NAME : EDWARD **Course : Computer Vision**
NIM : 2201741971 **Course Code : COMP7116**
CLASS : LB-08 **Faculty / Department : School of Computer Science**

1. **Point processing transformations (25%)**

These are four-point processing transformations:

- Thresholding
- Gray-scaling
- Negative transformation
- Histogram Equalization

a. Please explain the basic concepts of above point processing transformations. Complete your explanations with the mathematical equations of the transformations.

Answer:

a) Thresholding

Thresholding creates binary images from grey-level ones by turning all pixels below some threshold to zero and all pixels about that threshold to one.

If g(x, y) is a threshold version of f(x, y) at some global threshold T

$$g(x, y) = 1 \text{ if } f(x, y) \geq T$$

$$0 \text{ otherwise}$$

b) Gray-scaling

i) Logarithmic Transformations

The general form of the log transformation is

$$s = c * \log(1 + r)$$

The log transformation **maps a narrow range of low input grey level values into a wider range of output values.**

Usually set c to 1 .Grey levels must be in the range [0.0, 1.0].

ii) Power Law Transformation

Power law transformations have the following form

$$s = c * r^{\gamma}$$

Map a narrow range of dark input values into a wider range of output values or vice versa. Varying γ gives a whole family of curves.

c) Negative Transformation

Negative images are useful for enhancing white or grey detail embedded in dark regions of an image. It basically works by reversing the intensities of the image points. The formula:

$$s = \text{maxintensity} - r$$

$$s = (L - 1) - r$$

NAME : EDWARD **Course : Computer Vision**
NIM : 2201741971 **Course Code : COMP7116**
CLASS : LB-08 **Faculty / Department : School of Computer Science**

Here **L will be taken as 256** and **256-1 =255 gives us the max intensity**. **r is the intensity at the point to be considered**.

So, what happens is that, **the lighter pixels become dark** and **the darker picture becomes light**. And it results in image negative.

d) Histogram Equalization

Histogram Equalization is a computer image processing technique **used to improve contrast in images**. It accomplishes this by effectively spreading out the most frequent intensity values, i.e. **stretching out the intensity range of the image**.

This method usually increases the global contrast of images when its usable data is represented by close contrast values. This **allows for areas of lower local contrast to gain a higher contrast**.

Consider a discrete grayscale image {x} and **let n_i be the number of occurrences of Gray level i**. The probability of an occurrence of a pixel of level i in the image is

$$p_x(i) = p(x = i) = \frac{n_i}{n}, \quad 0 \leq i < L$$

L being the total number of gray levels in the image (typically 256), **n being the total number of pixels in the image**, pixel value i, normalized to [0,1].

Let define cumulative distribution function corresponding to px as

$$cdf_x(i) = \sum_{j=0}^{i} p_x(x = j)$$

which is also the image's accumulated normalized histogram.

Would like to create a transformation of the form y = T(x) to produce a new image {y}, with a flat histogram. Such an image would have a linearized cumulative distribution function (CDF) across the value range.

$$cdf_y(i) = iK$$

for some constant K. The properties of the CDF allow to perform such a transform (see Inverse distribution function); it is defined as:

$$cdf_y(y') = cdf_y(T(k)) = cdf_x(k)$$

where k is in the range [0,L]). Notice that T maps the levels into the range [0,1], since used a normalized histogram of {x}. **To map the values back into their original range, the following simple transformation needs to be applied on the result**:

$$y' = y \cdot (\max\{x\} - \min\{x\}) + \min\{x\}$$

NAME : EDWARD **Course : Computer Vision**
NIM : 2201741971 **Course Code : COMP7116**
CLASS : LB-08 **Faculty / Department : School of Computer Science**

b. Furthermore, please download the image from this URL: https://qrgo.page.link/YNzeX The first step is you load the image (lena_color.gif) and implement all above point processing transformations. It can be done by using OpenCV library.

Answer:

Load The Image:

```
In [25]: import numpy as np
         import cv2
         from matplotlib import pyplot as plt

         #Because the image file format is still in gif, then first I will change the file format to jpg in the following way:
         gif = cv2.VideoCapture('lena_color.gif')
         ret, frame = gif.read()
         cv2.imwrite('lena_color.jpg', frame)

         #After that, I use Test if the image can be read and shown after convert the format
         img = cv2.imread("lena_color.jpg")
```

a) Thresholding

```
In [26]: # 1) THRESHOLDING
           # Call the Threshold function, I use Binary Threshold
         ret,thresh1 = cv2.threshold(img2,127,255,cv2.THRESH_BINARY)
         plt.imshow(thresh1)
         plt.title('Lena Thresholding')
         plt.xticks([]), plt.yticks([])  # Hides the graph ticks and x / y axis
         plt.show()
```

b) Gray-scaling

```
In [27]: # 2) GrayScalling
           #Convert Image to Grayscale
         gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
         plt.imshow(gray)
         plt.title('Lena GrayScale')
         plt.xticks([]), plt.yticks([])  # Hides the graph ticks and x / y axis
         plt.imshow(gray, cmap='gray', vmin = 0, vmax = 255)

Out[27]: <matplotlib.image.AxesImage at 0x2322766df48>
```

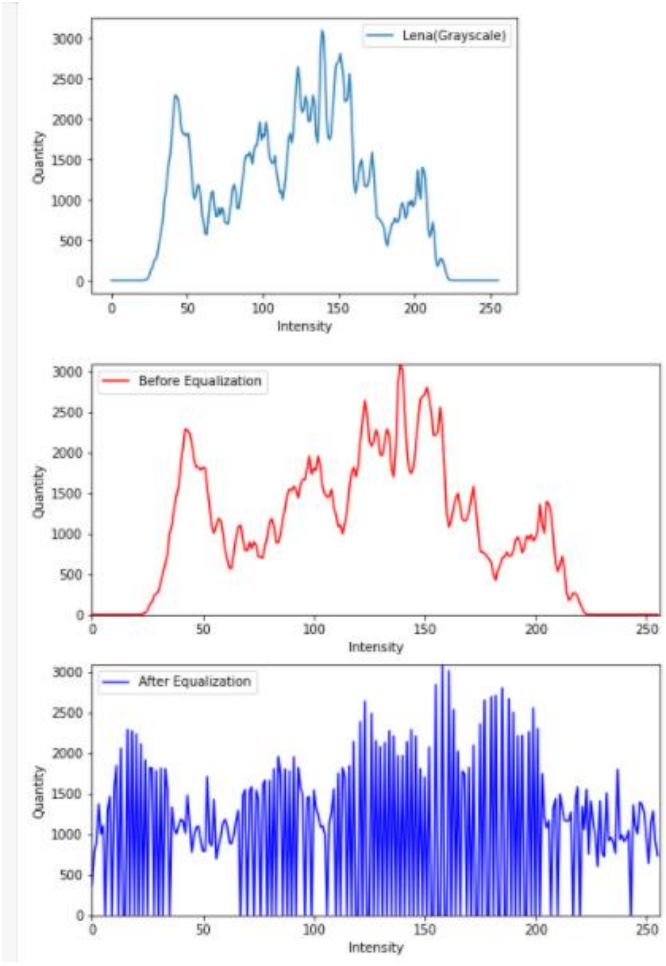
c) Negative Transformation

```
In [28]: # 3) Negative Transformation
           #Reverse the image with negative
         img_neg = 1 - img2
         plt.imshow(img_neg)
         plt.title('Lena Negative Transformation')
         plt.xticks([]), plt.yticks([])  # Hides the graph ticks and x / y axis
         plt.show()
```

NAME : EDWARD **Course : Computer Vision**
NIM : 2201741971 **Course Code : COMP7116**
CLASS : LB-08 **Faculty / Department : School of Computer Science**

d) Histogram Equalization

In [29]:
```
# 4) Histogram Equalization
#Getting image Height
gray_height = img.shape[0]
#Getting image width
gray_width = img.shape[1]

#Count Pixels for Each Intensities
gray_counter = np.zeros(256, dtype=int)

for i in range(gray_height):
    for j in range(gray_width):
        #Pixel Intensities on Coordinates(i,j)
        gray_counter[gray[i][j]] += 1
plt.figure(1)
plt.plot(gray_counter, label = 'Lena(Grayscale)')
plt.legend(loc='upper right')
plt.ylabel('Quantity')
plt.xlabel('Intensity')
plt.show()

# Apply Histogram Equalization
equ = cv2.equalizeHist(gray)
equ_counter = np.zeros(256, dtype=int)
for i in range(gray_height):
    for j in range(gray_width):
        equ_counter[equ[i][j]] += 1

# Plotting with matplotlib
plt.figure(1, figsize=(8, 8))

#Plot Before EqualizeHist
plt.subplot(2, 1, 1)
plt.plot(gray_counter, 'r', label='Before Equalization')
plt.legend(loc='upper left')
plt.ylabel('Quantity')
plt.xlabel('Intensity')
plt.axis([0, 256, 0, gray_counter.max() if (gray_counter.max() > equ_counter.max()) else equ_counter.max()])

#Plot After EqualizeHist
plt.subplot(2, 1, 2)
plt.plot(equ_counter, 'b', label='After Equalization')
plt.legend(loc='upper left')
plt.ylabel('Quantity')
plt.xlabel('Intensity')
plt.axis([0, 256, 0, gray_counter.max() if (gray_counter.max() > equ_counter.max()) else equ_counter.max()])
plt.show()
```



NAME : EDWARD **Course : Computer Vision**
NIM : 2201741971 **Course Code : COMP7116**
CLASS : LB-08 **Faculty / Department : School of Computer Science**

2. Convolution (25%)

Please download the image from this URL: https://qrgo.page.link/Vt7Ag The first step is you load the image (opera_house.jpg) and convert it into grayscale. It can be done by using Pillow library. And the two 3x3 kernels

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

1	0	-1
1	0	-1
1	0	-1

Please find the solution for: Convolution operation without border padding

a. Implement the above calculations using Numpy library.

Answer:

Load Image:

```
In [32]: from PIL import Image,ImageOps
         import matplotlib.pyplot as plt
         import numpy as np

         opera= Image.open('Sydney-Opera-House.jpg')
         plt.figure(figsize = (10,10))
         plt.xticks([]), plt.yticks([])
         plt.imshow(opera)
```

Convert to Grayscale:

```
In [34]: # converting to gray scale
         gray_img = ImageOps.grayscale(image)
         plt.figure(figsize = (10,10))
         plt.xticks([]), plt.yticks([])
         plt.imshow(gray_img,cmap = "gray")
```

Implementation of Convolution with 3x3 kernels:

```
In [35]: #a. Implement the above calculations using Numpy library.

#calculatint the final input shape
def getOutputShape(x,kernel_size):
    kernel_size = 3
    new_width = x[0] - kernel_size + 1
    new_height = x[1] - kernel_size + 1
    print("Input height : ",x[0]," Output height : ",new_width)
    print("Input width : ",x[1]," Output width : ",new_height)
    return new_width,new_height
getOutputShape(gray_img.size,3)

### convolution function
def conv(img,kernel):
    #converting image to numpy array.
    img = np.asarray(img)

    #calculate the new shape
    kernel_size = kernel.shape[0]
    new_shape = getOutputShape(img.shape,kernel_size)

    #output image array.
    out = np.zeros(shape = new_shape)

    for x in range(out.shape[0]):
        for y in range(out.shape[1]):
            sum = 0
            for i in range(kernel.shape[0]):
                for j in range(kernel.shape[1]):
                    sum += img[x+i][y+j] * kernel[i][j]
            # the below lines does 2 things
            # mods sum remember only +ve values of pixels are allowed
            # it also takes core that the pixel
            sum = np.abs(sum)
            if sum > 255.:
                sum = 255.0
            out[x,y] = sum
    return np.round(out)
```

NAME : EDWARD **Course : Computer Vision**
NIM : 2201741971 **Course Code : COMP7116**
CLASS : LB-08 **Faculty / Department : School of Computer Science**

b. Convert above Numpy array to images and show the results. The implementation can use Pillow library.

Answer:

```
In [26]: #b. Convert above Numpy array to images and show the results. The implementation can use Pillow library

         # convolution image 1.
         kernel_1 = np.ones((3,3)) * 1/9
         output_image_1 = conv(gray_img,kernel_1)
         plt.figure(figsize = (10,10))
         plt.title('Convolution Image 1')
         plt.xticks([]), plt.yticks([])
         plt.imshow(output_image_1,cmap = "gray")


         kernel_2 = np.array([[-1,0,1],[-1,0,1],[-1,0,1]])
         output_image_2 = conv(gray_img,kernel_2)
         plt.figure(figsize = (10,10))
         plt.title('Convolution Image 2')
         plt.xticks([]), plt.yticks([])
         plt.imshow(output_image_2,cmap = "gray")
```

```
Out[26]: <matplotlib.image.AxesImage at 0x1549d29ff48>
```

NAME : EDWARD **Course : Computer Vision**
NIM : 2201741971 **Course Code : COMP7116**
CLASS : LB-08 **Faculty / Department : School of Computer Science**

3. Harris corner detector (25%)

Please answer below questions related to Harris corner detection as follow:

a. Explain the basic idea to detect the Corner Points in an image

Answer:

Corners are mainly formed by the combination of two or more edges. These corners may or may not define the boundary of an image. **Harris corner detection algorithm is found out by calculating each pixel's gradient**. **If the absolute gradient values in both the directions are great, then consider the pixel as a corner**.

This method is followed by mathematical formulas. By taking into considerations of the differential of the corner score with respect to direction directly, **Harris and Stephens improved the Moravec's corner detector instead of using shifted patches**.

The accuracy of the extracted points will affect the accuracy of camera calibration parameters and its post processing problems of image quality.

b. If M is second moment matrix of image derivatives, please explain mathematically how to classify image points based on M Matrix!

Answer:

- **Step 1 (Defines the Derivatives):**

$$I_x = \frac{\partial I}{\partial x} \qquad I_y = \frac{\partial I}{\partial y} \qquad I_x I_y = \frac{\partial I \partial I}{\partial x \partial y}$$

- **Step 2 (Find out the product of derivatives of each pixel):**

$$I_x^2 = I_x \text{ X } I_x \qquad I_y = I_y \text{ X } I_y \qquad I_{xy} = I_x \text{ X } I_y$$

- **Step 3 (Calculate the covariance matrix M):**

$$M = \sum W(x,y) \begin{bmatrix} Ix^2 & IxIy \\ IxIy & Iy^2 \end{bmatrix}$$

W(x,y) is calculated by a windows function, that is:

$$E(u,v) = \sum W(x,y)\,[I(x+u,y+v) - I(x,y)]$$

Where:

I(x+u,y+v) is known as Shifted Intensity

I(x,y) is known as the Intensity

The Above Equation can be simplified as:

$$E(u,v) \approx [u \quad v]\, M \begin{bmatrix} u \\ v \end{bmatrix}$$

From that, we can conclude that:

$$M = E(u,v)\,[u \quad v] \begin{bmatrix} u \\ v \end{bmatrix}$$

NAME : EDWARD **Course : Computer Vision**
NIM : 2201741971 **Course Code : COMP7116**
CLASS : LB-08 **Faculty / Department : School of Computer Science**

c. Harris corner detector algorithm is based on above idea, together with additional step to enhance the detector. Please explain each step of Harris corner detector based on its original paper: C.Harris and M.Stephens. "A Combined Corner and Edge Detector", Proceedings of the 4th Alvey Vision Conference: pages 147—151, 1988

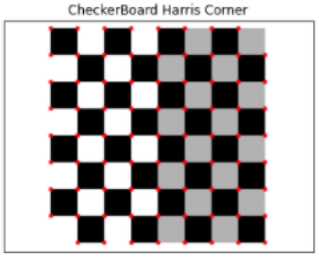
Answer:

- **Step 1 (Convert to grayscale from the original image)**
  - ➔ **because we need RGB channel for corner detection**, Moreover, it **reduces the size of information in the image** and it **helps to make the process fast**.
- **Step 2 (Apply Sobel operator)**
  - ➔ **To find the x and y gradient values for every pixel** in the grayscale image.
- **Step 3(Calculate Harris Value)**
  - ➔ **Each pixel in the grayscale image**, compute corner strength function.
- **Step 4 (Find all pixels that exceed a certain threshold and are the local maxima within a certain window)**
  - ➔ **To prevent redundant dupes of features**.
- **Step 5 (Compute Feature Descriptor)**
  - ➔ **Each pixel that meets the criteria in Step 5**, compute feature descriptor.

d. Please download the image from this URL: https://qrgo.page.link/jNBDK The first step is you load the image (checkerboard_101.png) and implement Harris corner detector. It can be done by using OpenCV library.

Answer:

In [27]:
```
import matplotlib.pyplot as plt
import numpy as np
import cv2

image = cv2.imread('checkerboard_101.png')
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
gray = cv2.cvtColor(image, cv2.COLOR_RGB2GRAY)
gray = np.float32(gray)
dst = cv2.cornerHarris(gray, 2, 3, 0.04)
dst = cv2.dilate(dst,None)

# Define thresh hold (thresh value inversely proportional to corner visibility )
thresh = 0.01*dst.max()

# Create copy of rgb image to draw corners on it.
corner_image = np.copy(image)

# Iterate through all the corners and draw them on the image
for j in range(0, dst.shape[0]):
    for i in range(0, dst.shape[1]):
        if(dst[j,i] > thresh):
            # image, center pt, radius, color, thickness
            cv2.circle( corner_image, (i, j), 1, (255,0,0), 10)


plt.imshow(corner_image)
plt.title('CheckerBoard Harris Corner')
plt.xticks([]), plt.yticks([])  # Hides the graph ticks and x / y axis
plt.show()
```



NAME : EDWARD **Course : Computer Vision**
NIM : 2201741971 **Course Code : COMP7116**
CLASS : LB-08 **Faculty / Department : School of Computer Science**

4. **Canny edge detector (25%)**

Canny edge detector was developed by John F. Canny in 1986 and uses a multi-stage algorithm to detect multiple edges in an image. Briefly describe the procedures involved in each stage, implement and apply it to this image from this URL: https://qrgo.page.link/HVHsH

a. Noise Reduction. For this purpose, what kind of filter is used? And why is this stage very important to do in the first place? Explain it!

Answer:

What kind of filter:

I am using a **Gaussian filter**.

Why this stage very important:

The edge detection algorithm is very sensitive to image noise. As a result, it is essential to filter it to prevent detecting false.

Implementation:

- **Apply Gaussian Blurring**

```
In [20]: def GaussianBlur(image):
             image = cv2.GaussianBlur(image, (3, 3), 0)
             return image
```

- **Result**

NAME : EDWARD **Course : Computer Vision**
NIM : 2201741971 **Course Code : COMP7116**
CLASS : LB-08 **Faculty / Department : School of Computer Science**

b. Finding Intensity Gradient of the Image. What does this stage produce? And what filters do you use to achieve it?

Answer:

The Stages Produce:

To detect the change of pixels' intensity on both X-axis and Y-axis The gradient detects the direction and the intensity of the edge by using edge-detection operators.

What kind of filter:

I used the **Sobel Filter**.

Implementation:

```
In [29]: def SobelFilter(image):
             image = Grayscale(GaussianBlur(image))
             plt.imshow(image, cmap = plt.get_cmap('gray'))
             plt.title('Gaussian Blur Result')
             plt.xticks([]), plt.yticks([])  # Hides the graph ticks and x / y axis
             plt.show()
             convolved = np.zeros(image.shape)
             G_x = np.zeros(image.shape)
             G_y = np.zeros(image.shape)
             size = image.shape
             kernel_x = np.array(([-1, 0, 1], [-2, 0, 2], [-1, 0, 1]))
             kernel_y = np.array(([-1, -2, -1], [0, 0, 0], [1, 2, 1]))
             for i in range(1, size[0] - 1):
                 for j in range(1, size[1] - 1):
                     G_x[i, j] = np.sum(np.multiply(image[i - 1 : i + 2, j - 1 : j + 2], kernel_x))
                     G_y[i, j] = np.sum(np.multiply(image[i - 1 : i + 2, j - 1 : j + 2], kernel_y))

             convolved = np.sqrt(np.square(G_x) + np.square(G_y))
             convolved = np.multiply(convolved, 255.0 / convolved.max())

             angles = np.rad2deg(np.arctan2(G_y, G_x))
             angles[angles < 0] += 180
             convolved = convolved.astype('uint8')
             return convolved, angles
```

Result:

NAME : EDWARD **Course : Computer Vision**
NIM : 2201741971 **Course Code : COMP7116**
CLASS : LB-08 **Faculty / Department : School of Computer Science**

c. Non-maximum Suppression. What is meant by this stage?

Answer:

Meant by this stage:

Is thinning technique. Once the edge direction is calculated we can trace them in the image. When looking at a pixel there are only four directions when checking the surrounding pixels in a reverse trigonometric direction:

- 0 degrees => horizontal direction
- 45 degrees => positive diagonal
- 90 degrees => vertical direction
- 135 degrees => negative diagonal

Implementation:

```
In [22]: def non_maximum_suppression(image, angles):
             size = image.shape
             suppressed = np.zeros(size)
             for i in range(1, size[0] - 1):
                 for j in range(1, size[1] - 1):
                     if (0 <= angles[i, j] < 22.5) or (157.5 <= angles[i, j] <= 180):
                         value_to_compare = max(image[i, j - 1], image[i, j + 1])
                     elif (22.5 <= angles[i, j] < 67.5):
                         value_to_compare = max(image[i - 1, j - 1], image[i + 1, j + 1])
                     elif (67.5 <= angles[i, j] < 112.5):
                         value_to_compare = max(image[i - 1, j], image[i + 1, j])
                     else:
                         value_to_compare = max(image[i + 1, j - 1], image[i - 1, j + 1])

                     if image[i, j] >= value_to_compare:
                         suppressed[i, j] = image[i, j]
             suppressed = np.multiply(suppressed, 255.0 / suppressed.max())
             return suppressed
```

Result:

NAME : EDWARD **Course : Computer Vision**
NIM : 2201741971 **Course Code : COMP7116**
CLASS : LB-08 **Faculty / Department : School of Computer Science**

d. Hysteresis Thresholding. What is the purpose of this stage? How do you perform double thresholding to achieve this goal?

Answer:

How I Perform Double Thresholding:

I am filtering the image. This is **accomplished by sorting all pixels into 2 categories (weak and strong)**. **If a pixel's value is greater than our high threshold, we consider it a strong one**. **If the pixel is brighter than the low threshold but darker than the high threshold, we consider the pixel to be weak**. **Pixels that are smaller than the low threshold are being ignored**.

Purpose of Edge tracking by hysteresis:

Now our image consists of weak pixels and strong pixels. We **must make all true edge pixels strong**(**even if now they are weak**) and **all the non-edge pixels should be transformed into zeros**. A weak pixel is transformed into a strong one only if it has a strong neighbour on any of the 8 directions.

Implementation:

In [23]:
```
def double_threshold_hysteresis(image, low, high):
    weak = 50
    strong = 255
    size = image.shape
    result = np.zeros(size)
    weak_x, weak_y = np.where((image > low) & (image <= high))
    strong_x, strong_y = np.where(image >= high)
    result[strong_x, strong_y] = strong
    result[weak_x, weak_y] = weak
    dx = np.array((-1, -1, 0, 1, 1, 1, 0, -1))
    dy = np.array((0, 1, 1, 1, 0, -1, -1, -1))
    size = image.shape

    while len(strong_x):
        x = strong_x[0]
        y = strong_y[0]
        strong_x = np.delete(strong_x, 0)
        strong_y = np.delete(strong_y, 0)
        for direction in range(len(dx)):
            new_x = x + dx[direction]
            new_y = y + dy[direction]
            if((new_x >= 0 & new_x < size[0] & new_y >= 0 & new_y < size[1]) and (result[new_x, new_y]  == weak)):
                result[new_x, new_y] = strong
                np.append(strong_x, new_x)
                np.append(strong_y, new_y)
    result[result != strong] = 0
    return result
```

Result: