Computer Vision Homework 10　D07922015　謝銘峰

Write a program to generate:

Original

Laplace Mask1 (0, 1, 0, 1, -4, 1, 0, 1, 0): 15

Laplace Mask2 (1, 1, 1, 1, -8, 1, 1, 1, 1):15

Minimum variance Laplacian: 20

Laplace of Gaussian: 3000

Difference of Gaussian: 1

(a) Laplace Mask 1 (0, 1, 0, 1, -4, 1, 0, 1, 0)

```python
def getLaplacianMask1Array(originalImage, threshold):
    """
    :type originalImage: Image (from PIL)
    :type threshold: float
    :return type: numpy array
    """
    from PIL import Image
    import numpy as np
    # Zero numpy array with the same size.
    LaplacianMask1 = np.zeros(originalImage.size)
    # Scan each column in original image.
    for c in range(originalImage.size[0]):
        # Scan each row in original image.
        for r in range(originalImage.size[1]):
            # Calculate x0, y0, x1, y1, x2, y2 and avoid out of image range.
            x0 = max(c - 1, 0)
            y0 = max(r - 1, 0)
            x1 = c
            y1 = r
            x2 = min(c + 1, originalImage.size[0] - 1)
            y2 = min(r + 1, originalImage.size[1] - 1)
            # Get 3x3 neighbors.
            neighbors = [originalImage.getpixel((x0, y0)), originalImage.getpixel((x1, y0)), originalImage.getpixel((x2, y0)),
            originalImage.getpixel((x0, y1)), originalImage.getpixel((x1, y1)), originalImage.getpixel((x2, y1)),
            originalImage.getpixel((x0, y2)), originalImage.getpixel((x1, y2)), originalImage.getpixel((x2, y2))]
            # Calculate Grandient magnitude of Laplacian mask 1 for (0, 1, 0, 1, -4, 1, 0, 1, 0).
            magnitude = (0) * neighbors[0] + (1) * neighbors[1] + (0) * neighbors[2] + \
                        (1) * neighbors[3] + (-4) * neighbors[4] + (1) * neighbors[5] + \
                        (0) * neighbors[6] + (1) * neighbors[7] + (0) * neighbors[8]
            # Binarize with threshold.
            if (magnitude >= threshold):
                LaplacianMask1[c, r] = 1
            elif (magnitude <= -threshold):
                LaplacianMask1[c, r] = -1
            else:
                LaplacianMask1[c, r] = 0
    return LaplacianMask1
```

Zerocrossing dector

```python
def zeroCrossingDetector(grandient, width, height):
    """
    :type grandient: numpy array
    :type width: int
    :type height: int
    :return type: Image (from PIL)
    """
    # New image with the same size and 'binary' format.
    zeroCrossingImage = Image.new('1', grandient.shape)
    # Scan each column in grandient array.
    for c in range(grandient.shape[0]):
        # Scan each row in grandient array.
        for r in range(grandient.shape[1]):
            # Record does it cross zero.
            cross = 1
            # If current location is high.
            if (grandient[c, r] == 1):
                # Scan its neighbors.
                for x in range(-width // 2, width // 2 + 1):
                    for y in range(-height // 2, height // 2 + 1):
                        # Avoid out of range.
                        destX = np.clip(c + x, 0, grandient.shape[0] - 1)
                        destY = np.clip(r + y, 0, grandient.shape[1] - 1)
                        # Check zero crossing.
                        if (grandient[destX, destY] == -1):
                            cross = 0
            # Put pixel to image.
            zeroCrossingImage.putpixel((c, r), cross)
    return zeroCrossingImage
```

(b) Laplace Mask 2 (1, 1, 1, 1, -8, 1, 1, 1, 1)

```python
def getLaplacianMask2Array(originalImage, threshold):
    """
    :type originalImage: Image (from PIL)
    :type threshold: float
    :return type: numpy array
    """
    from PIL import Image
    import numpy as np
    # Zero numpy array with the same size.
    LaplacianMask2 = np.zeros(originalImage.size)
    # Scan each column in original image.
    for c in range(originalImage.size[0]):
        # Scan each row in original image.
        for r in range(originalImage.size[1]):
            # Calculate x0, y0, x1, y1, x2, y2 and avoid out of image range.
            x0 = max(c - 1, 0)
            y0 = max(r - 1, 0)
            x1 = c
            y1 = r
            x2 = min(c + 1, originalImage.size[0] - 1)
            y2 = min(r + 1, originalImage.size[1] - 1)
            # Get 3x3 neighbors.
            neighbors = [originalImage.getpixel((x0, y0)), originalImage.getpixel((x1, y0)), originalImage.getpixel((x2, y0)),
            originalImage.getpixel((x0, y1)), originalImage.getpixel((x1, y1)), originalImage.getpixel((x2, y1)),
            originalImage.getpixel((x0, y2)), originalImage.getpixel((x1, y2)), originalImage.getpixel((x2, y2))]
            # Calculate Grandient magnitude of Laplacian mask 2 for (1, 1, 1, 1, -8, 1, 1, 1, 1).
            magnitude = (1) * neighbors[0] + (1) * neighbors[1] + (1) * neighbors[2] + \
                        (1) * neighbors[3] + (-8) * neighbors[4] + (1) * neighbors[5] + \
                        (1) * neighbors[6] + (1) * neighbors[7] + (1) * neighbors[8]
            magnitude = magnitude / 3
            # Binarize with threshold.
            if (magnitude >= threshold):
                LaplacianMask2[c, r] = 1
            elif (magnitude <= -threshold):
                LaplacianMask2[c, r] = -1
            else:
                LaplacianMask2[c, r] = 0
    return LaplacianMask2
```

(c) Minimum variance Laplacian: 20

```python
def getMinVarianceLaplacianArray(originalImage, threshold):
    """
    :type originalImage: Image (from PIL)
    :type threshold: float
    :return type: numpy array
    """
    from PIL import Image
    import numpy as np
    # Zero numpy array with the same size.
    minVarianceLaplacian = np.zeros(originalImage.size)
    # Scan each column in original image.
    for c in range(originalImage.size[0]):
        # Scan each row in original image.
        for r in range(originalImage.size[1]):
            # Calculate x0, y0, x1, y1, x2, y2 and avoid out of image range.
            x0 = max(c - 1, 0)
            y0 = max(r - 1, 0)
            x1 = c
            y1 = r
            x2 = min(c + 1, originalImage.size[0] - 1)
            y2 = min(r + 1, originalImage.size[1] - 1)
            # Get 3x3 neighbors.
            neighbors = [originalImage.getpixel((x0, y0)), originalImage.getpixel((x1, y0)), originalImage.getpixel((x2, y0)),
            originalImage.getpixel((x0, y1)), originalImage.getpixel((x1, y1)), originalImage.getpixel((x2, y1)),
            originalImage.getpixel((x0, y2)), originalImage.getpixel((x1, y2)), originalImage.getpixel((x2, y2))]
            # Calculate Grandient magnitude of Laplacian mask 2.
            magnitude = (2) * neighbors[0] + (-1) * neighbors[1] + (2) * neighbors[2] + \
                        (-1) * neighbors[3] + (-4) * neighbors[4] + (-1) * neighbors[5] + \
                        (2) * neighbors[6] + (-1) * neighbors[7] + (2) * neighbors[8]
            magnitude = magnitude / 3
            # Binarize with threshold.
            if (magnitude >= threshold):
                minVarianceLaplacian[c, r] = 1
            elif (magnitude <= -threshold):
                minVarianceLaplacian[c, r] = -1
            else:
                minVarianceLaplacian[c, r] = 0
    return minVarianceLaplacian
```

(d) Laplace of Gaussian

```
def getLaplacianOfGaussianArray(originalImage, threshold):
    """
    :type originalImage: Image (from PIL)
    :type threshold: float
    :return type: numpy array
    """
    from PIL import Image
    import numpy as np
    # Kernel of Laplacian of Gaussian.
    kernel = [  [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]]
    # Zero numpy array with the same size.
    LaplacianOfGaussianArray = np.zeros(originalImage.size)
    # Scan each column in original image.
    for c in range(originalImage.size[0]):
        # Scan each row in original image.
        for r in range(originalImage.size[1]):
            # Calculate x0-10, y0-10 and avoid out of image range.
            x = np.zeros(11)
            y = np.zeros(11)
            for i in range(11):
                x[i] = np.clip(c + (i - 5), 0, originalImage.size[0] - 1)
                y[i] = np.clip(r + (i - 5), 0, originalImage.size[1] - 1)
            # Get 11x11 neighbors.
            neighbors = np.zeros((11, 11))
            for i in range(11):
                for j in range(11):
                    neighbors[i, j] = originalImage.getpixel((x[i], y[j]))
            # Calculate Grandient magnitude of Laplacian of Gaussian.
            magnitude = 0
            for i in range(11):
                for j in range(11):
                    magnitude = magnitude + kernel[j][i] * neighbors[i, j]
            # Binarize with threshold.
            if (magnitude >= threshold):
                LaplacianOfGaussianArray[c, r] = 1
            elif (magnitude <= -threshold):
                LaplacianOfGaussianArray[c, r] = -1
            else:
                LaplacianOfGaussianArray[c, r] = 0
    return LaplacianOfGaussianArray
```

(e) Difference of Gaussian

```
def getDifferenceOfGaussianArray(originalImage, threshold):
    """
    :type originalImage: Image (from PIL)
    :type threshold: float
    :return type: numpy array
    """
    from PIL import Image
    import numpy as np
    # Kernel of Laplacian of Gaussian.
    kernel = [  [-1, -3, -4, -6, -7, -8, -7, -6, -4, -3, -1],
                [-3, -5, -8, -11, -13, -13, -13, -11, -8, -5, -3],
                [-4, -8, -12, -16, -17, -17, -17, -16, -12, -8, -4],
                [-6, -11, -16, -16, 0, 15, 0, -16, -16, -11, -6],
                [-7, -13, -17, 0, 85, 160, 85, 0, -17, -13, -7],
                [-8, -13, -17, 15, 160, 283, 160, 15, -17, -13, -8],
                [-7, -13, -17, 0, 85, 160, 85, 0, -17, -13, -7],
                [-6, -11, -16, -16, 0, 15, 0, -16, -16, -11, -6],
                [-4, -8, -12, -16, -17, -17, -17, -16, -12, -8, -4],
                [-3, -5, -8, -11, -13, -13, -13, -11, -8, -5, -3],
                [-1, -3, -4, -6, -7, -8, -7, -6, -4, -3, -1]]
    # Zero numpy array with the same size.
    DifferenceOfGaussianArray = np.zeros(originalImage.size)
    # Scan each column in original image.
    for c in range(originalImage.size[0]):
        # Scan each row in original image.
        for r in range(originalImage.size[1]):
            # Calculate x0-10, y0-10 and avoid out of image range.
            x = np.zeros(11)
            y = np.zeros(11)
            for i in range(11):
                x[i] = np.clip(c + (i - 5), 0, originalImage.size[0] - 1)
                y[i] = np.clip(r + (i - 5), 0, originalImage.size[1] - 1)
            # Get 11x11 neighbors.
            neighbors = np.zeros((11, 11))
            for i in range(11):
                for j in range(11):
                    neighbors[i, j] = originalImage.getpixel((x[i], y[j]))
            # Calculate Grandient magnitude of Difference of Gaussian.
            magnitude = 0
            for i in range(11):
                for j in range(11):
                    magnitude = magnitude + kernel[j][i] * neighbors[i, j]
            # Binarize with threshold.
            if (magnitude >= threshold):
                DifferenceOfGaussianArray[c, r] = 1
            elif (magnitude <= -threshold):
                DifferenceOfGaussianArray[c, r] = -1
            else:
                DifferenceOfGaussianArray[c, r] = 0
    return DifferenceOfGaussianArray
```

Execute process

```python
if __name__ == '__main__':
    from PIL import Image
    import numpy as np

    # Load image from file.
    originalImage = Image.open('lena.bmp')

    # Get grandient of Laplacian mask 1.
    LaplacianMask1Array = getLaplacianMask1Array(originalImage, 15)
    # Get zero crossing edge of Laplacian mask 1.
    LaplacianMask1Image = zeroCrossingDetector(LaplacianMask1Array, 3, 3)
    # Save Laplacian Mask 1 image.
    LaplacianMask1Image.save('Laplacian Mask 1.bmp')

    # Get grandient of Laplacian mask 2.
    LaplacianMask2Array = getLaplacianMask2Array(originalImage, 15)
    # Get zero crossing edge of Laplacian mask 2.
    LaplacianMask2Image = zeroCrossingDetector(LaplacianMask2Array, 3, 3)
    # Save Laplacian Mask 2 image.
    LaplacianMask2Image.save('Laplacian Mask 2.bmp')

    # Get grandient of min-Variance Laplacian.
    minVarianceLaplacianArray = getMinVarianceLaplacianArray(originalImage, 20)
    # Get zero crossing edge of min-Variance Laplacian.
    minVarianceLaplacianImage = zeroCrossingDetector(minVarianceLaplacianArray, 3, 3)
    # Save min-Variance Laplacian image.
    minVarianceLaplacianImage.save('min-Variance Laplacian.bmp')

    # Get grandient of Laplacian of Gaussian.
    LaplacianOfGaussianArray = getLaplacianOfGaussianArray(originalImage, 3000)
    # Get zero crossing edge of Laplacian of Gaussian.
    LaplacianOfGaussianImage = zeroCrossingDetector(LaplacianOfGaussianArray, 11, 11)
    # Save min-Variance Laplacian image.
    LaplacianOfGaussianImage.save('Laplacian of Gaussian.bmp')

    # Get grandient of Difference of Gaussian.
    DifferenceOfGaussianArray = getDifferenceOfGaussianArray(originalImage, 1)
    # Get zero crossing edge of Difference of Gaussian.
    DifferenceOfGaussianImage = zeroCrossingDetector(DifferenceOfGaussianArray, 11, 11)
    # Save min-Variance Difference image.
    DifferenceOfGaussianImage.save('Difference of Gaussian.bmp')
```