Computer Vision Homework 10

B03902042 宋子維

Description

In this homework, we are asked to implement RLaplacian, Minimum Variance Laplacian, Laplacian of Gaussian, and Difference of Gaussian zero crossing 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, **zero-crossing-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 \
                               [-kernel K] \
                               [-in-sigma IN_SIGMA] \
                               [-ex-sigma EX-SIGMA] \
                               [-thres THRES]
```

where METHOD can be one of Laplacian, min-var-Laplacian, LOG, and DOG; K is the optional argument for METHOD Laplacian; IN_SIGMA and EX_SIGMA are the optional argument for METHOD DOG, which can specify the inhibitory sigma and excitatory sigma for difference of Gaussian kernel.

Algorithms

First, I implement the function `convolve_whole(img, size, mask)`, which do convolution on `img` with respect to `mask`. When doing convolution, there may exist some points that is out of bound, and thus, I "flip" them to let them be the mirrored points with respect to boundary.

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

    return p

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

def convolve_whole(pix, size, mask):
    return {p: convolve(pix, size, p, mask) \
                for p in product(range(size[0]), range(size[1]))}
```

After doing convolution on image, we can find that which points are "zero-crossing".

```
def zero_crossing(M, size, thres):
    ret = {}
    for pos in product(range(size[0]), range(size[1])):
        ret[pos] = WHITE
        for offset in _8NEIGHBOR:
            cur = M[pos]
            try:
                neighbor = M[sumTuple(pos, offset)]
                if (cur > thres and neighbor < -thres):
                    ret[pos] = BLACK
            except:
                pass

    return ret
```

Hence, the difference between each zero-crossing detector is mask. Here are the results, masks and thresholds I use.

Laplacian

Kernel 0:

Figure 2-1: Laplacian with kernel 0 and threshold 15.

$$\text{kernel} = \begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

with threshold 15.

Kernel 1:

Figure 2-2: Laplacian with kernel 1 and threshold 15.

$$\text{kernel} = \begin{pmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \\ \frac{1}{3} & \frac{-8}{3} & \frac{1}{3} \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \end{pmatrix}$$

with threshold 15.

Minimum Variance Laplacian

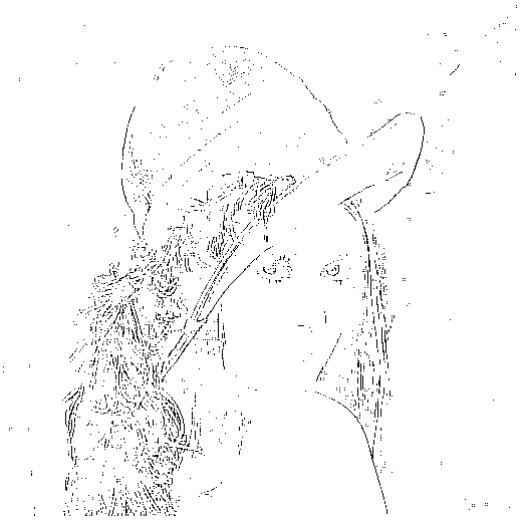

Figure 2-3: Minimum Variance Laplacian with threshold 20.

$$\text{kernel} = \begin{pmatrix} \frac{2}{3} & \frac{-1}{3} & \frac{2}{3} \\ \frac{-1}{3} & \frac{-4}{3} & \frac{-1}{3} \\ \frac{2}{3} & \frac{-1}{3} & \frac{2}{3} \end{pmatrix}$$

with threshold 20.

Laplacian of Gaussian

Figure 2-4: Laplacian of Gaussian with threshold 30.

$$\text{kernel} = \begin{pmatrix} 0 & 0 & 0 & -1 & -1 & -2 & -1 & -1 & 0 & 0 & 0 \\ 0 & 0 & -2 & -4 & -8 & -9 & -8 & -4 & -2 & 0 & 0 \\ 0 & -2 & -7 & -15 & -22 & -23 & -22 & -15 & -7 & -2 & 0 \\ -1 & -4 & -15 & -24 & -14 & -1 & -14 & -24 & -15 & -4 & -1 \\ -1 & -8 & -22 & -14 & 52 & 103 & 52 & -14 & -22 & -8 & -1 \\ -2 & -9 & -23 & -1 & 103 & 178 & 103 & -1 & -23 & -9 & -2 \\ -1 & -8 & -22 & -14 & 52 & 103 & 52 & -14 & -22 & -8 & -1 \\ -1 & -4 & -15 & -24 & -14 & -1 & -14 & -24 & -15 & -4 & -1 \\ 0 & -2 & -7 & -15 & -22 & -23 & -22 & -15 & -7 & -2 & 0 \\ 0 & 0 & -2 & -4 & -8 & -9 & -8 & -4 & -2 & 0 & 0 \\ 0 & 0 & 0 & -1 & -1 & -2 & -1 & -1 & 0 & 0 & 0 \end{pmatrix}$$

with threshold 30.

Differnece of Gaussian

Figure 2-5: Differnece of Gaussian with threshold 1.

The mask of Differnece of Gaussian is exactly the difference of two 2D Gaussian kernels. In this case, the inhibitory sigma is 1, excitatory sigma 3 and kernel size is 11×11.

```
def Gaussian(x, mean, std):
    return (1 / (sqrt(2 * pi) * std )) * \
            e ** ( (-(x - mean) ** 2 ) / (2 * std ** 2))

def Gaussian2d(x, y, std_x, std_y, mean_x=0, mean_y=0):
    return Gaussian(x, mean_x, std_x) * Gaussian(y, mean_y, std_y);

def DOG_mask(in_sigma, ex_sigma, size=11):
    l = int(-size / 2)
    r = int(size / 2) + 1

    mask = []
    for i, x in enumerate(range(l, r)):
        mask.append([])
        for y in range(l, r):
            mask[i].append(Gaussian2d(x, y, std_x=in_sigma, std_y=in_sigma) \
                           -Gaussian2d(x, y, std_x=ex_sigma, std_y=ex_sigma))

    return mask
```