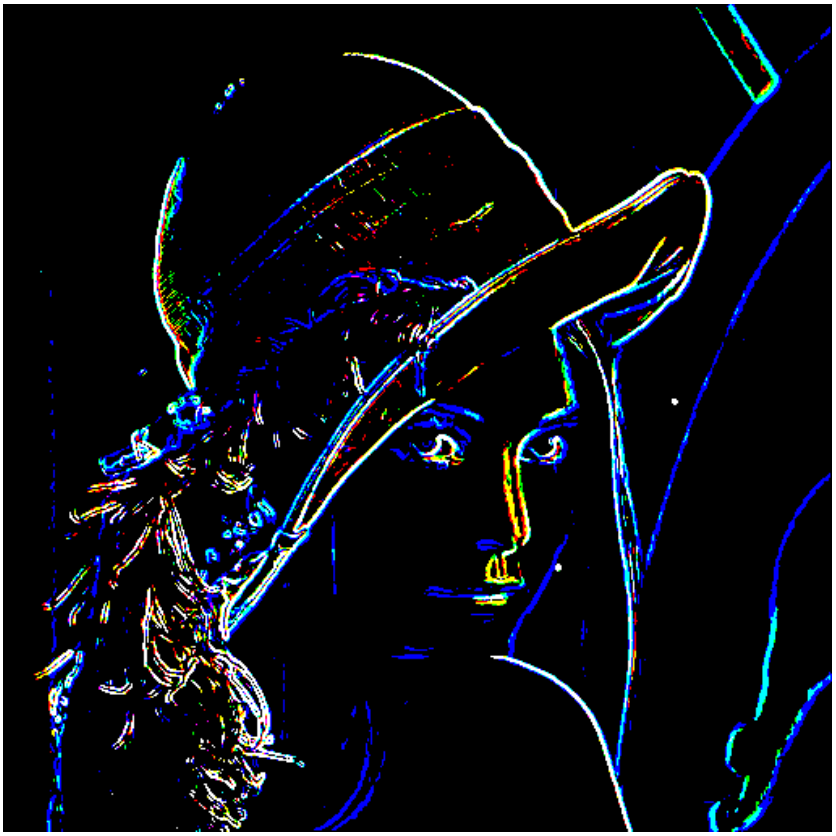
數位影像處理 DIP Homework Chapter 6_1

顏志憲 312512049

1. **Please use a gradient computed in RGB color vector space to detect the edge for the image, 'lenna-RGB.tif' as Fig. 6.44(b) in pp.453. Please describe your method, procedures, final gradient image and print out the source code? (3X10=40)**

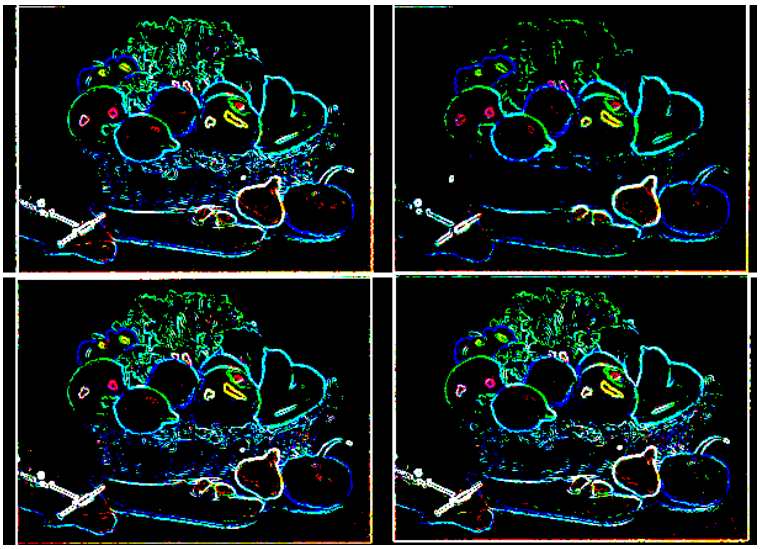


程式實現影像的邊緣檢測，結合了 Sobel 算子進行梯度計算，並透過 OpenCV 和 Matplotlib 處理及視覺化輸出。程式的設計主要包含幾個核心部分，從影像載入開始，經過邊緣檢測處理，最後輸出結果。首先會接收一個檔案路徑，並根據檔案副檔名來辨別影像的格式。如果檔案為 GIF 格式，則會逐幀提取所有影格並轉換為 RGB 格式，否則則直接載入靜態影像並轉為 RGB 色彩空間。這種設計保證了程式的通用性，適用於多種影像類型。在邊緣檢測的核心部分，程式使用 Sobel 算子來計算影像的梯度分量。Sobel 算子分別計算水平方向（x）和垂直方向（y）的梯度，再通過平方和開平方得到梯度幅值。接著，對梯度幅值進行正規化，將其轉換為 8 位影像以便後續處理。完成梯度計算後，程式會透過閾值分割來生成二值化的邊緣影像，強調出梯度幅值超過指定閾值的區域。這樣的處理方式能有效提取出影像中的邊緣特徵。梯度影像的結果顯示了影像中亮度快速變化的區域，這些區域通常對應於物體的輪廓或紋理邊緣。

```python
# !/usr/bin/env python3
import os
from PIL import Image
import cv2
import numpy as np
import matplotlib.pyplot as plt

class EdgeDetection:
    def __init__(self, file_path, show=True, save=True):
        self.file_path = file_path
        self.frames = []
        self.image_type = None
        self.show = show
        self.save = save
        self.main()

    def detect_file_type(self):
        _, ext = os.path.splitext(self.file_path)
        self.image_type = ext.lower()
        if self.image_type not in ['.gif', '.tif', '.png', '.jpg', '.jpeg']:
            raise ValueError("Unsupported file type! Supported types: .gif, .tif, .png, .jpg, .jpeg")

    def load_image(self):
        if self.image_type == '.gif':
            gif = Image.open(self.file_path)
            while True:
                frame = np.array(gif.convert("RGB"))
                self.frames.append(frame)
                try:
                    gif.seek(gif.tell() + 1)
                except EOFError:
                    break
        else:
            image = cv2.imread(self.file_path)
            if image is None:
                raise ValueError("Failed to load the image!")
            self.frames = [cv2.cvtColor(image, cv2.COLOR_BGR2RGB)]

    def compute_gradient(self, image):
        sobel_x = cv2.Sobel(image, cv2.CV_64F, 1, 0, ksize=3)
        sobel_y = cv2.Sobel(image, cv2.CV_64F, 0, 1, ksize=3)
        gradient_magnitude = np.sqrt(sobel_x**2 + sobel_y**2)
        return cv2.normalize(gradient_magnitude, None, 0, 255, cv2.NORM_MINMAX).astype(np.uint8)

    def detect_edges(self, threshold=50):
        edge_results = []
        for frame in self.frames:
            gradient_magnitude = self.compute_gradient(frame)
            _, edge_image = cv2.threshold(gradient_magnitude, threshold, 255, cv2.THRESH_BINARY)
            edge_results.append(edge_image)
        return edge_results
```

```python
def visualize_results(self, edge_results):
    for i, (original, edge) in enumerate(zip(self.frames, edge_results)):
        plt.figure(figsize=(12, 6))
        plt.subplot(1, 2, 1)
        plt.title(f'Original Image')
        plt.imshow(original)
        plt.axis('off')

        plt.subplot(1, 2, 2)
        plt.title(f'Edge Detected Image')
        plt.imshow(edge, cmap='gray')
        plt.axis('off')

        plt.show()

def save_results(self, edge_results, output_prefix="edge_result"):
    output_prefix = f"{output_prefix}_for_{os.path.basename(self.file_path)}"
    for i, edge_image in enumerate(edge_results):
        output_path = f"{output_prefix}_frame_{i + 1}.png"
        cv2.imwrite(output_path, edge_image)
        print(f"Saved: {output_path}")

def main(self, threshold=50, output_prefix="edge_result"):
    self.detect_file_type()
    self.load_image()
    edge_results = self.detect_edges(threshold=threshold)
    if self.show:
        self.visualize_results(edge_results)
    if self.save:
        self.save_results(edge_results, output_prefix=output_prefix)
```

2. Repeat (1) steps in the image ‘Visual resolution.gif’? (3X10=40)



程式碼與方法與(1)相同

3. Please comment and compare your two designs?

   從實驗的結果中發現，同樣的運算子在 RGB 不同成分組成中邊緣偵測的效果也有所不同，藍色成分邊緣更突顯落差較大的邊緣，紅色成分邊緣則是能連小細節也一同偵測，綠色邊緣則在兩者間，由結果也可表明 SOBEL 算子進邊緣偵測有很不錯的效果