Computer Vision Homework 9

B03902042 宋子維

Description

In this homework, we are asked to implement Robert, Prewitt, Sobel, Frei & Chen, Kirsch, Robinson, and Nevatia-Babu's edge detectors and use these to do edge detection on the gray level image, that is, *lena.bmp*, which is shown in *Figure 1-1*.

Figure 1-1: Gray level image.

Programming

I use python to implement the algorithms. There is one python program, namely, **edge-detector.py**, where I use **pillow** to process basic image I/O. In the program, there are some basic functions and instructions:

1. `PIL.Image.open(img)`: load the image `img` and return a pillow **Image** object.

2. `pix = Image.load()`: return the **PixelAccess** object of **Image** object to `pix`, which offers us to use `pix[x, y]` to access the pixel value at position (x, y).

3. `PIL.Image.new(mode, size)`: create a new pillow **Image** object given its mode and size.

4. `Image.size`: pair (width, height) of **Image** object.

5. `sumTuple((a, b), (c, d))`: return the tuple (a+c, b+d).

6. `product(range(a), range(b))`: return the list of cartesian product of set $\{0, 1, \cdots, a-1\}$ and set $\{0, 1, \cdots, b-1\}$.

Usage

The usages of **edge-detector.py** is in the following:

```
python3.5 edge-detector.py [-h] -input INPUT \
                               -output OUTPUT \
                               -method METHOD \
                               [-thres THRES]
```

where METHOD can be one of Roberts, Prewitt, Sobel, FreiChen, Kirsch, Robinson and NB.

Result

First of all, the thresholds I use are listed in the following:

- Roberts: 12
- Prewitt: 24
- Sobel: 38
- Frei and Chen: 30
- Kirsch: 135
- Robinson: 43
- Nevatia-Babu: 12500

And the kernels for each operator are totally the same with the ones in the slide.

Figure 2-1: Roberts operator with threshold 12

Figure 2-2: Prewitt operator with threshold 24

Figure 2-3: Sobel operator with threshold 38

Figure 2-4: Frei & Chen operator with threshold 30

Figure 2-5: Kirsch operator with threshold 135

Figure 2-6: Robinson operator with threshold 43

Figure 2-7: Nevatia-Babu operator with threshold 12000

Algorithm

First, all of these edge detectors need to do convolution on image, so I first implement the function of convolution on a point:

```
def flip(p, bound):
    if p < 0:
        return -p-1
    elif p >= bound:
        return 2 * bound - 1 - p

    return p

# M: original image
# size: size of image

def convolve(M, size, pos, mask):
    ret = 0;
    X, Y = int(len(mask)/2), int(len(mask[0])/2)
    for (j, row) in enumerate(mask):
        for (i, value) in enumerate(row):
            offset = (i-X, j-Y)
            (x, y) = sumTuple(pos, offset)
            x = flip(x, size[0])
            y = flip(y, size[1])
            ret += M[x, y] * value
    return ret
```

where `flip()` return the mirrored point if the input is out of range. `convolve()` return the convolution value at `pos` respect to `mask`.

There are two types of gradient magnitude in this assignment:

1. $G = \sqrt{\sum_k G_k^2}$

2. $G = \max_k G_k$

Roberts, Prewitt, Sobel and Frei & Chen edge detectors belong to Type 1; Kirsch, Robinson, Nevatia-Babu edge detectors belong to Type 2. The main difference between them is the mask. I implement the function `edge_detect()`, which do convolution on whole image and calculate the gradient magnitude of each position after convolution. For each operator, simply pass the mask and type (sum of square or max) corresponding to it, and then the function will return the image after doing edge detection.

```
edge = lambda x, T: (WHITE if x < T else BLACK)

def edge_detect(pix, size, mask, thres, type):
    if type == 'sum':
        return {p: edge(sqrt(sum( \
                    [convolve(pix, size, p, m)**2 for m in mask])), thres) \
                        for p in product(range(size[0]), range(size[1]))}
    elif type == 'max':
        return {p: edge(max( \
                    [convolve(pix, size, p, m) for m in mask]), thres) \
                        for p in product(range(size[0]), range(size[1]))}
```

where `edge()` simply return WHITE if the gradient magnitude is less than threshold; otherwise, BLACK.