✦ Member-only story

Digital Image Processing using Fourier Transform in Python

Craig Chen · Follow

7 min read · Feb 17, 2020

Listen Share ••• More

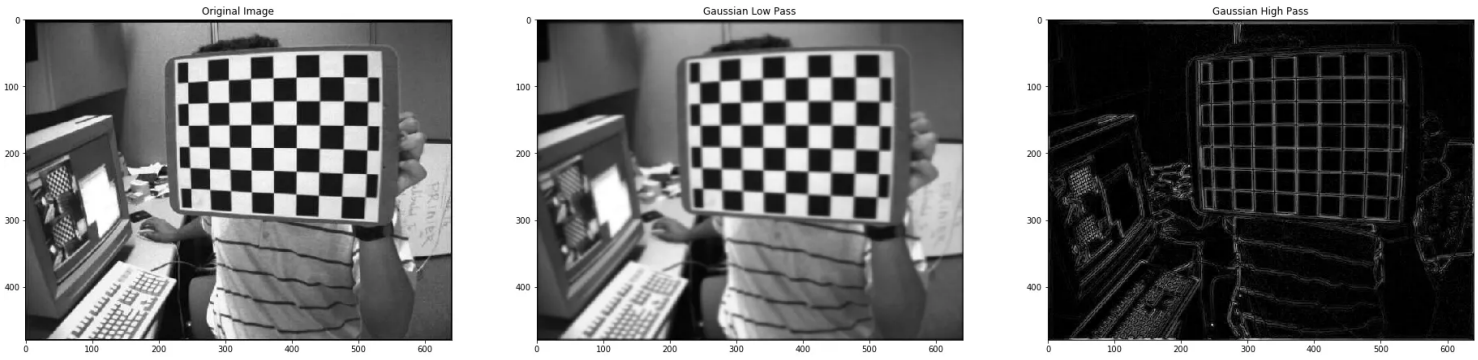
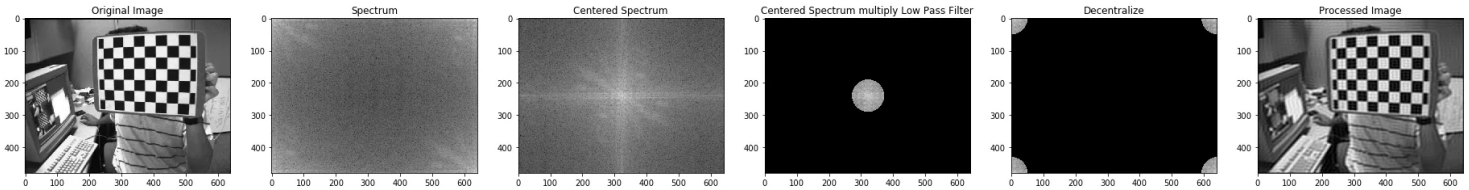


Figure (a): (from left to right) (1) Original image (2) With Gaussian Low Pass Filter (3) With Gaussian High Pass Filter. The original image in this post comes from OpenCV Github example.

Digital images are now part of our daily life. People can hardly live without it. Therefore, digital image processing becomes more and more important these days. How to increase the resolution of images or reduce noises of images are always hot topics. Fourier Transformation can help us out. We can utilize Fourier Transformation to transform our image information - gray scaled pixels into frequencies and do further process.

Today, I'll talk about how to utilize Fast Fourier Transformation in digital image processing, and how to implement it in Python. The process flow is as following (from left to right):



1. Implement Fast Fourier Transformation to transform gray scaled image into frequency
2. Visualize and Centralize zero-frequency component
3. Apply low/high pass filter to filter frequencies
4. Decentralize
5. Implement inverse Fast Fourier Transformation to generate image data

Fast Fourier Transformation

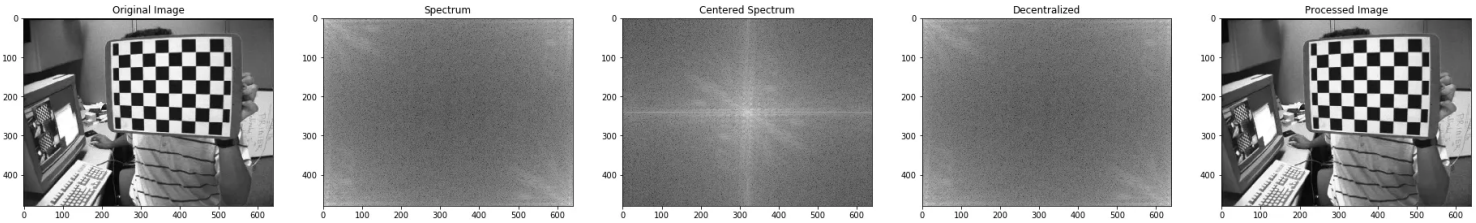


Figure (c): (From left to right) (1) Original image (2) FFT spectrum visual output (3) Centralized (4) Decentralized (5) Inverse FFT

Digital images, unlike light wave and sound wave in real life, are discrete because pixels are not continuous. That means we should implement Discrete Fourier Transformation (DFT) instead of Fourier Transformation. However, DFT process is often too slow to be practical. That is the reason why I chose Fast Fourier Transformation (FFT) to do the digital image processing.

Step 1: Compute the 2-dimensional Fast Fourier Transform.

The result from FFT process is a complex number array which is very difficult to visualize directly. Therefore, we have to transform it into 2-dimension space. Here are two ways that we can visualize this FFT result: 1. Spectrum 2. Phase angle.

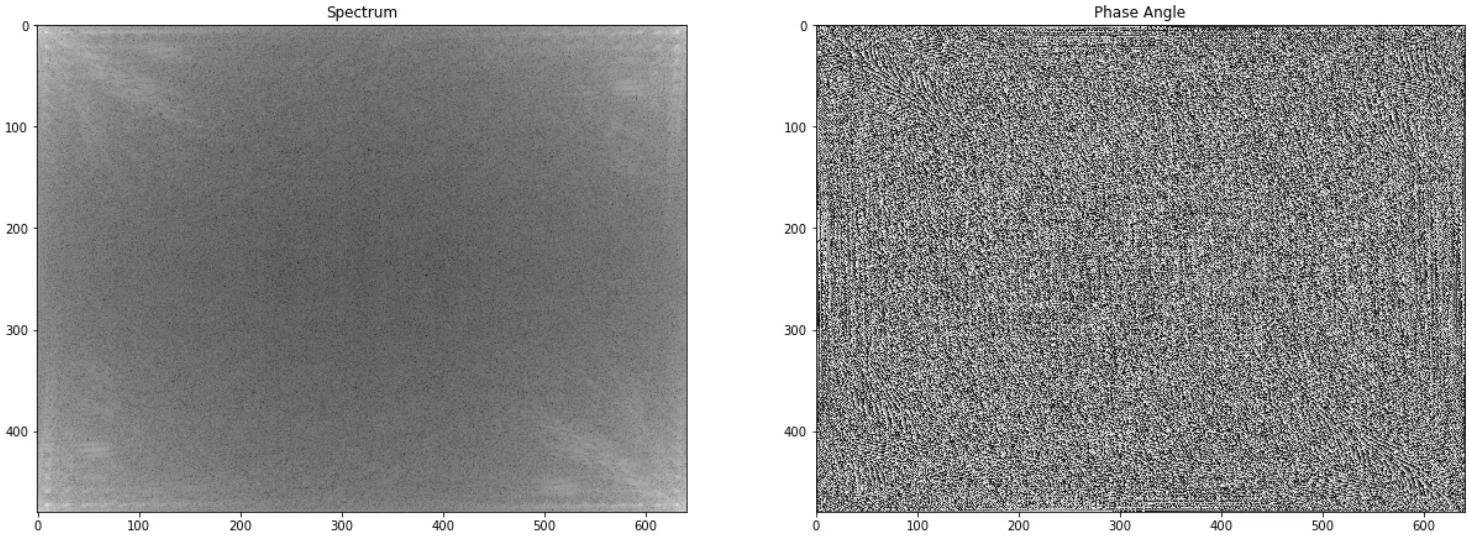


Figure (d): (from left to right) (1) Spectrum (2) Phase Angle

From ***Figure (d)(1),*** there are some symmetric patterns on the four corners. These patterns can be translated to the center of the image in the next step.

The white area in the spectrum image show the high power of frequency. The corners in the spectrum image represent low frequencies. Therefore, combining two points above, the white area on the corner indicates that there is high energy in low/zero frequencies which is a very normal situation for most images.

On the other side, it is hard to identify any noticeable patterns from ***Figure (d)(2).*** This did not indicate that the phase angle of FFT is totally useless because the phase preserves the shape characteristics which is an indispensable information for an image.

Step 2: Shift the zero-frequency component to the center of the spectrum.

2-D FFT has translation and rotation properties, so we can shift frequency without losing any piece of information. I shifted the zero-frequency component to the center of the spectrum which makes the spectrum image more visible for human. Moreover, this translation could help us implement high/low-pass filter easily.

Step 3: Inverse of Step 2. Shift the zero-frequency component back to original location

Step 4: Inverse of Step 1. Compute the 2-dimensional inverse Fast Fourier Transform.

The processes of step 3 and step 4 are converting the information from spectrum back to gray scale image. It could be done by applying inverse shifting and inverse FFT operation.

Code

In Python, we could utilize Numpy - numpy.fft to implement FFT operation easily.

After understanding the basic theory behind Fourier Transformation, it is time to figure out how to manipulate spectrum output to process images. First, we need to understand the low/high pass filter.

Low Pass Filter

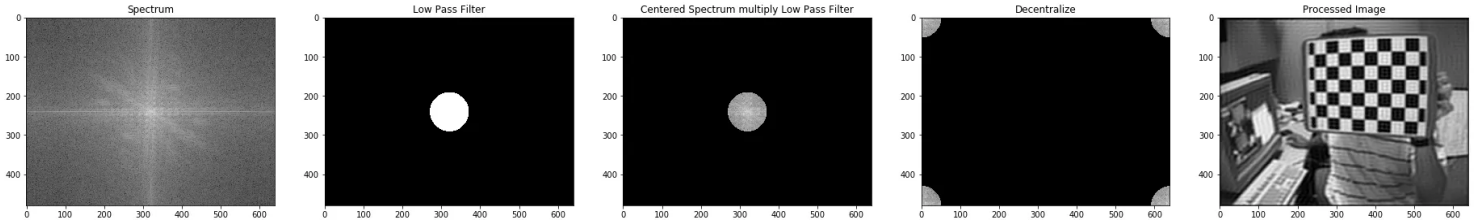


Figure (e):

Low pass filter is a filter that only allow low frequencies to pass through. Low frequencies in images mean pixel values that are changing slowly. For example, smooth area with slightly color changing in the image such as the center of new blank white paper is considered as a low frequency content.

Since the output of low pass filter only allow low frequencies to pass through, the high frequencies contents such as noises are blocked which make processed image has less noisy pixels. Therefore, low pass filter is highly used to remove the noises in images.

High Pass Filter

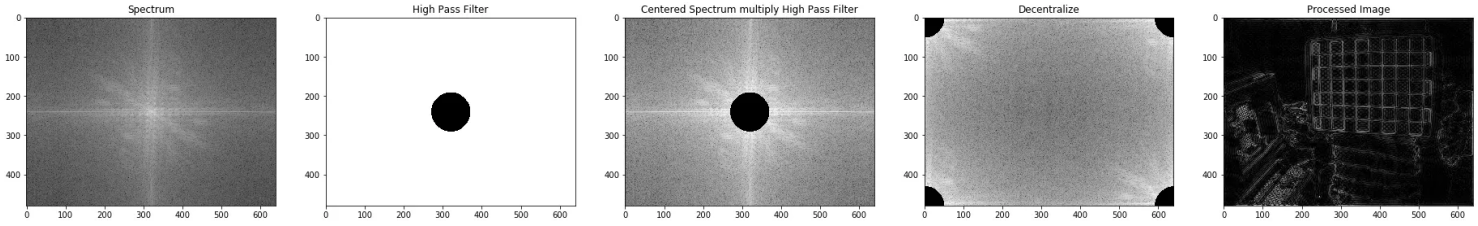


Figure (f)

High Pass filter, on the contrary, is a filter that only allow high frequencies to pass through. High frequencies in images mean pixel values that are changing dramatically. For example, Edge areas in the image with huge color changing such as the edge between two overlap white and black paper is consider as the high frequency content.

The output from high pass filter captures the edges in image which could be used to sharpen the original image with proper overlap calculation. This will enhance sharpness in original image making edges more clear.

From ***Figure(e)(5)*** and ***Figure(f)(5)***, we could notice that these two filters present different characteristics. Low pass filter tends to preserve overall information in an image. On the other hand, high pass filter is trying to identify changes in an image.

After realized how the low/high pass filter works in previous section, let's move on to get the right shape of filter.

Ideal Filter

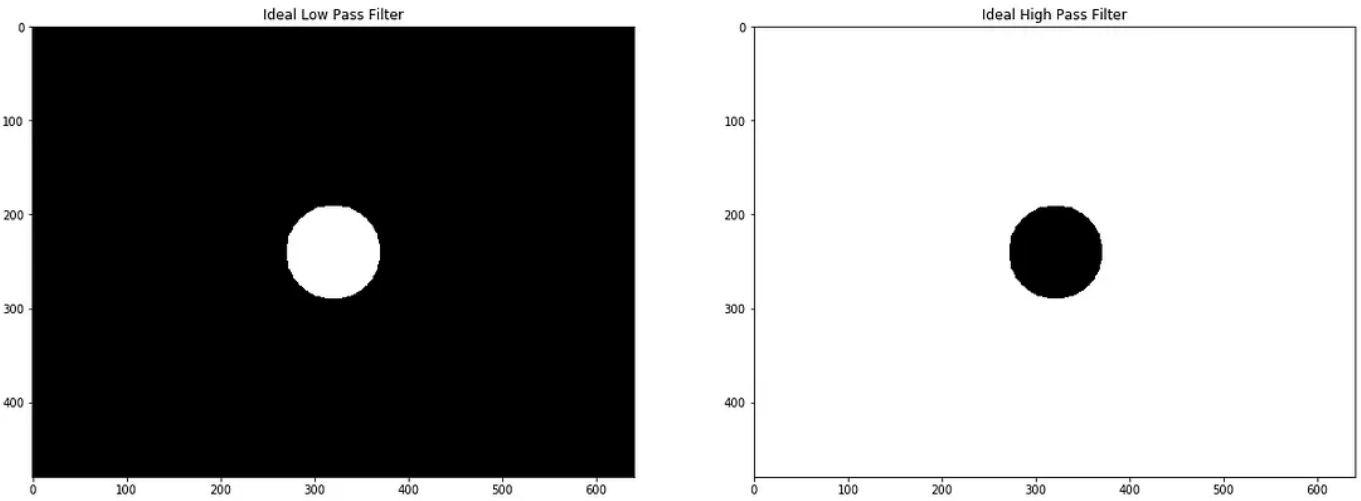


Figure (g): (from left to right) (1) Low pass filter with D_0**=50** (2) High pass filter with D_0**=50**

$$H(u,v) = \begin{cases} 1 & \text{if } D(u,v) \leq D_0 \\ 0 & \text{if } D(u,v) > D_0 \end{cases}$$

Formula (a): Formula for **ideal low pass filter** where D_0 is a positive constant and D(u, v) is the distance between a point (u, v) in the frequency domain and the center of the frequency rectangle

The idea which behinds ideal filter is very simple: Given a radius value D_0 ***as a threshold,*** low pass filter ***Figure (g)(1)*** has ***H(u, v)*** equals to 1 under the threshold, and ***H(u, v)*** equals to 0 when above the threshold.

$$H(u,v) = \begin{cases} 0 & \text{if } D(u,v) \leq D_0 \\ 1 & \text{if } D(u,v) > D_0 \end{cases}$$

Formula (b): Formula for **ideal high pass filter** where D_0 is a positive constant and D(u, v) is the distance between a point (u, v) in the frequency domain and the center of the frequency rectangle

On the contrary, high pass filter ***Figure (g)(2)*** has ***H(u, v)*** equals to 0 under threshold, and ***H(u, v)*** equals to 1 when above the threshold.

Butterworth Filter

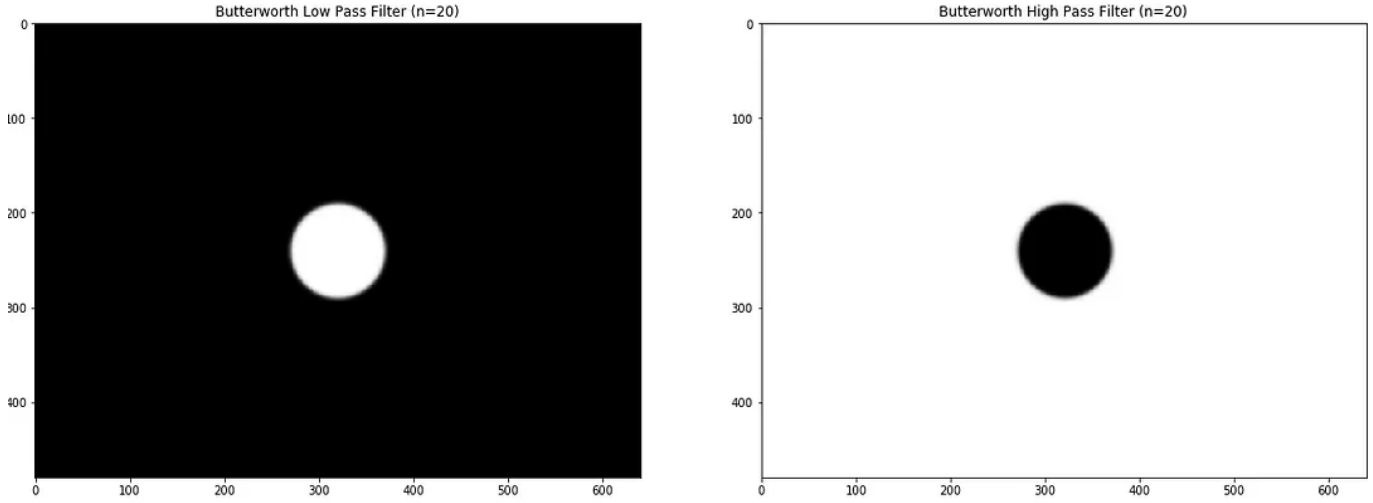


Figure (h): (from left to right) (1) Butterworth low pass filter with n=20 and D_0**=50**(2) Butterworth high pass filter with n=20 and D_0**=50**

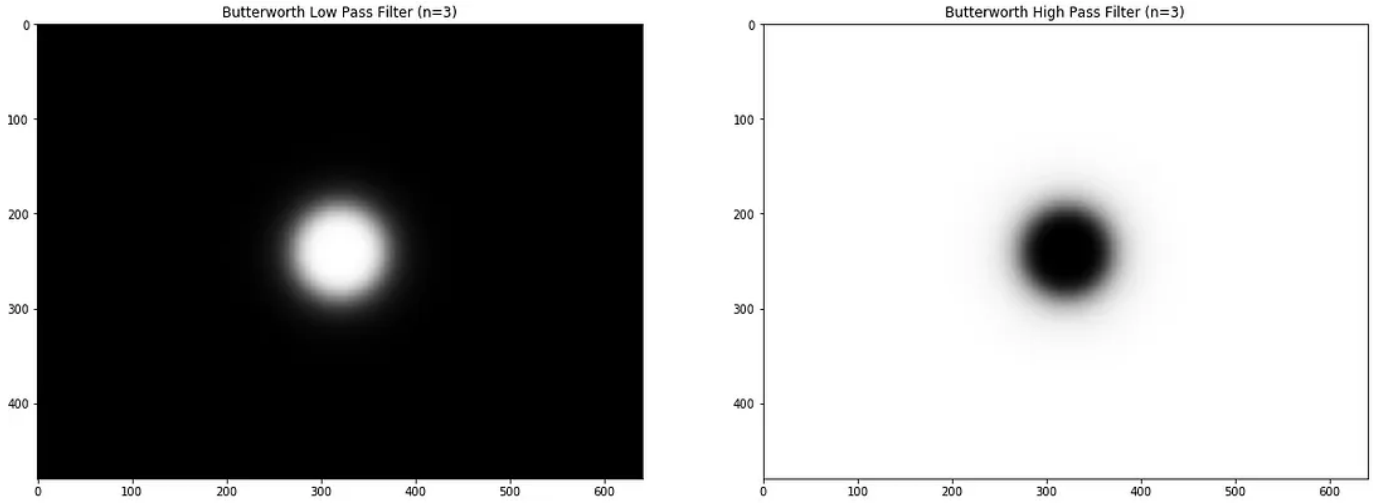


Figure (i): (from left to right) (1) Butterworth low pass filter with n=3 (2) Butterworth high pass filter with n=3

$$H(u, v) = \frac{1}{1 + [D(u, v)/D_0]^{2n}}$$

Formula (c): Formula for **Butterworth low pass filter** where D_0 is a positive constant and D(u, v) is the distance between a point (u, v) in the frequency domain and the center of the frequency rectangle

Unlike an ideal filter, a Butterworth filter does not have a sharp discontinuity that gives a clear cutoff between passed and filtered frequencies. Butterworth filter introduces a new ***parameter n*** in the function. While manipulating ***n,*** it affects the clearness of the cutoff between passed and filtered frequencies. ***Figure(h)*** and ***Figure(i)***

$$H(u,v) = \frac{1}{1 + [D_0/D(u,v)]^{2n}}$$

Formula (d): Formula for **Butterworth high pass filter** where D_0 is a positive constant and D(u, v) is the distance between a point (u, v) in the frequency domain and the center of the frequency rectangle

Gaussian Filter

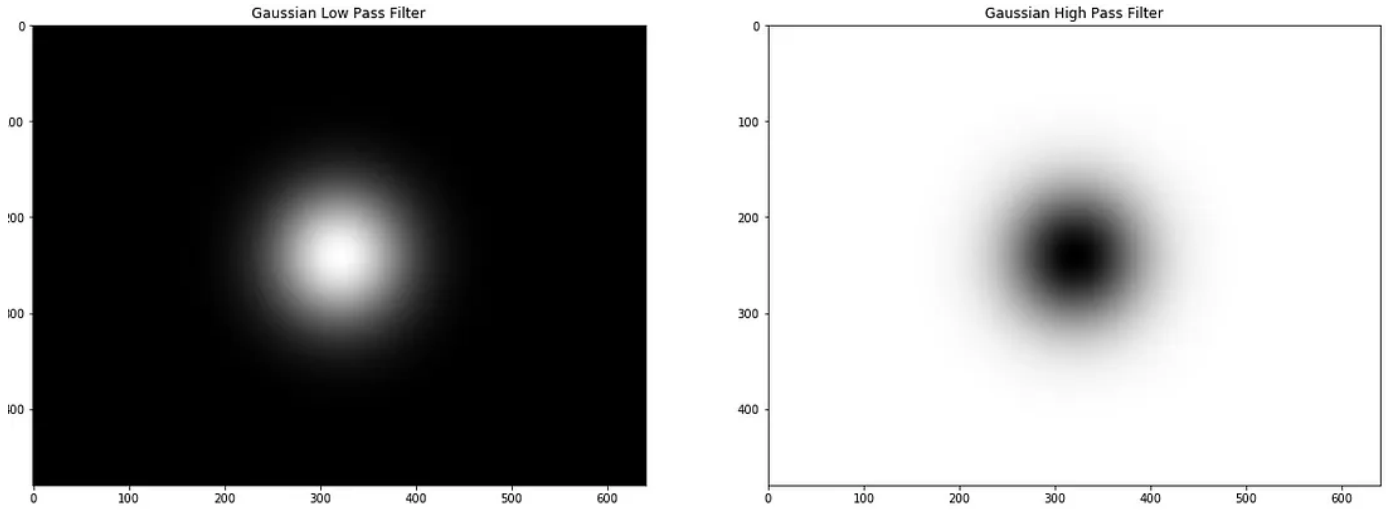


Figure (j): (from left to right) (1) Gaussian low pass filter with D_0**=50** (2) Gaussian high pass filter with D_0**=50**

$$H(u, v) = e^{-D^2(u, v)/2D_0^2}$$

Formula (e): Formula for **Gaussian low pass filter** where D_0 is a positive constant and D(u, v) is the distance between a point (u, v) in the frequency domain and the center of the frequency rectangle

Gaussian filter is a smoother cutoff version than Butterworth. The cutoff between passed and filtered frequencies is very blurry which leads to smoother processed images.

$$H(u, v) = 1 - e^{-D^2(u,v)/2D_0^2}$$

Formula (f): Formula for **Gaussian high pass filter** where D_0 is a positive constant and D(u, v) is the distance between a point (u, v) in the frequency domain and the center of the frequency rectangle

Filter Comparison

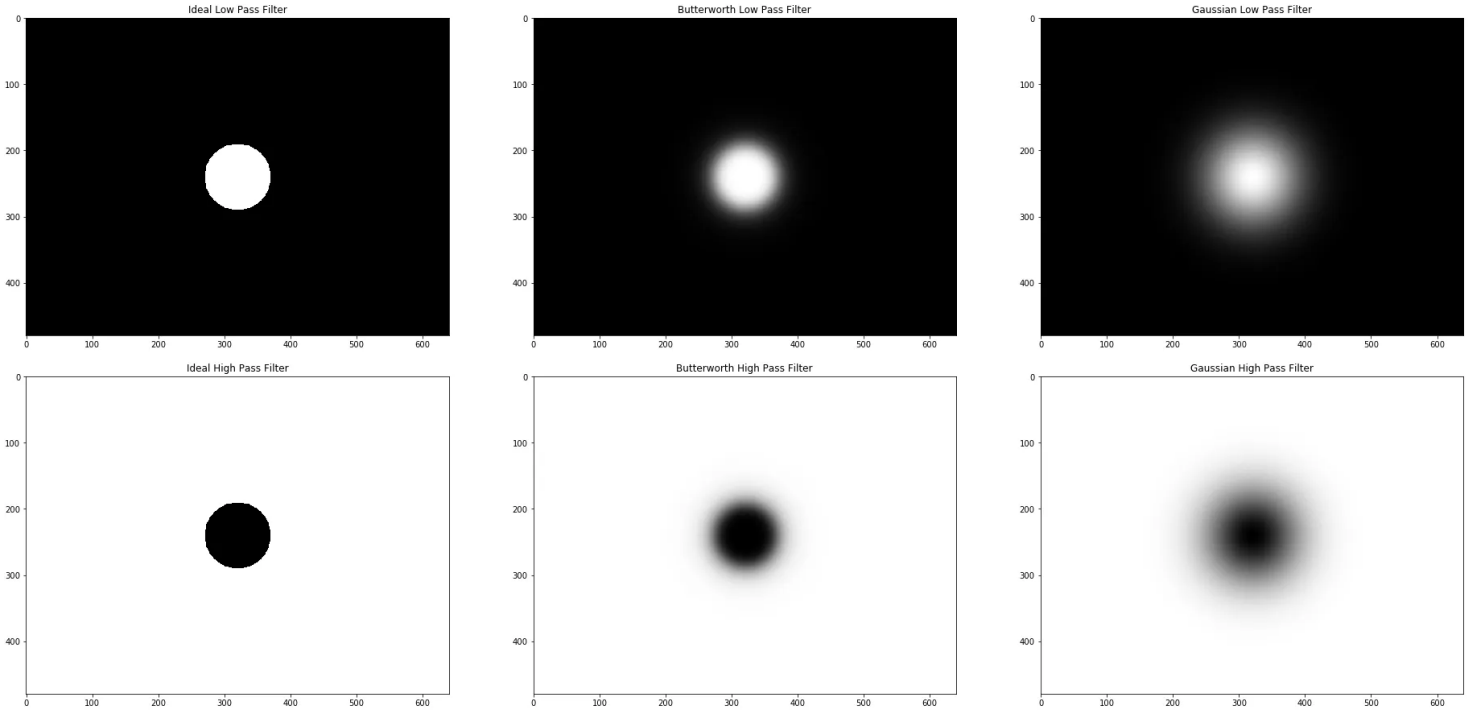


Figure (k): (from left to right) Ideal filter, Butterworth filter (n=10) and Gaussian Filter with D_0**=50**

I put all different filters in ***Figure (k)*** to have a summary of what we have in filters design. From left to right, the circle becomes blurry on its edge which will lead to different impact on output results.

> *Butterworth filter basically is a filter between ideal filter and Gaussian filter.*

Low Pass Filter Results

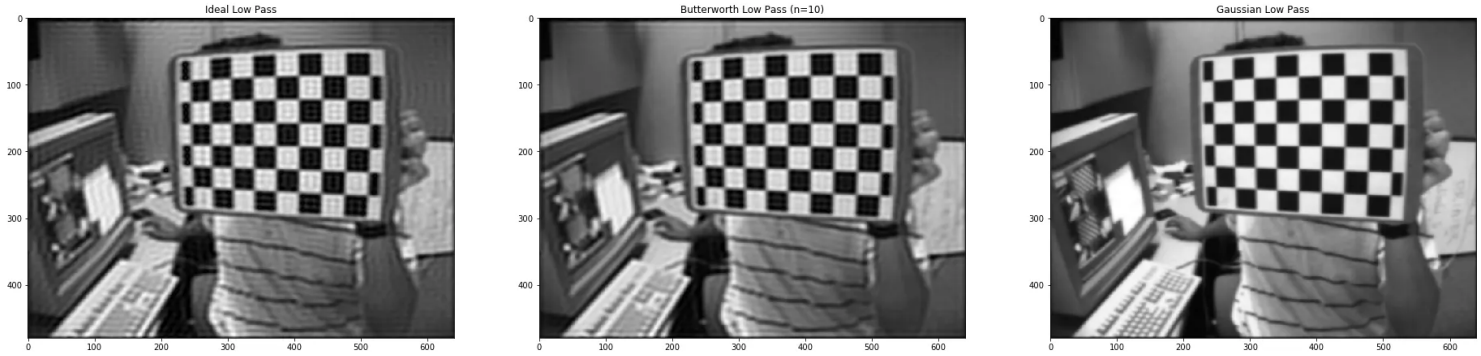


Figure (l): (from left to right) (1) Output using ideal filter (2) Output using Butterworth Filter (3) Output using Gaussian filter with D_0**=50**

Figure (l) shows that all three filters are low pass filter because the output image preserves overall image information. Moreover, we could easily notice that the Gaussian filter performed better than other two filters due to low distortion. The reason why the ideal filter has a lot of waves noise is that the design of ideal filter blocks **ALL** information that is outside of certain radius from origin point. Therefore, some information will be discontinued sharply without any smooth out. On the contrary, Butterworth and Gaussian filter are smoothly blocking information that is outside of certain radius from origin point which makes image more smoothly with less distortion.

High Pass Filter Results

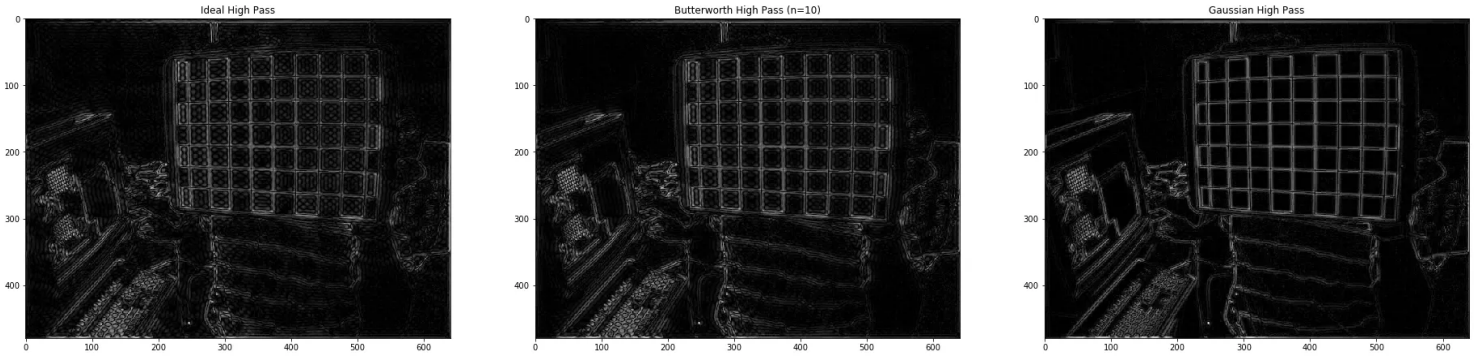


Figure (m): (from left to right) (1) Output using ideal filter (2) Output using Butterworth Filter (3) Output using Gaussian filter with D_0**=50**

Filters in *Figure (m)*, without doubt, are high pass filter because the output results only captured edges. The differences in high pass results between filters are similar to low pass filter results. There are a lot of distortions in an ideal filter result when compares to a Butterworth filter and a Gaussian filter.

Full Code

The End

Fourier Transformation is a very powerful tool for us to manipulate 2-dimension information. FT allows us to process image in another dimension which brings more flexibility. In this article, I go through some basic procedures using Fourier Transformation to process image. Hope you enjoy it.

Thanks for reading it. Happy coding!

Python Opencv Fourier Transform Filters Image Processing

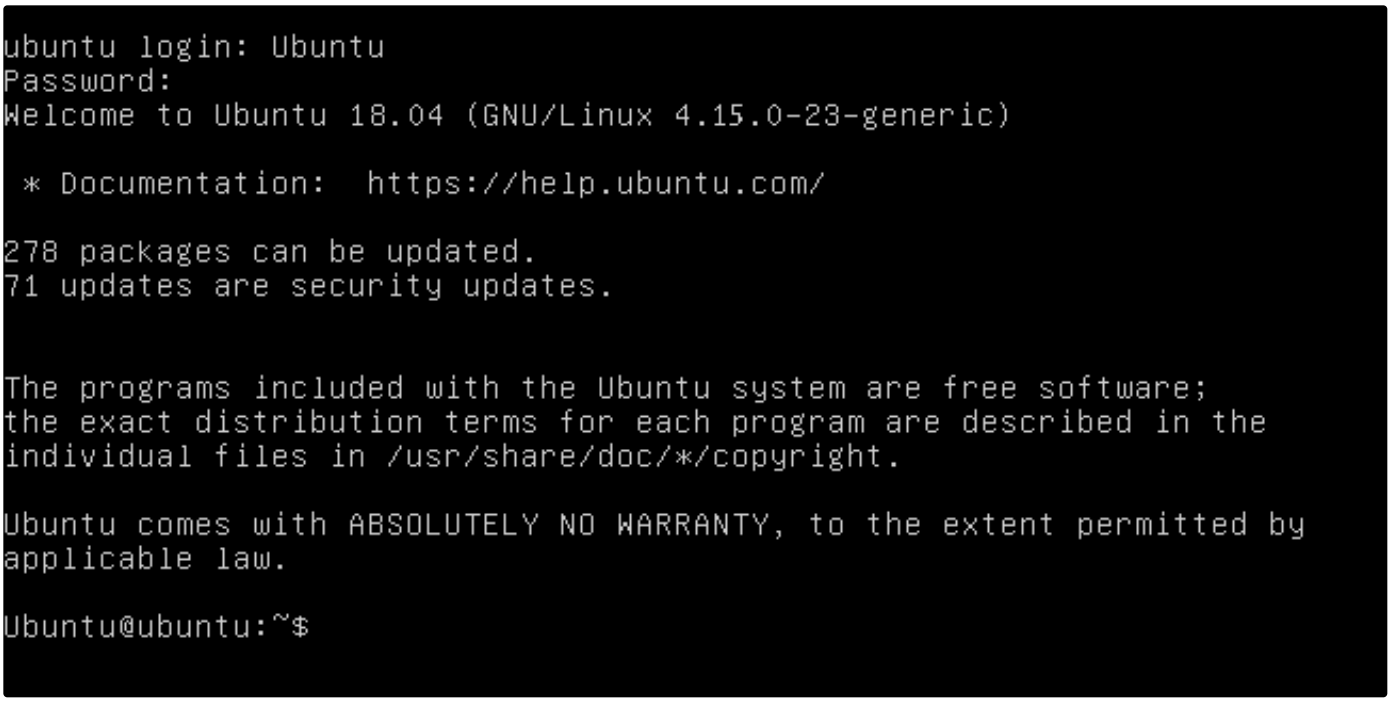
Follow

Written by Craig Chen

114 Followers

Quantitative Analyst / Data Scientist

More from Craig Chen



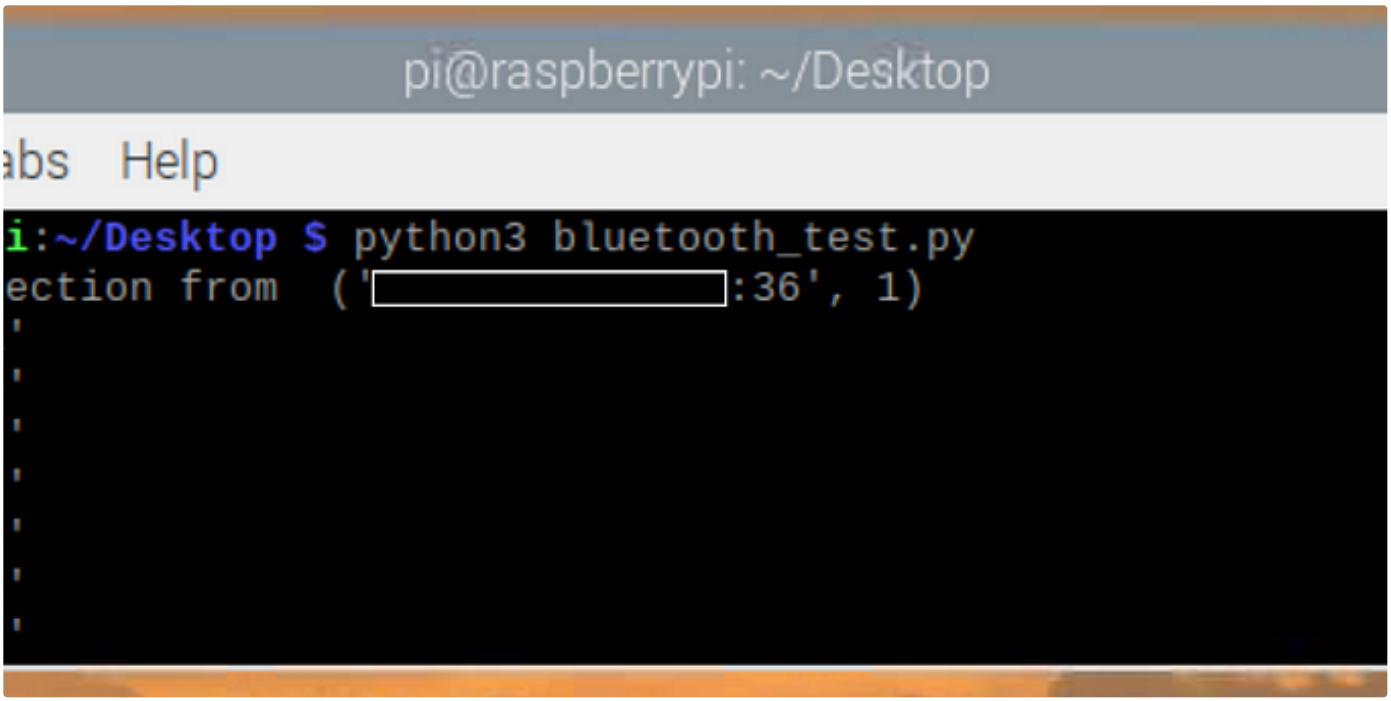
 Craig Chen

How to get my Ubuntu 18.0.4 GUI back from TTY?

A month ago, I installed Ubuntu for my computer for the first time. As a novice in Ubuntu, I encountered some issues about managing it...

· 2 min read · Aug 11, 2018

222 2



Craig Chen

Access Raspberry Pi terminal via Bluetooth on Android Device

It's really common and easy to use SSH to access Raspberry Pi through Internet like WiFi or Ethernet. However, if you don't have internet...

· 3 min read · Dec 25, 2019

57 2

Craig Chen

How to install third-party command line software on AWS Lambda function using Python Chalice

AWS Lambda is a powerful FaaS (Function as a service) service and is flexible to deal with most of tasks. It comes with many useful...

· 4 min read · Sep 20, 2019

Craig Chen

Convert PDF to image on AWS Lambda using Python

My experience about how to convert PDF file to image and how to deal with pdf.la file not found on AWS Lambda function

· 2 min read · Oct 8, 2019

8 3

See all from Craig Chen

Recommended from Medium

Rashida Nasrin Sucky in Towards Data Science

Morphological Operations for Image Preprocessing in OpenCV, in Detail

Erosion, dilation, opening, closing, morphological gradient, tophat / whitehat, and blackhat explained with examples

· 8 min read · Dec 27, 2022

Evgenii Munin in Better Programming

Camera Calibration on a Chessboard With Python and OpenCV

Estimate the intrinsic camera matrix and evaluate its precision

· 4 min read · Feb 22

Amit Chauhan

OpenCV: Adaptive and Otsu Threshold in Image Processing with Python

Image pre-processing techniques in artificial intelligence

✨ · 3 min read · Mar 20

12

Sunny Kumar

Image Preprocessing Using opencv

It is impossible to imagine an image based ML model, be it classification, detection or segmentation without image manipulation. Opencv...

5 min read · May 23

97

Dmitrii Eliuseev in Dev Genius

5 Open-Source Deep Learning Tools for Imaging Super-Resolution

Almost everybody has photos or videos made 10–20 years ago. These pictures were made in a time when 1600×1200 or even 640×480 image...

· 10 min read · Oct 14, 2022

67 4