HOMOMORPHIC FILTERING (GAUSSIAN FILTER)

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

def normalize(img):
    nImg = np.zeros(img.shape)#, dtype='uint8')
    max_ = img.max()
    min_ = img.min()
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            nImg[i][j] = (img[i][j]-min_)/(max_-min_) * 255
    return np.array(nImg, dtype='uint8')

def gaussianhomomorphic(gl, gh, c, d0, shape):
    im_H = shape[0]
    im_W = shape[1]
    centerx = im_H//2
    centery = im_W//2
    g = np.zeros((shape),np.float32)
    ds = d0**2
    for i in range(im_H):
        for j in range(im_W):
            u = i - centerx
            v = j - centery
            a = 1 - np.exp(((-c*(u**2+v**2))/ds))
            g[i,j] = (gh - gl) * a + gl
    return g

img = cv2.imread('prova.jpg',0)
logimage = np.log1p(img)
f = np.fft.fft2(logimage)
fshift = np.fft.fftshift(f)
magnitude = np.abs(fshift)
phase = np.angle(fshift)
gl = 0.5, gh = 1.2, c = 0.1, d0 = 50
shape = img.shape
ffilter = gaussianhomomorphic(gl, gh, c, d0, shape)
newmagnitude = np.multiply(magnitude,ffilter)
newimg = np.multiply(newmagnitude,np.exp(1j*phase))
spatialoutput = np.real(np.fft.ifft2(np.fft.ifftshift(newimg)))
output = np.expm1(spatialoutput)
plt.imshow(output,gray)
plt.show()
```

Fig1.1: Input Image

Fig1.2:Output Image

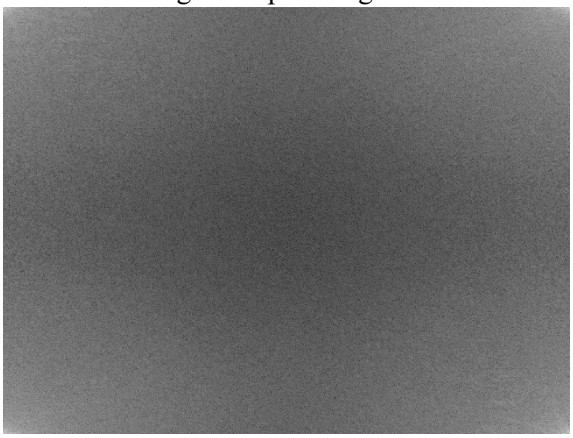

Fig1.3: Magnitude Spectrum

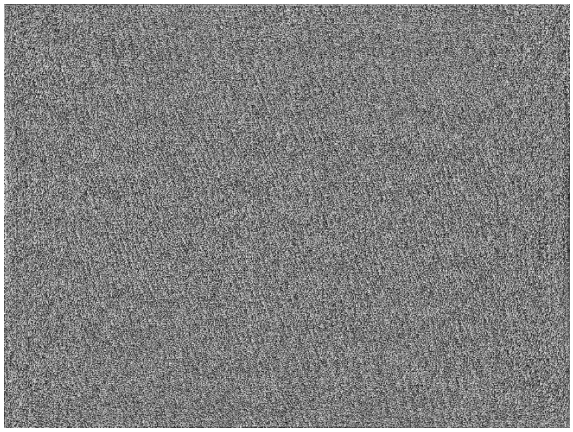

Fig1.4: Phase

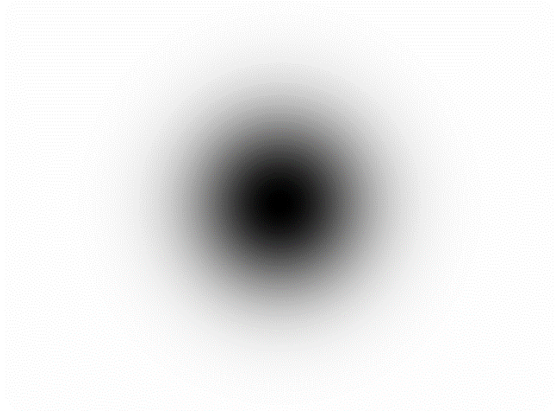

Fig1.3: Gaussian Kernel

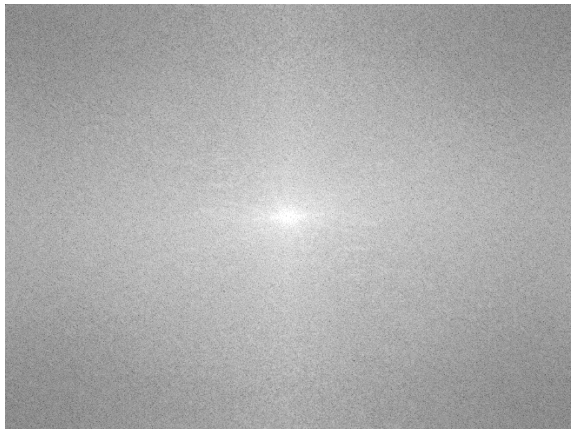

Fig1.4: Center Shifted Magnitude

INVERSE FILTERING

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
def normalize(img):
    nImg = np.zeros(img.shape)#, dtype='uint8')
    max_ = img.max()
    min_ = img.min()
    for i in range(img.shape[0]):
        for j in range(img.shape[1]):
            nImg[i][j] = (img[i][j]-min_)/(max_-min_) * 255
    return np.array(nImg, dtype='uint8')
def zero_pad(image, shape, position='corner'):
    shape = np.asarray(shape, dtype=int)
    imshape = np.asarray(image.shape, dtype=int)
    if np.alltrue(imshape == shape):
        return image
    if np.any(shape <= 0):
        raise ValueError("ZERO_PAD: null or negative shape given")
    dshape = shape - imshape
    if np.any(dshape < 0):
        raise ValueError("ZERO_PAD: target size smaller than source one")
    pad_img = np.zeros(shape, dtype=image.dtype)
    idx, idy = np.indices(imshape)
    if position == 'center':
        if np.any(dshape % 2 != 0):
            raise ValueError("ZERO_PAD: source and target shapes have different parity.")
        offx, offy = dshape // 2
    else:
        offx, offy = (0, 0)
    pad_img[idx + offx, idy + offy] = image
    return pad_img
def psf2otf(psf, shape):
    if np.all(psf == 0):
        return np.zeros_like(psf)
    inshape = psf.shape
    # Pad the PSF to outsize
    psf = zero_pad(psf, shape, position='corner')
    f = np.fft.fft2(psf)
    otf = np.fft.fftshift(f)
    return np.abs(otf)
def butterworth(d0,n,shape):
    im_H = shape[0]
    im_W = shape[1]
```

```
    centerx = im_H//2
    centery = im_W//2
    g = np.zeros((shape),np.float32)
    for i in range(im_H):
        for j in range(im_W):
            u = i - centerx
            v = j - centery
            p = 1+(((u**2+v**2)**0.5)/d0)**(2*n)
            q = 1/p
            g[i,j] = q
    return g
def motion_blurr(img):
    im_H = img.shape[0]
    im_W = img.shape[1]
    ksize = 5
    padding = (ksize-1)//2
    img = cv2.copyMakeBorder(img, padding, padding, padding, padding, cv2.BORDER_REPLICATE)
    output_H = (im_H + ksize-1)
    output_W = (im_W + ksize-1)
    result = np.zeros((output_H,output_W),np.float32)
    motion_blurr_filter = np.array([[1,0,0,0,0],
                                    [0,1,0,0,0],
                                    [0,0,1,0,0],
                                    [0,0,0,1,0],
                                    [0,0,0,0,1]])
    for x in range(padding,output_H-padding):
        for y in range(padding,output_W-padding):
            a = 0
            for i in range(-padding,padding+1):
                for j in range(-padding,padding+1):
                    a += motion_blurr_filter[i+padding,j+padding]*img[x-i,y-j]
            result[x,y] = a/5
            result[x,y] /= 255
    return (result,motion_blurr_filter)
img = cv2.imread('lena.png',0)
blurred_image,psf = motion_blurr(img)
f = np.fft.fft2(blurred_image)
fshift = np.fft.fftshift(f)
magnitude = np.abs(fshift)
phase = np.angle(fshift)
shape = blurred_image.shape
otf = psf2otf(psf,blurred_image.shape)
newmagnitude = magnitude/otf
d0 = 52, n = 2
bfilter = butterworth(d0, n, shape)
newmagnitude *= bfilter
newimg = np.multiply(newmagnitude,np.exp(1j*phase))
```

```
spatialoutput = np.real(np.fft.ifft2(np.fft.ifftshift(newimg)))
plt.imshow(spatialoutput,'gray')
plt.show()
```

Fig2.1: Input Image

Fig2.2: Blurred Image

Fig2.3: Restored Image