Computer vision homework 9

B04902028 資工三 洪浩翔

Original Lena image

1. Robert edge detection:

Robert edge detection with threshold = 12

```python
def robert(image , threshold):
  robert_image = image.copy()
  (h , w) = image.size
  for i in range(h):
    for j in range(w):
      tmp1_1 = 0
      tmp1_2 = 0
      tmp2_1 = 0
      tmp2_2 = 0
      if (i+1 >= h) and (j+1 >= w):
        tmp1_1 = image.getpixel((i , j))
        tmp1_2 = 0
        tmp2_1 = 0
        tmp2_2 = 0
      elif (i+1 >= h) and (j+1 < w):
        tmp1_1 = image.getpixel((i , j))
        tmp1_2 = 0
        tmp2_1 = image.getpixel((i , j+1))
        tmp2_2 = 0
      elif (i+1 < h) and (j+1 >= w):
        tmp1_1 = image.getpixel((i , j))
        tmp1_2 = 0
        tmp2_1 = 0
        tmp2_2 = image.getpixel((i+1 , j))
      else:
        tmp1_1 = image.getpixel((i , j))
        tmp1_2 = image.getpixel((i+1 , j+1))
        tmp2_1 = image.getpixel((i , j+1))
        tmp2_2 = image.getpixel((i+1 , j))
      r1 = -(tmp1_1) + (tmp1_2)
      r2 = -(tmp2_1) + (tmp2_2)
      value = math.sqrt(r1**2 + r2**2)
      if value > threshold:
        robert_image.putpixel((i , j) , 0)
      else:
        robert_image.putpixel((i , j) , 255)

  return robert_image
```

Code of Robert

2. Prewitt edge detection:

Prewitt edge detection with threshold = 24

```
def prewitt(image , threshold , tmp_list):
  prewitt_image = image.copy()
  (h , w) = image.size
  #print 'check:' , tmp_list
  for i in range(h):
    for j in range(w):
      p1 = -(tmp_list[i][j] + tmp_list[i][j+1] + tmp_list[i][j+2]) + (tmp_list[i+2][j] + tmp_list[i+2][j+1] + tmp_list[i+2][j+2])
      p2 = -(tmp_list[i][j] + tmp_list[i+1][j] + tmp_list[i+2][j]) + (tmp_list[i][j+2] + tmp_list[i+1][j+2] + tmp_list[i+2][j+2])
      value = math.sqrt(p1 ** 2 + p2 ** 2)
      if value > threshold:
        prewitt_image.putpixel((i , j) , 0)
      else:
        prewitt_image.putpixel((i , j) , 255)

  return prewitt_image
```

Code of prewitt

3. Sobel edge detection:

Sobel edge detection with threshold = 38

```python
def Sobel(image , threshold , tmp_list):
  Sobel_image = image.copy()
  (h , w) = image.size
  for i in range(h):
    for j in range(w):
      s1 = -(tmp_list[i][j] + 2*tmp_list[i][j+1] + tmp_list[i][j+2]) + (tmp_list[i+2][j] + 2*tmp_list[i+2][j+1] + tmp_list[i+2][j+2])
      s2 = -(tmp_list[i][j] + 2*tmp_list[i+1][j] + tmp_list[i+2][j]) + (tmp_list[i+2][j] + 2*tmp_list[i+2][j+1] + tmp_list[i+2][j+2])
      value = math.sqrt(s1 ** 2 + s2 ** 2)
      if value > threshold:
        Sobel_image.putpixel((i , j) , 0)
      else:
        Sobel_image.putpixel((i , j) , 255)

  return Sobel_image
```

Code of sobel

4. Frei & Chen edge detection:

Frei & Chen edge detection with threshold = 30

```python
def Frei_and_Chen(image , threshold , tmp_list):
  Frei_and_Chen_image = image.copy()
  (h , w) = image.size
  for i in range(h):
    for j in range(w):
      s1 = -(tmp_list[i][j] + math.sqrt(2)*tmp_list[i][j+1] + tmp_list[i][j+2]) + (tmp_list[i+2][j] + math.sqrt(2)*tmp_list[i+2][j+1] + tmp_list[i+2][j+2])
      s2 = -(tmp_list[i][j] + math.sqrt(2)*tmp_list[i+1][j] + tmp_list[i+2][j]) + (tmp_list[i+2][j] + math.sqrt(2)*tmp_list[i+2][j+1] + tmp_list[i+2][j+2])
      value = math.sqrt(s1 ** 2 + s2 ** 2)
      if value > threshold:
        Frei_and_Chen_image.putpixel((i , j) , 0)
      else:
        Frei_and_Chen_image.putpixel((i , j) , 255)

  return Frei_and_Chen_image
```

Code of Frei & Chen

5. Kirsch edge detection:

Kirsch edge detection with threshold = 135

```
def Kirsch(image , threshold , tmp_list):
  Kirsch_image = image.copy()
  (h , w) = image.size
  k = [0]*8
  for i in range(h):
    for j in range(w):
      k[0] = ((-3)*tmp_list[i][j+1] + (-3)*tmp_list[i][j] + (-3)*tmp_list[i+1][j] + (-3)*tmp_list[i+2][j] + (-3)*tmp_list[i+2][j+1]  +
              5*tmp_list[i+2][j+2] + 5*tmp_list[i+1][j+2] + 5*tmp_list[i][j+2])
      k[1] = ((-3)*tmp_list[i+2][j+2] + (-3)*tmp_list[i][j] + (-3)*tmp_list[i+1][j] + (-3)*tmp_list[i+2][j] + (-3)*tmp_list[i+2][j+1]  +
              5*tmp_list[i][j+1] + 5*tmp_list[i+1][j+2] + 5*tmp_list[i][j+2])
      k[2] = ((-3)*tmp_list[i+2][j+2] + (-3)*tmp_list[i+1][j+2] + (-3)*tmp_list[i+1][j] + (-3)*tmp_list[i+2][j] + (-3)*tmp_list[i+2][j+1]  +
              5*tmp_list[i][j+1] + 5*tmp_list[i][j] + 5*tmp_list[i][j+2])
      k[3] = ((-3)*tmp_list[i+2][j+2] + (-3)*tmp_list[i+1][j+2] + (-3)*tmp_list[i][j+2] + (-3)*tmp_list[i+2][j] + (-3)*tmp_list[i+2][j+1]  +
              5*tmp_list[i][j+1] + 5*tmp_list[i][j] + 5*tmp_list[i+1][j])
      k[4] = ((-3)*tmp_list[i+2][j+2] + (-3)*tmp_list[i+1][j+2] + (-3)*tmp_list[i][j+2] + (-3)*tmp_list[i][j+1] + (-3)*tmp_list[i+2][j+1]  +
              5*tmp_list[i+2][j] + 5*tmp_list[i][j] + 5*tmp_list[i+1][j])
      k[5] = ((-3)*tmp_list[i+2][j+2] + (-3)*tmp_list[i+1][j+2] + (-3)*tmp_list[i][j+2] + (-3)*tmp_list[i][j+1] + (-3)*tmp_list[i][j]  +
              5*tmp_list[i+2][j] + 5*tmp_list[i+2][j+1] + 5*tmp_list[i+1][j])
      k[6] = ((-3)*tmp_list[i+1][j] + (-3)*tmp_list[i+1][j+2] + (-3)*tmp_list[i][j+2] + (-3)*tmp_list[i][j+1] + (-3)*tmp_list[i][j]  +
              5*tmp_list[i+2][j] + 5*tmp_list[i+2][j+1] + 5*tmp_list[i+2][j+2])
      k[7] = ((-3)*tmp_list[i+1][j] + (-3)*tmp_list[i+2][j] + (-3)*tmp_list[i][j+2] + (-3)*tmp_list[i][j+1] + (-3)*tmp_list[i][j]  +
              5*tmp_list[i+1][j+2] + 5*tmp_list[i+2][j+1] + 5*tmp_list[i+2][j+2])
      value = max(k)
      if value > threshold:
        Kirsch_image.putpixel((i , j) , 0)
      else:
        Kirsch_image.putpixel((i , j) , 255)

  return Kirsch_image
```

Code of Kirsch

6. Robinson edge detection:

Robinson edge detection with threshold = 43

```
def Robinson(image , threshold , tmp_list):
  Robinson_image = image.copy()
  (h , w) = image.size
  k = [0]*8
  for i in range(h):
    for j in range(w):
      k[0] = (-1)*tmp_list[i][j] + (-2)*tmp_list[i+1][j] + (-1)*tmp_list[i+2][j] + tmp_list[i][j+2] + 2*tmp_list[i+1][j+2] + tmp_list[i+2][j+2]
      k[4] = -k[0]
      k[1] = (-1)*tmp_list[i+1][j] + (-2)*tmp_list[i+2][j] + (-1)*tmp_list[i+2][j+1] + tmp_list[i][j+1] + 2*tmp_list[i][j+2] + tmp_list[i+1][j+2]
      k[5] = -k[1]
      k[2] = (-1)*tmp_list[i+2][j] + (-2)*tmp_list[i+2][j+1] + (-1)*tmp_list[i+2][j+2] + tmp_list[i][j] + 2*tmp_list[i][j+1] + tmp_list[i][j+2]
      k[6] = -k[2]
      k[3] = (-1)*tmp_list[i+2][j+1] + (-2)*tmp_list[i+2][j+2] + (-1)*tmp_list[i+1][j+2] + tmp_list[i+1][j] + 2*tmp_list[i][j] + tmp_list[i][j+1]
      k[7] = -k[3]
      value = max(k)
      if value > threshold:
        Robinson_image.putpixel((i , j) , 0)
      else:
        Robinson_image.putpixel((i , j) , 255)

  return Robinson_image
```

Code of Robinson

7. Nevatia-Babu's edge detectors:

Nevatia-Babu's edge detection with threshold = 12500

```
def Nevatia_Babu(image , threshold , tmp_list_big):
  Nevatia_Babu_image = image.copy()
  (h , w) = image.size
  k = [0]*6
  for i in range(h):
    for j in range(w):
      k[0] = (100*tmp_list_big[i][j] + 100*tmp_list_big[i][j+1] + 100*tmp_list_big[i][j+2] + 100*tmp_list_big[i][j+3] + 100*tmp_list_big[i][j+4] +
              100*tmp_list_big[i+1][j] + 100*tmp_list_big[i+1][j+1] + 100*tmp_list_big[i+1][j+2] + 100*tmp_list_big[i+1][j+3] + 100*tmp_list_big[i+1][j+4] -
             (100*tmp_list_big[i+3][j] + 100*tmp_list_big[i+3][j+1] + 100*tmp_list_big[i+3][j+2] + 100*tmp_list_big[i+3][j+3] + 100*tmp_list_big[i+3][j+4]) -
             (100*tmp_list_big[i+4][j] + 100*tmp_list_big[i+4][j+1] + 100*tmp_list_big[i+4][j+2] + 100*tmp_list_big[i+4][j+3] + 100*tmp_list_big[i+4][j+4]))
      k[1] = (100*tmp_list_big[i][j] + 100*tmp_list_big[i][j+1] + 100*tmp_list_big[i][j+2] + 100*tmp_list_big[i][j+3] + 100*tmp_list_big[i][j+4] +
              100*tmp_list_big[i+1][j] + 100*tmp_list_big[i+1][j+1] + 100*tmp_list_big[i+1][j+2] + 78*tmp_list_big[i+1][j+3] + (-32)*tmp_list_big[i+1][j+4] +
              100*tmp_list_big[i+2][j] + 92*tmp_list_big[i+2][j+1] + (-92)*tmp_list_big[i+2][j+3] - 100*tmp_list_big[i+2][j+4] +
              32*tmp_list_big[i+3][j] + (-78)*tmp_list_big[i+3][j+1] + (-100)*tmp_list_big[i+3][j+2] + (-100)*tmp_list_big[i+3][j+3] + (-100)*tmp_list_big[i+3][j+4] -
              (100*tmp_list_big[i+4][j] + 100*tmp_list_big[i+4][j+1] + 100*tmp_list_big[i+4][j+2] + 100*tmp_list_big[i+4][j+3] + 100*tmp_list_big[i+4][j+4]))
      k[2] = (100*tmp_list_big[i][j] + 100*tmp_list_big[i][j+1] + 100*tmp_list_big[i][j+2] + 32*tmp_list_big[i][j+3] - 100*tmp_list_big[i][j+4] +
              100*tmp_list_big[i+1][j] + 100*tmp_list_big[i+1][j+1] + 92*tmp_list_big[i+1][j+2] - 78*tmp_list_big[i+1][j+3] - 100*tmp_list_big[i+1][j+4] +
              100*tmp_list_big[i+2][j] + 100*tmp_list_big[i+2][j+1] + (-100)*tmp_list_big[i+2][j+3] - 100*tmp_list_big[i+2][j+4] +
              100*tmp_list_big[i+3][j] + (78)*tmp_list_big[i+3][j+1] + (-92)*tmp_list_big[i+3][j+2] + (-100)*tmp_list_big[i+3][j+3] + (-100)*tmp_list_big[i+3][j+4] +
              100*tmp_list_big[i+4][j] + (-32)*tmp_list_big[i+4][j+1] + (-100)*tmp_list_big[i+4][j+2] - 100*tmp_list_big[i+4][j+3] - 100*tmp_list_big[i+4][j+4])
      k[3] = ((-100)*tmp_list_big[i][j] + (-100)*tmp_list_big[i][j+1] + (100)*tmp_list_big[i][j+3] + (100)*tmp_list_big[i][j+4] +
              (-100)*tmp_list_big[i+1][j] + (-100)*tmp_list_big[i+1][j+1] + 100*tmp_list_big[i+1][j+3] + 100*tmp_list_big[i+1][j+4] +
              (-100)*tmp_list_big[i+2][j] + (-100)*tmp_list_big[i+2][j+1] + 100*tmp_list_big[i+2][j+3] + 100*tmp_list_big[i+2][j+4] +
              (-100)*tmp_list_big[i+3][j] + (-100)*tmp_list_big[i+3][j+1] + 100*tmp_list_big[i+3][j+3] + 100*tmp_list_big[i+3][j+4] +
              (-100)*tmp_list_big[i+4][j] + (-100)*tmp_list_big[i+4][j+1] + 100*tmp_list_big[i+4][j+3] + 100*tmp_list_big[i+4][j+4])
      k[4] = ((-100)*tmp_list_big[i][j] + 32*tmp_list_big[i][j+1] + 100*tmp_list_big[i][j+2] + 100*tmp_list_big[i][j+3] + 100*tmp_list_big[i][j+4] +
              (-100)*tmp_list_big[i+1][j] + (-78)*tmp_list_big[i+1][j+1] + 92*tmp_list_big[i+1][j+2] + 100*tmp_list_big[i+1][j+3] + 100*tmp_list_big[i+1][j+4] +
              (-100)*tmp_list_big[i+2][j] + (-100)*tmp_list_big[i+2][j+1] + 100*tmp_list_big[i+2][j+3] + 100*tmp_list_big[i+2][j+4] +
              (-100)*tmp_list_big[i+3][j] + (-100)*tmp_list_big[i+3][j+1] + (-92)*tmp_list_big[i+3][j+2] + 78*tmp_list_big[i+3][j+3] + 100*tmp_list_big[i+3][j+4] +
              (-100)*tmp_list_big[i+4][j] + (-100)*tmp_list_big[i+4][j+1] + (-100)*tmp_list_big[i+4][j+2] - 32*tmp_list_big[i+4][j+3] + 100*tmp_list_big[i+4][j+4])
      k[5] = (100*tmp_list_big[i][j] + 100*tmp_list_big[i][j+1] + 100*tmp_list_big[i][j+2] + 100*tmp_list_big[i][j+3] + 100*tmp_list_big[i][j+4] +
              (-32)*tmp_list_big[i+1][j] + 78*tmp_list_big[i+1][j+1] + 100*tmp_list_big[i+1][j+2] + 100*tmp_list_big[i+1][j+3] + 100*tmp_list_big[i+1][j+4] +
              (-100)*tmp_list_big[i+2][j] + (-92)*tmp_list_big[i+2][j+1] + 92*tmp_list_big[i+2][j+3] + 100*tmp_list_big[i+2][j+4] +
              (-100)*tmp_list_big[i+3][j] + (-100)*tmp_list_big[i+3][j+1] + (-100)*tmp_list_big[i+3][j+2] + (-78)*tmp_list_big[i+3][j+3] + 32*tmp_list_big[i+3][j+4] +
              (-100)*tmp_list_big[i+4][j] + (-100)*tmp_list_big[i+4][j+1] + (-100)*tmp_list_big[i+4][j+2] - 100*tmp_list_big[i+4][j+3] - 100*tmp_list_big[i+4][j+4])
      value = max(k)
      if value > threshold:
        Nevatia_Babu_image.putpixel((i , j) , 0)
      else:
        Nevatia_Babu_image.putpixel((i , j) , 255)

  return Nevatia_Babu_image
```

Code of Nevatia-Babu's