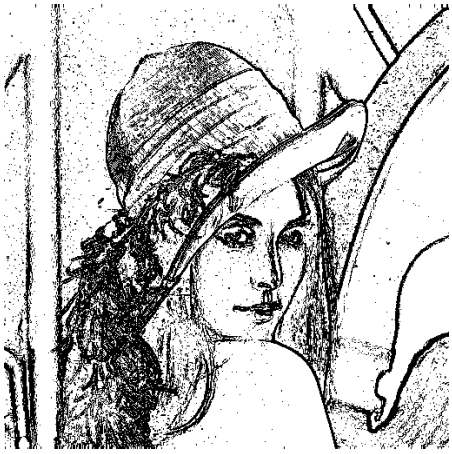
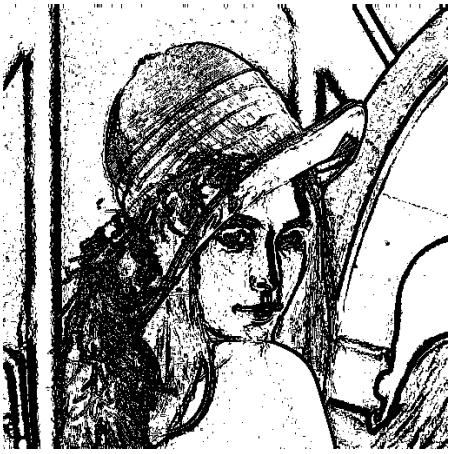
Computer Vision Homework 9　D07922015　謝銘峰

Write a program to generate:

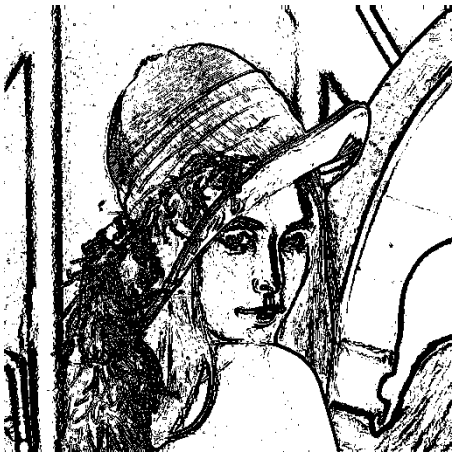
Original



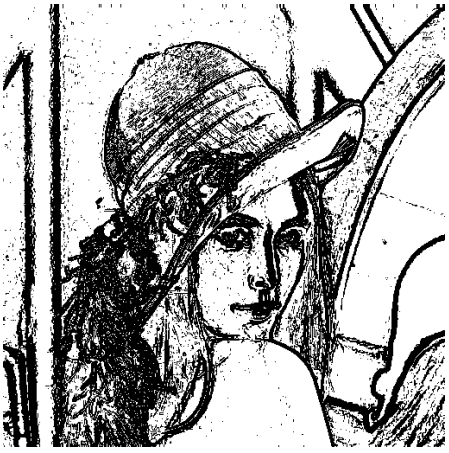
Robert's Operator: 12



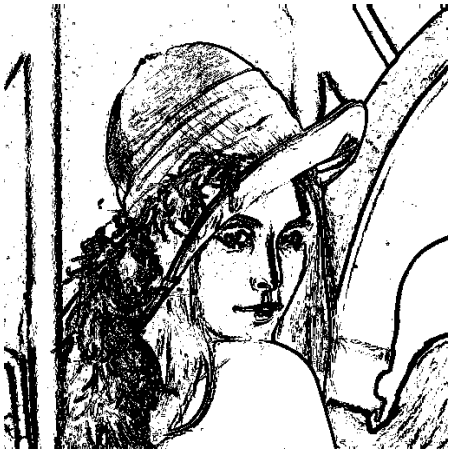
Prewitt's Edge Detector: 24



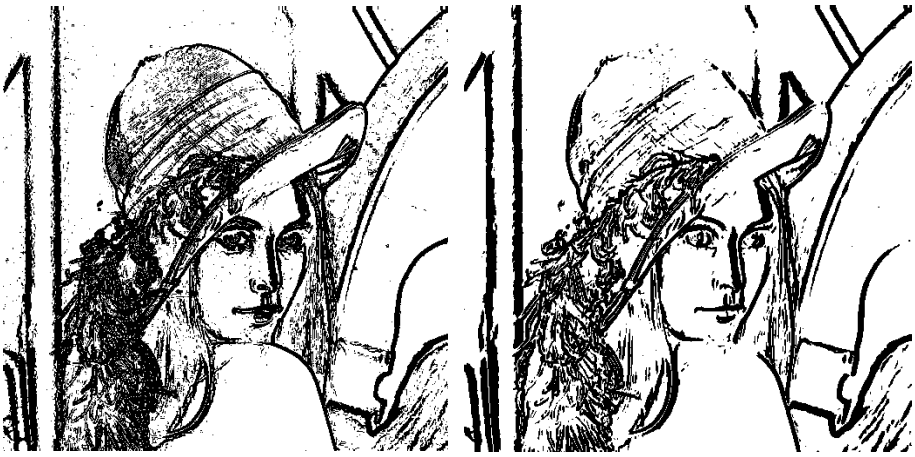
Sobel's Edge Detector: 38



Frei and Chen's Gradient Operator: 30



Kirsch's Compass Operator: 135



Robinson’s Compass Operator: 43  Nevatia-Babu 5x5 Operator: 12500

## (a) Robert operator

```python
def getRobertsImage(originalImage, threshold):
    """
    :type originalImage: Image (from PIL)
    :type threshold: float
    :return type: Image (from PIL)
    """
    from PIL import Image
    import math
    # New image with the same size and 'binary' format.
    robertsImage = Image.new('1', 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 and avoid out of image range.
            x0 = c
            y0 = r
            x1 = min(c + 1, originalImage.size[0] - 1)
            y1 = min(r + 1, originalImage.size[1] - 1)
            # Calculate r1 and r2 of Robert.
            r1 = -originalImage.getpixel((x0, y0)) + originalImage.getpixel((x1, y1))
            r2 = -originalImage.getpixel((x1, y0)) + originalImage.getpixel((x0, y1))
            # Calulate Grandient magnitude.
            magnitude = int(math.sqrt(r1 ** 2 + r2 ** 2))
            # Binarize with threshold.
            if (magnitude >= threshold):
                robertsImage.putpixel((c, r), 0)
            else:
                robertsImage.putpixel((c, r), 1)
    return robertsImage
```

## (b) Prewitt's edge detector

```
def getPrewittImage(originalImage, threshold):
    """
    :type originalImage: Image (from PIL)
    :type threshold: float
    :return type: Image (from PIL)
    """
    from PIL import Image
    import math
    # New image with the same size and 'binary' format.
    prewittImage = Image.new('1', 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)
            # Calculate p1 and p2 of Prewitt.
            p1 = -originalImage.getpixel((x0, y0)) - originalImage.getpixel((x1, y0)) - originalImage.getpixel((x2, y0))\
                 + originalImage.getpixel((x0, y2)) + originalImage.getpixel((x1, y2)) + originalImage.getpixel((x2, y2))
            p2 = -originalImage.getpixel((x0, y0)) - originalImage.getpixel((x0, y1)) - originalImage.getpixel((x0, y2))\
                 + originalImage.getpixel((x2, y0)) + originalImage.getpixel((x2, y1)) + originalImage.getpixel((x2, y2))
            # Calulate Grandient magnitude.
            magnitude = int(math.sqrt(p1 ** 2 + p2 ** 2))
            # Binarize with threshold.
            if (magnitude >= threshold):
                prewittImage.putpixel((c, r), 0)
            else:
                prewittImage.putpixel((c, r), 1)
    return prewittImage
```

## (c) Sobel's edge detector

```
def getSobelImage(originalImage, threshold):
    """
    :type originalImage: Image (from PIL)
    :type threshold: float
    :return type: Image (from PIL)
    """
    from PIL import Image
    import math
    # New image with the same size and 'binary' format.
    sobelImage = Image.new('1', 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)
            # Calculate p1 and p2 of Sobel.
            p1 = -originalImage.getpixel((x0, y0)) - 2 * originalImage.getpixel((x1, y0)) - originalImage.getpixel((x2, y0))\
                 + originalImage.getpixel((x0, y2)) + 2 * originalImage.getpixel((x1, y2)) + originalImage.getpixel((x2, y2))
            p2 = -originalImage.getpixel((x0, y0)) - 2 * originalImage.getpixel((x0, y1)) - originalImage.getpixel((x0, y2))\
                 + originalImage.getpixel((x2, y0)) + 2 * originalImage.getpixel((x2, y1)) + originalImage.getpixel((x2, y2))
            # Calulate Grandient magnitude.
            magnitude = int(math.sqrt(p1 ** 2 + p2 ** 2))
            # Binarize with threshold.
            if (magnitude >= threshold):
                sobelImage.putpixel((c, r), 0)
            else:
                sobelImage.putpixel((c, r), 1)
    return sobelImage
```

## (d) Frei and Chen's gradient operator

```
def getFreiChenImage(originalImage, threshold):
    """
    :type originalImage: Image (from PIL)
    :type threshold: float
    :return type: Image (from PIL)
    """
    from PIL import Image
    import math
    # New image with the same size and 'binary' format.
    FreiChenImage = Image.new('1', 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)
            # Calculate p1 and p2 of FreiChen.
            p1 = -originalImage.getpixel((x0, y0)) - math.sqrt(2) * originalImage.getpixel((x1, y0)) - originalImage.getpixel((x2, y0))\
                 + originalImage.getpixel((x0, y2)) + math.sqrt(2) * originalImage.getpixel((x1, y2)) + originalImage.getpixel((x2, y2))
            p2 = -originalImage.getpixel((x0, y0)) - math.sqrt(2) * originalImage.getpixel((x0, y1)) - originalImage.getpixel((x0, y2))\
                 + originalImage.getpixel((x2, y0)) + math.sqrt(2) * originalImage.getpixel((x2, y1)) + originalImage.getpixel((x2, y2))
            # Calulate Grandient magnitude.
            magnitude = int(math.sqrt(p1 ** 2 + p2 ** 2))
            # Binarize with threshold.
            if (magnitude >= threshold):
                FreiChenImage.putpixel((c, r), 0)
            else:
                FreiChenImage.putpixel((c, r), 1)
    return FreiChenImage
```

# (e) Kirsch's compass operator

```
def getKirschImage(originalImage, threshold):
    """
    :type originalImage: Image (from PIL)
    :type threshold: float
    :return type: Image (from PIL)
    """
    from PIL import Image
    import numpy as np
    import math
    # New image with the same size and 'binary' format.
    KirschImage = Image.new('1', 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)
            # Calculate k0-k7 of Kirsch.
            k = np.zeros(8)
            k[0] = -3 * originalImage.getpixel((x0, y0)) - 3 * originalImage.getpixel((x1, y0)) + 5 * originalImage.getpixel((x2, y0))\
                   - 3 * originalImage.getpixel((x0, y1)) + 5 * originalImage.getpixel((x2, y1))\
                   - 3 * originalImage.getpixel((x0, y2)) - 3 * originalImage.getpixel((x1, y2)) + 5 *originalImage.getpixel((x2, y2))
            k[1] = -3 * originalImage.getpixel((x0, y0)) + 5 * originalImage.getpixel((x1, y0)) + 5 * originalImage.getpixel((x2, y0))\
                   - 3 * originalImage.getpixel((x0, y1)) + 5 * originalImage.getpixel((x2, y1))\
                   - 3 * originalImage.getpixel((x0, y2)) - 3 * originalImage.getpixel((x1, y2)) - 3 *originalImage.getpixel((x2, y2))
            k[2] = 5 * originalImage.getpixel((x0, y0)) + 5 * originalImage.getpixel((x1, y0)) + 5 * originalImage.getpixel((x2, y0))\
                   - 3 * originalImage.getpixel((x0, y1)) - 3 * originalImage.getpixel((x2, y1))\
                   - 3 * originalImage.getpixel((x0, y2)) - 3 * originalImage.getpixel((x1, y2)) - 3 *originalImage.getpixel((x2, y2))
            k[3] = 5 * originalImage.getpixel((x0, y0)) + 5 * originalImage.getpixel((x1, y0)) - 3 * originalImage.getpixel((x2, y0))\
                   + 5 * originalImage.getpixel((x0, y1)) - 3 * originalImage.getpixel((x2, y1))\
                   - 3 * originalImage.getpixel((x0, y2)) - 3 * originalImage.getpixel((x1, y2)) - 3 *originalImage.getpixel((x2, y2))
            k[4] = 5 * originalImage.getpixel((x0, y0)) - 3 * originalImage.getpixel((x1, y0)) - 3 * originalImage.getpixel((x2, y0))\
                   + 5 * originalImage.getpixel((x0, y1)) - 3 * originalImage.getpixel((x2, y1))\
                   + 5 * originalImage.getpixel((x0, y2)) - 3 * originalImage.getpixel((x1, y2)) - 3 *originalImage.getpixel((x2, y2))
            k[5] = -3 * originalImage.getpixel((x0, y0)) - 3 * originalImage.getpixel((x1, y0)) - 3 * originalImage.getpixel((x2, y0))\
                   + 5 * originalImage.getpixel((x0, y1)) - 3 * originalImage.getpixel((x2, y1))\
                   + 5 * originalImage.getpixel((x0, y2)) + 5 * originalImage.getpixel((x1, y2)) - 3 *originalImage.getpixel((x2, y2))
            k[6] = -3 * originalImage.getpixel((x0, y0)) - 3 * originalImage.getpixel((x1, y0)) - 3 * originalImage.getpixel((x2, y0))\
                   - 3 * originalImage.getpixel((x0, y1)) - 3 * originalImage.getpixel((x2, y1))\
                   + 5 * originalImage.getpixel((x0, y2)) + 5 * originalImage.getpixel((x1, y2)) + 5 *originalImage.getpixel((x2, y2))
            k[7] = -3 * originalImage.getpixel((x0, y0)) - 3 * originalImage.getpixel((x1, y0)) - 3 * originalImage.getpixel((x2, y0))\
                   - 3 * originalImage.getpixel((x0, y1)) + 5 * originalImage.getpixel((x2, y1))\
                   - 3 * originalImage.getpixel((x0, y2)) + 5 * originalImage.getpixel((x1, y2)) + 5 *originalImage.getpixel((x2, y2))
            # Calulate Grandient magnitude.
            magnitude = max(k)
            # Binarize with threshold.
            if (magnitude >= threshold):
                KirschImage.putpixel((c, r), 0)
            else:
                KirschImage.putpixel((c, r), 1)
    return KirschImage
```

# (f) Robinson's compass operator

```python
def getRobinsonImage(originalImage, threshold):
    """
    :type originalImage: Image (from PIL)
    :type threshold: float
    :return type: Image (from PIL)
    """
    from PIL import Image
    import numpy as np
    import math
    # New image with the same size and 'binary' format.
    RobinsonImage = Image.new('1', 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)
            # Calculate r0~r7 of Robinson.
            k = np.zeros(8)
            k[0] = -1 * originalImage.getpixel((x0, y0)) - 2 * originalImage.getpixel((x0, y1)) - 1 * originalImage.getpixel((x0, y2))\
                + 1 * originalImage.getpixel((x2, y0)) + 2 * originalImage.getpixel((x2, y1)) + 1 *originalImage.getpixel((x2, y2))
            k[1] = -1 * originalImage.getpixel((x0, y1)) - 2 * originalImage.getpixel((x0, y2)) - 1 * originalImage.getpixel((x1, y2))\
                + 1 * originalImage.getpixel((x1, y0)) + 2 * originalImage.getpixel((x2, y0)) + 1 *originalImage.getpixel((x2, y1))
            k[2] = -1 * originalImage.getpixel((x0, y2)) - 2 * originalImage.getpixel((x1, y2)) - 1 * originalImage.getpixel((x2, y2))\
                + 1 * originalImage.getpixel((x0, y0)) + 2 * originalImage.getpixel((x1, y0)) + 1 *originalImage.getpixel((x2, y0))
            k[3] = -1 * originalImage.getpixel((x1, y2)) - 2 * originalImage.getpixel((x2, y2)) - 1 * originalImage.getpixel((x2, y1))\
                + 1 * originalImage.getpixel((x0, y1)) + 2 * originalImage.getpixel((x0, y0)) + 1 *originalImage.getpixel((x1, y0))
            k[4] = -1 * originalImage.getpixel((x2, y0)) - 2 * originalImage.getpixel((x2, y1)) - 1 * originalImage.getpixel((x2, y2))\
                + 1 * originalImage.getpixel((x0, y0)) + 2 * originalImage.getpixel((x0, y1)) + 1 *originalImage.getpixel((x0, y2))
            k[5] = -1 * originalImage.getpixel((x1, y0)) - 2 * originalImage.getpixel((x2, y0)) - 1 * originalImage.getpixel((x2, y1))\
                + 1 * originalImage.getpixel((x0, y1)) + 2 * originalImage.getpixel((x0, y2)) + 1 *originalImage.getpixel((x1, y2))
            k[6] = -1 * originalImage.getpixel((x0, y0)) - 2 * originalImage.getpixel((x1, y0)) - 1 * originalImage.getpixel((x2, y0))\
                + 1 * originalImage.getpixel((x0, y2)) + 2 * originalImage.getpixel((x1, y2)) + 1 *originalImage.getpixel((x2, y2))
            k[7] = -1 * originalImage.getpixel((x0, y1)) - 2 * originalImage.getpixel((x0, y0)) - 1 * originalImage.getpixel((x1, y0))\
                + 1 * originalImage.getpixel((x1, y2)) + 2 * originalImage.getpixel((x2, y2)) + 1 *originalImage.getpixel((x2, y1))
            # Calulate Grandient magnitude.
            magnitude = max(k)
            # Binarize with threshold.
            if (magnitude >= threshold):
                RobinsonImage.putpixel((c, r), 0)
            else:
                RobinsonImage.putpixel((c, r), 1)
    return RobinsonImage
```

# Execute process

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

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

    # Get Robert image.
    robertsImage = getRobertsImage(originalImage, 12)
    # Get Prewitt image.
    prewittImage = getPrewittImage(originalImage, 24)
    # Get Sobel image.
    sobelImage = getSobelImage(originalImage, 38)
    # Get FreiChen image.
    FreiChenImage = getFreiChenImage(originalImage, 30)
    # Get Kirsch image.
    KirschImage = getKirschImage(originalImage, 135)
    # Get Robinson image.
    RobinsonImage = getRobinsonImage(originalImage, 43)
    # Get NevatiaBabu image.
    NevatiaBabuImage = getNevatiaBabuImage(originalImage, 12500)

    # Save Robert image.
    robertsImage.save('Robert.bmp')
    # Save Prewitt image.
    prewittImage.save('Prewitt.bmp')
    # Save Sobel image.
    sobelImage.save('Sobel.bmp')
    # Save FreiChen image.
    FreiChenImage.save('FreiChen.bmp')
    # Save Kirsch image.
    KirschImage.save('Kirsch.bmp')
    # Save Robinson image.
    RobinsonImage.save('Robinson.bmp')
    # Save NevatiaBabu image.
    NevatiaBabuImage.save('NevatiaBabu.bmp')
```