Image Processing Prog1

- Introduction / Objectives

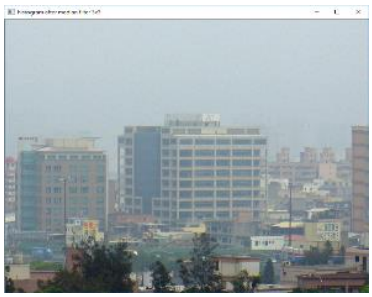
The main purpose of this program is to do image enhance. After I review the course slide, I use some processing technique to completing this assignment. For example: intensity transformation, spatial filter and histogram equalization.

- A review of the methods you have used

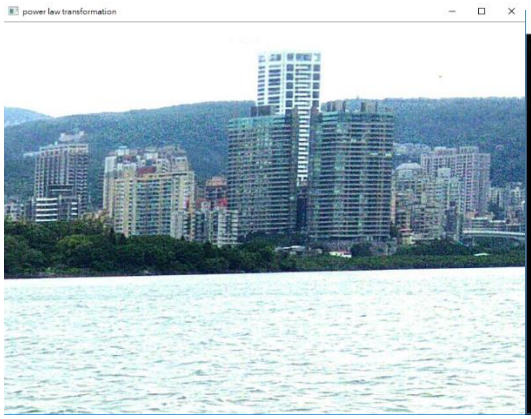
- Contrast stretching
- Power-law transformation
- Histogram equalization
- Gaussian filter
- Median filter
- Convolution
- Bilateral filter

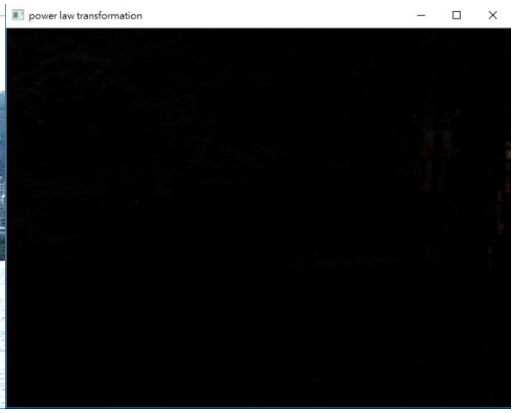
- A explanation of the experiments you have done, and the results

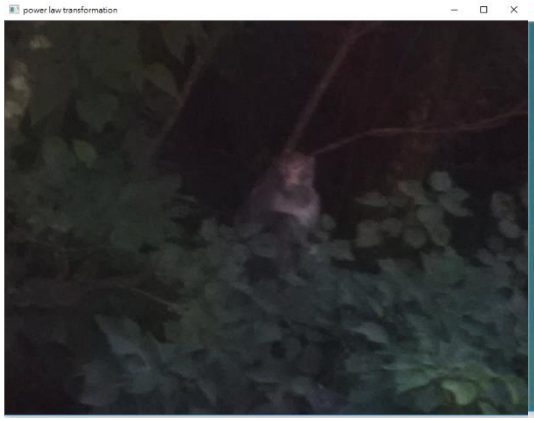
- Contrast stretching

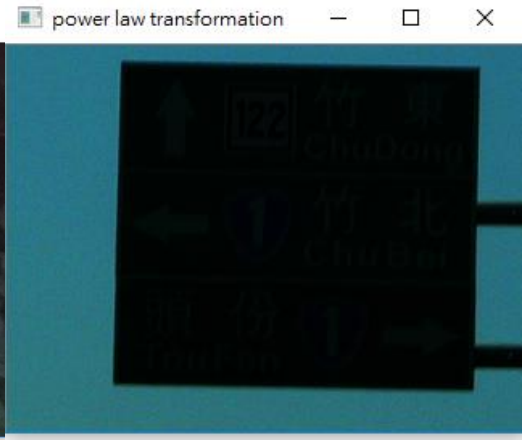


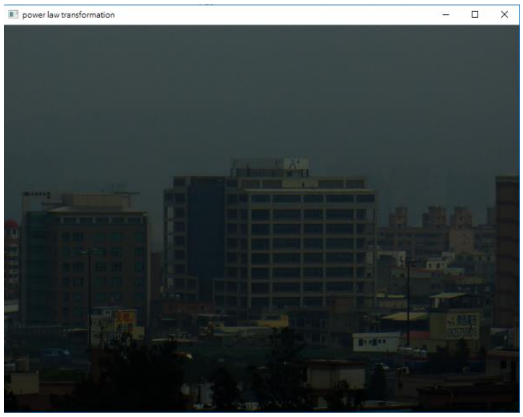
- Power law transformation (gamma=3)



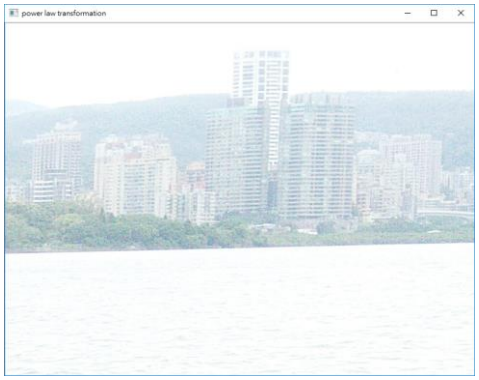


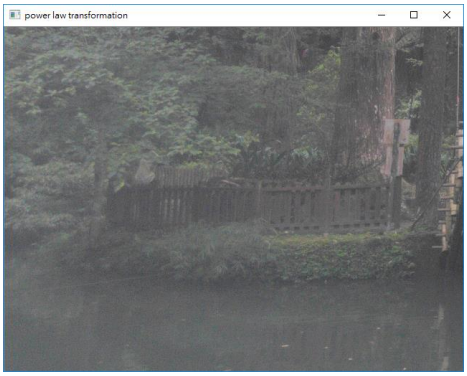


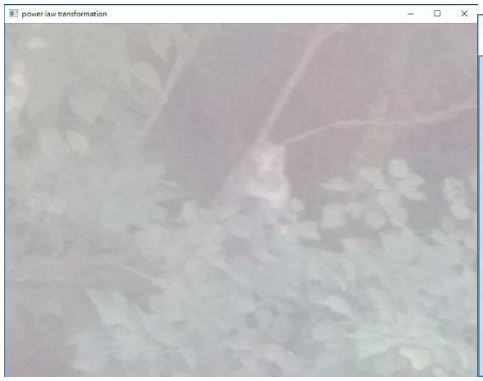


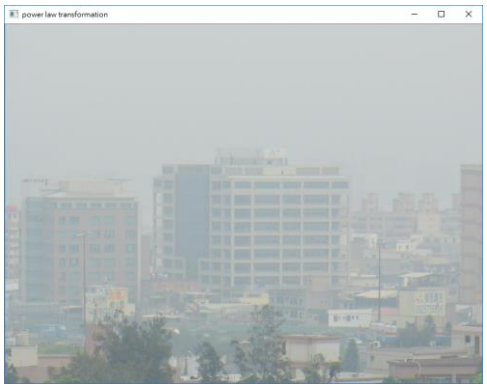


- Power law transformation (gamma=0.5)

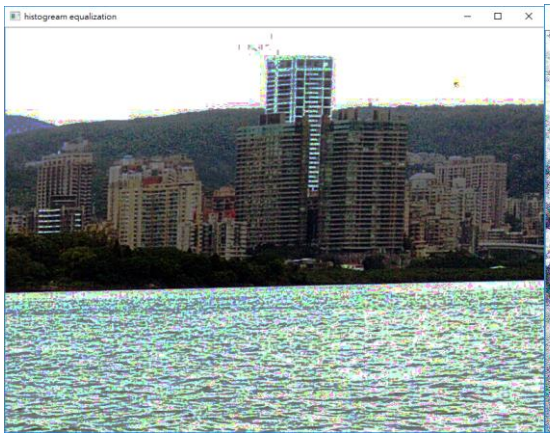


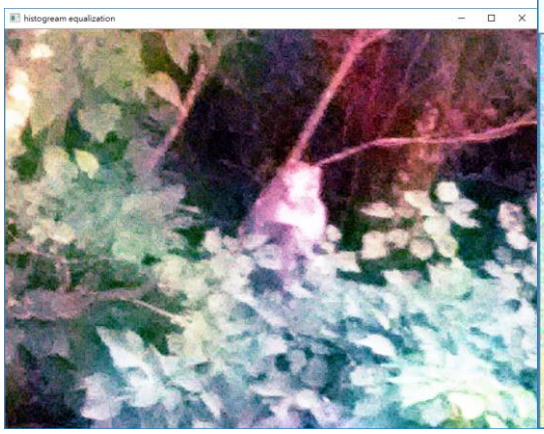


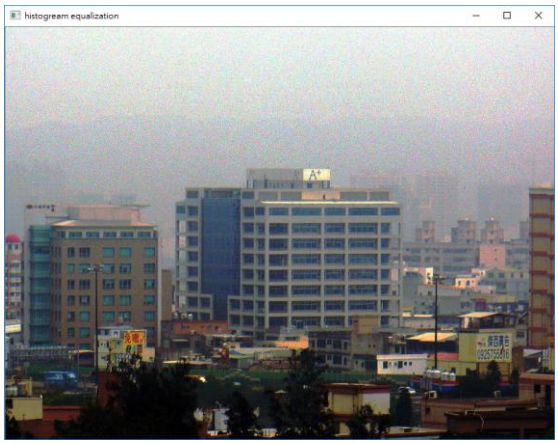




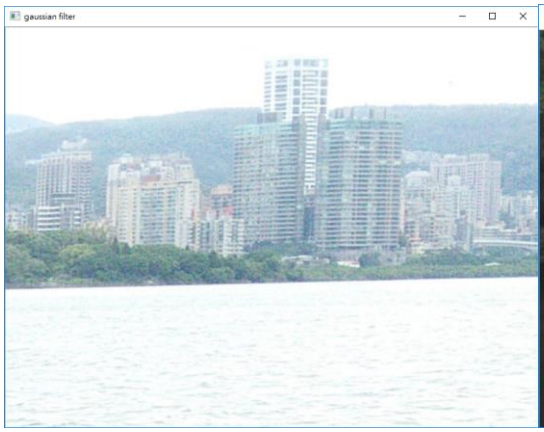
- Histogram equalization

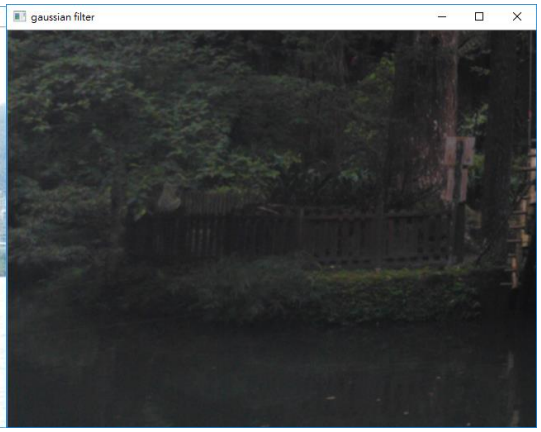


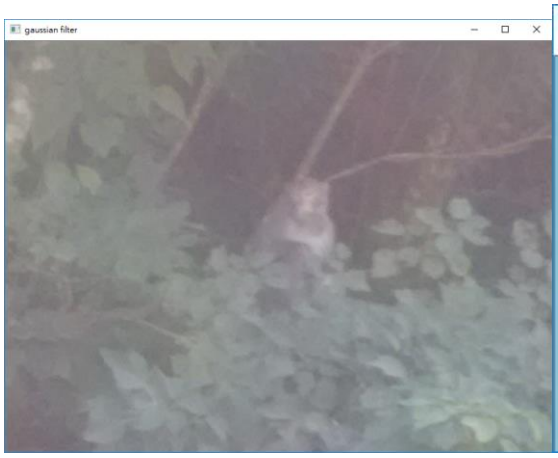


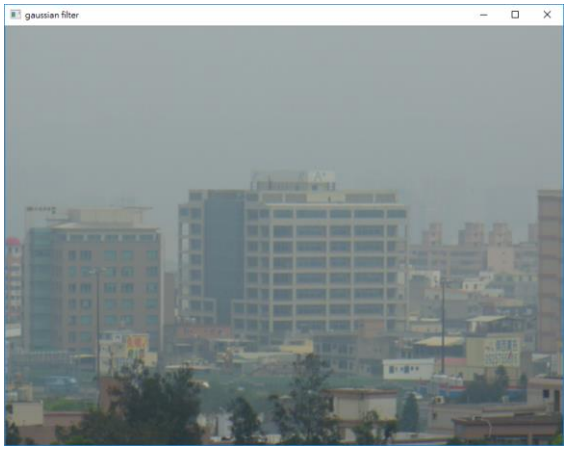


- Gaussian Filter sigma=1 kernel_size=3x3
  I also try the different sigma kernel_size, but the result of experiment isn't good. So I don't put the result of attempts.

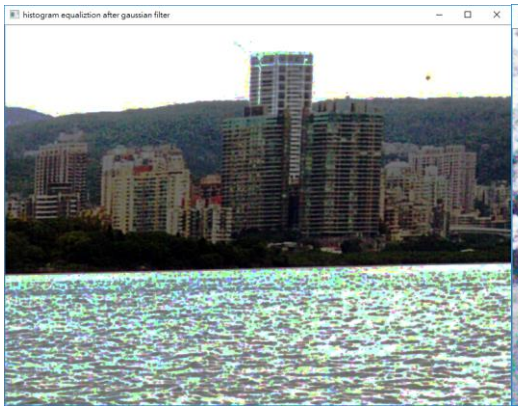


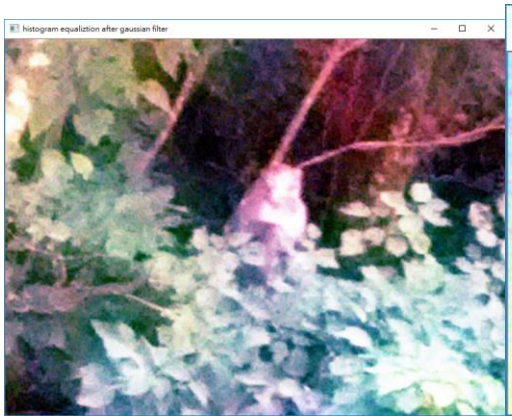


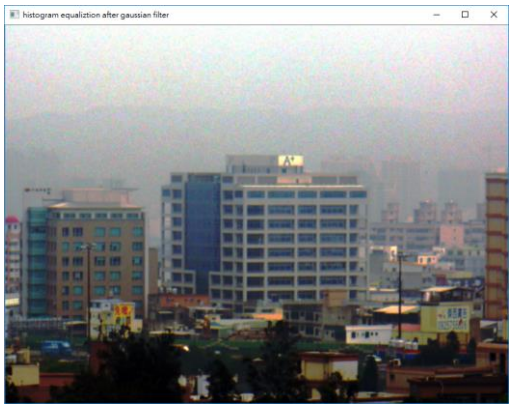




- Histogram equalization after Gaussian filter

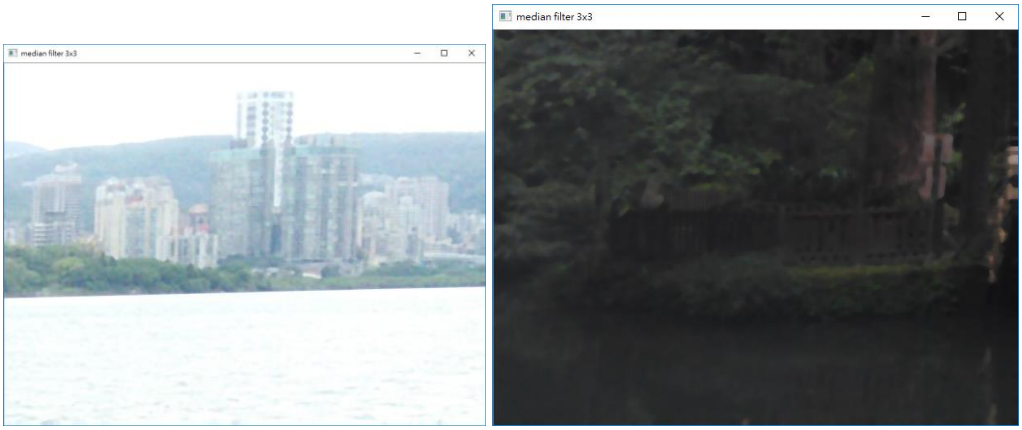




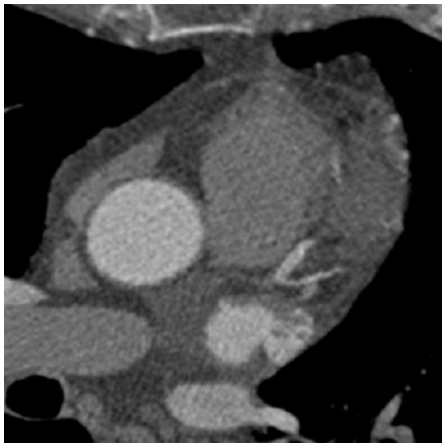
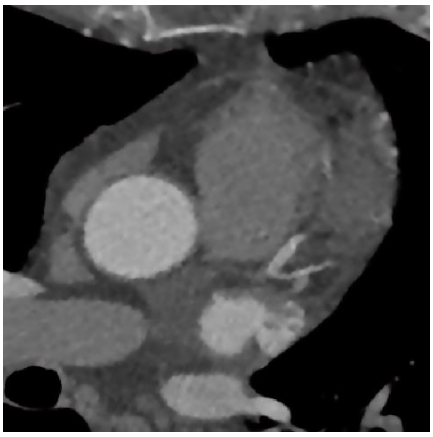


- Median filter by size 3x3

  I also try the larger size (ex: 5x5), but the result is too blurred. So I don’t use it.



- Gaussian filter vs Bilateral Filter

- Discussions: Your observations, interpretations of results, and remaining questions

Based on my experiment, I find some points and make the interpretations of results:

1. If the pixel of image is concentrate on some pixel value, I will use the histogram equalization. Like image 5 there is a good performance. But it may enhance the noise in some images like image 1,2,3.
2. If the color of image is too dark or too white, I will use the power-law transformation to color adjustment. Like image 3 there is a good performance. I also try the gamma>1 and gamma<1 transformation to observe the difference. Then, I find that it is not suitable for using gamma<1 power-law transformation if the color of image is bright/white originally.
3. There is the best performance in image 2 after using contrast stretching. I think the color contrast of image 2 isn't obviously. So when I use the contrast stretching, there is a good result.
4. Although I do the Gaussian filter or median filter to de-noise before using the intensity transformation, the result is not change a lot.
5. Compared Gaussian filter and Bilateral Filter, I find that there is the better performance by using Bilateral Filter on CT image/image 6. I think the reason is Bilateral Filter keeps the edge gradient nicely. However, in Gaussian filter the change of edge is linear. Therefore, the gradient is used to show a gradual state, and if this is shown in the image, it is edge distortion.

What techniques do I try, there are good performance in the images:

Image 1: power-law transformation

Image 2: contrast stretching

Image 3: power-law transformation

Image 4: I don’t find it. But in my attempt contrast stretching is good by comparison

Image 5: histogram equalization

Image 6: Bilateral filter

Remaining questions:

1. Though I use some transformation after using de-noise method, the result is not change a lot. Maybe I don’t use the correct combination.
2. I don’t find the suitable image processing technique for image 4.

- Code List

main.py

If I want to use some combinations of image processing technique (ex : I use the de-noise filter before I use the transformation), I will move the code block to the place I need.

```
import cv2
import numpy as np
from lib import
contrast_stretch,power_law_transformation,hisogram_equaliztion,gaussian_
smoothing_filter,convolution,median_filter,bilateralfilter

for i in range(1,6,1):
    origin=cv2.imread('p1im'+str(i)+'.png')

    img=contrast_stretch(origin)
    cv2.imshow('constrat stretching',img)
    cv2.waitKey(0)

    img=power_law_transformation(origin,3)
    cv2.imshow('power law transformation',img)
    cv2.waitKey(0)

    r,g,b=cv2.split(origin)
```

```
    cdf_r=hisogram_equaliztion(r)
    cdf_g=hisogram_equaliztion(g)
    cdf_b=hisogram_equaliztion(b)
    equ=cv2.merge((cdf_r[r],cdf_g[g],cdf_b[b]))
    cv2.imshow('histogream equalization',equ)
    cv2.waitKey(0)



sigma_s = 2.8
sigma_r = 0.1*255
for i in range(1,7,1):
    origin=cv2.imread('p1im'+str(i)+'.png')

    img=median_filter(origin,3)
    cv2.imshow('meidan filter',img)
    cv2.waitKey(0)

    kernel=gaussian_smoothing_filter(3,1)
    img=convolution(kernel,origin)
    cv2.imshow('gaussian filter',img)
    cv2.waitKey(0)

    img_gray=cv2.cvtColor(origin,cv2.COLOR_BGR2GRAY)
    img = bilateralfilter(origin,img_gray,sigma_s,sigma_r)
    cv2.imshow('bilateral filter',img)
    cv2.waitKey(0)
```

Lib.py

```
import numpy as np
import math
import cv2

def contrast_stretch(img):
    M,N,C=img.shape
    min_val=256
    max_val=0
    mins=[]
```

```
    maxs=[]
    for c in range(C):
        for i in range(M):
            if min(img[i,:,c])<min_val:
                min_val=min(img[i,:,c])
            if max(img[i,:,c])>max_val:
                max_val=max(img[i,:,c])
        mins.append(min_val)
        maxs.append(max_val)

    for c in range(C):
        for i in range(M):
            for j in range(N):
                img[i,j,c]=(img[i,j,c]-mins[c])/(maxs[c]-mins[c])*255

    return img

def power_law_transformation(img,gamma):
    return np.array(255*(img/255)**gamma,dtype='uint8')

def hisogram_equaliztion(img):
    hist,bins = np.histogram(img.flatten(),256,[0,256])
    cdf = hist.cumsum()
    cdf_m = np.ma.masked_equal(cdf,0)
    cdf_m = (cdf_m - cdf_m.min())*255/(cdf_m.max()-cdf_m.min())
    cdf = np.ma.filled(cdf_m,0).astype('uint8')
    return cdf

def gaussian_smoothing_filter(size,sigma):
    smooth_filter=np.zeros((size,size))
    size=int(size/2)
    i=0
    for s in range(-size,size+1,1):
        j=0
        for t in range(-size,size+1,1):
            smooth_filter[i,j]=1/(2*math.pi*sigma**2)*math.exp(-
((s**2+t**2)/2*sigma**2))
            j+=1
```

```
        i+=1
    smooth_filter/=smooth_filter.sum()
    return smooth_filter

def convolution(kernel,img):
    m,n,c=img.shape
    output=np.zeros((m,n,c),dtype='uint8')
    k_h,k_w=kernel.shape
    pad=k_h//2
    img=cv2.copyMakeBorder(img,pad,pad,pad,pad,cv2.BORDER_REPLICATE)
    for c in range(c):
        for i in range(pad,m+pad):
            for j in range(pad,n+pad):
                    roi=img[i-pad:i+pad+1,j-pad:j+pad+1,c]
                    val=(roi*kernel).sum()
                    output[i-pad,j-pad,c]=val
    return output

def median_filter(img,pad):
    img=cv2.copyMakeBorder(img,pad,pad,pad,pad,cv2.BORDER_REPLICATE)
    m,n,channel=img.shape
    output=np.zeros((m,n,channel),dtype='uint8')
    for c in range(channel):
        for i in range(pad,m+pad):
            for j in range(pad,n+pad):
                median=np.median(img[i-pad:i+pad+1,j-pad:j+pad+1,c])
                output[i-pad,j-pad,c]=median
    return output

def bilateralfilter(image, texture, sigma_s, sigma_r):
    r = int(np.ceil(3 * sigma_s))
    # Image padding
    if image.ndim == 3:
        h, w, ch = image.shape
        I = np.pad(image, ((r, r), (r, r), (0, 0)),
'symmetric').astype(np.float32)
    elif image.ndim == 2:
        h, w = image.shape
```

```
        I = np.pad(image, ((r, r), (r, r)), 'symmetric').astype(np.float32)
    else:
        print('Input image is not valid!')
        return image
    # Check texture size and do padding
    if texture.ndim == 3:
        ht, wt, cht = texture.shape
        if ht != h or wt != w:
            print('The guidance image is not aligned with input image!')
            return image
        T = np.pad(texture, ((r, r), (r, r), (0, 0)),
'symmetric').astype(np.int32)
    elif texture.ndim == 2:
        ht, wt = texture.shape
        if ht != h or wt != w:
            print('The guidance image is not aligned with input image!')
            return image
        T = np.pad(texture, ((r, r), (r, r)), 'symmetric').astype(np.int32)
    # Pre-compute
    output = np.zeros_like(image)
    scaleFactor_s = 1 / (2 * sigma_s * sigma_s)
    scaleFactor_r = 1 / (2 * sigma_r * sigma_r)
    # A lookup table for range kernel
    LUT = np.exp(-np.arange(256) * np.arange(256) * scaleFactor_r)
    # Generate a spatial Gaussian function
    x, y = np.meshgrid(np.arange(2 * r + 1) - r, np.arange(2 * r + 1) - r)
    kernel_s = np.exp(-(x * x + y * y) * scaleFactor_s)
    # Main body
    if I.ndim == 2 and T.ndim == 2:     # I1T1 filter
        for y in range(r, r + h):
            for x in range(r, r + w):
                wgt = LUT[np.abs(T[y - r:y + r + 1, x - r:x + r + 1] - T[y,
x])] * kernel_s
                output[y - r, x - r] = np.sum(wgt * I[y - r:y + r + 1, x - r:x
+ r + 1]) / np.sum(wgt)
    elif I.ndim == 3 and T.ndim == 2:     # I3T1 filter
        for y in range(r, r + h):
            for x in range(r, r + w):
```

```
                wgt = LUT[abs(T[y - r:y + r + 1, x - r:x + r + 1] - T[y, x])] *
kernel_s
                wacc = np.sum(wgt)
                output[y - r, x - r, 0] = np.sum(wgt * I[y - r:y + r + 1, x -
r:x + r + 1, 0]) / wacc
                output[y - r, x - r, 1] = np.sum(wgt * I[y - r:y + r + 1, x -
r:x + r + 1, 1]) / wacc
                output[y - r, x - r, 2] = np.sum(wgt * I[y - r:y + r + 1, x -
r:x + r + 1, 2]) / wacc
    elif I.ndim == 3 and T.ndim == 3:     # I3T3 filter
        for y in range(r, r + h):
            for x in range(r, r + w):
                wgt = LUT[abs(T[y - r:y + r + 1, x - r:x + r + 1, 0] - T[y, x,
0])] * \
                      LUT[abs(T[y - r:y + r + 1, x - r:x + r + 1, 1] - T[y, x,
1])] * \
                      LUT[abs(T[y - r:y + r + 1, x - r:x + r + 1, 2] - T[y, x,
2])] * \
                      kernel_s
                wacc = np.sum(wgt)
                output[y - r, x - r, 0] = np.sum(wgt * I[y - r:y + r + 1, x -
r:x + r + 1, 0]) / wacc
                output[y - r, x - r, 1] = np.sum(wgt * I[y - r:y + r + 1, x -
r:x + r + 1, 1]) / wacc
                output[y - r, x - r, 2] = np.sum(wgt * I[y - r:y + r + 1, x -
r:x + r + 1, 2]) / wacc
    elif I.ndim == 2 and T.ndim == 3:     # I1T3 filter
        for y in range(r, r + h):
            for x in range(r, r + w):
                wgt = LUT[abs(T[y - r:y + r + 1, x - r:x + r + 1, 0] - T[y, x,
0])] * \
                      LUT[abs(T[y - r:y + r + 1, x - r:x + r + 1, 1] - T[y, x,
1])] * \
                      LUT[abs(T[y - r:y + r + 1, x - r:x + r + 1, 2] - T[y, x,
2])] * \
                      kernel_s
                output[y - r, x - r] = np.sum(wgt * I[y - r:y + r + 1, x - r:x
+ r + 1]) / np.sum(wgt)
```

```
    else:
        print('Something wrong!')
        return image

    return output
```