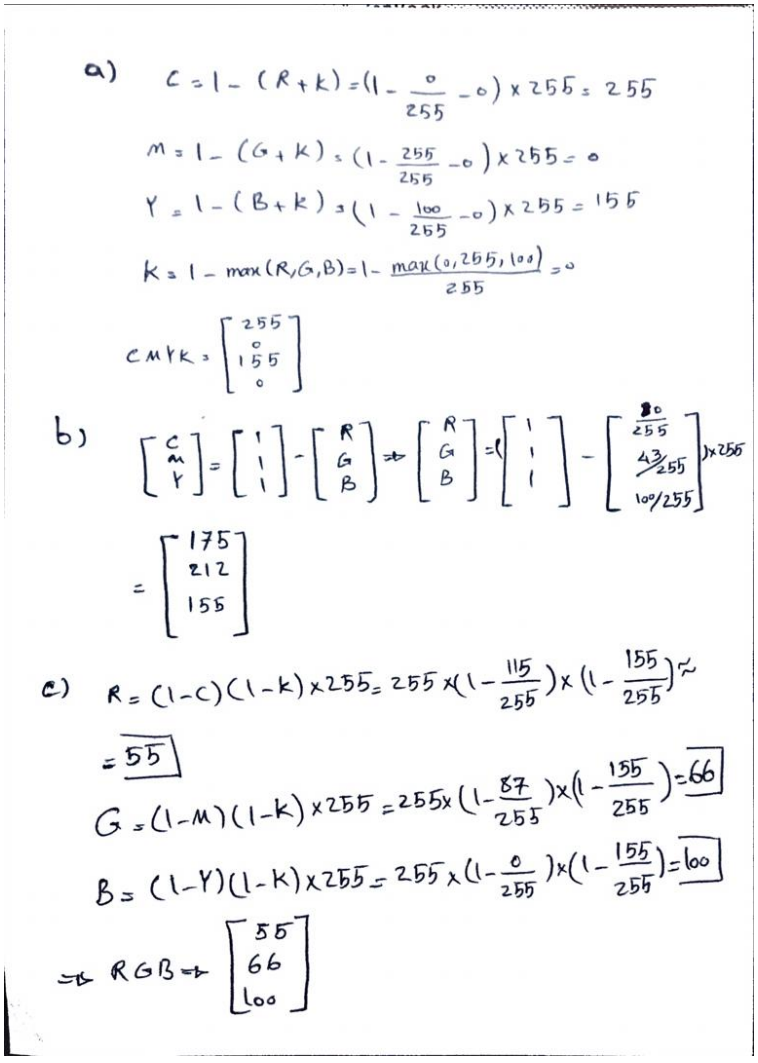
HW7

November 9, 2020

Zahra Hosseini - 96531226

1 PART 1

1.1 a



2 PART 2

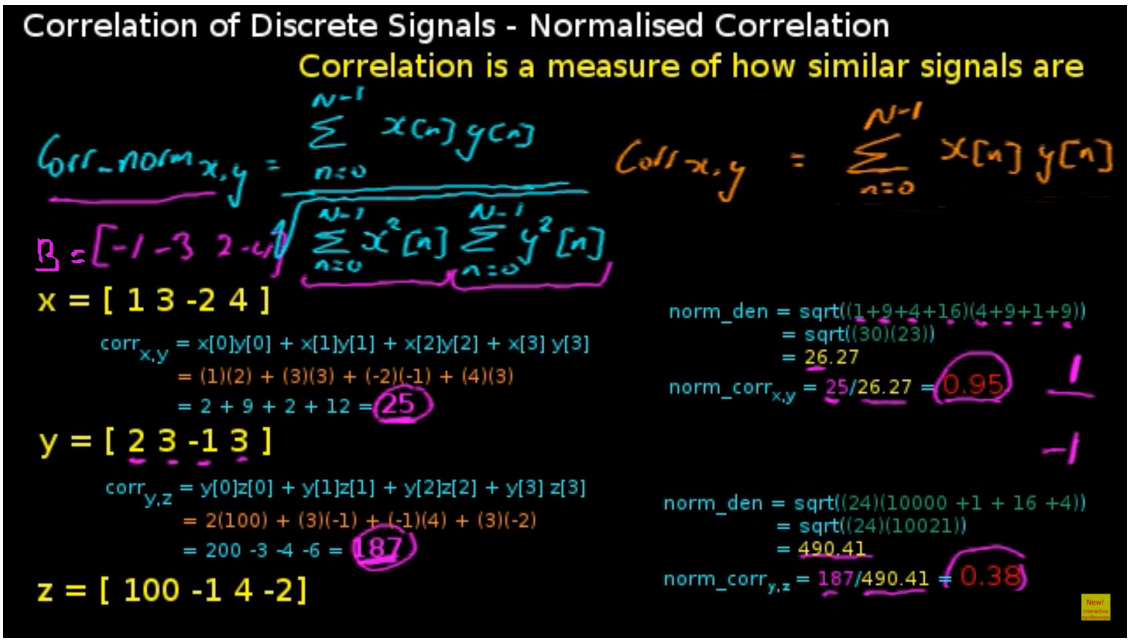
The correlation between two signals or simply the cross correlation is a standard tool for evaluating the degree to which two signals are similar. It is an elementary approach to match two image patches, for feature detection as well as a component of more sophisticated techniques .

$$Cross\ correlation = \sum_{i=1}^{n}(X_i \breve{} \overline{X})(Y_i \breve{} \overline{Y})$$

Normalized cross correlation (NCC) has been commonly used as a metric to evaluate the degree of similarity (or dissimilarity) between two compared images. The main advantage of the normalized cross correlation over the ordinary cross correlation is that it is less sensitive to linear changes in the amplitude of illumination in the two compared images. Furthermore, the Normalized Cross Correlation is confined in the range between –1 and 1. The setting of detection threshold value is much simpler than the cross correlation. The Normalized Cross Correlation does not have a minimal frequency domain expression. It cannot be directly computed using the more efficient FFT (Fast Fourier Transform) in the spectral domain

$$Normalized\ cross\ correlation = \frac{\sum_{i=1}^{n}(X_i \breve{} \overline{X})(Y_i \breve{} \overline{Y})}{\sqrt{\left[\sum_{i=1}^{n}(X_i \breve{} \overline{X})^2 \ \sum_{i=1}^{n}(Y_i \breve{} \overline{Y})^2\right]}}$$

A numerical example:



Sources:
link1,link2, link3

3 PART 3

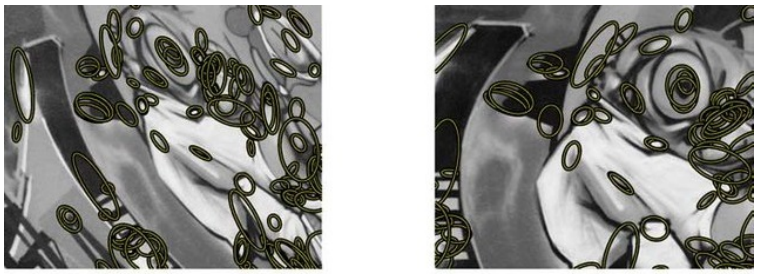
It uses Hessian matrix to detect key points and remove edges:

$$\mathcal{H} = \begin{bmatrix} I_{xx} & I_{xy} \\ I_{xy} & I_{yy} \end{bmatrix}$$
$$\det(\mathcal{H}) = I_{xx}I_{yy} - (wI_{xy})^2$$
$$trace(\mathcal{H}) = I_{xx} + I_{yy}$$

Corner is identified by its large variation in both x and y directions. Mathematically we can build what so called hessian matrix that state the variation (derivative) in x, y and xy direction.

Hessian matrix describes the 2nd order local image intensity variations around the selected voxel. For the obtained Hessian matrix, eigenvector decomposition extracts an orthonormal coordinate system that is aligned with the second order structure of the image. Having the eigenvalues and knowing the (assumed) model of the structure to be detected and the resulting theoretical behavior of the eigenvalues, the decision can be made if the analyzed voxel belongs to the structure being searched.

But in the Harris ,The basic idea of its second moment matrix, $M = \begin{bmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{bmatrix}$
, edges and other unstable points can be removed via M. Interest points are either detected with the Harris detector or with a detector based on the Hessian matrix. In both cases scale-selection is based on the Laplacian, and the shape of the elliptical region is determined with the second moment matrix of the intensity gradient
all of the other steps in hessian algorithm is same as harris. the hessian uses The second partial derivative, it causes to chose local maximum with a lower threshold.



(a) **Harris-Affine**

(b) **Hessian-Affine**

Sources:
link1 , link2, link3

4 PART 4

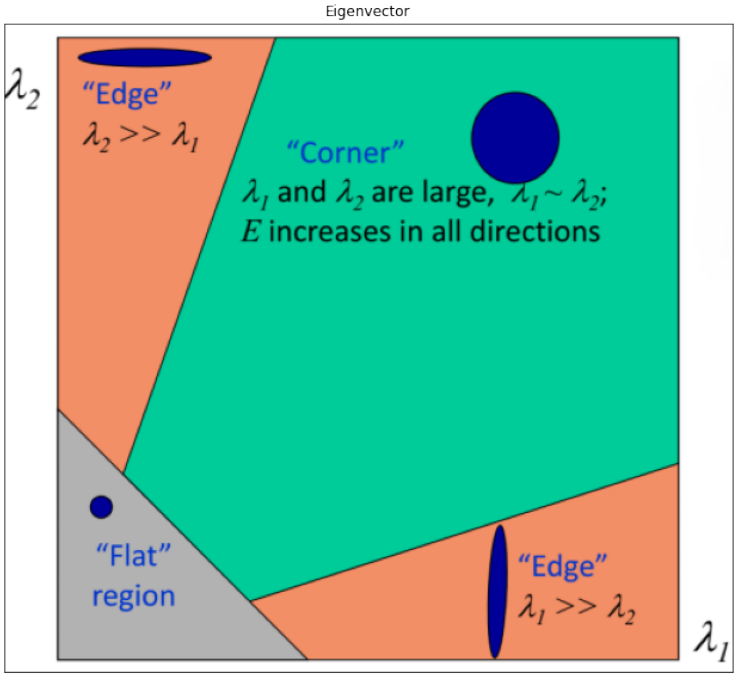
M1 —> The green circle —> Corner

M2 —> The blue circle —> Edge

M3 —> The red circle —> Edge

M4 —> The yellow circle —> Flat

```
M1 values:
 [92.93739228 50.87260772]
M1 vectors:
 [[ 0.89183951  0.45235195]
 [-0.45235195  0.89183951]]
M2 values:
 [1.63540288e+02 1.05011879e-01]
M2 vectors:
 [[ 0.99999912  0.00132774]
 [-0.00132774  0.99999912]]
M3 values:
 [  0.169902 164.401498]
M3 vectors:
 [[-0.99999544  0.00302014]
 [-0.00302014 -0.99999544]]
M4 values:
 [0.14344888 0.32345112]
M4 vectors:
 [[-0.9987461   0.05006215]
 [-0.05006215 -0.9987461 ]]
```



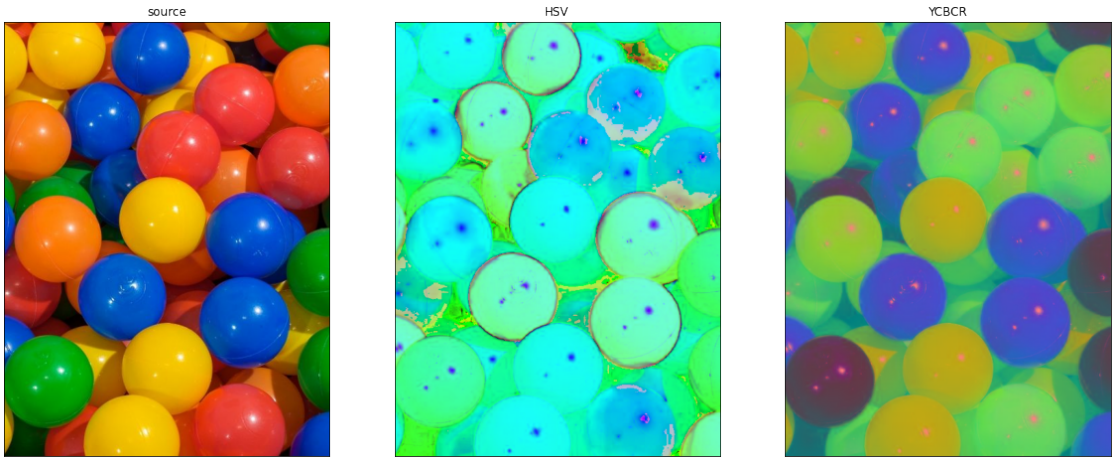
[1]:
```
import numpy as np
M1=np.array([
    [84.33 ,-16.97],
    [-16.97,59.48]
])
M2=np.array([
    [163.54 ,-0.217],
    [-0.217,0.1053]
])
M3=np.array([
    [0.1714,-0.496],
    [-0.496,164.4]
])
M4=np.array([
    [0.1439,-0.009],
    [-0.009,0.323]
])
values1,vectors1=np.linalg.eig(M1)
values2,vectors2=np.linalg.eig(M2)
values3,vectors3=np.linalg.eig(M3)
values4,vectors4=np.linalg.eig(M4)

print("M1 values:\n",values1)
print("M1 vectors:\n",vectors1)
print("M2 values:\n",values2)
print("M2 vectors:\n",vectors2)
print("M3 values:\n",values3)
print("M3 vectors:\n",vectors3)
print("M4 values:\n",values4)
print("M4 vectors:\n",vectors4)
```

5 PART 5A

In this question i converted BGR to HSV and Ycbcr. Sources: link1

[17]:
```
def convert_to_hsv(image):
    '''
    Converts the color space of the input image to the HSV color space.

    Parameters:
        image (numpy.ndarray): The input image.

    Returns:
        numpy.ndarray: The result image.
```

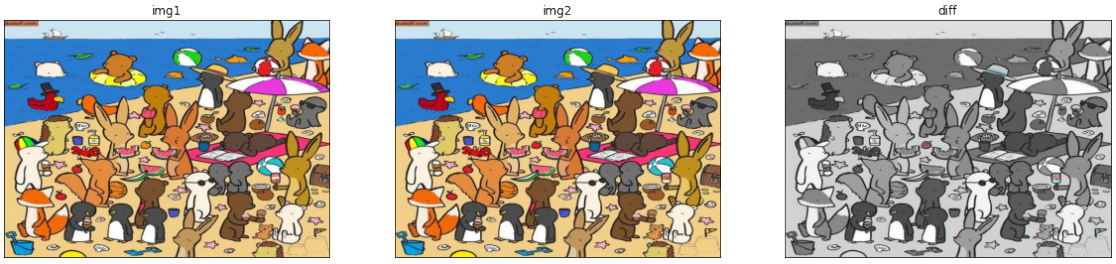
```
    '''

    out_img = image.copy()

    #Write your code here

    out_img=cv2.cvtColor(out_img,cv2.COLOR_BGR2HSV)

    return out_img

def convert_to_ycbcr(image):
    '''
    Converts the color space of the input image to the YCbCr color space.

    Parameters:
        image (numpy.ndarray): The input image.

    Returns:
        numpy.ndarray: The result image.
    '''

    out_img = image.copy()

    #Write your code here

    out_img=cv2.cvtColor(out_img,cv2.COLOR_BGR2YCrCb)

    return out_img
```



6 PART 5B

I have done this implementation according to the session 12 of vision class.

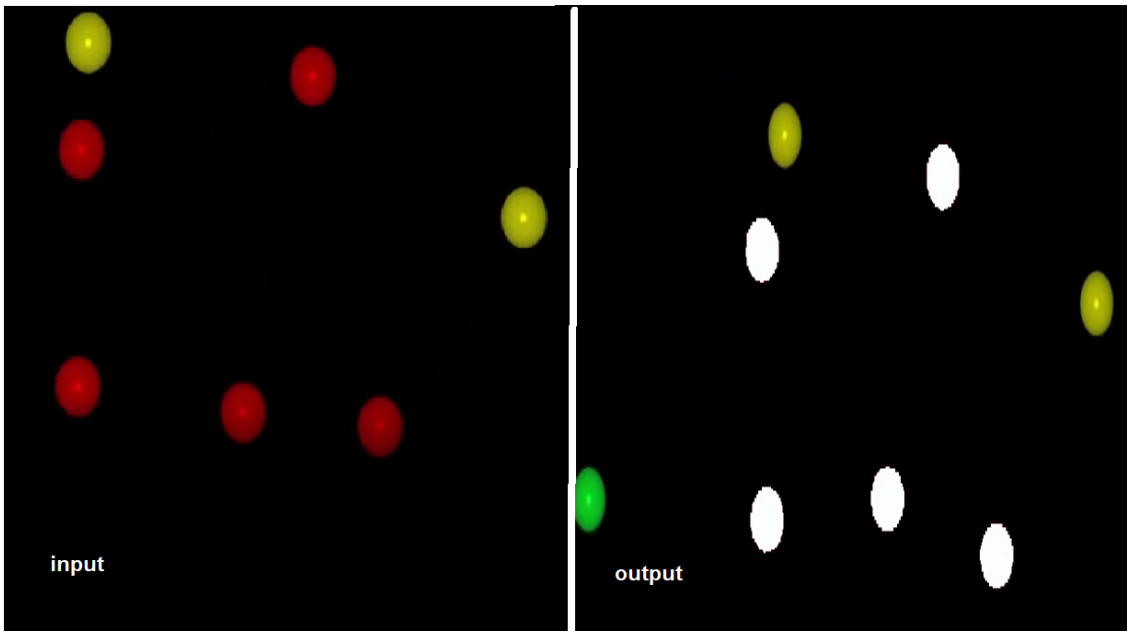
[130]:
```python
def get_dif(image1, image2):
    '''
    Creates a new image that differences between two input images are shown.

    Parameters:
        image1 (numpy.ndarray): The first input image.
        image2 (numpy.ndarray): The second input image.

    Returns:
        numpy.ndarray: The result difference image.
    '''

    out_img = image1.copy()

    #Write your code here
    # convert to 3 equal channels
    out_img = cv2.merge((out_img, out_img, out_img))
    for i in range(image1.shape[0]):
        for j in range(image1.shape[1]):
            out_img[i,j]=out_img[i,j]*[1,0,0]

    out_img2 = image2.copy()
    out_img2 = cv2.merge((out_img2, out_img2, out_img2))
    for i in range(image2.shape[0]):
        for j in range(image2.shape[1]):
            out_img2[i,j]=out_img2[i,j]*[0,1,1]

    result= out_img2+out_img
    return result
```



7 PART 6

In this question ,I have Converted image to HSV plane using cvtColor() function then Extract red color from it using inRange() function.
Sources: Link1,link2,link3

[132]:
```
def process_frame(frame):
    '''
    Converts red circles in the input image to white circles.

    Parameters:
        frame (numpy.ndarray): The input frame.

    Returns:
        numpy.ndarray: The result output frame.
    '''

    result = frame.copy()

    #Write your code here
    img_hsv=cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

    # lower mask (0-10)
    lower_red = np.array([0,50,50])
    upper_red = np.array([10,255,255])
    mask0 = cv2.inRange(img_hsv, lower_red, upper_red)

    # upper mask (170-180)
    lower_red = np.array([170,50,50])
    upper_red = np.array([180,255,255])
    mask1 = cv2.inRange(img_hsv, lower_red, upper_red)

    # join my masks
    mask = mask0+mask1

    # set my output img to zero everywhere except my mask
    output_img = frame.copy()
    output_img[np.where(mask==255)] = 255

    # or your HSV image, which I *believe* is what you want


    return output_img
```



8 PART 7

this question has diffrent steps to implement:

1. Color to grayscale
2. Spatial derivative calculation
3. Structure tensor setup
4. Harris response calculation
5. Non-maximum suppression
   I implemented them separately.

[134]:
```python
def gradient_x(img):
    ##Sobel operator kernels
    kernel_x = np.array([[-1, 0, 1],[-2, 0, 2],[-1, 0, 1]])
    return cv2.filter2D(img,-1,kernel_x)

def gradient_y(img):
    ##Sobel operator kernels
    kernel_y = np.array([[1, 2, 1], [0, 0, 0], [-1, -2, -1]])
    return cv2.filter2D(img,-1,kernel_y)

def gaussian_kernel(size, sigma=1):
    gaussian = None
    gaussian = np.zeros((size,size), np.float)
    coe=1/(2*((np.pi)*(sigma**2)))
    co2_std=2*(sigma**2)
    for i in range(size):
```

```
        for j in range(size):
            gaussian[i,j]+=(coe*np.exp((-(i**2+j**2)/co2_std)))
    return gaussian
```

Implement this function for Harris detection.

[135]:
```
def harris_points(image):
    '''
    Gets corner points by applying the harris detection algorithm.

    Parameters:
        image (numpy.ndarray): The input image.

    Returns:
        numpy.ndarray: The result image.
    '''

    out_img = image.copy()

    #Write your code here
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    I_x = gradient_x(np.float32(gray))
    I_y = gradient_y(np.float32(gray))
    kernel_size=3
    sigma=1
    kernel=gaussian_kernel(kernel_size,sigma)
    img_smoothed = cv2.filter2D(np.float32(image),-1,np.float32(kernel))

    Ixx = cv2.filter2D(np.float32(I_x**2),-1,np.float32(kernel))
    Ixy = cv2.filter2D(np.float32(I_y*I_x),-1,np.float32(kernel))
    Iyy = cv2.filter2D(np.float32(I_y**2),-1,np.float32(kernel))

    k=0.05

    # determinant
    detA = Ixx * Iyy - Ixy ** 2
    # trace
    traceA = Ixx + Iyy

    harris_response = detA - k * (traceA ** 2)

    print(np.argmax(harris_response))
    for rowindex, response in enumerate(harris_response):
        for colindex, r in enumerate(response):
            if r > 72180:
                # this is a corner
                cv2.circle(out_img,(rowindex, colindex),2,(255,0,0), -1)
```

```
                #out_img[rowindex, colindex]=[255,0,0]



    return out_img
```