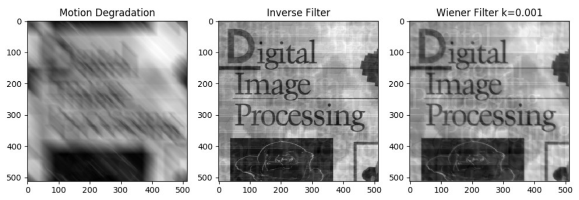
Then simply divided back



I found we do not need to change the sign of `a` and `b` when inversing it.

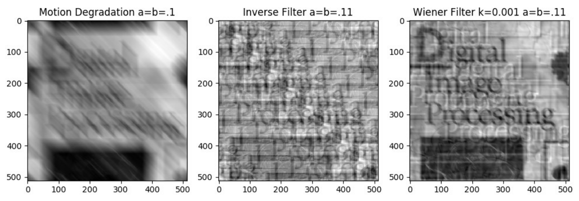


And inverse the filter with Wiener

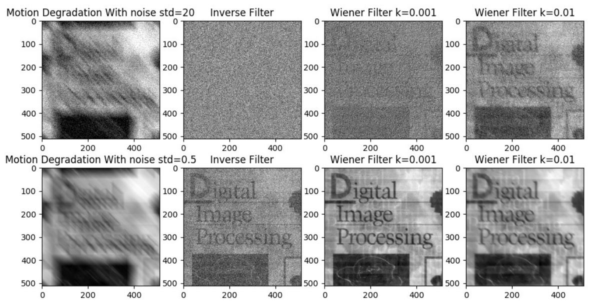


Wiener get bad reconstruction image, of course, because we know there is no noise and exact parameters of motion blur.

Let me show that the parameter is a little different from the truth. In the below result image, Wiener use K to make it more robust.



Next, I add gaussian noise and try it again.

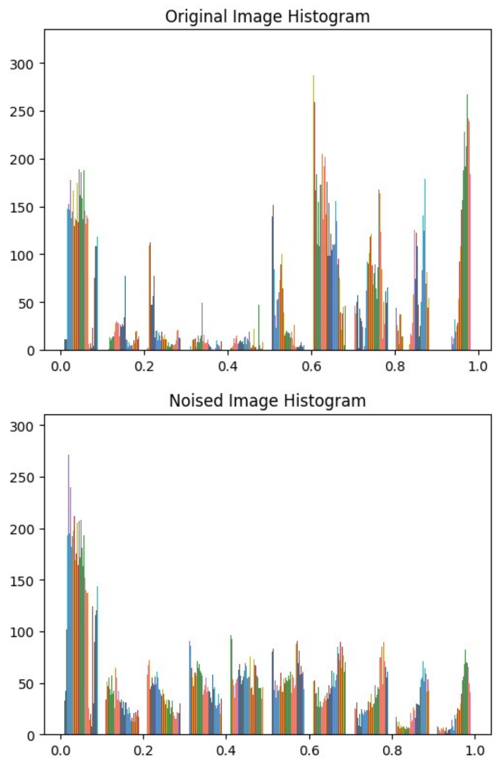


We can conclude some concepts:

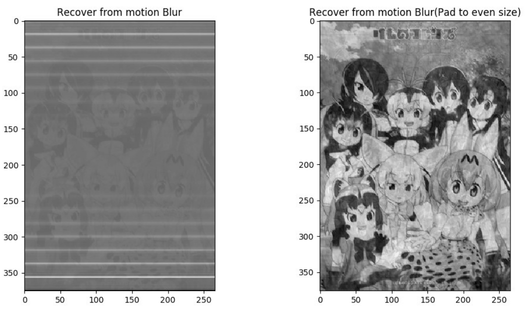
- The K value in Wiener filter play a important role to make model robust
- The lower K value get more detail after restoring. It is equivalent to inverse filter when K=0. However inverse filter is sensitive to noise.
- The higher the K value, more robust the model has. But the image will blurred simultaneously.

More things

I apply uniform noise on book image. We can find the histogram after applying noise become flatter and more even than untreated image.



And I found that we should set image to even size before applying FFT.



Program usage

Install on local machine

```
./setup.sh
```

Install by Docker

```
docker build -t linnil1/2019imageprocess .
docker run -ti --rm \
    --user $(id -u) \
    -v $PWD:/app \
    -v /tmp/.X11-unix:/tmp/.X11-unix \
    -e DISPLAY=$DISPLAY \
    linnil1/2019imageprocess python3 qt.py
```

Without QT

You can use command line as a postfix image processor.(Same as HW2)

For example

- `python3 hw4_np.py --read ../hw2/data/kemono_friends.jpg --graya --homomorphic 0.3 1.7 150 --show`
- `python3 hw4_np.py --read data/C1HW04_IMG02_2019.bmp --graya --motionblur 0.05 .05 --gaussiannoise 0 0.05 --motionwiener .01 .05 .05 --show` You can use `python3 hw4_np.py --help` to see more.

QT

`python3 qt.py` or `./run.sh`

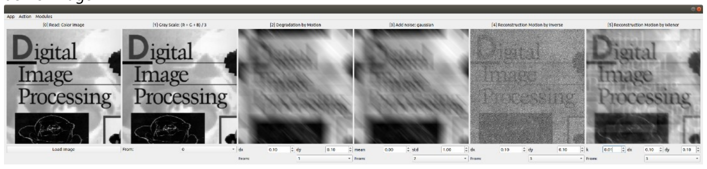
A dynamic UI image processor. You can decide what module you want to use in the processing pipeline. Also, you can add your own module painless. The advantage of modularization is extendable and easy to manage. Further more, you can redesign your UI without changing the core function. Just think about professional software like GIMP. Is it complicated?

> the interface was too complicated. Try to make it simpler with single input and single output

Note:

- All filter should run under gray scale image.
- The parameters of gaussian is cutoff value not sigma.
- To display the Difference operation in convenience, the code is not only `img1 - img2` . Instead, I use `(img1 - img2) / 2 + 127` .

demo Image



[0] Read: Color Image
[1] Gray Scale: (R + G + B) / 3 — From: 0
[2] Degradation by Motion — dx 0.10, dy 0.10, From: 1
[3] Add noise: gaussian — mean 0.00, std 1.00, From: 2
[4] Reconstruction Motion by Inverse — dx 0.10, dy 0.10, From: 3
[5] Reconstruction Motion by Wiener — k 0.01, dx 0.10, dy 0.10, From: 3