นาย ศุภากร ยุคลธง 550610548

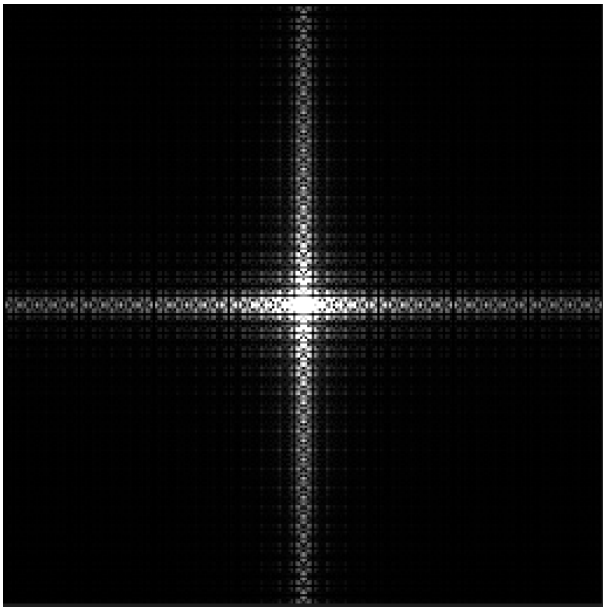
Digital Image Processing (261453)

Computer Assignment 2

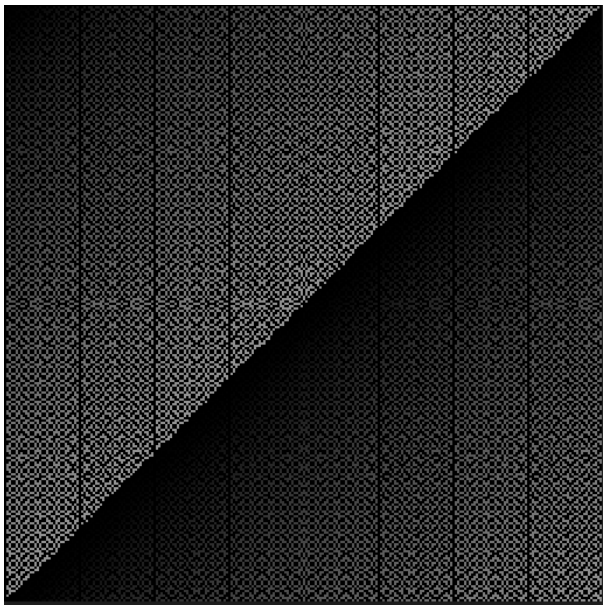
1. Properties of the Fourier Transform

1.1) ทำการแปลง Fourier Transform ของภาพ “Cross.pgm” (200×200) เนื่องจากในการใช้ FFT จำเป็นต้องให้ภาพมีขนาดที่อยู่ในรูปของ 2^n ดังนั้น อาจจะต้องทำการ Pad ก่อน และให้แสดงภาพผลลัพธ์ในรูปของ amplitude และ phase spectra

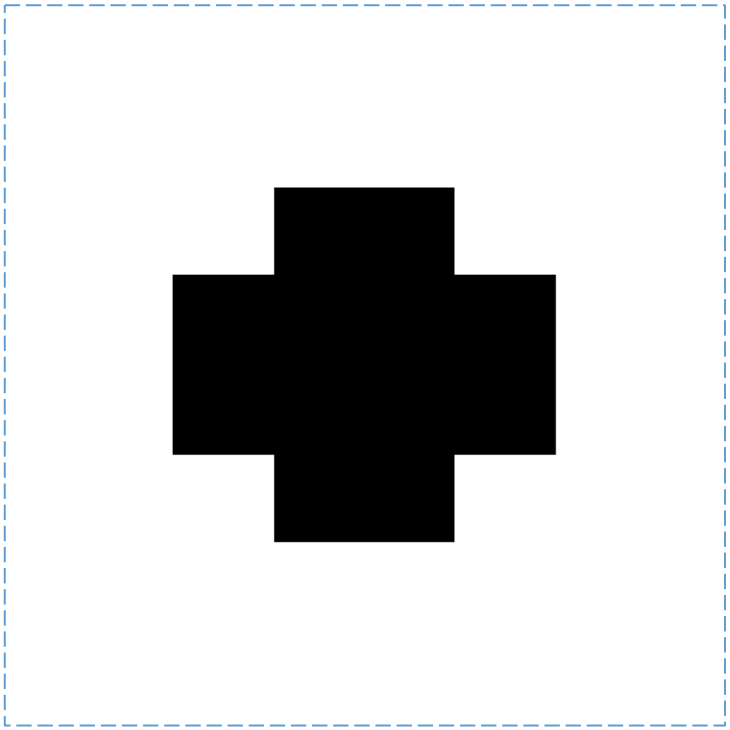
Amplitude

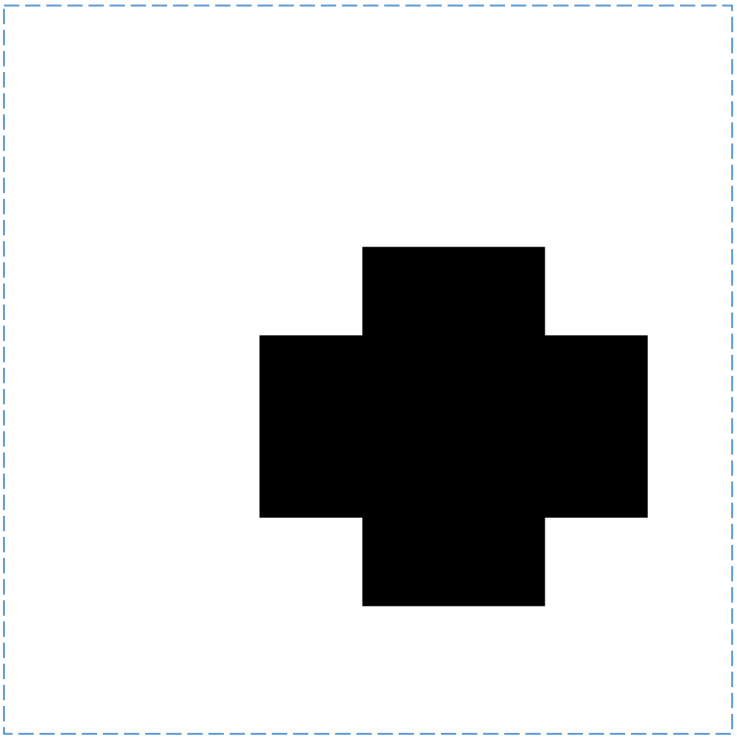


Phase spectra



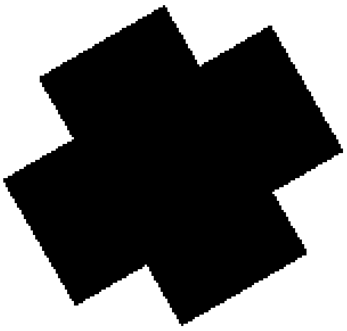
1.2) ให้ทำการคูณ phase spectrum ที่ได้ในข้อ 1.1 ด้วย complex number ค่าหนึ่ง เพื่อที่ว่าเมื่อทำการ inverse Fourier transform แล้ว ภาพผลลัพธ์ที่ได้ ย้ายด้วยจำนวนในแกน *x* และ *y* เป็น (20,30)



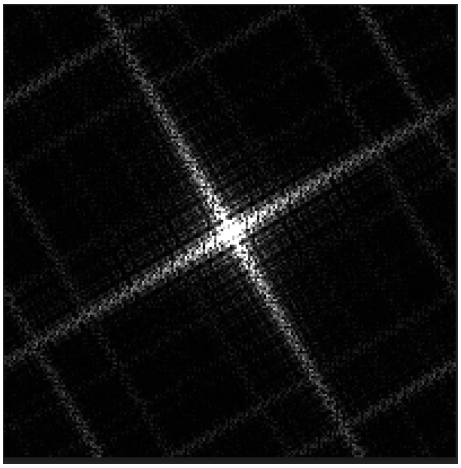


1.3) ทำการหมุนภาพ “Cross.pgm” ไป 30 องศา และแสดงผลของการแปลงฟูเรียร์ ให้ทำการวิเคราะห์ว่าเกิดอะไรขึ้น

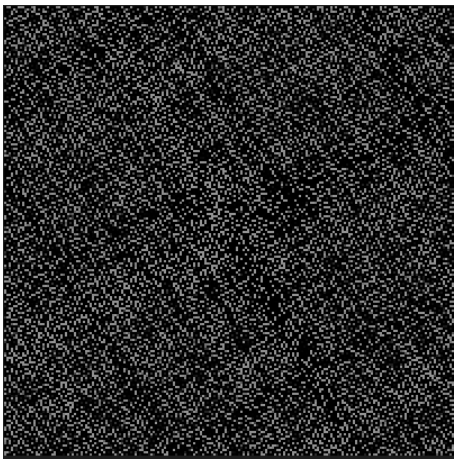
ภาพที่ได้จากการหมุน



Amplitude



Phase spectra

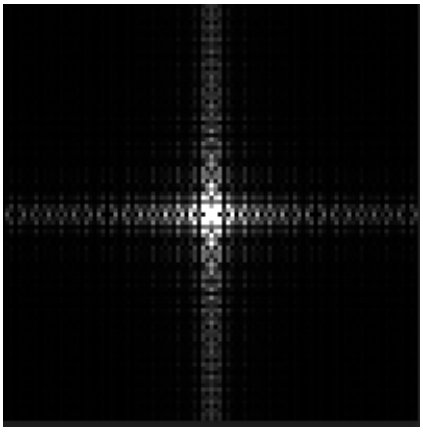


จากการหมุน จะสังเกตเห็นได้ว่าจะทำให้ Amplitude และ Phase ของรูปเปลี่ยนไป โดยที่ Amplitude นั้นหมุนไปตามแนวที่รูปเกิดการหมุนเป็นฯไปตามคุณสมบัติของ การหมุน

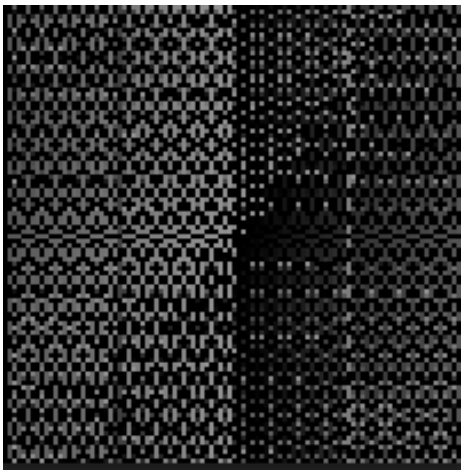
1.4) ทำการ Down-sample “Cross.pgm” เพื่อให้รูปมีขนาด 100×100 หลังจากนั้นทำการแปลง Fourier Transform ให้แสดงภาพผลลัพธ์ในรูปของ amplitude และ phase spectra ให้ทำการวิเคราะห์ว่าเกิดอะไรขึ้น

ภาพที่ได้จากการหมุน

Amplitude



Phase spectra



จะเห็นได้ว่าทั้ง Amplitude และ Spectrum ไม่มีการเปลี่ยนแปลงเลย เพราะ การ scaling ภาพ ไม่มีผลต่อการเปลี่ยนแปลง

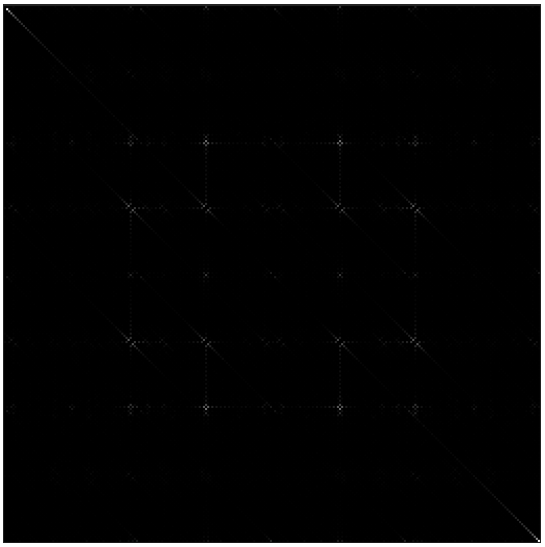
1.5) ใช้การแปลง inverse Fourier Transform ของผลลัพธ์ที่ได้ในข้อ 1.1 โดยที่

1.5.1) ไม่ใช้ข้อมูล phase

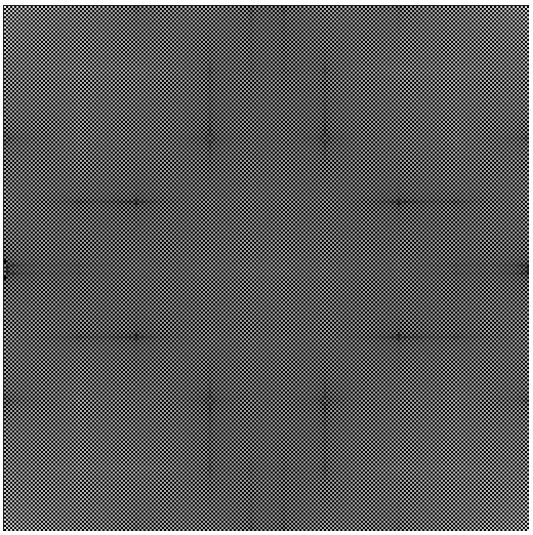
1.5.2) ไม่ใช้ข้อมูล amplitude

ให้ทำการวิเคราะห์ว่าเกิดอะไรขึ้น

ใช้แต่ Phase



ใช้แต่ Amplitude

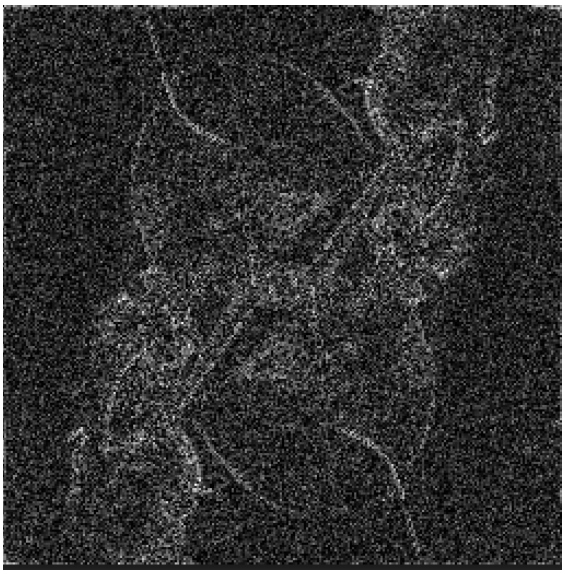


การที่ใช้แต่ข้อมูล Amplitude จะเห็นแต่ข้อมูลของความเข้มแสง โดยมีแค่ DC Component เท่านั้นที่เด่นชัด แต่ถ้าใช้ข้อมูล Phase จะทำให้สามารถเห็นขอบของวัตถุในภาพ ทำให้เราจึงทราบว่า ข้อมูล Phase มีข้อมูลของขอบวัตถุประกอบอยู่ด้วย

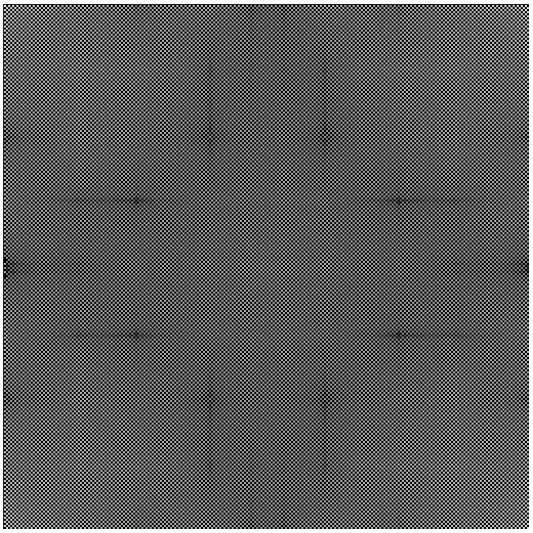
1.6) ให้ทำการทดลองเช่นเดียวกับข้อ 1.5 ด้วยภาพ “Lenna.pgm” (256×256)

ภาพ Lenna.pgm

ใช้แต่ Phase



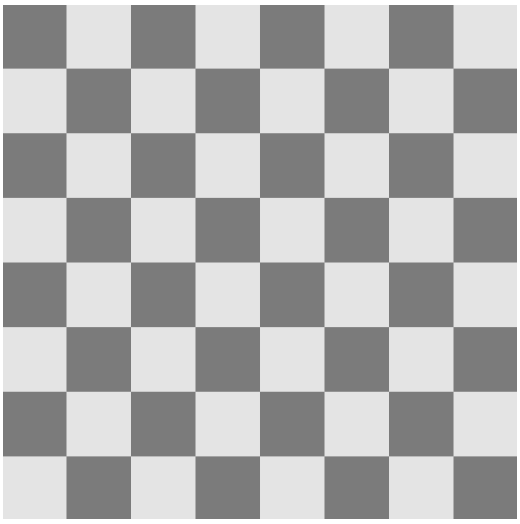
ใช้แต่ Amplitude



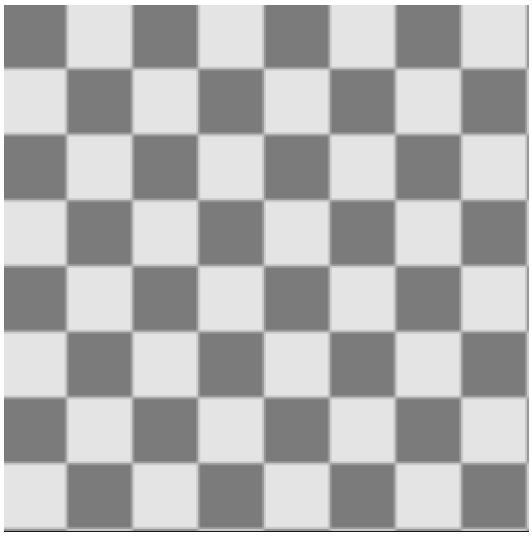
การที่ใช้แต่ข้อมูล Amplitude จะเห็นแต่ข้อมูลของความเข้มแสง โดยมีแค่ DC Component เท่านั้นที่เด่นชัด แต่ถ้าใช้ข้อมูล Phase จะทำให้สามารถเห็นขอบของวัตถุในภาพ ทำให้เราจึงทราบว่า ข้อมูล Phase มีข้อมูลของขอบวัตถุประกอบอยู่ด้วย

1.7) ทำ convolution ภาพ “Chess.pgm” (256×256) ด้วย mask หรือ kernel ขนาดเล็กอันหนึ่ง เพื่อทำการ Blur ภาพ และให้ทำการ filter ใน frequency domain ด้วย Fourier transform ของ Kernel นั้นด้วย เพื่อทำการ Blur ภาพด้วย เปรียบเทียบผลที่ได้ทั้งสองแบบ ว่ามีความแตกต่างหรือหรือเหมือนกันอย่างไรภาพ

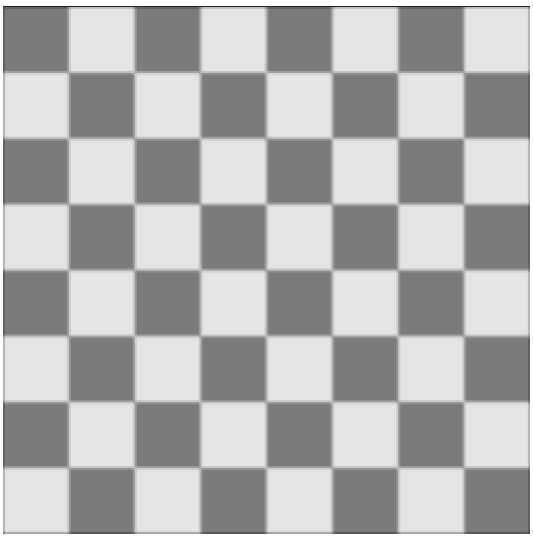
Chess.pgm



Blur in FFT



Blur with Convolution



จากผลการทดลองเห็นว่า รูปที่ได้จากการเบลอด้วยทั้งสองวิธีนั้น ได้ผลลัพธ์ที่เหมือนกัน

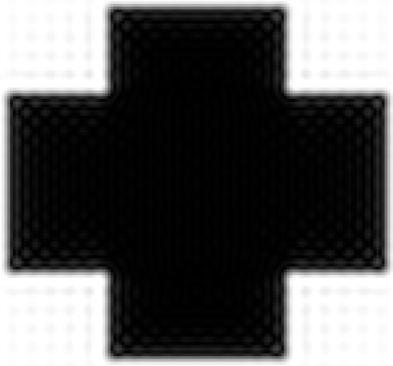
2. Filter Design

2.1) ให้ใช้ ideal low-pass filter กับภาพ “Cross.pgm” โดยที่เปลี่ยน cutoff frequency และให้ศึกษา ringing effect ที่เกิดขึ้น หลังจากนั้นให้ทำการทดลองซ้ำด้วย Non-ideal filter อื่น

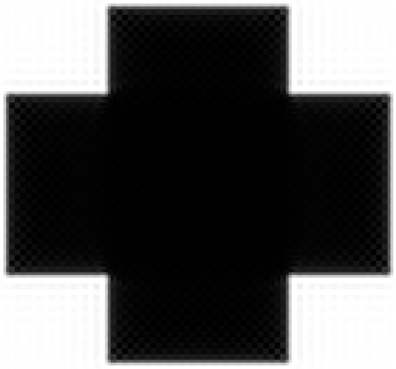
Ideal low-pass filter

Original

Ideal low pass filter with Cut off 20



Ideal low pass filter with Cut off 30



Ideal low pass filter with Cut off 50

จากการทดลองพบว่า เมื่อรูปต้นฉบับได้ถูกนำไปผ่าน Ideal low pass filter ที่ความถี่ Cut off ต่ำๆ ปรากฏว่าภาพที่ได้จะเบลอขึ้น ทำให้ขอบของวัตถุไม่ชัดเจน และยังทำให้เกิด Ringing Effect อีกด้วย แต่เมื่อนำไปผ่านความถี่ Cut off โดยเพิ่มค่าขึ้นเรื่อยปรากฏว่าภาพที่ได้นั้น ค่อยๆ ชัดขึ้นตามการเพิ่มค่าเลขความถี่ Cut off

Butterworth low-pass filter ที่ Order(n) = 2

Original

Cut off 20

Cut off 30

Cut off 50

จะพบว่าเมื่อใช้ Butter worth low-pass filter ภาพที่ได้นั้นคือ ภาพที่มีความเบลอ และไม่มีการเกิด Ringing Effect ถ้า N มีค่าเหมาะสม เมื่อมีการเพิ่ม Cut off ขึ้นเรื่อยๆ รูปที่ได้ก็จะมีความคล้ายคลึงกับต้นฉบับมากขึ้นเมื่อเพิ่มค่านี้เช่นกัน

Butterworth low-pass filter ที่เปลี่ยนค่า N แต่ cut off คงที่

Original

N = 2

N = 20

N = 50

จะเห็นว่า Butter worth low-pass filter ก็ทำให้เกิด Ringing Effect ได้ถ้าเราเพิ่มค่า N ขึ้นไปเรื่อย ดังนั้น เราควรจะเลือกค่า N ที่เหมาะสม ส่วนใหญ่นิยมใช้ N = 2

Gaussian Low-pass filter

Original

Cut off 20

Cut off 30

Cut off 50

จากการนำรูปภาพไปผ่าน Gaussian low-pass Filter แล้วพบว่าภาพที่ได้มีความเบลอเกิดขึ้นบนภาพ แต่ถ้าเพิ่มค่าความถี่ Cut off ขึ้นอีก เราจะพบว่ารูปภาพมีความใกล้เคียงกับรูปต้นฉบับไปเรื่อยๆตามค่าที่เพิ่มขึ้น และก็ยังพบอีกว่า ไม่มีการเกิด Ringing Effect ขึ้นมาเลย

2.2) ให้ใช้ low-pass filter หลายแบบ โดยที่เปลี่ยนค่าตัวแปรต่างๆ รวมถึงขนาดของ filter ด้วย กับภาพ “Chess_noise.pgm” และ “Lenna_noise.pgm” เพื่อลด noise และให้เปรียบเทียบผลกับ Median filter ที่มีการเปลี่ยนขนาด และให้ใช้ RMS ในการคำนวณหาความแตกต่างระหว่างผลลัพธ์ที่ได้กับภาพที่ไม่มี Noise “Chess.pgm” and “Lenna.pgm”

ในการคำนวณ RMS ใช้สมการการคำนวณดังนี้

$$RMS = \sqrt{\frac{\sum_{i=0}^{n}\sum_{j=0}^{m}\left(f_1(i,j) - f_2(i,j)\right)^2}{m * n}}$$

โดยที่

n และ m คือขนาดของด้านในแนวแกน x และแกน y ตามลำดับ

f_1 และ f_2 คือ ค่า Grey Scale ในตำแหน่งนั้นๆของรูปที่ 1 กับ รูปที่ 2 โดยที่ i และ j เป็นตำแหน่งของ pixel

ภาพ Lenna.pgm

ภาพ Lenna_noise.pgm

ภาพ lenna_noise ที่ผ่าน Ideal low-pass filter

Cut off 20

Cut off 30

Cut off 50

ภาพที่ 1	เทียบกับภาพที่ 2	RMS
Lenna.pgm	Lenna.pgm	0.0
Lenna.pgm	Lenna_noise.pgm	25.2807822355
Lenna.pgm	Ideal low pass with cut off 20	17.6081566376
Lenna.pgm	Ideal low pass with cut off 30	14.1999086404
Lenna.pgm	Ideal low pass with cut off 50	12.6860422885

ภาพ lenna_noise ที่ผ่าน Butter worth low-pass filter ที่ n = 2

Cut off 20

Cut off 30

Cut off 50

ภาพที่ 1	เทียบกับภาพที่ 2	RMS
Lenna.pgm	Lenna.pgm	0.0
Lenna.pgm	Lenna_noise.pgm	25.2807822355
Lenna.pgm	butterworth with cut off 20	15.5516716847
Lenna.pgm	butterworth with cut off 30	12.6251371773
Lenna.pgm	butterworth with cut off 50	11.084632587

ภาพ lenna_noise ที่ผ่าน Gaussian low-pass filter

Cut off 20

Cut off 30

Cut off 50

ภาพที่ 1	เทียบกับภาพที่ 2	RMS
Lenna.pgm	Lenna.pgm	0.0
Lenna.pgm	Lenna_noise.pgm	25.2807822355
Lenna.pgm	Gaussian with cut off 20	14.4654091589
Lenna.pgm	Gaussian with cut off 30	11.933575849
Lenna.pgm	Gaussian with cut off 50	11.2171429091

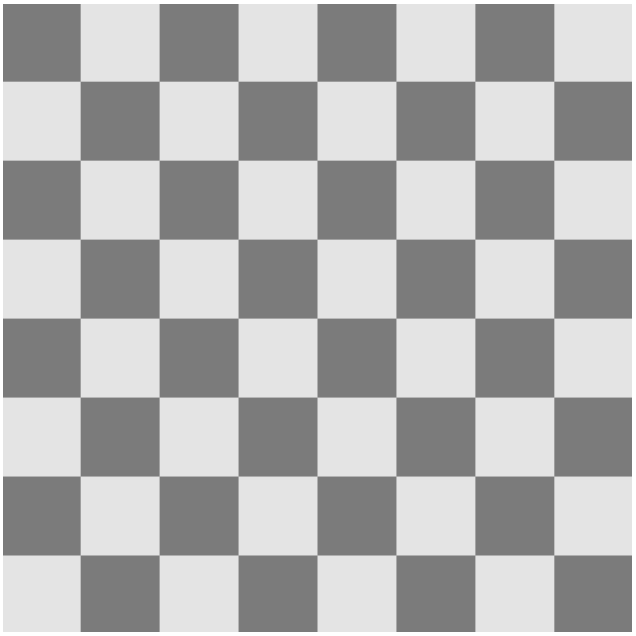
ภาพ lenna_noise ที่ผ่าน Median filter

Median Filter

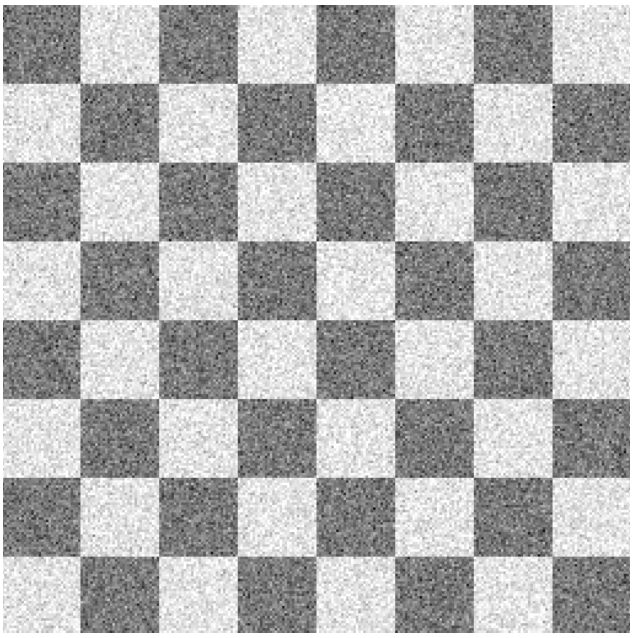
ภาพที่ 1	เทียบกับภาพที่ 2	RMS
Lenna.pgm	Lenna.pgm	0.0
Lenna.pgm	Lenna_noise.pgm	25.2807822355
Lenna.pgm	Median Filter	13.5181113365

จากการทดลองเราได้ภาพที่ผ่าน Filter ต่างๆ และค่า RMS ซึ่งค่า RMS อาจจะสามารถบอกได้ว่ารูปภาพที่มีค่าน้อยกว่าค่า RMS ของ Lenna_noise แสดงถึงตัว Filter ที่ใช้มีความสามารถในการลด noise ได้ และยิ่งถ้าความต่างของรูป (RMS) ที่ผ่าน Filter นั้นๆมีค่าเข้าใกล้ 0 ก็แสดงว่ามีความสามารถในการลด Noise ได้มาก ในการทดลองนี้พบว่า Gaussian with cutoff 50 ดีที่สุด(การทดลองนี้ไม่ได้ครอบคลุมทุก Cutoff อาจจะมีค่าที่ดีกว่านี้)

ภาพ Chess.pgm

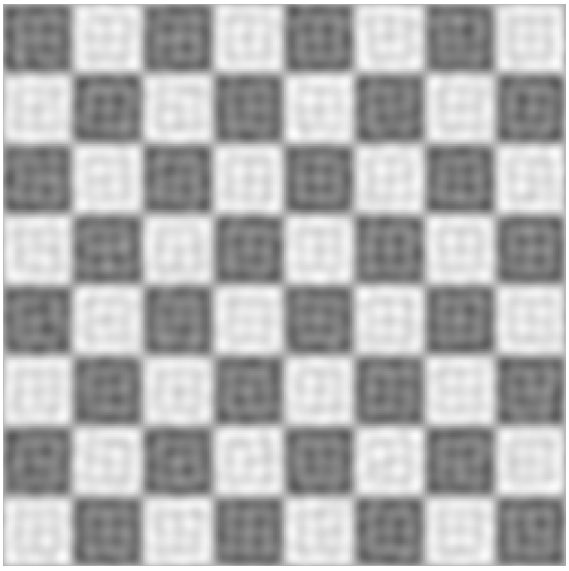


ภาพ Chess_noise.pgm

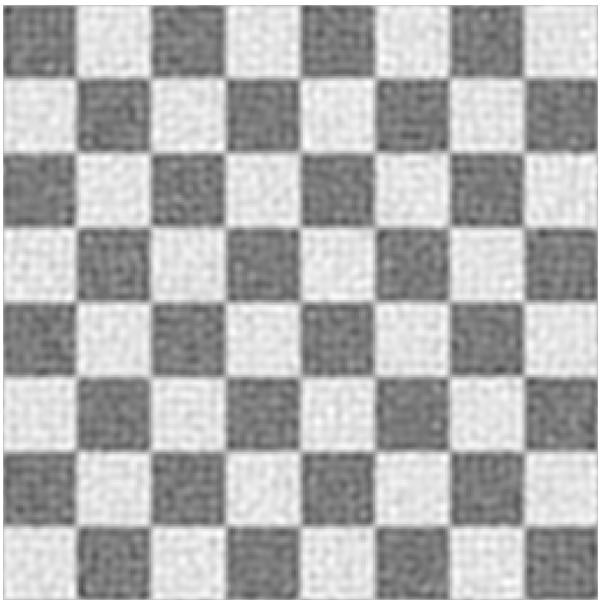


ภาพ Chess_noise ที่ผ่าน Ideal low-pass filter

Cut off 20



Cut off 30



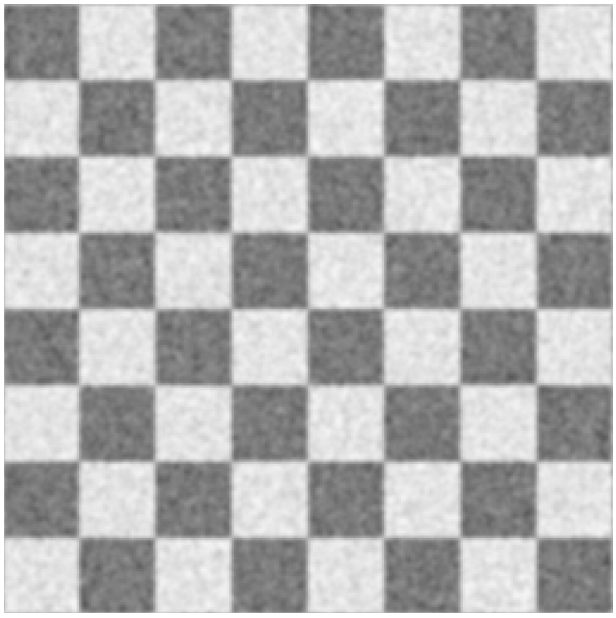
Cut off 50

ภาพที่ 1	เทียบกับภาพที่ 2	RMS
Chess.pgm	Chess.pgm	0.0
Chess.pgm	Chess_noise.pgm	24.5852060758
Chess.pgm	Ideal low pass with cut off 20	26.131443233
Chess.pgm	Ideal low pass with cut off 30	19.446980314
Chess.pgm	Ideal low pass with cut off 50	16.8509834342

ภาพ Chess_noise ที่ผ่าน Butter worth low-pass filter

Cut off 20

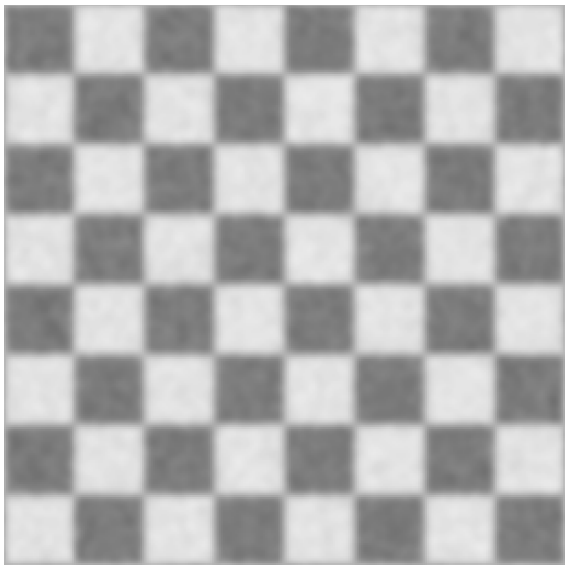
Cut off 30



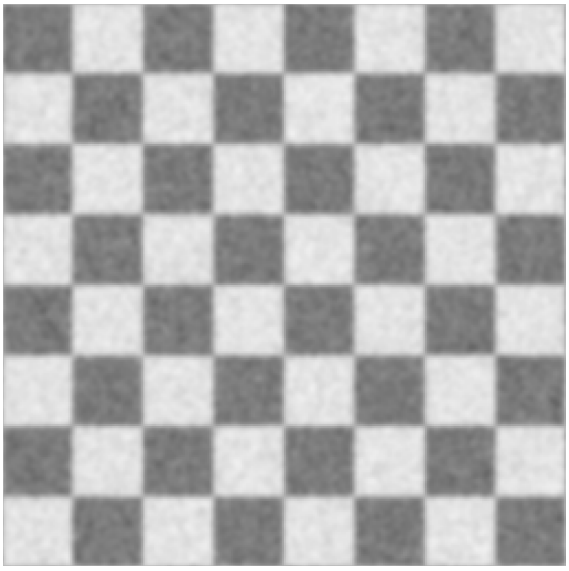
Cut off 50

ภาพที่ 1	เทียบกับภาพที่ 2	RMS
Chess.pgm	Chess.pgm	0.0
Chess.pgm	Chess_noise.pgm	24.5852060758
Chess.pgm	Butterworth with cut off 20	22.0184176253
Chess.pgm	Butterworth with cut off 30	18.0382847013
Chess.pgm	Butterworth with cut off 50	14.6207029111

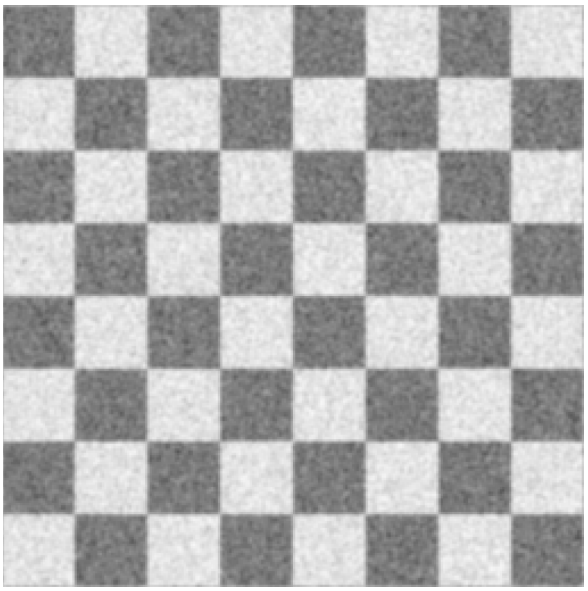
ภาพ Chess_noise ที่ผ่าน Gaussian low-pass filter



Cut off 20



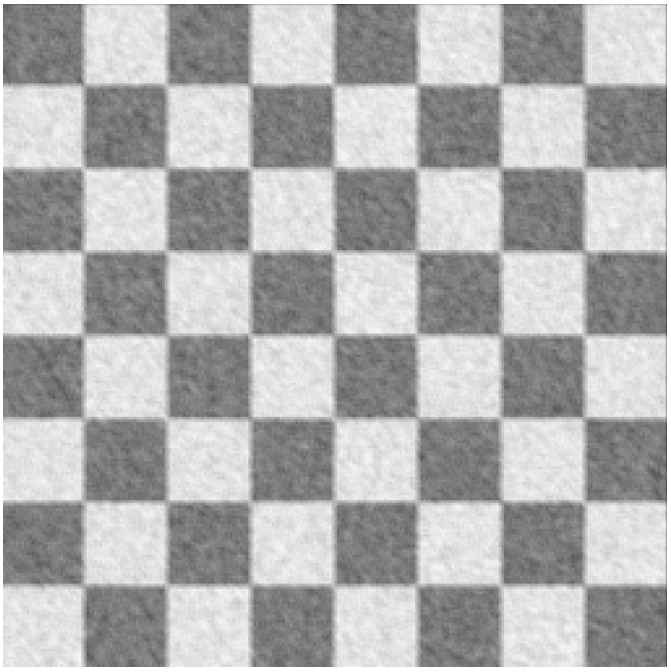
Cut off 30



Cut off 50

ภาพที่ 1	เทียบกับภาพที่ 2	RMS
Chess.pgm	Chess.pgm	0.0
Chess.pgm	Chess_noise.pgm	24.5852060758
Chess.pgm	Gaussian with cut off 20	20.7212630599
Chess.pgm	Gaussian with cut off 30	17.0200391356
Chess.pgm	Gaussian with cut off 50	14.0904350358

ภาพ Chess_noise ที่ผ่าน Median filter



Median Filter

ภาพที่ 1	เทียบกับภาพที่ 2	RMS
Chess.pgm	Chess.pgm	0.0
Chess.pgm	Chess_noise.pgm	25.2807822355
Chess.pgm	Median Filter	17.9855237606

จากการทดลองเราได้ภาพที่ผ่าน Filter ต่างๆ และค่า RMS ซึ่งค่า RMS อาจจะสามารถบอกได้ว่ารูปภาพที่มีค่าน้อยกว่าค่า RMS ของ Chess_noise แสดงถึงตัว Filter ที่ใช้มีความสามารถในการลด noise ได้ และยิ่งถ้าความต่างของรูป (RMS) ที่ผ่าน Filter นั้นๆมีค่าเข้าใกล้ 0 ก็แสดงว่ามีความสามารถในการลด Noise ได้มาก ในการทดลองนี้พบว่า Gaussian with cutoff 50 ดีที่สุด(การทดลองนี้ไม่ได้ครอบคลุมทุก Cutoff อาจจะมีค่าที่ดีกว่านี้)

และเรายังพบอีกว่าในส่วนของ Ideal low pass ที่ cut off มีค่า 20 นั้นจะทำให้เกิด Ringing Effect อย่างเห็นได้ชัดและเมื่อสังเกตค่า RMS ของมัน ก็พบว่ามีค่ามากกว่า RMS ของ Chess_noise.pgm เสียอีก

Link โค้ด → https://github.com/SupakornYu/ImageProcessingHomework-HW2/blob/master/img_main2.py

ใช้ภาษา Python ในการพัฒนา

Library ที่เขียนไว้มีดังนี้

```python
import numpy as np
import matplotlib.pyplot as plt
import math
import cmath

def padwithzeros(vector, pad_width, iaxis, kwargs):
    vector[:pad_width[0]] = 255
    vector[-pad_width[1]:] = 255
    return vector

def padwithzerooos(vector, pad_width, iaxis, kwargs):
    vector[:pad_width[0]] = 0
    vector[-pad_width[1]:] = 0
    return vector

class ImageLibFourier:
    def readPGMImage(self,path):
        file = open(path, "rb")
        pgmVer = file.readline().split()
        pgmComment = []
        while True:
            pgmComment_eachline = file.readline()
            if(pgmComment_eachline[0]==35):
                pgmComment.append(pgmComment_eachline)
            else:
                break
        pgmSize = pgmComment_eachline.split()
        pgmGreyscale = file.readline().split()
        pgmDataList = []
        htg = np.zeros((256),dtype=np.int32)
        np.set_printoptions(suppress=True)
        for j in range(int(pgmSize[1])):
            pgmDataX = []
            for i in range(int(pgmSize[0])):
                byte = file.read(1)
                chrToInt = ord(byte)
                pgmDataX.append(chrToInt)
                htg[chrToInt] = htg[chrToInt]+1
            pgmDataList.append(pgmDataX)
        file.close()
        pgmData = np.asarray(pgmDataList,dtype=np.int32)
        return pgmVer,pgmComment,pgmSize,pgmGreyscale,pgmData,htg

    def buildPGMFile(self,fileName,width,height,greyLevel,pgmData):
        f = open(str(fileName)+".pgm","wb")
        f.write("P5\n");
        f.write("# "+str(fileName)+"\n");
        f.write(str(width)+" "+str(height)+"\n"+str(greyLevel[0])+"\n");
        for i in range(int(height)):
            for j in range(int(width)):
                if pgmData[i][j]<0:
                    pgmData[i][j] = 0
                elif pgmData[i][j]>int(greyLevel[0]):
                    pgmData[i][j] = int(greyLevel[0])
                f.write(chr(pgmData[i][j]));
        f.close()



    def padImage(self,path,padWidthEachSide):
        pgmVer,pgmComment,pgmSize,pgmGreyscale,pgmData,htg = myLib.readPGMImage(path)
        pgmData = np.lib.pad(pgmData, padWidthEachSide, padwithzeros)
        pgmSize[0] = int(pgmSize[0]) + (int(padWidthEachSide)*2) #x axis
        pgmSize[1] = int(pgmSize[1]) + (int(padWidthEachSide)*2)
```

```
        path = path.replace(".pgm", "");
        self.buildPGMFile(path+"Padding",pgmSize[0],pgmSize[1],pgmGreyscale,pgmData)

    def convertToFourier(self,pgmData):
        return np.fft.fft2(pgmData)

    def scalepgmData(self,pgmData):
        pgmData = np.log(pgmData)
        return pgmData

    def moveAxispgmDataBeforeFourier(self,pgmData,pgmSize):
        for j in range(int(pgmSize[1])):
            for i in range(int(pgmSize[0])):
                x = math.pow(-1.0,float(i+j))
                pgmData[i][j] = float(pgmData[i][j])*x
        return pgmData

    def pgmDataToPhasePicWithScale(self,filename,pgmDataFourier,pgmSize,pgmGreyscale):
        #pgmDataFourier = np.log(np.angle(pgmDataFourier))
        pgmDataFourier = np.angle(pgmDataFourier)
        #pgmDataFourier = 1+np.log2(1+pgmDataFourier)
        pgmDataFourier = np.round(pgmDataFourier*(float(pgmGreyscale[0])/(np.amax(pgmDataFourier)-np.amin(pgmDataFourier))),0).astype(int)
        #pgmDataFourierPhase = np.around(np.angle(pgmDataFourier)*(16),0).astype(int)
        self.buildPGMFile(filename+"phase",pgmSize[0],pgmSize[1],pgmGreyscale,pgmDataFourier)

    def pgmDataToAmplitudePicWithScale(self,filename,pgmDataFourier,pgmSize,pgmGreyscale):
        pgmDataFourier = np.round(np.abs(pgmDataFourier/float(pgmGreyscale[0])),0).astype(int)
        #pgmDataFourier = np.around(1+np.log2(1+np.abs(pgmDataFourier)),0).astype(int)
        self.buildPGMFile(filename+"Amplitude",pgmSize[0],pgmSize[1],pgmGreyscale,pgmDataFourier)
        #np.set_printoptions(threshold=np.nan)
```

```
    def shiftAxisInFourier(self,filename,a,b):
        pgmVer,pgmComment,pgmSize,pgmGreyscale,pgmData,htg = self.readPGMImage(filename)
        #pgmData = myLib.moveAxispgmDataBeforeFourier(pgmData,pgmSize)
        pgmDataFourier = self.convertToFourier(pgmData)
        #print pgmDataFourier
        for j in range(int(pgmSize[1])):
            for i in range(int(pgmSize[0])):
                #realSim = np.cos(-2*np.pi*(((a*i)/int(pgmSize[0]))+((b*j)/int(pgmSize[1]))))
                #complexSim = np.sin(-2*np.pi*(((a*i)/int(pgmSize[0]))+((b*j)/int(pgmSize[1]))))*1j
                #pgmDataFourier[i][j] = (realSim+complexSim)* pgmDataFourier[i][j]
                pgmDataFourier[i][j] = pgmDataFourier[i][j] * cmath.exp(-2j*np.pi*((a*i/float(pgmSize[0]))+(b*j/float(pgmSize[1]))))
        pgmDataInverse = np.round(np.abs(np.fft.ifft2(pgmDataFourier)),0).astype(int)
        #print pgmDataInverse
        filename = filename.replace(".pgm", "");
        self.buildPGMFile(filename+"ShiftAxis",pgmSize[0],pgmSize[1],pgmGreyscale,pgmDataInverse)

    def rotatePic(self,filename,angle):
        angle = np.deg2rad(angle)
        pgmVer,pgmComment,pgmSize,pgmGreyscale,pgmData,htg = self.readPGMImage(filename)
        pgmDataNew  = np.full((int(pgmSize[0]), int(pgmSize[0])), 255, dtype=np.int)

        for j in range(int(pgmSize[1])):
            for i in range(int(pgmSize[0])):
                xAxis = (i-100)*np.cos(angle)-(j-100)*np.sin(angle)
                yAxis = (i-100)*np.sin(angle)+(j-100)*np.cos(angle)
                xAxis = xAxis + 100
                yAxis = yAxis + 100
                #print xAxis
                #print yAxis
                if xAxis >= 200 or yAxis >= 200 or xAxis < 0 or yAxis < 0:
                    pgmDataNew[i][j] = 255
                else:
```

```
                    pgmDataNew[i][j] = pgmData[xAxis][yAxis]
        filename = filename.replace(".pgm", "");
        self.buildPGMFile(filename+"RotatePic",pgmSize[0],pgmSize[1],pgmGreyscale,pgmDataNew)
        #print pgmDataNew

    def downSample(self,filename,ratio):
        pgmVer,pgmComment,pgmSize,pgmGreyscale,pgmData,htg = self.readPGMImage(filename)
        newPgmSize = int(int(pgmSize[0])*ratio)
        pgmDataNew  = np.full((newPgmSize, newPgmSize), 255, dtype=np.int)
        for j in range(newPgmSize):
            for i in range(newPgmSize):
                pgmDataNew[i][j] = pgmData[i*(1/ratio)][j*(1/ratio)]
        filename = filename.replace(".pgm", "");
        self.buildPGMFile(filename+"DownSamplePic",newPgmSize,newPgmSize,pgmGreyscale,pgmDataNew)

    def inverseFourierPgmWithOutAmplitude(self,filename):
        pgmVer,pgmComment,pgmSize,pgmGreyscale,pgmData,htg = myLib.readPGMImage(filename)
        pgmData = self.moveAxispgmDataBeforeFourier(pgmData,pgmSize)
        #print pgmData
        pgmDataFourier = self.convertToFourier(pgmData)
        pgmDataFourier = np.angle(pgmDataFourier)

        #pgmDataFourier = np.around(np.abs(np.fft.ifft2(pgmDataFourier)),0).astype(float)
        pgmDataFourier = np.abs(np.fft.ifft2(pgmDataFourier))
        pgmDataFourier = np.round(pgmDataFourier*(float(pgmGreyscale[0])/(np.amax(pgmDataFourier)-np.amin(pgmDataFourier))),0).astype(int)
        #print np.min(pgmDataFourier)
        pgmDataFourier = self.moveAxispgmDataBeforeFourier(pgmDataFourier,pgmSize)
        filename = filename.replace(".pgm", "");
        self.buildPGMFile(filename+"PgmWithOutAmplitude",pgmSize[0],pgmSize[1],pgmGreyscale,pgmDataFourier)

    def inverseFourierPgmWithOutPhase(self,filename):
        pgmVer,pgmComment,pgmSize,pgmGreyscale,pgmData,htg = myLib.readPGMImage(filename)
```

```
        pgmData = self.moveAxispgmDataBeforeFourier(pgmData,pgmSize)
        pgmDataFourier = self.convertToFourier(pgmData)
        pgmDataFourier = np.abs(pgmDataFourier)

        #pgmDataFourier = np.around(np.abs(np.fft.ifft2(pgmDataFourier)),0).astype(float)
        pgmDataFourier = np.log2(np.abs(np.fft.ifft2(pgmDataFourier)))
        pgmDataFourier = np.round(pgmDataFourier*(float(pgmGreyscale[0])/(np.amax(pgmDataFourier)-np.amin(pgmDataFourier))),0).astype(int)
        #pgmDataFourier = np.around(np.abs(pgmDataFourier/float(pgmGreyscale[0])),0)
        #pgmDataFourier = np.around(pgmDataFourier*(float(pgmGreyscale[0])/(np.amax(pgmDataFourier)-np.amin(pgmDataFourier))),0).astype(int)
        print np.min(pgmDataFourier)
        pgmDataFourier = self.moveAxispgmDataBeforeFourier(pgmDataFourier,pgmSize)
        filename = filename.replace(".pgm", "");
        self.buildPGMFile(filename+"PgmWithOutPhase",pgmSize[0],pgmSize[1],pgmGreyscale,pgmDataFourier)

    def convolutionWithKernel(self,inputFileName,kernelName,kernel,extendPadding):
        pgmVer,pgmComment,pgmSize,pgmGreyscale,pgmData,htg = self.readPGMImage(str(inputFileName)+".pgm")
        pgmData = np.lib.pad(pgmData, extendPadding, padwithzerooos)
        pgmDataCon = np.zeros((int(pgmSize[1]),int(pgmSize[0])),dtype=np.float)
        pgmDataCon = np.lib.pad(pgmDataCon, extendPadding, padwithzerooos)
        pgmDataCon.fill(255)
        #print pgmDataCon.shape
        for i in range(1,int(pgmSize[1])+1):
            for j in range(1,int(pgmSize[0])+1):
                temp = 0
                #XYY
                #YYY
                #YYY
                temp += pgmData[i][j]*kernel[1][1]
                temp += pgmData[i-1][j-1]*kernel[0][0]
                temp += pgmData[i-1][j]*kernel[0][1]
                temp += pgmData[i+1][j+1]*kernel[2][2]
                temp += pgmData[i][j-1]*kernel[1][0]
```

```
                temp += pgmData[i][j+1]*kernel[1][2]
                temp += pgmData[i+1][j-1]*kernel[2][0]
                temp += pgmData[i+1][j]*kernel[2][1]
                temp += pgmData[i+1][j+1]*kernel[2][2]
                pgmDataCon[i][j] = temp
        pgmDataCon = np.delete(pgmDataCon, int(pgmSize[1])+1, 0)
        #print pgmDataCon.shape
        pgmDataCon = np.delete(pgmDataCon, 0, 0)
        #print pgmDataCon.shape
        pgmDataCon = np.delete(pgmDataCon, int(pgmSize[0])+1, 1)
        #print pgmDataCon.shape
        pgmDataCon = np.delete(pgmDataCon, 0, 1)
        #print pgmDataCon.shape
        pgmDataCon = np.round(pgmDataCon,0).astype(int)
        self.buildPGMFile(str(inputFileName)+"Con"+str(kernelName),pgmSize[0],pgmSize[1],pgmGreyscale,pgmDataCon)

    def convolutionWithKernelFrequencyDomain(self,inputFileName,kernelName,kernel):
        pgmVer,pgmComment,pgmSize,pgmGreyscale,pgmData,htg = self.readPGMImage(str(inputFileName)+".pgm")
        pgmData = self.moveAxispgmDataBeforeFourier(pgmData,pgmSize)
        pgmDataFourier = self.convertToFourier(pgmData)

        kernel = np.lib.pad(kernel, 127, padwithzerooos)
        kernel = np.delete(kernel, 0, 1)
        kernel = np.delete(kernel, 0, 0)
        kernel = self.moveAxispgmDataBeforeFourier(kernel,pgmSize)
        kernelFourier = self.convertToFourier(kernel)
        pgmResult = kernelFourier*pgmDataFourier
        pgmResult = np.abs(np.fft.ifft2(pgmResult))
        pgmResult = np.abs(np.round(self.moveAxispgmDataBeforeFourier(pgmResult,pgmSize),0).astype(int))
        #print pgmResult
        self.buildPGMFile(str(inputFileName)+"ConFre"+str(kernelName),pgmSize[0],pgmSize[1],pgmGreyscale,pgmResult)
        #print kernel.shape
```

```
    def idealLowPassFilter(self,filename,cutoff):
        pgmVer,pgmComment,pgmSize,pgmGreyscale,pgmData,htg = self.readPGMImage(str(filename)+".pgm")
        pgmData = self.moveAxispgmDataBeforeFourier(pgmData,pgmSize)
        pgmDataFourier = self.convertToFourier(pgmData)
        hFilter = np.zeros((int(pgmSize[0]),int(pgmSize[1])),dtype=np.float)
        hFilter.fill(0)
        m = float(pgmSize[1])
        n = float(pgmSize[0])
        for j in range(int(pgmSize[1])):
            for i in range(int(pgmSize[0])):
                dUV = float(np.sqrt(((float(j)-(m/2.0))**2.0)+((float(i)-(n/2.0))**2.0)))
                if dUV <= float(cutoff):
                    hFilter[i][j] = 1.0
                elif dUV > float(cutoff):
                    hFilter[i][j] = 0.0
        pgmResult = pgmDataFourier*hFilter
        pgmResult = np.abs(np.fft.ifft2(pgmResult))
        pgmResult = self.moveAxispgmDataBeforeFourier(pgmResult,pgmSize)
        pgmResult = np.abs(np.round(pgmResult,0).astype(int))
        self.buildPGMFile(str(filename)+"idealLowPassFilter"+str(cutoff),pgmSize[0],pgmSize[1],pgmGreyscale,pgmResult)

    def butterWorthLowPassFilter(self,filename,cutoff,order):
        pgmVer,pgmComment,pgmSize,pgmGreyscale,pgmData,htg = self.readPGMImage(str(filename)+".pgm")
        pgmData = self.moveAxispgmDataBeforeFourier(pgmData,pgmSize)
        pgmDataFourier = self.convertToFourier(pgmData)
        hFilter = np.zeros((int(pgmSize[0]),int(pgmSize[1])),dtype=np.float)
        hFilter.fill(0)
        m = float(pgmSize[1])
        n = float(pgmSize[0])
        for j in range(int(pgmSize[1])):
            for i in range(int(pgmSize[0])):
                dUV = float(np.sqrt(((float(j)-(m/2.0))**2.0)+((float(i)-(n/2.0))**2.0)))
                hFilter[i][j] = 1.0/(1.0+((dUV/float(cutoff))**(2.0*order)))
```

```
        pgmResult = pgmDataFourier*hFilter
        pgmResult = np.abs(np.fft.ifft2(pgmResult))
        pgmResult = self.moveAxispgmDataBeforeFourier(pgmResult,pgmSize)
        pgmResult = np.abs(np.round(pgmResult,0).astype(int))
        self.buildPGMFile(str(filename)+"ButterWorthLowPassFilter"+str(cutoff)+"Order"+str(order),pgmSize[0],pgmSize[1],pgmGreyscale,pgmResult)

    def GaussianLowPassFilter(self,filename,cutoff):
        pgmVer,pgmComment,pgmSize,pgmGreyscale,pgmData,htg = self.readPGMImage(str(filename)+".pgm")
        pgmData = self.moveAxispgmDataBeforeFourier(pgmData,pgmSize)
        pgmDataFourier = self.convertToFourier(pgmData)
        hFilter = np.zeros((int(pgmSize[0]),int(pgmSize[1])),dtype=np.float)
        hFilter.fill(0)
        m = float(pgmSize[1])
        n = float(pgmSize[0])
        for j in range(int(pgmSize[1])):
            for i in range(int(pgmSize[0])):
                dUV = float(np.sqrt(((float(j)-(m/2.0))**2.0)+((float(i)-(n/2.0))**2.0)))
                hFilter[i][j] = np.exp(-1.0*((dUV**2.0)/(2.0*(cutoff**2.0))))
        pgmResult = pgmDataFourier*hFilter
        pgmResult = np.abs(np.fft.ifft2(pgmResult))
        pgmResult = self.moveAxispgmDataBeforeFourier(pgmResult,pgmSize)
        pgmResult = np.abs(np.round(pgmResult,0).astype(int))
        self.buildPGMFile(str(filename)+"GaussianLowPassFilter"+str(cutoff),pgmSize[0],pgmSize[1],pgmGreyscale,pgmResult)

    def medianFilter(self,filename):
        kernel = np.array([[1,1,1],[1,1,1],[1,1,1]],dtype=np.float)
        kernel = (1.0/9.0)*kernel #median filter kernel
        self.convolutionWithKernel(filename,"MedianFilter",kernel,1)

    def rootMeanSquare(self,originalFileName,afterFileName):
        pgmVerOri,pgmCommentOri,pgmSizeOri,pgmGreyscaleOri,pgmDataOri,htgOri = self.readPGMImage(str(originalFileName)+".pgm")
        pgmVerAf,pgmCommentAf,pgmSizeAf,pgmGreyscaleAf,pgmDataAf,htgAf = self.readPGMImage(str(afterFileName)+".pgm")
```

```
        rootMeanSquareValue = 0.0
        for j in range(int(pgmSizeOri[1])):
            for i in range(int(pgmSizeOri[0])):
                rootMeanSquareValue = rootMeanSquareValue + ((pgmDataOri[i][j]-pgmDataAf[i][j])**2)

        rootMeanSquareValue = rootMeanSquareValue/(float(pgmSizeOri[1])*float(pgmSizeOri[0]))
        rootMeanSquareValue = np.sqrt(rootMeanSquareValue)
        return rootMeanSquareValue
```

การเรียกใช้ Library ในแต่ล่ะข้อ

1.1)

```
myLib = ImageLibFourier()
myLib.padImage("Cross.pgm",28)
pgmVer,pgmComment,pgmSize,pgmGreyscale,pgmData,htg = myLib.readPGMImage("CrossPadding.pgm")
pgmData = myLib.moveAxispgmDataBeforeFourier(pgmData,pgmSize)
pgmDataFourier = myLib.convertToFourier(pgmData)
myLib.pgmDataToPhasePicWithScale("CrossPadding",pgmDataFourier,pgmSize,pgmGreyscale)
myLib.pgmDataToAmplitudePicWithScale("CrossPadding",pgmDataFourier,pgmSize,pgmGreyscale)
```

1.2)

```
myLib = ImageLibFourier()
myLib.padImage("Cross.pgm",28)
myLib.shiftAxisInFourier("CrossPadding.pgm",20,30)
pgmVer,pgmComment,pgmSize,pgmGreyscale,pgmData,htg = myLib.readPGMImage("CrossPaddingShiftAxis.pgm")
pgmData = myLib.moveAxispgmDataBeforeFourier(pgmData,pgmSize)
pgmDataFourier = myLib.convertToFourier(pgmData)
myLib.pgmDataToPhasePicWithScale("CrossPaddingShiftAxis",pgmDataFourier,pgmSize,pgmGreyscale)
myLib.pgmDataToAmplitudePicWithScale("CrossPaddingShiftAxis",pgmDataFourier,pgmSize,pgmGreyscale)
```

1.3)

```
myLib = ImageLibFourier()
myLib.rotatePic("Cross.pgm",-30)
pgmVer,pgmComment,pgmSize,pgmGreyscale,pgmData,htg = myLib.readPGMImage("CrossRotatePic.pgm")
pgmData = myLib.moveAxispgmDataBeforeFourier(pgmData,pgmSize)
pgmDataFourier = myLib.convertToFourier(pgmData)
myLib.pgmDataToPhasePicWithScale("CrossRotatePic",pgmDataFourier,pgmSize,pgmGreyscale)
myLib.pgmDataToAmplitudePicWithScale("CrossRotatePic",pgmDataFourier,pgmSize,pgmGreyscale)
```

1.4)

```
myLib = ImageLibFourier()
myLib.downSample("Cross.pgm",0.5)
pgmVer,pgmComment,pgmSize,pgmGreyscale,pgmData,htg = myLib.readPGMImage("CrossDownSamplePic.pgm")
pgmData = myLib.moveAxispgmDataBeforeFourier(pgmData,pgmSize)
pgmDataFourier = myLib.convertToFourier(pgmData)
myLib.pgmDataToPhasePicWithScale("CrossDownSamplePic",pgmDataFourier,pgmSize,pgmGreyscale)
myLib.pgmDataToAmplitudePicWithScale("CrossDownSamplePic",pgmDataFourier,pgmSize,pgmGreyscale)
```

1.5)

```
myLib = ImageLibFourier()
myLib.inverseFourierPgmWithOutAmplitude("CrossPadding.pgm")
myLib.inverseFourierPgmWithOutPhase("CrossPadding.pgm")
```

1.6)

```
myLib = ImageLibFourier()
myLib.inverseFourierPgmWithOutAmplitude("Lenna.pgm")
myLib.inverseFourierPgmWithOutPhase("Lenna.pgm")
```

1.7)

```
kernel = np.array([[1,2,1],[2,4,2],[1,2,1]],dtype=np.float)
kernel = (1.0/16.0)*kernel
myLib = ImageLibFourier()
myLib.convolutionWithKernel("Chess","Blur",kernel,1)
myLib.convolutionWithKernelFrequencyDomain("Chess","BlurInFourier",kernel)
```

2.1)

```
myLib = ImageLibFourier()
myLib.idealLowPassFilter("Cross",20)
myLib.idealLowPassFilter("Cross",30)
myLib.idealLowPassFilter("Cross",50)
myLib.butterWorthLowPassFilter("Cross",20,2)
myLib.butterWorthLowPassFilter("Cross",30,2)
myLib.butterWorthLowPassFilter("Cross",50,2)
myLib.butterWorthLowPassFilter("Cross",20,20)
myLib.butterWorthLowPassFilter("Cross",20,5)
myLib.butterWorthLowPassFilter("Cross",20,100)
myLib.GaussianLowPassFilter("Cross",20)
myLib.GaussianLowPassFilter("Cross",30)
myLib.GaussianLowPassFilter("Cross",50)
```

2.2)

```
myLib = ImageLibFourier()


myLib.medianFilter("Lenna_noise")
print myLib.rootMeanSquare("Lenna","Lenna_noiseConMedianFilter")


myLib.idealLowPassFilter("Lenna_noise",20)
myLib.idealLowPassFilter("Lenna_noise",30)
myLib.idealLowPassFilter("Lenna_noise",50)


myLib.butterWorthLowPassFilter("Lenna_noise",20,2)
myLib.butterWorthLowPassFilter("Lenna_noise",30,2)
myLib.butterWorthLowPassFilter("Lenna_noise",50,2)


myLib.GaussianLowPassFilter("Lenna_noise",20)
myLib.GaussianLowPassFilter("Lenna_noise",30)
myLib.GaussianLowPassFilter("Lenna_noise",50)


print myLib.rootMeanSquare("Lenna","Lenna_noise")
print myLib.rootMeanSquare("Lenna","Lenna_noiseidealLowPassFilter20")
print myLib.rootMeanSquare("Lenna","Lenna_noiseidealLowPassFilter30")
print myLib.rootMeanSquare("Lenna","Lenna_noiseidealLowPassFilter50")


print myLib.rootMeanSquare("Lenna","Lenna_noise")
print myLib.rootMeanSquare("Lenna","Lenna_noiseButterWorthLowPassFilter20Order2")
print myLib.rootMeanSquare("Lenna","Lenna_noiseButterWorthLowPassFilter30Order2")
print myLib.rootMeanSquare("Lenna","Lenna_noiseButterWorthLowPassFilter50Order2")
```

```
print myLib.rootMeanSquare("Lenna","Lenna_noise")

print myLib.rootMeanSquare("Lenna","Lenna_noiseGaussianLowPassFilter20")

print myLib.rootMeanSquare("Lenna","Lenna_noiseGaussianLowPassFilter30")

print myLib.rootMeanSquare("Lenna","Lenna_noiseGaussianLowPassFilter50")









myLib.idealLowPassFilter("Chess_noise",20)

myLib.idealLowPassFilter("Chess_noise",30)

myLib.idealLowPassFilter("Chess_noise",50)


myLib.butterWorthLowPassFilter("Chess_noise",20,2)

myLib.butterWorthLowPassFilter("Chess_noise",30,2)

myLib.butterWorthLowPassFilter("Chess_noise",50,2)


myLib.GaussianLowPassFilter("Chess_noise",20)

myLib.GaussianLowPassFilter("Chess_noise",30)

myLib.GaussianLowPassFilter("Chess_noise",50)


print myLib.rootMeanSquare("Chess","Chess_noise")

print myLib.rootMeanSquare("Chess","Chess_noiseidealLowPassFilter20")

print myLib.rootMeanSquare("Chess","Chess_noiseidealLowPassFilter30")

print myLib.rootMeanSquare("Chess","Chess_noiseidealLowPassFilter50")


print myLib.rootMeanSquare("Chess","Chess_noise")

print myLib.rootMeanSquare("Chess","Chess_noiseButterWorthLowPassFilter20Order2")

print myLib.rootMeanSquare("Chess","Chess_noiseButterWorthLowPassFilter30Order2")
```

```
print myLib.rootMeanSquare("Chess","Chess_noiseButterWorthLowPassFilter50Order2")


print myLib.rootMeanSquare("Chess","Chess_noise")

print myLib.rootMeanSquare("Chess","Chess_noiseGaussianLowPassFilter20")

print myLib.rootMeanSquare("Chess","Chess_noiseGaussianLowPassFilter30")

print myLib.rootMeanSquare("Chess","Chess_noiseGaussianLowPassFilter50")



myLib.medianFilter("Chess_noise")

print myLib.rootMeanSquare("Chess","Chess_noiseConMedianFilter")
```