[Computer Vision I] Homework 9

學號: R07943087 姓名: 林啟源

Write a program to generate images and histograms:

```python
def main(Config):
    # Read in Image
    Img = cv2.imread(Config.init_pict)
    Width, Height, Channel = Img.shape
    print("Image width =", Width, ", Image height =", Height, "\n")

    # (a) Robert's Operator: 12
    Img_Robert = Robert(Img, Width, Height, 12)
    cv2.imwrite("Lena_Robert_12.bmp", Img_Robert)
    print("Lena_Robert_12.bmp")
    # (b) Prewitt's Edge Detector: 24
    Img_Prewitt = Prewitt(Img, Width, Height, 24)
    cv2.imwrite("Lena_Prewitt_24.bmp", Img_Prewitt)
    print("Lena_Prewitt_24.bmp")
    # (c) Sobel's Edge Detector: 38
    Img_Sobel = Sobel(Img, Width, Height, 38)
    cv2.imwrite("Lena_Sobel_38.bmp", Img_Sobel)
    print("Lena_Sobel_38.bmp")
    # (d) Frei and Chen's Gradient Operator: 30
    Img_Frei_Chen = Frei_Chen(Img, Width, Height, 30)
    cv2.imwrite("Lena_Frei_Chen_30.bmp", Img_Frei_Chen)
    print("Lena_Frei_Chen_30.bmp")
    # (e) Kirsch's Compass Operator: 135
    Img_Kirsch = Kirsch(Img, Width, Height, 135)
    cv2.imwrite("Lena_Kirsch_135.bmp", Img_Kirsch)
    print("Lena_Kirsch_135.bmp")
    # (f) Robinson's Compass Operator: 43
    Img_Robinson = Robinson(Img, Width, Height, 43)
    cv2.imwrite("Lena_Robinson_43.bmp", Img_Robinson)
    print("Lena_Robinson_43.bmp")
    # (g) Nevatia-Babu 5x5 Operator: 12500
    Img_Nevatia_Babu = Nevatia_Babu(Img, Width, Height, 12500)
    cv2.imwrite("Lena_Nevatia-Babu_12500.bmp", Img_Nevatia_Babu)
    print("Lena_Nevatia-Babu_12500.bmp")
```

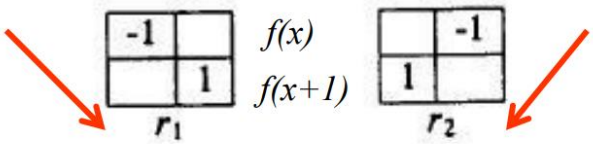
(a) Robert's Operator: 12

```python
# Robert's Operator
def Robert(img, width, height, threshold):
    img_copy = img[:, :, 0]
    img_robert = np.ones((width, height), np.uint8)
    img_robert = img_robert*255
    mask1 = np.array([[-1, 0],
                      [ 0, 1]])
    mask2 = np.array([[ 0,-1],
                      [ 1, 0]])

    for w in range(width-1):
        for h in range(height-1):
            r1 = np.sum(mask1 * img_copy[w:w+2, h:h+2])
            r2 = np.sum(mask2 * img_copy[w:w+2, h:h+2])
            gm = math.sqrt(r1**2 + r2**2)
            if (gm>threshold):
                img_robert[w, h] = 0

    return img_robert
```

$f'(x) \approx f(x+1) - f(x)$



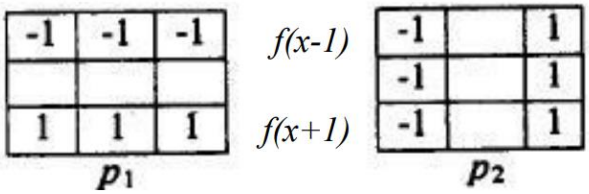
gradient magnitude: $\sqrt{r_1^2 + r_2^2}$

Lena_Robert_12

(b) Prewitt's Edge Detector: 24

```
33  # Prewitt's Edge Detector
34  def Prewitt(img, width, height, threshold):
35      img_copy = img[:, :, 0]
36      img_prewitt = np.ones((width+2, height+2), np.uint8)
37      img_prewitt = img_prewitt*255
38      mask1 = np.array([[-1, -1, -1],
39                        [ 0,  0,  0],
40                        [ 1,  1,  1]])
41      mask2 = np.array([[-1,  0,  1],
42                        [-1,  0,  1],
43                        [-1,  0,  1]])
44
45      for w in range(1, width-1, 1):
46          for h in range(1, height-1, 1):
47              p1 = np.sum(mask1 * img_copy[w-1:w+2, h-1:h+2])
48              p2 = np.sum(mask2 * img_copy[w-1:w+2, h-1:h+2])
49              gm = math.sqrt(p1**2 + p2**2)
50              if (gm>threshold):
51                  img_prewitt[w, h] = 0
52
53      return img_prewitt[1:width+1, 1:height+1]
```

$f'(x) \approx f(x+1) - f(x-1)$ **Threshold=24**



gradient magnitude: $\sqrt{p_1^2 + p_2^2}$

Lena_Prewitt_24

(c) Sobel's Edge Detector: 38

```python
# Sobel's Edge Detector
def Sobel(img, width, height, threshold):
    img_copy = img[:, :, 0]
    img_sobel = np.ones((width+2, height+2), np.uint8)
    img_sobel = img_sobel*255
    mask1 = np.array([[-1, -2, -1],
                      [ 0,  0,  0],
                      [ 1,  2,  1]])
    mask2 = np.array([[-1,  0,  1],
                      [-2,  0,  2],
                      [-1,  0,  1]])

    for w in range(1, width-1, 1):
        for h in range(1, height-1, 1):
            s1 = np.sum(mask1 * img_copy[w-1:w+2, h-1:h+2])
            s2 = np.sum(mask2 * img_copy[w-1:w+2, h-1:h+2])
            gm = math.sqrt(s1**2 + s2**2)
            if (gm>threshold):
                img_sobel[w, h] = 0

    return img_sobel[1:width+1, 1:height+1]
```

$f'(x) \approx f(x+1) - f(x-1)$ **Threshold=38**

-1	-2	-1
1	2	1

s_1 $f(x-1)$ $f(x+1)$

-1		1
-2		2
-1		1

s_2

gradient magnitude: $\sqrt{s_1^2 + s_2^2}$

Lena_Sobel_38

(d) Frei and Chen's Gradient Operator: 30

```python
# Frei and Chen's Gradient Operator
def Frei_Chen(img, width, height, threshold):
    img_copy = img[:, :, 0]
    img_frei_chen = np.ones((width+2, height+2), np.uint8)
    img_frei_chen = img_frei_chen*255
    r2 = math.sqrt(2)
    mask1 = np.array([[-1,-r2, -1],
                      [ 0,  0,  0],
                      [ 1, r2,  1]])
    mask2 = np.array([[-1,  0,  1],
                      [-r2, 0, r2],
                      [-1,  0,  1]])

    for w in range(1, width-1, 1):
        for h in range(1, height-1, 1):
            s1 = np.sum(mask1 * img_copy[w-1:w+2, h-1:h+2])
            s2 = np.sum(mask2 * img_copy[w-1:w+2, h-1:h+2])
            gm = math.sqrt(s1**2 + s2**2)
            if (gm>threshold):
                img_frei_chen[w, h] = 0

    return img_frei_chen[1:width+1, 1:height+1]
```

Frei and Chen gradient operator (ppt p.87) Threshold=30

$f(x-1)$

-1	$-\sqrt{2}$	-1
1	$\sqrt{2}$	1

f_1

-1		1
$-\sqrt{2}$		$\sqrt{2}$
-1		1

f_2

$f(x+1)$

gradient magnitude: $\sqrt{f_1^2 + f_2^2}$ $f'(x) \approx f(x+1) - f(x-1)$

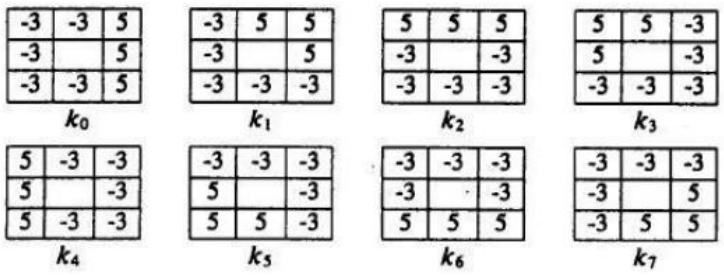
Lena_Frei_Chen_30

(e) Kirsch's Compass Operator: 135

```python
# Kirsch's Compass Operator
def Kirsch(img, width, height, threshold):
    img_copy = img[:, :, 0]
    img_kirsch = np.ones((width+2, height+2), np.uint8)
    img_kirsch = img_kirsch*255
    Kir = np.array([ [[-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]] ])

    for w in range(1, width-1, 1):
        for h in range(1, height-1, 1):
            k_list = []
            for k in range(8):
                ksum = np.sum(Kir[k] * img_copy[w-1:w+2, h-1:h+2])
                k_list.append(ksum)
                gmax = max(k_list)
            if (gmax>threshold):
                img_kirsch[w, h] = 0

    return img_kirsch[1:width+1, 1:height+1]
```



gradient magnitude: $\max_{n,n=0,\ldots,7} k_n$

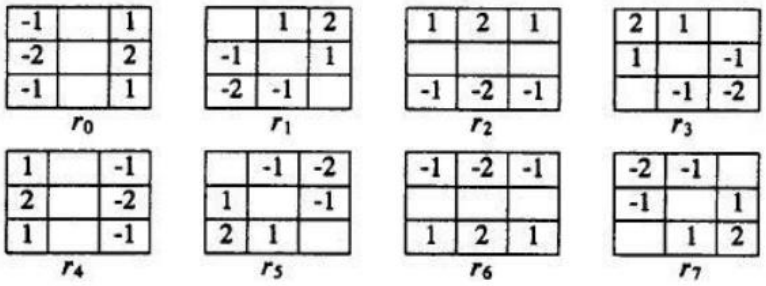
Lena_Kirsch_135

(f) Robinson's Compass Operator: 43

```python
# Robinson's Compass Operator
def Robinson(img, width, height, threshold):
    img_copy = img[:, :, 0]
    img_robinson = np.ones((width+2, height+2), np.uint8)
    img_robinson = img_robinson*255
    Rob = np.array([ [[-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]] ])

    for w in range(1, width-1, 1):
        for h in range(1, height-1, 1):
            r_list = []
            for r in range(8):
                rsum = np.sum(Rob[r] * img_copy[w-1:w+2, h-1:h+2])
                r_list.append(rsum)
                gmax = max(r_list)
            if (gmax>threshold):
                img_robinson[w, h] = 0

    return img_robinson[1:width+1, 1:height+1]
```



gradient magnitude: $\max_{n,n=0,\ldots,7} r_n$

Lena_Robinson_43

(g) Nevatia-Babu 5x5 Operator: 12500

```python
# Nevatia-Babu 5x5 Operator
def Nevatia_Babu(img, width, height, threshold):
    img_copy = img[:, :, 0]
    img_nevatia_babu = np.ones((width+4, height+4), np.uint8)
    img_nevatia_babu = img_nevatia_babu*255
    NB_mask = np.array([
        [[ 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], [ 100, 100, 100,  78, -32], [ 100,  92,   0, -92,-100], [  32, -78,-100,-100,-100], [-100,-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, -32,-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,   0, 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, -32, 100]],
        [[ 100, 100, 100, 100, 100], [ -32,  78, 100, 100, 100], [-100, -92,   0,  92, 100], [-100,-100,-100, -78,  32], [-100,-100,-100,-100,-100]] ])

    for w in range(2, width-2, 1):
        for h in range(2, height-2, 1):
            nb_list = []
            for nb in range(6):
                nbsum = np.sum(NB_mask[nb] * img_copy[w-2:w+3, h-2:h+3])
                nb_list.append(nbsum)
                gmax = max(nb_list)
            if (gmax>threshold):
                img_nevatia_babu[w, h] = 0

    return img_nevatia_babu[2:width+2, 2:height+2]
```

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

0°

100	100	100	100	100
100	100	100	78	-32
100	92	0	-92	-100
32	-78	-100	-100	-100
-100	-100	-100	-100	-100

30°

100	100	100	32	-100
100	100	92	-78	-100
100	100	0	-100	-100
100	78	-92	-100	-100
100	-32	-100	-100	-100

60°

-100	-100	0	100	100
-100	-100	0	100	100
-100	-100	0	100	100
-100	-100	0	100	100
-100	-100	0	100	100

-90°

-100	32	100	100	100
-100	-78	92	100	100
-100	-100	0	100	100
-100	-100	-92	78	100
-100	-100	-100	-32	100

-60°

100	100	100	100	100
-32	78	100	100	100
-100	-92	0	92	100
-100	-100	-100	-78	32
-100	-100	-100	-100	-100

-30°

gradient magnitude: $\max_{n,n=0,\ldots,5} N_n$

169	169	169	1

Lena_Nevatia-Babu_12500