Hw4_Color edge detection

- Student's name : 簡茂芳
- registration number: 409410035

- Data due: 6/5
- Data handed in: 6/5

A. Technical description

一，

本次作業我使用 Sobel 算子來實作。下面大致列出其優缺點:

【Sobel 算子優缺點】

優點：

1. 簡單高效：Sobel 算子使用簡單的卷積操作，計算速度較快。
2. 邊緣定位準確：Sobel 算子可以檢測出圖像中的邊緣，並提供較好的邊緣定位信息。

缺點：

1. 對噪音敏感：Sobel 算子對噪音比較敏感，噪音會導致邊緣偵測結果產生誤差。
2. 邊緣粗糙：Sobel 算子會產生較粗的邊緣，對於某些細微的邊緣特徵可能無法準確捕捉。

二，

由於彩色圖片有多個通道，若分別對各個通道做 sobel 在做合併的話，可能會產生較多噪聲。所以，我在看了相關的網站後，決定將彩色圖像轉換

為灰度圖像，即使用灰度轉換方法(由公式: Gray = *R*0.299 + G*0.587 + B*0.114*)，將彩色圖像轉換為灰度圖像，然後對灰度圖像應用 Sobel 算子進行邊緣檢測。

三，程式碼

1. 先讀圖片並詢問是否做模糊處理

```
79  if __name__=='__main__':
80      # Load color images
81      image1 = cv2.imread('baboon.png')
82      image2 = cv2.imread('peppers.png')
83      image3 = cv2.imread('pool.png')
84
85      apply_blur = input("Do you want to apply median blur (filter size is 7x7)? (y/n): ")
```

2. 處理 **Color edge detection**

```
88      # Perform edge detection on the color images
89      edges1 = color_edge_detection(image1, apply_blur)
90      edges2 = color_edge_detection(image2, apply_blur)
91      edges3 = color_edge_detection(image3, apply_blur)
```

color_edge_detection

```python
48  def color_edge_detection(image, apply_blur):
49      # Convert image to grayscale
50      gray = image[:, :, 0] * 0.299 + image[:, :, 1] * 0.587 + image[:, :, 2] * 0.114
51
52      if apply_blur.lower() == 'y':
53          # Apply median blur
54          blur_ksize = 7
55          blurred_image = median_blur(gray, blur_ksize)
56      else:
57          blurred_image = gray
58
59      # Apply Sobel operator for x-axis gradient
60      sobel_x = sobel_operator(blurred_image, 'x')
61
62      # Apply Sobel operator for y-axis gradient
63      sobel_y = sobel_operator(blurred_image, 'y')
64
65      # Compute gradient magnitude
66      magnitude = np.sqrt(sobel_x**2 + sobel_y**2)
67      # direction = np.arctan2(sobel_y, sobel_x)
68
69      return magnitude
```

首先，將圖片轉成灰階 (line 50)，再根據用戶是否做模糊處理，選 ”y” 的話，執行 line 53~55 ，反之，執行 line 57。處理完 magnitude 後傳回。

median_blur

```python
6   def median_blur(image, ksize=7):
7       # Get image dimensions
8       height, width = image.shape[:2]
9
10      # Pad the image(上下左右各 ksize//2 像素) to handle borders
11      padded_image = np.pad(image, ((ksize//2, ksize//2), (ksize//2, ksize//2)), mode='constant')
12
13      # Create a blank output image
14      output_image = np.zeros_like(image)
15
16      # Apply median blur
17      for i in range(height):
18          for j in range(width):
19              # patch 是指 image 中以 (i,j) 為中心的 ksize x ksize 區域
20              patch = padded_image[i:i+ksize, j:j+ksize]
21              median_value = np.median(patch)
22              output_image[i, j] = median_value
23
24      return output_image
```

將圖片做 padding 後，再 apply median blur (詳見註解)

sobel_operator

```python
def sobel_operator(image, axis):
    # Define Sobel kernels
    if axis == 'x':
        kernel = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])
    elif axis == 'y':
        kernel = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]])

    # Perform convolution
    filtered_image = np.zeros_like(image)
    height, width = image.shape
    for i in range(1, height-1):
        for j in range(1, width-1):
            # patch 是指 image 中以 (i,j) 為中心的 3x3 區域
            patch = image[i-1:i+2, j-1:j+2]
            filtered_value = np.sum(patch * kernel)
            filtered_image[i, j] = filtered_value

    return filtered_image
```

判斷對哪一個方向做 sobel_operator 後，選取對應的 kernal 進行卷積，並回傳。

3. 處理輸出

```python
    # Display original images and processed images
    fig, axs = plt.subplots(2, 3, figsize=(12, 8))
```

配置畫布大小 (2 x 3)

```python
    # Show original Image 1
    axs[0, 0].imshow(bgr_to_rgb(image1))
    axs[0, 0].set_title('Image 1')
    axs[0, 0].axis('off')

    # Show original Image 2
    axs[0, 1].imshow(bgr_to_rgb(image2))
    axs[0, 1].set_title('Image 2')
    axs[0, 1].axis('off')

    # Show original Image 3
    axs[0, 2].imshow(bgr_to_rgb(image3))
    axs[0, 2].set_title('Image 3')
    axs[0, 2].axis('off')

```

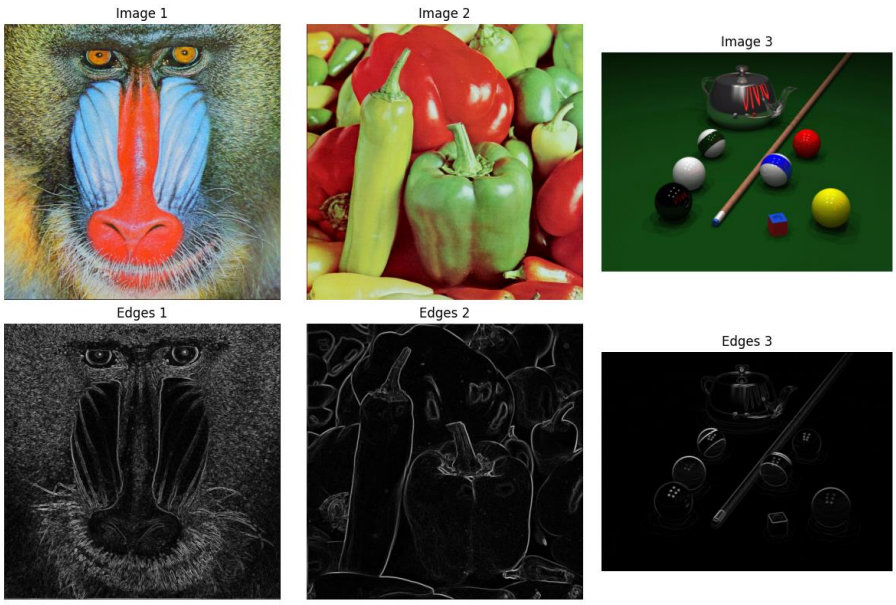
第一個 row 放原圖

```
110     # Show edges of Image 1
111     axs[1, 0].imshow(edges1, cmap='gray')
112     axs[1, 0].set_title('Edges 1')
113     axs[1, 0].axis('off')
114
115     # Show edges of Image 2
116     axs[1, 1].imshow(edges2, cmap='gray')
117     axs[1, 1].set_title('Edges 2')
118     axs[1, 1].axis('off')
119
120     # Show edges of Image 3
121     axs[1, 2].imshow(edges3, cmap='gray')
122     axs[1, 2].set_title('Edges 3')
123     axs[1, 2].axis('off')
124
125     # Adjust layout and display the plot
126     plt.tight_layout()
127     plt.show()
```

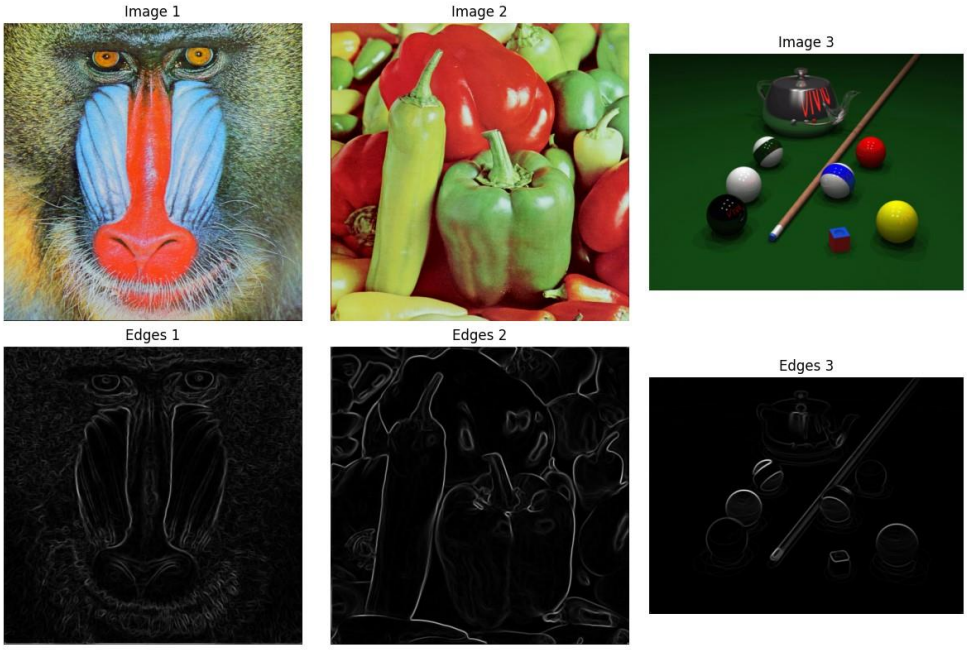
第二個 row 放原圖的 edge，並將圖片 show

B. Experimental results

No apply median blur



apply median blur



C. Discussions

在進行圖像處理的過程中，我選擇將彩色圖片轉換為灰階圖像，並應用 Sobel 算子進行邊緣檢測，提取圖像中的重要邊緣信息。

首先，將彩色圖片轉換為灰階圖像的目的是為了將彩色信息轉化為亮度值信息。這樣可以使我們更專注於圖像中的邊緣特徵，並簡化計算過程。轉換後的灰階圖像保留了原始圖像中的亮度差異，並使得邊緣在亮度上更加明顯。

接下來，我使用 Sobel 算子進行邊緣檢測。Sobel 算子可以根據圖像中像素的灰度變化情況計算出每個像素的梯度值。通過計算梯度，我可以得到圖像中的邊緣強度和方向信息。應用 Sobel 算子後，圖像中的邊緣被突出顯示，使我們能夠更清晰地看到物體的輪廓和邊界。

另外，我也嘗試了使用中值模糊（median blur）和不使用中值模糊的方式。由於中值模糊，可以通過將每個像素的值替換為其鄰近像素值的中值，來減少圖像中的噪點。對於某些圖像，如撞球圖的點和 baboon 的紋路，可能存在較多的噪點或細微變化，這些噪點或細節可能會對邊緣檢測結果產生干擾。

經過實驗比較，我發現在不使用中值模糊的情況下，邊緣檢測結果受到噪點的干擾，可能會產生一些雜散的邊緣，而且細微的紋理細節也可能被視為邊緣。而在應用了中值模糊後，噪點得到了一定程度的去除，圖像變得更加平滑，邊緣檢測結果也更加乾淨和精確。特別是在撞球圖的點和 baboon 的紋路中，經過中值模糊處理後，噪點被消除，紋理變得更加平滑，讓邊緣更加突出。

總結來說，將彩色圖片轉為灰階並應用 Sobel 算子進行邊緣檢測是一個有效的方法，可以從圖像中提取出重要的邊緣信息。同時，使用中值模糊可以在一定程度上去除噪點，使邊緣檢測結果更加準確和清晰。

D. References and Appendix

1. 影像邊緣偵測

   https://steam.oxxostudio.tw/category/python/ai/opencv-edge-detection.html

2. Edge Detection Using OpenCV

   https://learnopencv.com/edge-detection-using-opencv/