CS-663 Assignment 3 Q1

Soham Naha (193079003)
Akshay Bajpai (193079002)
Mohit Agarwala (19307R004)

October 18, 2020

1 (30 points) Harris Corner Detection

Input Image :

- 1/data/boat.mat

Assume the pixel dimensions to be equal along both axes, i.e., assume an aspect ratio of 1:1 for the axes.
Shift and rescale the intensities in the image to lie within the range [0, 1] .
Implement the Harris corner detector algorithm. The parameters underlying this algorithm are: two Gaussian smoothing levels involved in computing the structure tensor (the first Gaussian to smooth the image before computing the gradient, the second Gaussian for the weighted averaging to compute the structure tensor), the constant k in the corner-ness measure. Tune these three parameters to get the best results.

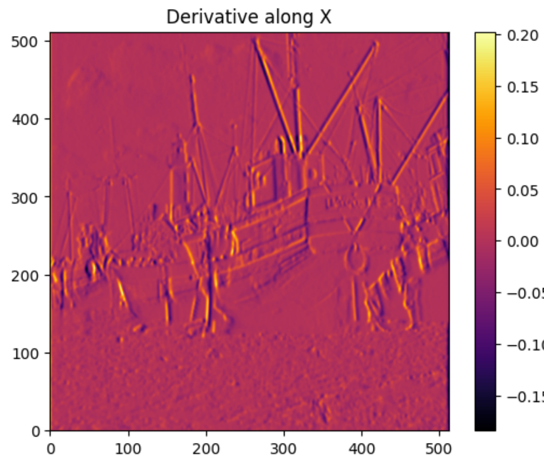
- (16 points) Write a function myHarrisCornerDetector.m to implement this.
- (4 points) Display the derivative images, corresponding to the partial derivatives along the X and Y axes.
- (4 points) Display an image (along with a colormap) of the principal eigenvalue of the structure tensor evaluated at each pixel. Display another image (along with a colormap) of the other eigenvalue of the structure tensor evaluated at each pixel.
- (6 points) Display the image (along with a colormap) of the Harris corner-ness measure. Tune the free parameters such that positive values in this image should correspond to "corner" structures in the image. Report all three parameter values used.

```python
import numpy as np
import cv2
import matplotlib.pyplot as plt
import h5py
from tqdm import tqdm

def normalize_image(image):
    """Normalize the image to remain between 0-1
    Take the image and use the Min-Max normalization criterion to normalize the images

    inputs: image(input image to be normalized)
    outputs: normalized(normalized image)
    """
    out = image.copy()
    normalized = (out-np.min(out))/(np.max(out)-np.min(out))
    return normalized

def get_image(filename):
    """Extract the image from mat file
    inputs: filename(filepath of the .mat file)
    outputs: the image as numpy array after normalizing
    """
    f = h5py.File(filename,"r")
    out = f.get('imageOrig')
    out = np.array(out)
```

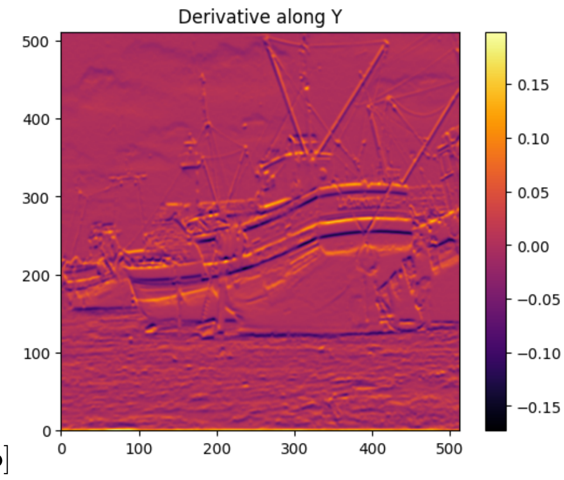
```python
    return normalize_image(out)

def dnorm(x,sigma,mu=0):
    """Calculates the 1D Gaussian kernel
    inputs: x(kernel position),sigma(standard deviation of the kernel),mu(optional, default=0)
    outputs: 1D kernel
    """
    return (1.0/(np.sqrt(2*np.pi)*sigma))*np.e**(-(((x - mu)/sigma)**2) / 2)

def gaussian_kernel(ksize,sigma):
    """Calculates and returns 2D the Gaussian Kernel
    inputs: ksize(Gaussian Kernel Size),sigma(standard deviation)
    outputs: kernel_2D (2D gaussian kernel)
    """
    kernel_1D = np.linspace(-(ksize // 2), ksize // 2, ksize)
    for i in range(ksize):
        kernel_1D[i] = dnorm(kernel_1D[i], sigma=sigma)

    # computers outer product of two 1-D gaussian kernels
    # to produce a 2D Gaussian Kernel
    kernel_2D = np.outer(kernel_1D.T, kernel_1D.T)
    kernel_2D *= 1.0/np.sum(kernel_2D)
    return kernel_2D

def convolution(image,kernel_size,sigma):
    """Performs Convolution of the image with a Gaussian kernel
    inputs: image(input image),kernel_size(Gaussian kernel size),
            sigma(Gaussian Kernel Standard deviation)
    outputs: output (convoluted image with the kernel)
    """
    kernel = gaussian_kernel(kernel_size,sigma=sigma)
    image_row, image_col = image.shape
    kernel_row, kernel_col = kernel.shape
    output = np.zeros(image.shape)

    # CREATING ZERO-PADDED IMAGE
    pad_height = int((kernel_row - 1) / 2)
    pad_width = int((kernel_col - 1) / 2)
    padded_image = np.zeros((image_row + (2 * pad_height), image_col + (2 * pad_width)))
    padded_image[pad_height:padded_image.shape[0] - pad_height, pad_width:padded_image.shape[1]
                -pad_width] = image

    # CONVOLUTION OPERATION DONE HERE
    for row in tqdm(range(image_row),desc="Gaussian Convolution"):
        for col in range(image_col):
            output[row, col] = np.sum(kernel * padded_image[row:row + kernel_row,
                                                col:col + kernel_col])
    output = (output/np.max(output))

    return output

def calculate_gradient(image):
    """Calculate the image gradient
    inputs: image (input image)
    outputs: dy(gradient along Y),dx(gradient along X),
             Ixx(square of gradient along X),Iyy(square of gradient along Y),
             Ixy(product of the gradient along X and Y)
    """
    dy, dx = np.gradient(image)
    Ixx = dx**2
    Ixy = dy*dx
    Iyy = dy**2
    return (dy,dx,Ixx,Iyy,Ixy)

def cornernessMeasure(Sxx,Syy,Sxy,k):
```

```python
    """Calculate the cornerness of the window
    inputs: Sxx,Syy,Sxy,k(cornerness constant)
    outputs: det(determinant of matrix),trace(trace of matrix),r(cornerness measure)
    """
    det = (Sxx * Syy) - (Sxy**2)
    trace = Sxx + Syy
    r = det - k*(trace**2)
    return r,det,trace

def findCorners(filename, window_size_blur, sigma_weights, k, thresh):
    """
    Finds and returns list of corners and new image with corners drawn
    :param img: The original image
    :param window_size: The size (side length) of the sliding window
    :param k: Harris corner constant. Usually 0.04 - 0.06
    :param thresh: The threshold above which a corner is counted
    :return:
    """

    img = get_image(filename)
    sigma_window = 0.3*((window_size_blur-1)*0.5 - 1) + 0.8
    img = convolution(img,window_size_blur,sigma_window)
    #img = img*255.0
    img = np.rot90(img)

    #Find x and y derivatives
    (dy,dx,Ixx,Iyy,Ixy) = calculate_gradient(img)
    height,width = img.shape

    plt.figure()
    plt.imshow(dy,cmap='inferno',origin="lower")
    plt.title("Derivative along Y")
    plt.colorbar()
    plt.savefig('../images/y_derivative.png',cmap='inferno',bbox_inches="tight")

    plt.figure()
    plt.imshow(dx,cmap='inferno',origin="lower")
    plt.title("Derivative along X")
    plt.colorbar()
    plt.savefig('../images/x_derivative.png',cmap='inferno',bbox_inches="tight")

    cornerList = []
    offset = int(window_size_blur/2)

    eigvalues = np.zeros((height,width,2))
    corner_img =np.zeros((height,width))

    #Loop through image and find our corners
    min_r = 1000000
    max_r = 0
    for y in tqdm(range(offset, height-offset),desc="Finding Corners..."):
        for x in range(offset, width-offset):
            #Calculate sum of squares
            weight_kernel = gaussian_kernel(2*offset+1,sigma=sigma_weights)
            windowIxx = Ixx[y-offset:y+offset+1, x-offset:x+offset+1]*weight_kernel
            windowIxy = Ixy[y-offset:y+offset+1, x-offset:x+offset+1]*weight_kernel
            windowIyy = Iyy[y-offset:y+offset+1, x-offset:x+offset+1]*weight_kernel
            Sxx = windowIxx.sum()
            Sxy = windowIxy.sum()
            Syy = windowIyy.sum()

            #Find determinant and trace, use to get corner response
            r,det,trace = cornernessMeasure(Sxx,Syy,Sxy,k)
            eigvalues[x,y,0] = (trace + np.sqrt(trace**2 - 4*det))/2
            eigvalues[x,y,1] = (trace - np.sqrt(trace**2 - 4*det))/2
```

```
            corner_img[x,y]=r

            if(r>max_r):
                max_r = r
            if (r<min_r):
                min_r = r
            #If corner response is over threshold, color the point and add to corner list
            if r > thresh:
                cornerList.append([x, y, r])

    print("Minimum Cornerness Value :",min_r)
    print("Maximum Cornerness Value :",max_r)
    return img, cornerList,corner_img ,eigvalues

def plotCorners(finalImg,cornerList):
    plt.figure()
    plt.imshow(finalImg,cmap="gray",origin="lower")
    plt.colorbar()
    plt.title("Red Corners in Image")
    for i in cornerList:
        k,j,l = i
        plt.scatter(k,j,color="r",s=0.3,marker="*")
    plt.savefig('../images/Harris.png',bbox_inches='tight')

def plotEigenValues(eigvalues):
    plt.figure()
    plt.imshow(np.rot90(eigvalues[:,:,0]),cmap='inferno')
    plt.title("Eigen Value 1")
    plt.colorbar()
    plt.savefig('../images/Eigen_value1.png',cmap='inferno',bbox_inches='tight')
    plt.figure()
    plt.imshow(np.rot90(eigvalues[:,:,1]),cmap='inferno')
    plt.title("Eigen Value 2")
    plt.colorbar()
    plt.savefig('../images/Eigen_value2.png',cmap='inferno',bbox_inches='tight')

def plotCornernessMeasure(corner_img):
    plt.figure()
    plt.imshow(np.rot90(corner_img),cmap='hot')
    plt.title("Cornerness Measure Plot")
    plt.colorbar()
    #plt.show()
    plt.savefig('../images/Cornerness.png',cmap='hot')

def HarrisCornerDetector(filename,window_size_blur, sigma_weights, k, thresh):
    img, cornerList,corner_img ,eigvalues = findCorners(filename,window_size_blur, sigma_weights, k, thresh)
    plotCorners(img,cornerList)
    plotEigenValues(eigvalues)
    plotCornernessMeasure(corner_img)



if __name__=="__main__":
    filename="../data/boat.mat"
    window_size = 7
    k = 0.06
    thresh = 0.8*10**(-5)
    sigma_weights = 1.2
    print("Window Size: " + str(window_size))
    print("K-value for Cornerness measure: " + str(k))
    print("Corner Response Threshold:" + str(thresh))
    HarrisCornerDetector(filename, int(window_size),sigma_weights, float(k), thresh)
```

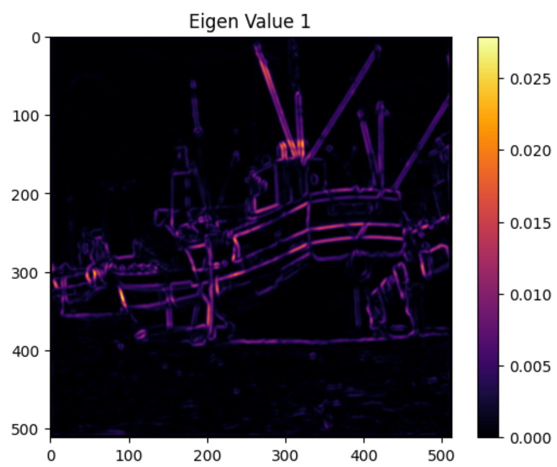


[a]

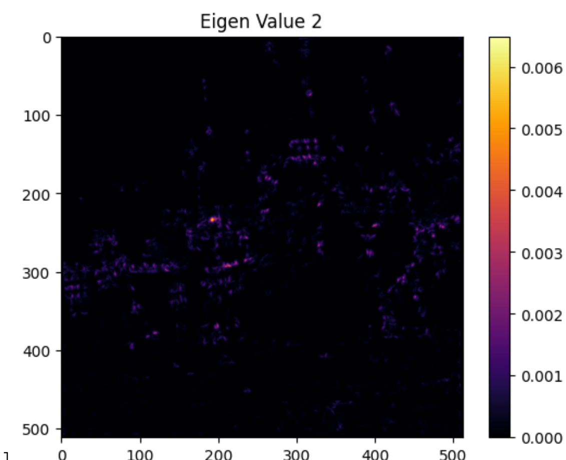


[b]

Figure 1: (a) X-derivative of 1/data/boat.mat (b) Y-derivative of 1/data/boat.mat

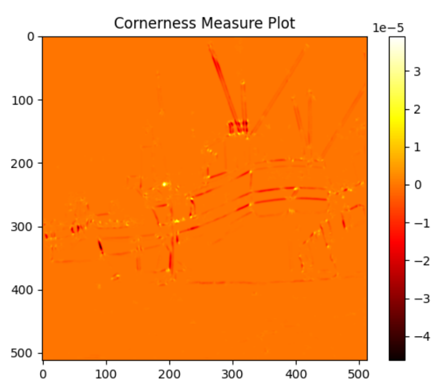


[a]

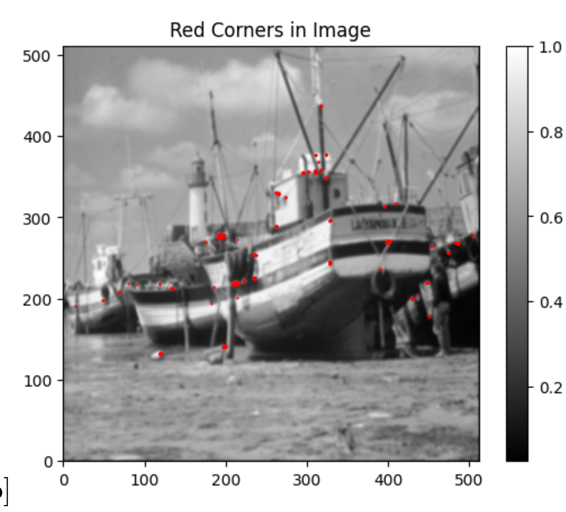


[b]

Figure 2: (a) Eigen Value 1 of 1/data/boat.mat (b) Eigen Value 2 of 1/data/boat.mat



[a]



[b]

Figure 3: (a) Cornerness Measure Image of 1/data/boat.mat (b) Corner Plot on image of 1/data/boat.mat

Observation

The three parameters tuned are:

- Window Size of Gaussian to smooth the image : 7
- Gaussian Kernel Standard Deviation for weights : 1.2
- Cornerness constant (k): 0.06
- Threshold value for calculating corners : 0.8×10^{-5}

myMainScript.py

```
from myHarrisCornerDetection import HarrisCornerDetector

filename="../data/boat.mat"
window_size = 7
k = 0.06
thresh = 0.8*10**(-5)
sigma_weights = 1.2
print("Window Size: " + str(window_size))
print("K-value for Cornerness measure: " + str(k))
print("Corner Response Threshold:" + str(thresh))
HarrisCornerDetector(filename, int(window_size),sigma_weights, float(k), thresh)
```