Corner detection

The goal of this assignment is to detect the corners using the techniques Shi-tomasi and harris corner detection.

Shi-Tomasi corner detection :

1st step:

The first step is to find the moment matrix for every pixel in the image and then find the eigenvalues for every pixel. The eigenvalues (lambda_1 and lambda_2) define the state of the pixel in the image. To find the corner we need we need to get the minimum lambda for every pixel, it is considered as the shi-tomasi cornerness measure.

The following python function gets the shi-tomasi cornerness measure for every pixel. In this function we first find the x derivative and y derivative of every pixel using derivative of gaussian kernel. We used gaussian kernel because it is more robust to noise. So, Ix and Iy contains the derivatives of every pixel.

The next thing is to find the sum of Ix^2 , iy^2 and Ixy in small window (we’ve taken 3*3) for every pixel.these are the elements from the moment matrix. From these eigenvalues can be calculated by using the formula.

$$\lambda_{\pm} = \frac{1}{2}\left[(h_{11} + h_{22}) \pm \sqrt{4h_{12}h_{21} + (h_{11} - h_{22})^2}\right]$$

The Shi-tomasi cornerness measure for every pixel is the minimum of lambda values and we return those values from this function.

```
def Shi_Tomasi_cornerness_measure(im, window_size, sigma = 3) :

    Ix = zeros(im.shape)
    filters.gaussian_filter(im, (sigma,sigma), (0,1), Ix) # storing x derivate of the

    Iy = zeros(im.shape)
    filters.gaussian_filter(im, (sigma,sigma), (1,0), Iy) # storing y derivate of the

    # Calculating the elements of the momment matrix
    Ixx = ndimage.uniform_filter(Ix * Ix, window_size)
    Ixx = window_size*Ixx # sum(Ixx around 3*3 window)

    Iyy = ndimage.uniform_filter(Iy * Iy, window_size)
    Iyy = window_size*Iyy # sum(Iyy around 3*3 window)

    Ixy = ndimage.uniform_filter(Ix * Iy, window_size)
    Ixy = window_size*Ixy # sum(Ixy around 3*3 window)

    # Finding the eigen values for every pixel
    lambda_1 = 1/2*( (Ixx + Iyy) + np.sqrt(4*Ixy*Ixy + (Ixx - Iyy)**2) )
    lambda_2 = 1/2*( (Ixx + Iyy) - np.sqrt(4*Ixy*Ixy + (Ixx - Iyy)**2) )

    # lambda min is the cornerness measure for Shi_tomasi method
    lambda_min = zeros(im.shape)
    lambda_min = np.minimum(lambda_1, lambda_2)

    return (lambda_min)
```

2nd step:

```
def Shi_Tomasi_coords(lambda_min, threshold, min_dist):
    coords = np.argwhere(lambda_min > threshold)
    lambda_values = [lambda_min[c[0],c[1]] for c in coords] # getting the lambda valu
    index = argsort(lambda_values)  #sort the index values of lambda_values
    window = zeros(lambda_min.shape) #window by taking every value as one
    window[min_dist:-min_dist, min_dist:-min_dist] = 1
    # select the best points taking min_distance into account
    localMax_coords = []
    for i in index:
        if (window[coords[i,0],coords[i,1]] == 1):
            localMax_coords.append(coords[i])
            window[(coords[i,0]-min_dist):(coords[i,0]+min_dist),
            (coords[i,1]-min_dist):(coords[i,1]+min_dist)] = 0
    return (localMax_coords) # returning the localmax of lambda coords
```

The goal of this function is to find the local_maximum (In small patch) of minimum lambda_values which are greater than the user specified threshold value.

First we've taken the coords of the lambda_min which are greater than the threshold. And then we sort the indices of lambda_min values of the coords in decreasing order. And then we've taken the window size with same dimensions of the lambda_min and put all values as one except at the boundaries (to get rid of padding problem).

The index is a 1d array containing the sorted indices of lambda_values. So if the corresponding coordinate of window matrix is equal to one we are taking the coordinate as the local_max coordinate and then we put zeros to 10*10 patch around this local_max coordinate as zero (In this we can't get every coordinate is the local_max). And then we are returning the local_max coordinate.

3rd step:

```
def plot_Shi_Tomasi_coords(image,coords):
    figure()
    gray()
    imshow(image)
    plot([p[1] for p in coords],[p[0] for p in coords], 'r.')
    axis('off')
    show()
```

The goal of this function is to plot the red dots on the local_max coordinates in the gray scale image.

Results:

```
im1 = array(Image.open('Image1.jpg').convert('L'))
lambda_min = Shi_Tomasi_cornerness_measure(im1, sigma = 3, window_size = 9)
coords = Shi_Tomasi_coords(lambda_min, threshold=122, min_dist=10)
plot_Shi_Tomasi_coords(im1, coords)
```

```
im2 = array(Image.open('Image2.jpg').convert('L'))
lambda_min = Shi_Tomasi_cornerness_measure(im2, sigma = 3, window_size = 9)
coords = Shi_Tomasi_coords(lambda_min, threshold=34, min_dist=10)
plot_Shi_Tomasi_coords(im2, coords)
```

```
im3 = array(Image.open('Image3.jpg').convert('L'))
lambda_min = Shi_Tomasi_cornerness_measure(im3, sigma = 3, window_size = 9)
coords = Shi_Tomasi_coords(lambda_min, threshold=18, min_dist=12)
plot_Shi_Tomasi_coords(im3, coords)
```

Harris corner detection:

1st step:

```python
def Harris_corner_cornerness_measure(im, sigma, alpha, window_size) :

    # Calculating the gradients of the image
    Ix = zeros(im.shape)
    filters.gaussian_filter(im, (sigma,sigma), (0,1), Ix) # storing x derivate of the
    Iy = zeros(im.shape)
    filters.gaussian_filter(im, (sigma,sigma), (1,0), Iy) # storing y derivate of the

    # Calculating the elements of the momment matrix
    Ixx = ndimage.uniform_filter(Ix * Ix, window_size)
    Ixx = window_size*Ixx # sum(Ixx around 3*3 window)

    Iyy = ndimage.uniform_filter(Iy * Iy, window_size)
    Iyy = window_size*Iyy # sum(Iyy around 3*3 window)

    Ixy = ndimage.uniform_filter(Ix * Iy, window_size)
    Ixy = window_size*Ixy # sum(Ixy around 3*3 window)

    # Finding the eigen values for every pixel
    lambda_1 = 1/2*( (Ixx + Iyy) + np.sqrt(4*Ixy*Ixy + (Ixx - Iyy)**2) )
    lambda_2 = 1/2*( (Ixx + Iyy) - np.sqrt(4*Ixy*Ixy + (Ixx - Iyy)**2) )

    f = zeros(im.shape)
    f = (lambda_1 * lambda_2) - alpha * ((lambda_1 + lambda_2) ** 2)
    #f = (lambda_1*lambda_2) /(lambda_1 + lambda_2)

    return (f)
```

In harris cornerness measure finding function everything is same as in shi-tomasi cornerness measure calculation except instead taking lambda_min as the cornerness measure we use f value. The formula for f value is

$$f = \lambda_1\lambda_2 - \kappa(\lambda_1 + \lambda_2)^2$$

$$= determinant(H) - \kappa\,(trace(H))^2$$

After returning f measure, repeat the step2 and step3 process in shi-tomasi corner method for harris corner method. The k value we used here are different for different images.

Results:

```
f = Harris_corner_cornerness_measure(im1, sigma = 3, alpha = 0.03, window_size = 9)
coords = harris_coords(f, threshold=10000, min_dist=15)
plot_harris_coords(im1, coords)
```

```
f = Harris_corner_cornerness_measure(im2, sigma = 3, alpha = 0.03, window_size = 9)
coords = harris_coords(f, threshold=800, min_dist=13)
plot_harris_coords(im2, coords)
```

```
f = Harris_corner_cornerness_measure(im3, sigma = 3, alpha = 0.02, window_size = 9)
coords = harris_coords(f, threshold=1000, min_dist=13)
plot_harris_coords(im3, coords)
```

Conclusion :

The threshold values are different for different images in both Shi-tomasi and harris_corner detection. The x derivative and y derivative can be calculated by using sobel kernel also but the results aren’t getting good because sobel is not robust to noise while gaussian is.