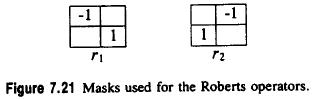
**#Assignment IX**

R08521609 土木所碩二 王澤庠

Algorithms and code fragment:

All the details are demonstrated within the code.

1. Robert’s operator

def RobertsImage(originImg, threshold):

    robertsImg = Image.new('1', originImg.size)

    for c in range(originImg.size[0]):

        for r in range(originImg.size[1]):

            #define the coordinate

            x0,y0 = c,r

            x1,y1 = min(c + 1, originImg.size[0] - 1), min(r + 1, originImg.size[1] - 1)

            # Calculate r1 and r2 of Robert.

            r1 = -originImg.getpixel((x0, y0)) + originImg.getpixel((x1, y1))

            r2 = -originImg.getpixel((x1, y0)) + originImg.getpixel((x0, y1))

            # calculate the magnitude

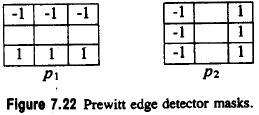
            mag = int(math.sqrt(r1\*r1 + r2\*r2))

            pixVal = 0 if mag >= threshold else 1

            robertsImg.putpixel((c, r), pixVal)

    return robertsImg

1. Prewitt’s edge operator



def PrewittImage(originImg, threshold):

    prewittImg = Image.new('1', originImg.size)

    for c in range(originImg.size[0]):

        for r in range(originImg.size[1]):

            #define the coordinate

            x0,y0 = max(c - 1, 0),max(r - 1, 0)

            x1,y1 = c,r

            x2,y2 = min(c + 1, originImg.size[0] - 1),min(r + 1, originImg.size[1] - 1)

            # Calculate p1 and p2 of Prewitt.

            p1 = -originImg.getpixel((x0, y0)) - originImg.getpixel((x1, y0)) - originImg.getpixel((x2, y0))\

                 + originImg.getpixel((x0, y2)) + originImg.getpixel((x1, y2)) + originImg.getpixel((x2, y2))

            p2 = -originImg.getpixel((x0, y0)) - originImg.getpixel((x0, y1)) - originImg.getpixel((x0, y2))\

                 + originImg.getpixel((x2, y0)) + originImg.getpixel((x2, y1)) + originImg.getpixel((x2, y2))

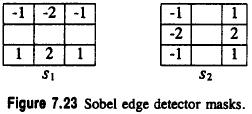
            # calculate the magnitude

            mag = int(math.sqrt(p1\*p1 + p2\*p2))

            pixVal = 0 if mag >= threshold else 1

            prewittImg.putpixel((c, r), pixVal)

    return prewittImg

1. Sobel’s edge operator

def SobelImage(originImg, threshold):

    sobelImg = Image.new('1', originImg.size)

    for c in range(originImg.size[0]):

        for r in range(originImg.size[1]):

            #define the coordinate

            x0,y0 = max(c - 1, 0),max(r - 1, 0)

            x1,y1 = c,r

            x2,y2 = min(c + 1, originImg.size[0] - 1),min(r + 1, originImg.size[1] - 1)

            # Calculate p1 and p2 of Sobel.

            p1 = -originImg.getpixel((x0, y0)) - originImg.getpixel((x1, y0)) - originImg.getpixel((x2, y0))\

                 + originImg.getpixel((x0, y2)) + originImg.getpixel((x1, y2)) + originImg.getpixel((x2, y2))

            p2 = -originImg.getpixel((x0, y0)) - originImg.getpixel((x0, y1)) - originImg.getpixel((x0, y2))\

                 + originImg.getpixel((x2, y0)) + originImg.getpixel((x2, y1)) + originImg.getpixel((x2, y2))

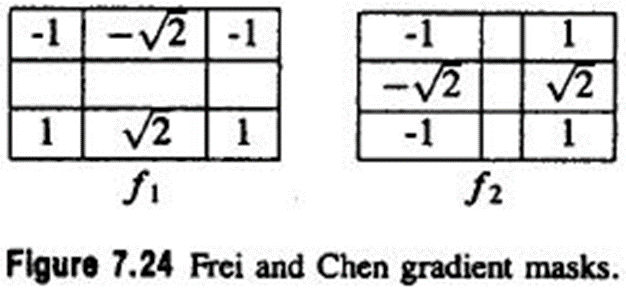
            # calculate the magnitude

            mag = int(math.sqrt(p1\*p1 + p2\*p2))

            pixVal = 0 if mag >= threshold else 1

            sobelImg.putpixel((c, r), pixVal)

    return sobelImg

1. Frei and Chen’s gradient operator

def FreiChenImage(originImg, threshold):

    FreiChenImg = Image.new('1', originImg.size)

    for c in range(originImg.size[0]):

        for r in range(originImg.size[1]):

            #define the coordinate

            x0,y0 = max(c - 1, 0),max(r - 1, 0)

            x1,y1 = c,r

            x2,y2 = min(c + 1, originImg.size[0] - 1),min(r + 1, originImg.size[1] - 1)

            # Calculate p1 and p2 of FreiChen.

            p1 = -originImg.getpixel((x0, y0)) - math.sqrt(2) \* originImg.getpixel((x1, y0)) - originImg.getpixel((x2, y0))\

                 + originImg.getpixel((x0, y2)) + math.sqrt(2) \* originImg.getpixel((x1, y2)) + originImg.getpixel((x2, y2))

            p2 = -originImg.getpixel((x0, y0)) - math.sqrt(2) \* originImg.getpixel((x0, y1)) - originImg.getpixel((x0, y2))\

                 + originImg.getpixel((x2, y0)) + math.sqrt(2) \* originImg.getpixel((x2, y1)) + originImg.getpixel((x2, y2))

            # calculate the magnitude

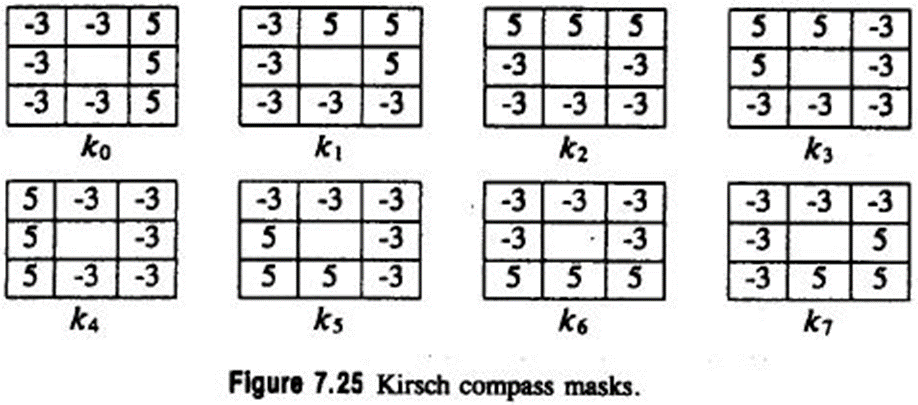
            mag = int(math.sqrt(p1\*p1 + p2\*p2))

            pixVal = 0 if mag >= threshold else 1

            FreiChenImg.putpixel((c, r), pixVal)

    return FreiChenImg

1. Kirsch’s compass operator



def KirschImage(originImg, threshold):

    #define the masks

    k0 = np.array([

        [-3, -3, 5],

        [-3, 0, 5],

        [-3, -3, 5]

    ])

    k1 = np.array([

        [-3, 5, 5],

        [-3, 0, 5],

        [-3, -3, -3]

    ])

    k2 = np.array([

        [5, 5, 5],

        [-3, 0, -3],

        [-3, -3, -3]

    ])

    k3 = np.array([

        [5, 5, -3],

        [5, 0, -3],

        [-3, -3, -3]

    ])

    k4 = np.array([

        [5, -3, -3],

        [5, 0, -3],

        [5, -3, -3]

    ])

    k5 = np.array([

        [-3, -3, -3],

        [5, 0, -3],

        [5, 5, -3]

    ])

    k6 = np.array([

        [-3, -3, -3],

        [-3, 0, -3],

        [5, 5, 5]

    ])

    k7 = np.array([

        [-3, -3, -3],

        [-3, 0, 5],

        [-3, 5, 5]

    ])

    mat\_k = np.array([k0,k1,k2,k3,k4,k5,k6,k7])

    KirschImg = Image.new('1', originImg.size)

    for c in range(originImg.size[0]):

        for r in range(originImg.size[1]):

            #define the coordinate

            x0,y0 = max(c - 1, 0),max(r - 1, 0)

            x1,y1 = c,r

            x2,y2 = min(c + 1, originImg.size[0] - 1),min(r + 1, originImg.size[1] - 1)

            # Calculate k0-k7 of Kirsch.

            k = np.zeros(8)

            for i in range(len(k)):

                k[i] = mat\_k[i][0][0] \* originImg.getpixel((x0, y0)) + mat\_k[i][0][1] \* originImg.getpixel((x1, y0)) + mat\_k[i][0][2] \* originImg.getpixel((x2, y0)) \

                 + mat\_k[i][1][0] \* originImg.getpixel((x0, y1)) + mat\_k[i][1][2] \* originImg.getpixel((x2, y1)) \

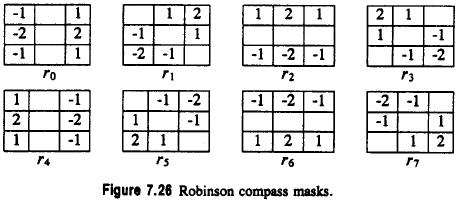
                 + mat\_k[i][2][0] \* originImg.getpixel((x0, y2)) + mat\_k[i][2][1] \* originImg.getpixel((x1, y2)) + mat\_k[i][2][2] \* originImg.getpixel((x2, y2))

            mag = np.amax(k) # Calulate Grandient mag.

            pixVal = 0 if mag >= threshold else 1

            KirschImg.putpixel((c, r), pixVal)

    return KirschImg

1. Robinson’s compass operator

def RobinsonImage(originImg, threshold):

    #define the masks

    k0 = np.array([

        [-1, 0, 1],

        [-2, 0, 2],

        [-1, 0, 1]

    ])

    k1 = np.array([

        [0, 1, 2],

        [-1, 0, 1],

        [-2, -1, 0]

    ])

    k2 = np.array([

        [1, 2, 1],

        [0, 0, 0],

        [-1, -2, -1]

    ])

    k3 = np.array([

        [2, 1, 0],

        [1, 0, -1],

        [0, -1, -2]

    ])

    k4 = np.array([

        [1, 0, -1],

        [2, 0, -2],

        [1, 0, -1]

    ])

    k5 = np.array([

        [0, -1, -2],

        [1, 0, -1],

        [2, 1, 0]

    ])

    k6 = np.array([

        [-1, -2, -1],

        [0, 0, 0],

        [1, 2, 1]

    ])

    k7 = np.array([

        [-2, -1, 0],

        [-1, 0, 1],

        [0, 1, 2]

    ])

    mat\_k = np.array([k0,k1,k2,k3,k4,k5,k6,k7])

    RobinsonImg = Image.new('1', originImg.size)

    for c in range(originImg.size[0]):

        for r in range(originImg.size[1]):

            #define the coordinate

            x0,y0 = max(c - 1, 0),max(r - 1, 0)

            x1,y1 = c,r

            x2,y2 = min(c + 1, originImg.size[0] - 1),min(r + 1, originImg.size[1] - 1)

            # Calculate k0-k7 of Robinson.

            k = np.zeros(8)

            for i in range(len(k)):

                k[i] = mat\_k[i][0][0] \* originImg.getpixel((x0, y0)) + mat\_k[i][0][1] \* originImg.getpixel((x1, y0)) + mat\_k[i][0][2] \* originImg.getpixel((x2, y0)) \

                 + mat\_k[i][1][0] \* originImg.getpixel((x0, y1)) + mat\_k[i][1][2] \* originImg.getpixel((x2, y1)) \

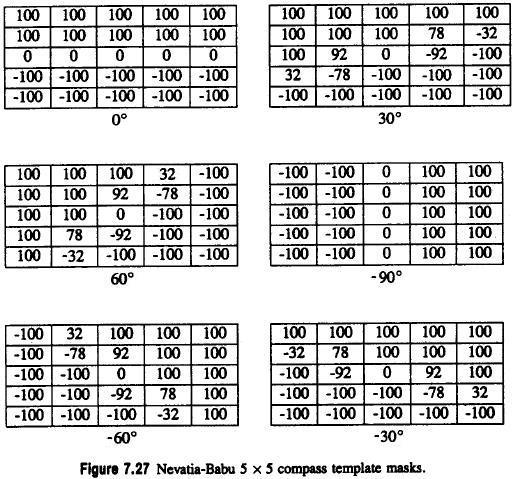
                 + mat\_k[i][2][0] \* originImg.getpixel((x0, y2)) + mat\_k[i][2][1] \* originImg.getpixel((x1, y2)) + mat\_k[i][2][2] \* originImg.getpixel((x2, y2))

            mag = np.amax(k) # Calulate Grandient mag.

            pixVal = 0 if mag >= threshold else 1

            RobinsonImg.putpixel((c, r), pixVal)

    return RobinsonImg

1. Nevatia-Babu operator

def NevatiaBabuImage(originImg, threshold):

    #define the masks

    k0 = np.array([

        [100, 100, 100, 100, 100],

        [100, 100, 100, 100, 100],

        [0, 0, 0, 0, 0],

        [-100, -100, -100, -100, -100],

        [-100, -100, -100, -100, -100],

    ])

    k1 = np.array([

        [100, 100, 100, 100, 100],

        [100, 100, 100, 78, -32],

        [100, 92, 0, -92, -100],

        [32, -78, -100, -100, -100],

        [-100, -100, -100, -100, -100]

    ])

    k2 = np.array([

        [100, 100, 100, 32, -100],

        [100, 100, 92, -78, -100],

        [100, 100, 0, -100, -100],

        [100, 78, -92, -100, -100],

        [100, -32, -100, -100, -100]

    ])

    k3 = np.array([

        [-100, -100, 0, 100, 100],

        [-100, -100, 0, 100, 100],

        [-100, -100, 0, 100, 100],

        [-100, -100, 0, 100, 100],

        [-100, -100, 0, 100, 100]

    ])

    k4 = np.array([

        [-100, 32, 100, 100, 100],

        [-100, -78, 92, 100, 100],

        [-100, -100, 0, 100, 100],

        [-100, -100, -92, 78, 100],

        [-100, -100, -100, -32, 100]

    ])

    k5 = np.array([

        [100, 100, 100, 100, 100],

        [-32, 78, 100, 100, 100],

        [-100, -92, 0, 92, 100],

        [-100, -100, -100, -78, 32],

        [-100, -100, -100, -100, -100]

    ])

    mat\_k = np.array([k0,k1,k2,k3,k4,k5])

    NevatiaBabuImg = Image.new('1', originImg.size)

    for c in range(originImg.size[0]):

        for r in range(originImg.size[1]):

            #define the coordinate

            x0,y0 = max(c - 2, 0),max(r - 2, 0)

            x1,y1 = max(c - 1, 0),max(r - 1, 0)

            x2,y2 = c,r

            x3,y3 = min(c + 1, originImg.size[0] - 1),min(r + 1, originImg.size[1] - 1)

            x4,y4 = min(c + 2, originImg.size[0] - 1),min(r + 2, originImg.size[1] - 1)

            # Get 5x5 neighbors.

            neighbors = [originImg.getpixel((x0, y0)), originImg.getpixel((x1, y0)), originImg.getpixel((x2, y0)), originImg.getpixel((x3, y0)), originImg.getpixel((x4, y0)),

                        originImg.getpixel((x0, y1)), originImg.getpixel((x1, y1)), originImg.getpixel((x2, y1)), originImg.getpixel((x3, y1)), originImg.getpixel((x4, y1)),

                        originImg.getpixel((x0, y2)), originImg.getpixel((x1, y2)), originImg.getpixel((x2, y2)), originImg.getpixel((x3, y2)), originImg.getpixel((x4, y2)),

                        originImg.getpixel((x0, y3)), originImg.getpixel((x1, y3)), originImg.getpixel((x2, y3)), originImg.getpixel((x3, y3)), originImg.getpixel((x4, y3)),

                        originImg.getpixel((x0, y4)), originImg.getpixel((x1, y4)), originImg.getpixel((x2, y4)), originImg.getpixel((x3, y4)), originImg.getpixel((x4, y4))]

            # Calculate k0-k7 of NevatiaBabu.

            k = np.zeros(6)

            for i in range(len(k)):

                k[i] = mat\_k[i][0][0] \* neighbors[0] + mat\_k[i][0][1] \* neighbors[1] + mat\_k[i][0][2] \* neighbors[2] + mat\_k[i][0][3] \* neighbors[3] + mat\_k[i][0][4] \* neighbors[4]\

                 + mat\_k[i][1][0] \* neighbors[5] + mat\_k[i][1][1] \* neighbors[6] + mat\_k[i][1][2] \* neighbors[7] + mat\_k[i][1][3] \* neighbors[8] + mat\_k[i][1][4] \* neighbors[9] \

                 + mat\_k[i][2][0] \* neighbors[10] + mat\_k[i][2][1] \* neighbors[11] + mat\_k[i][2][2] \* neighbors[12] + mat\_k[i][2][3] \* neighbors[13] + mat\_k[i][2][4] \* neighbors[14] \

                 + mat\_k[i][3][0] \* neighbors[15] + mat\_k[i][3][1] \* neighbors[16] + mat\_k[i][3][2] \* neighbors[17] + mat\_k[i][3][3] \* neighbors[18] + mat\_k[i][3][4] \* neighbors[19] \

                 + mat\_k[i][4][0] \* neighbors[20] + mat\_k[i][4][1] \* neighbors[21] + mat\_k[i][4][2] \* neighbors[22] + mat\_k[i][4][3] \* neighbors[23] + mat\_k[i][4][4] \* neighbors[24]

            # Calulate Grandient mag.

            mag = max(k)

            pixVal = 0 if mag >= threshold else 1

            NevatiaBabuImg.putpixel((c, r), pixVal)

    return NevatiaBabuImg

main:

if \_\_name\_\_ == '\_\_main\_\_':

    originImg = Image.open('lena.bmp')

    robertsImage = RobertsImage(originImg, 30).save('Robert.bmp')

    prewittImage = PrewittImage(originImg, 24).save('Prewitt.bmp')

    sobelImage = SobelImage(originImg, 38).save('Sobel.bmp')

    FreiChenImage = FreiChenImage(originImg, 30).save('FreiChen.bmp')

    KirschImage = KirschImage(originImg, 135).save('Kirsch.bmp')

    RobinsonImage = RobinsonImage(originImg, 43).save('Robinson.bmp')

    NevatiaBabuImage = NevatiaBabuImage(originImg, 12500).save('NevatiaBabu.bmp')