Assignment 3

160209F

HETTIARACHCHI H.H.K.B.D.

ENTC

1. Gaussian Smoothing Filter

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

def gaussianFilter(img ,kernel_size,sigma):
    p=kernel_size//2
    img_transformed = np.zeros(img.shape)
    kernel=np.zeros((kernel_size,kernel_size))
    kernel1=np.zeros((1,kernel_size))
    kernel2=np.zeros((kernel_size,1))
    for t in range(kernel_size):
        x=t-p
        kernel1[0][t]=(np.exp(-((x**2)/(2*(sigma**2))))/np.sqrt(2*np.pi*(sigma**2)))
        kernel2[t] = (np.exp(-((x**2)/(2*(sigma**2))))/np.sqrt(2*np.pi*(sigma**2)))
    kernel=np.matmul(kernel2,kernel1)
    kernel_sum=np.sum(kernel)
    kernel=kernel/kernel_sum
    a, b = img.shape[0], img.shape[1]
    for i in range(3):
        for x in range(p, a - p):
            for y in range(p, b - p):
                region = img[x - p:x + p + 1, y - p:y + p + 1, i]
                region_sum=np.sum(np.multiply(region,kernel))
                img_transformed[x, y, i] = region_sum
    img_transformed = img_transformed.astype('uint8')
    return img_transformed
img=cv.imread("images\im07small.png",cv.IMREAD_COLOR)
kernel_size=5
img_transformed=gaussianFilter(img ,5,1.8)
img=cv.cvtColor(img,cv.COLOR_BGR2RGB)
img_transformed=cv.cvtColor(img_transformed,cv.COLOR_BGR2RGB)
f,axarr=plt.subplots(1,2)
axarr[0].imshow(img)
axarr[0].set_title('Original image')
axarr[1].imshow(img_transformed)
axarr[1].set_title('Sigma of transformed image : '+str(1.8)+"\nKernel size: "+str(5))
plt.show()
```

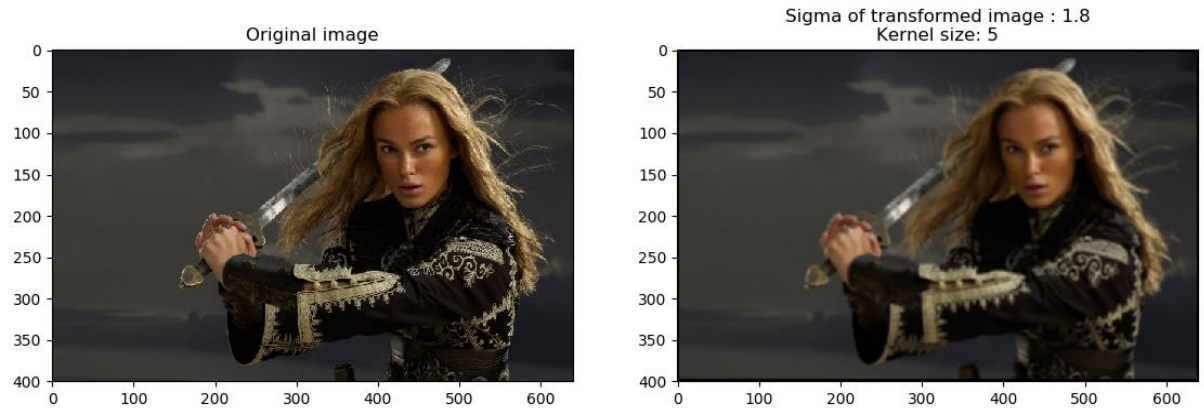

Figure 1

In Gaussian smoothing, the Gaussian kernel is created by *gaussianFilter* function with the arguments image, kernel size and the sigma value for the filter. The image is more smoothened and blurred with the increment of the kernel size and the sigma. This is used in feature enhancer as DoG because only the high frequency spatial information are suppressed by the Gaussian filter. In *gaussianFilter* function, 2D gaussian kernel has been obtained by multiplying 1D gaussian kernel.

2. Implementation of Difference of Gaussian

```
import cv2 as cv
import matplotlib.pyplot as plt
import numpy as np
import GaussianFilter as gf

def DoG(img,k,sigma,kernel_size1, kernel_size2):
    img_trans1=gf.Gaussianfilter(img,kernel_size1,sigma)
    img_trans2=gf.Gaussianfilter(img,kernel_size2,k*sigma)
    img_transformed=img_trans2-img_trans1
    return (img_transformed)
img=cv.imread("images\im07small.png",cv.IMREAD_COLOR)
kernel_size=5
img_transformed=DoG(img ,1.5,1.8,9,13)
img=cv.cvtColor(img,cv.COLOR_BGR2RGB)
img_transformed=cv.cvtColor(img_transformed,cv.COLOR_BGR2RGB)
f,axarr=plt.subplots(1,2)
axarr[0].imshow(img)
axarr[0].set_title('Original image')
axarr[1].imshow(img_transformed)
axarr[1].set_title('Sigma of transformed image : '+str(1.8)+"\nKernel size: "+str(5))
plt.show()
```

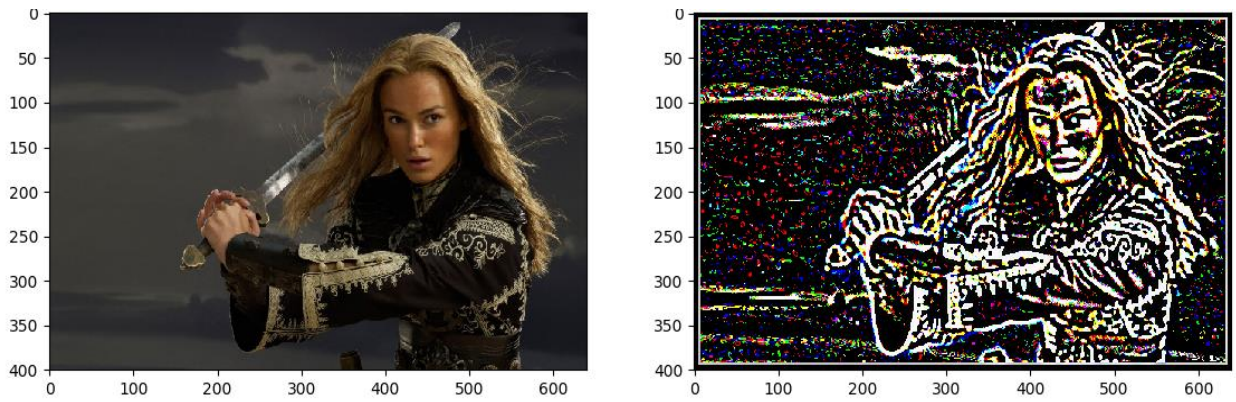

Figure 2

The Difference of Gaussian is a feature enhancement obtaining by subtracting two gaussian blurred images. Here the DoG a is obtained by subtracting the Gaussian filtered image having a sigma of 1.8 and kernel of size 9 by the Gaussian filtered image having a sigma of 2.7 and kernel size of 13. DoG acts as a bandpass filter since the spatial information are preserved by this method. As the output image, DoG can be used for feature detection like edges, corners and so on.

3. Feature detection

```
def ScaleSpace(img,k,sigma0,nims):
    keypoints=[]
    img_mid = np.zeros(img.shape)
    downs=1
    s1,s2=img.shape
    while(max(s1,s2)>=(1.5)*sigma0*(k**4)):
        sigma = sigma0
        layers=[]
        for var3 in range(nims):
           layers.append(DoG(img,k,sigma))
           sigma=k*sigma
        for var1 in range(layers[0].shape[0]):
                for var2 in range(layers[0].shape[1]):
                    for var3 in range(1,nims-1):
                            points = []
                            for varx in range(max(var1 - 1, 0), min(var1 + 2,
layers[0].shape[0])):
                                for vary in range(max(var2 - 1, 0), min(var2 + 2,
layers[0].shape[1])):
                                    if(varx,vary)!=(var1,var2):
                                        points.append(layers[var3-1][varx, vary])
                                        points.append(layers[var3][varx, vary])
                                        points.append(layers[var3+1][varx, vary])
                            if (layers[var3][var1, var2] >max(points) or layers[var3][var1, var2]
< min(points)):
                                keypoints.append((( downs*var1, downs*var2), downs*(
sigma0*(k**var3))))
        sigma0=sigma0*(k**2)
        img=img[::2,::2]
        s1=s1//2
        s2=s2//2
        downs=downs*2


    img1=cv.imread("img1.ppm",cv.IMREAD_COLOR)
    print(len(keypoints))
    print(img1.shape[0]*img1.shape[1])
    for keyp in keypoints:
        cv.circle(img1, (keyp[0][1],keyp[0][0]), int(keyp[1]*(2**0.5)),(0,255,0))
    cv.imshow("Image",img1)
    cv.waitKey(0)
img=cv.imread("img1.ppm",cv.IMREAD_GRAYSCALE)
kernel_size=5
ScaleSpace(img,1.5,1.8,4)
```

Figure 3

4. Plot of circles

The Gaussian filter is applied 5 times with different sigma values with the starting value of 1.8 and increasing factor of 1.5. Then in each octave difference between these Gaussian images is obtained. Local extremes of these DoG images are considered as key points with scale depending on the sigma and amount by which the original image is down sampled at the detected stage.

5. Comparison with Harris corner detection

```
import cv2 as cv
import matplotlib.pyplot as plt
import numpy as np
def Harris(img):
    gray = cv.cvtColor(img,cv.COLOR_BGR2GRAY)
    gray = np.float32(gray)
    dst = cv.cornerHarris(gray,2,3,0.04)
    dst = cv.dilate(dst,None)
    img[dst>0.01*dst.max()]=[0,255,0]
    cv.imshow('dst',img)
    cv.waitKey(0)
    cv.destroyAllWindows()
img1=cv.imread("\imagesimg1.ppm",cv.IMREAD_COLOR)
img2=img[::2,::2,:]
Harr(img1)
Harr(img2)
```

Figure 5

Figure 4

The Harris Corner Detector is just a mathematical way of determining which windows produce large variations when moved in any direction. With each window, a score R is associated. Based on this score, you can figure out which ones are corners and which ones are not.

$$R = \det M - k(trace\, M)^2$$

$$\det M = \lambda_1 \lambda_2$$

$$trace\, M = \lambda_1 + \lambda_2$$

Here, when we apply the Harris corner detection method for different scales as shown in *Figure 5* and *Figure 6* , the same corners in different scales has been variated because of non-invariance on scale changes of Harris corner detection. But in blob detection using scale space extrema features are detected at different scales. That is why we can see circles with different radii at different locations. With Harris corner detector one can only detect key points but with Scale space extrema one can assign a scale to a key point.