Computer Vision Homework 9

R08922079 資工所一 洪浩翔

Part 0

```python
def readImg(filename='lena.bmp'):
    #read img
    image = cv2.imread(filename, cv2.IMREAD_GRAYSCALE)
    return image

def gradient(img, kernel1, kernel2, thres, name):
    result = img.copy()
    img = np.pad(img, ((1,1),(1,1)), 'constant', constant_values=0)
    w,h = img.shape
    for i in range(1,w-1):
        for j in range(1,h-1):
            r1 = np.sum(img[i-1:i+2,j-1:j+2]*kernel1)
            r2 = np.sum(img[i-1:i+2,j-1:j+2]*kernel2)
            value = math.sqrt(r1**2 + r2**2)
            if value > thres:
                result[i-1,j-1] = 0
            else:
                result[i-1,j-1] = 255
    cv2.imwrite(name+'.jpg', result)
    return result
```

```python
def familyGrad(img, kernelFamily, thres, name):
    result = img.copy()
    img = np.pad(img, ((1,1),(1,1)), 'constant', constant_values=0)
    w, h = img.shape
    for i in range(1,w-1):
        for j in range(1,h-1):
            recordK = []
            for k in kernelFamily:
                recordK.append(np.sum(img[i-1:i+2,j-1:j+2]*k))
            g = max(recordK)
            if g > thres:
                result[i-1,j-1] = 0
            else:
                result[i-1,j-1] = 255
    cv2.imwrite(name+'.jpg', result)
    return result
```

Part 1

Original Lena

Robert's Operator: 12

Prewitt's Edge Detector: 24

Sobel's Edge Detector: 38

Frei and Chen's Gradient Operator: 30

Kirsch's Compass Operator: 135

Robinson's Compass Operator: 43

Nevatia-Babu 5x5 Operator: 12500

Code

```
def robert(img, thres=12):
    kernelR1 = np.array([[0,0,0],[0,-1,0],[0,0,1]])
    kernelR2 = np.array([[0,0,0],[0,0,-1],[0,1,0]])
    result = gradient(img.copy(), kernelR1, kernelR2, thres, "robert")
    return result

def prewitt(img, thres=24):
    kernelP1 = np.array([[-1,-1,-1],[0,0,0],[1,1,1]])
    kernelP2 = np.array([[-1,0,1],[-1,0,1],[-1,0,1]])
    result = gradient(img.copy(), kernelP1, kernelP2, thres, "prewitt")
    return result

def sobel(img, thres=38):
    kernelP1 = np.array([[-1,-2,-1],[0,0,0],[1,2,1]])
    kernelP2 = np.array([[-1,0,1],[-2,0,2],[-1,0,1]])
    result = gradient(img.copy(), kernelP1, kernelP2, thres, "sobel")
    return result

def frei(img, thres=30):
    kernelP1 = np.array([[-1,-math.sqrt(2),-1],[0,0,0],[1,math.sqrt(2),1]])
    kernelP2 = np.array([[-1,0,1],[-math.sqrt(2),0,math.sqrt(2)],[-1,0,1]])
    result = gradient(img.copy(), kernelP1, kernelP2, thres, "frei")
    return result
```

```
def kirsch(img, thres=135):
    kernelFamily = [[[-3,-3,5],[-3,0,5],[-3,-3,5]],[[-3,5,5],[-3,0,5],[-3,-3,-3]],
    [[5,5,5],[-3,0,-3],[-3,-3,-3]],[[5,5,-3],[5,0,-3],[-3,-3,-3]],[[5,-3,-3],[5,0,-3],[5,-3,-3]],
    [[-3,-3,-3],[5,0,-3],[5,5,-3]],[[-3,-3,-3],[-3,0,-3],[5,5,5]],[[-3,-3,-3],[-3,0,5],[-3,5,5]]]
    kernelFamily = np.array(kernelFamily)
    result = familyGrad(img, kernelFamily, thres, "kirsch")
    return result

def robinson(img, thres=43):
    kernelFamily = [[[-1,0,1],[-2,0,2],[-1,0,1]],[[0,1,2],[-1,0,1],[-2,-1,0]],
    [[1,2,1],[0,0,0],[-1,-2,-1]],[[2,1,0],[1,0,-1],[0,-1,-2]],[[1,0,-1],[2,0,-2],[1,0,-1]],
    [[0,-1,-2],[1,0,-1],[2,1,0]],[[-1,-2,-1],[0,0,0],[1,2,1]],[[-2,-1,0],[-1,0,1],[0,1,2]]]
    kernelFamily = np.array(kernelFamily)
    result = familyGrad(img, kernelFamily, thres, "robinson")
    return result
```

```python
def nevatia(img, thres=12500):
    kernelFamily = [
    [[100,100,100,100,100],[100,100,100,100,100],[0,0,0,0,0],[-100,-100,-100,-100,-100],[-100,-100,
    [[100,100,100,100,100],[100,100,100,78,-32],[100,92,0,-92,-100],[32,-78,-100,-100,-100],[-100,-
    [[100,100,100,32,-100],[100,100,92,-78,-100],[100,100,0,-100,-100],[100,78,-92,-100,-100],[100,
    [[-100,-100,0,100,100],[-100,-100,0,100,100],[-100,-100,0,100,100],[-100,-100,0,100,100],[-100,
    [[-100,32,100,100,100],[-100,-78,92,100,100],[-100,-100,0,100,100],[-100,-100,-92,78,100],[-100
    [[100,100,100,100,100],[-32,78,100,100,100],[-100,-92,0,92,100],[-100,-100,-100,-78,32],[-100,-
    kernelFamily = np.array(kernelFamily)
    result = img.copy()
    img = np.pad(img, ((2,2),(2,2)), 'constant', constant_values=0)
    w, h = img.shape
    for i in range(2,w-2):
        for j in range(2,h-2):
            recordK = []
            for k in kernelFamily:
                recordK.append(np.sum(img[i-2:i+3,j-2:j+3]*k))
            g = max(recordK)
            if g > thres:
                result[i-2,j-2] = 0
            else:
                result[i-2,j-2] = 255
    cv2.imwrite('nevatia.jpg', result)
    return result
```

```python
def main():
    img = readImg()

    _ = robert(img)
    _ = prewitt(img)
    _ = sobel(img)
    _ = frei(img)
    _ = kirsch(img)
    _ = robinson(img)
    _ = nevatia(img)

if __name__ == "__main__":
    main()
```