Computer Vision hw9

B05902050

黃子源

I use PIL to complete the homework. In my program, I use function getpixel() and putpixel() to get the value of every pixel.

I create a list named mask to represent the 3x3 block coordinate which the original is in the center.

All the thresholds I use is from the course website which TA recommends to use.

Robert's Operator:

I just simply get the value of the 2 masks r1, r2, and compute the value of (r1^2 + r2^2)^0.5, I use math.sqrt() function to calculate the root value.

Principal code fragment:

```
robert = Image.new("1", (w, h))
for i in range(w-1):
    for j in range(h-1):
        r1 = img.getpixel((i + 1,j + 1)) - img.getpixel((i,j))
        r2 = img.getpixel((i + 1,j)) - img.getpixel((i,j + 1))
        gradiant = math.sqrt(r1*r1 + r2*r2)
        if gradiant > 12:
            robert.putpixel((i,j),1)
        else:
            robert.putpixel((i,j),0)
robert.save('robert.bmp')
```

Prewitt's Edge Detector and Sobel's Edge Detector:

I use the following formula to calculate the value of the gradients of two detectors.

P_1	P_2	P_3
P_4	P_5 $f(x,y)$	P_6
P_7	P_8	P_9

$$\nabla f(x,y) \cong |G_x| + |G_y|$$
$$\cong |(P_7 + wP_8 + P_9) - (P_1 + wP_2 + P_3)|$$
$$+ |(P_3 + wP_6 + P_9) - (P_1 + wP_4 + P_7)|$$

w：以P_5為中心的4-connected

- Prewitt edge detector : $w = 1$
- Sobel edge detector : $w = 2$

Frei and Chen's Gradient operator:

I use the formula above and take 'w' as 2^0.5 to get f1 and f2. After that, I use math.sqrt() to compute (f1^2 + f2^2)^0.5

Principal code fragment:

```
prewitt = Image.new("1", (w,h))
sobel = Image.new("1", (w,h))
frei_and_chen = Image.new("1", (w,h))
for i in range(1,w-1):
    for j in range(1,h-1):
        p = [0 for n in range(9)]
        for k in range(9):
            p[k] =  img.getpixel((i + mask[k][0],j + mask[k][1]))
        gradiantp = abs((p[6] + p[7] + p[8]) - (p[0] + p[1] + p[2])) + abs((p[2] + p[5] + p[8]) - (p[0] + p[3] + p[
        gradiants = abs((p[6] + 2*p[7] + p[8]) - (p[0] + 2*p[1] + p[2])) + abs((p[2] + 2*p[5] + p[8]) - (p[0] + 2*p
        f1 = (p[6] + math.sqrt(2)*p[7] + p[8]) - (p[0] + math.sqrt(2)*p[1] + p[2])
        f2 = (p[2] + math.sqrt(2)*p[5] + p[8]) - (p[0] + math.sqrt(2)*p[3] + p[6])
        gradiantf = math.sqrt(f1*f1 + f2*f2)
        if gradiantp > 24:
            prewitt.putpixel((i,j),1)
        else:
            prewitt.putpixel((i,j),0)
        if gradiants > 38:
            sobel.putpixel((i,j),1)
        else:
            sobel.putpixel((i,j),0)
        if gradiantf > 30:
            frei_and_chen.putpixel((i,j),1)
        else:
            frei_and_chen.putpixel((i,j),0)
```

Kirsch's Compass Operator:

I take this mask as the sum value of 8-connectivity neighbor multiply -3, and plus every 3 connected pixels multiply 8. After calculating the value of 8 masks, take the maxima one as gradient.

Principal code fragment:

```
kirsch = Image.new("1", (w,h))
for i in range(1,w-1):
    for j in range(1,h-1):
        p = [0 for n in range(9)]
        kinitial = 0
        k = [0 for n in range(8)]
        for kk in range(9):
            p[kk] =  img.getpixel((i + mask[kk][0],j + mask[kk][1]))
            if kk != 4:
                kinitial += -3*p[kk]
        k[0] = kinitial + 8*(p[2] + p[5] + p[8])
        k[1] = kinitial + 8*(p[1] + p[2] + p[5])
        k[2] = kinitial + 8*(p[0] + p[1] + p[2])
        k[3] = kinitial + 8*(p[0] + p[1] + p[3])
        k[4] = kinitial + 8*(p[0] + p[3] + p[6])
        k[5] = kinitial + 8*(p[3] + p[6] + p[7])
        k[6] = kinitial + 8*(p[6] + p[7] + p[8])
        k[7] = kinitial + 8*(p[5] + p[7] + p[8])
        gradiantk = max(k)
        if gradiantk > 135:
            kirsch.putpixel((i,j),1)
        else:
            kirsch.putpixel((i,j),0)
kirsch.save('kirsch.bmp')
```

Robinson’s Compass Operator:

This part I just simply calculate the 8 masks’ value with brute force and take the largest one as gradient.

Principal code fragment:

```
robinson = Image.new("1", (w,h))
for i in range(1,w-1):
    for j in range(1,h-1):
        p = [0 for n in range(9)]
        r = [0 for n in range(4)]
        for k in range(9):
            p[k] =  img.getpixel((i + mask[k][0],j + mask[k][1]))
        r[0] = abs((p[2] + 2*p[5] + p[8]) - (p[0] + 2*p[3] + p[6]))
        r[1] = abs((p[1] + 2*p[2] + p[5]) - (p[3] + 2*p[6] + p[7]))
        r[2] = abs((p[0] + 2*p[1] + p[2]) - (p[6] + 2*p[7] + p[8]))
        r[3] = abs((p[3] + 2*p[0] + p[1]) - (p[5] + 2*p[8] + p[7]))
        gradiantr = max(r)
        if gradiantr > 43:
            robinson.putpixel((i,j),1)
        else:
            robinson.putpixel((i,j),0)
robinson.save('robinson.bmp')
```

Nevatia-Babu 5x5 Operator:

This part I calculate the value of n0 and n3 first because they are the simplest. Then I calculate the value of the remaining masks using n0 and n3, and take the maximum as gradient.

Principal code fragment:

```
nevatia_babu = Image.new("1", (w,h))
for i in range(2,w-2):
    for j in range(2,h-2):
        p = [0 for m in range(25)]
        n = [0 for m in range(6)]
        ninitial = 0
        nvertical = 0
        count = 0
        for k in range(-2,3):
            for l in range(-2,3):
                p[count] = img.getpixel((i + l,j + k))
                if k < 0:
                    ninitial += 100*p[count]
                elif k > 0:
                    ninitial -= 100*p[count]
                if l < 0:
                    nvertical -= 100*p[count]
                elif l > 0:
                    nvertical += 100*p[count]
                count += 1
        n[0] = ninitial
        n[1] = n[0] - 22*p[8] - 132*p[9] + 100*p[10] + 92*p[11] - 92*p[13] - 100*p[14] + 132*p[15] + 22*p[16]
        n[2] = n[1] - 68*p[3] - 200*p[4] - 8*p[7] - 156*p[8] - 68*p[9] + 8*p[11] - 8*p[13] + 68*p[15] + 156*p[16]
        n[3] = nvertical
        n[4] = n[3] + 132*p[1] + 100*p[2] + 22*p[6] + 92*p[7] - 92*p[17] - 22*p[18] - 100*p[22] -132*p[23]
        n[5] = n[0] - 132*p[5] - 22*p[6] - 100*p[10] - 92*p[11] + 92*p[13] + 100*p[14] + 22*p[18] + 132*p[19]
        gradiantn = max(n)
        #print(gradiantn)
        if gradiantn > 12500:
            nevatia_babu.putpixel((i,j),1)
        else:
            nevatia_babu.putpixel((i,j),0)
nevatia_babu.save('nevatia_babu.bmp')
```

Result:

Robert: Prewitt: Sobel:

Kirsch: Robinson: Nevatia-Babu:

Frei and Chen: