Assignment 1 – report
Siddhant Barua
Cwid = 10439929

```
from PIL import Image, ImageDraw

import random as rn

import numpy as np

import math as mt

import itertools as it


def suppress(Hessian_imgae, other_extreme_final_i, row, col):
    for i in range(1, row - 1):
        for j in range(1, col - 1):
            test_me = [Hessian_imgae[i - 1][j - 1], Hessian_imgae[i - 1][j],
Hessian_imgae[i - 1][j + 1],
                       Hessian_imgae[i][j - 1],
                       Hessian_imgae[i][j], Hessian_imgae[i][j + 1], Hessian_imgae[i +
1][j - 1],
                       Hessian_imgae[i + 1][j],
                       Hessian_imgae[i + 1][j + 1]]

 # what happens here is as follows [ 1 , 2 , 3]                  [ 0 , 0 , 0]
 #                                  [ 1 , 5 , 1]  => this becomes [ 0 , 5 , 0]
#                                   [ 1 , 1 , 1]                  [ 0 , 0 , 0]
# this is non maximum suppresssion
            maximum = 0
            for z in test_me:
                if z > maximum:
                    maximum = z
            test_me2 = maximum

            if Hessian_imgae[i][j] != test_me2:

                other_extreme_final_i[i - 1][j - 1] = 0  # if it's not the max value
in the [size,size] matrix in
                # locations arround the center pixel , it is set to 0 or black/ lowest
luminosity
            else:
                other_extreme_final_i[i - 1][j - 1] = Hessian_imgae[i][j]  # else set
it to the max value at the center

    other_extreme_final_i = other_extreme_final_i.astype(np.uint8)
    other_extreme_final_i = Image.fromarray(other_extreme_final_i)

    return other_extreme_final_i;


def convolve_matrix(iMg, Filt):
    valA, valB = iMg.shape

    # this is convolution ideas from the following source :
    # https: // matthew - brett.github.io / teaching / smoothing_intro.html
    # https://stackoverflow.com/questions/2448015/2d-convolution-using-python-and-
numpy/42579291#42579291
    # Ghttps://stackoverflow.com/questions/43086557/convolve2d-just-by-using-numpy

    # formula h[m,n] =  SUMMATION   (g[k,l] + f[m-k, n-l])
    #                            k,l

    Out_img = np.zeros(((valA - 2 * int((int(mt.sqrt(Filt.size)) - 1) / 2)),
                        (valB - 2 * int((int(mt.sqrt(Filt.size)) - 1) / 2))))
    for i in range((valA - 2 * int((int(mt.sqrt(Filt.size)) - 1) / 2))):
```

Assignment 1 – report
Siddhant Barua
Cwid = 10439929

```
        for j in range((valB - 2 * int((int(mt.sqrt(Filt.size)) - 1) / 2))):
            for k in range(int(mt.sqrt(Filt.size))):
                for l in range(int(mt.sqrt(Filt.size))):
                    Out_img[i][j] = Out_img[i][j] + (Filt[k][l] * iMg[i + k][j + l])
    return Out_img


def add_padd(given_img, given_size):
    temp_imgarr = given_img
    size = given_size
    row, column = temp_imgarr.shape
    filth_num = int((size - 1) / 2)

    out_row = row + 2 * (filth_num)
    out_column = column + 2 * (filth_num)

    flin_im = np.zeros((out_row, out_column))

    # this is where padding takes place
    # this block adds padding to the original image , the borders of the image
    # i.e. the rows and column values are copied to the added border. - this is called
replication of border pixels

    '''
    consider a matrix as follows [[1,2],
                                  [1,2,]]

    '''
    i = 0

    # this add's zeroes to the borders of the image
    # [[0, 0, 0, 0],
    # [0, 1, 2, 0],
    # [0, 1, 2, 0],
    # [0, 0, 0, 0]]

    while i < row:
        j = 0
        while j < (column):
            flin_im[i + filth_num][j + filth_num] = temp_imgarr[i][j]
            j += 1
        i += 1

    i = 0

    # this copies the pixel value from location img[1,1] to the borders of the image
i.e [0,0]
    # [[1, 0, 0, 0],
    # [0, 1, 2, 0],
    # [0, 1, 2, 0],
    # [0, 0, 0, 0]]

    while i < filth_num:
        j = 0
        while j < filth_num:
            flin_im[i][j] = flin_im[filth_num][filth_num]
            j += 1
        i += 1

    i = 0

    # this copies the pixel value from location img[1,2] to the borders of the image
i.e [0,4]
```

Assignment 1 – report
Siddhant Barua
Cwid = 10439929

```
    # [[1, 0, 0, 2],
    # [0, 1, 2, 0],
    # [0, 1, 2, 0],
    # [0, 0, 0, 0]]

    while i < filth_num:
        j = (out_column - filth_num)
        while j < out_column:
            flin_im[i][j] = flin_im[filth_num][out_column - filth_num - 1]
            j += 1
        i += 1

    i = (out_row - filth_num)

    # this copies the pixel value from location img[2,1] to the borders of the image
i.e [3,0]
    # [[1, 0, 0, 2],
    # [0, 1, 2, 0],
    # [0, 1, 2, 0],
    # [1, 0, 0, 0]]

    while i < out_row:
        j = 0
        while j < filth_num:
            flin_im[i][j] = flin_im[out_row - filth_num - 1][filth_num]
            j += 1
        i += 1

    i = (out_row - filth_num)

    # this copies the pixel value from location img[2,2] to the borders of the image
i.e [3,3]
    # [[1, 0, 0, 2],
    # [0, 1 2, 0],
    # [0, 1 2, 0],
    # [1, 0, 0, 2]]

    while i < out_row:
        j = (out_column - filth_num)
        while j < out_column:
            flin_im[i][j] = flin_im[out_row - filth_num - 1][out_column - filth_num -
1]
            j += 1
        i += 1

    i = 0

    # this block makes sure the top border of the padded image has the same pixel
    # values as the original image at the border , this is done for dynamically
changing
    # sizes of the image and it's applied gauss filter
    # [[1, 1, 2, 2],
    # [0, 1,  2, 0],
    # [0, 1,  2, 0],
    # [1, 0, 0, 2]]

    while i < filth_num:
        j = filth_num
        while j < (out_column - filth_num):
            flin_im[i][j] = flin_im[filth_num][j]
            j += 1
        i += 1
```

Assignment 1 – report
Siddhant Barua
Cwid = 10439929

```
    i = (out_row - filth_num)

    # this block makes sure the bottom border of the padded image has the same pixel
    # values as the original image at the border , this is done for dynamically
changing
    # sizes of the image and it's applied gauss filter
    # [[1, 1, 2, 2],
    # [0, 1,  2, 0],
    # [0, 1,  2, 0],
    # [1, 1, 2, 2]]

    while i < out_row:
        j = filth_num
        while j < (out_column - filth_num):
            flin_im[i][j] = flin_im[out_row - filth_num - 1][j]
            j += 1
        i += 1

    i = filth_num

    # this block makes sure the left border of the padded image has the same pixel
    # values as the original image at the border , this is done for dynamically
changing
    # sizes of the image and it's applied gauss filter i.e the entire left column
    # is changed to the same pixel values as the original image at the left border /
1st column
    # [[1, 1, 2, 2],
    # [1, 1,  2, 0],
    # [1, 1,  2, 0],
    # [1, 1, 2, 2]]

    while i < (out_row - filth_num):
        j = 0
        while j < filth_num:
            flin_im[i][j] = flin_im[i][filth_num]
            j += 1
        i += 1

    i = filth_num

    # this block makes sure the right border of the padded image has the same pixel
    # values as the original image at the border , this is done for dynamically
changing
    # sizes of the image and it's applied gauss filter i.e the right left column/ i =3
column/ 4th column
    # is changed to the same pixel values as the original image at the right border /
i = 3rd column /4th column
    # [[1, 1, 2, 2],
    # [1, 1,  2, 0],
    # [1, 1,  2, 0],
    # [1, 1, 2, 2]]

    while i < (out_row - filth_num):
        j = (out_column - filth_num)
        while j < out_column:
            flin_im[i][j] = flin_im[i][out_column - filth_num - 1]
            j += 1
        i += 1

    return flin_im


def main():
```

Assignment 1 – report
Siddhant Barua
Cwid = 10439929

```
    input_image = Image.open("road.png")  # uses PIL to open the image and store it in
a local variable
    input_image.show()

    size = 5  # sizez of the gaussian filter

    sigma = 1  # Sigma value of the gaussian filter

    temp_imgarr = np.array(input_image)  # numpy function that basically typecasts the
input image to a matrix format

    gauss_filter = np.zeros((size, size))  # creates a 5 X 5 matrix filled with zeroes

    filter_number = int((size - 1) / 2)

    y, x = np.ogrid[float(-filter_number):float(filter_number + 1), float(-
filter_number):float(filter_number + 1)]
    '''ogrid[-2:3, -2,3)] which returns 2 axes y and x and in this case are returned
as
        y = [[-2.],   x= [[-2., -1., 0., 1., 2.]]
            [-1.],
            [ 0.],
            [ 1.],
            [ 2.]]
     '''

    sum = 0

    i = 0
    j = 0
    reset = i = j

    '''
    basically the formula G(x,y) = {[1 / ( 2 * Pi * Sigma^2)] * e^[ - ( x^2 + y^2 )/(2
* Sigma^2)]}
    '''
    while i < size:
        j = 0
        while j < size:
            gauss_filter[i][j] = mt.exp((-((x[0][j] ** 2) + (y[i][0] ** 2)) / (2 *
(sigma ** 2)))) * (
                    1 / (2 * mt.pi * (sigma ** 2)))
            sum = sum + gauss_filter[i][j]
            j = j + 1
        i = i + 1

    i = reset
    ''' sec: https://stackoverflow.com/questions/17190649/how-to-obtain-a-gaussian-
filter-in-python '''

    while i < size:
        j = 0
        while j < size:
            gauss_filter[i][j] = gauss_filter[i][j] / sum
            j = j + 1
        i = i + 1

    # calls the add padd function wich replicates the borders

    almost_final = add_padd(temp_imgarr, size)

    fully_final = convolve_matrix(almost_final, gauss_filter)
    fully_final = Image.fromarray(fully_final)
```

Assignment 1 – report
Siddhant Barua
Cwid = 10439929

```
    hess_img = np.array(fully_final)

    val1, val2 = hess_img.shape

    extremely_final_image = np.zeros((val1, val2))
    other_extreme_final_i = extremely_final_image

    # we know that the hessin determinant is of the form [Ixx  Ixy]
    #                                                     [Ixy  Iyy]

    Sobel_X = [[-1, 0, +1], [-2, 0, +2], [-1, 0, +1]]
    Sobel_Y = [[+1, +2, +1], [0, 0, 0], [-1, -2, -1]]
    Sobel_Y = np.array(Sobel_Y)
    Sobel_X = np.array(Sobel_X)

    # this finds Ixx
    IMGOUT = add_padd(hess_img, 3)
    convolve = convolve_matrix(IMGOUT, Sobel_X)
    IMGOUT = add_padd(convolve, 3)
    XX = convolve_matrix(IMGOUT, Sobel_X)

    # this finds Iyy
    IMGOUT = add_padd(hess_img, 3)
    convolve = convolve_matrix(IMGOUT, Sobel_Y)
    IMGOUT = add_padd(convolve, 3)
    YY = convolve_matrix(IMGOUT, Sobel_Y)

    # this finds Ixy

    IMGOUT = add_padd(hess_img, 3)
    convolve = convolve_matrix(IMGOUT, Sobel_Y)
    IMGOUT = add_padd(convolve, 3)
    XY = convolve_matrix(IMGOUT, Sobel_X)

    for i in range(0, val1):
        for j in range(0, val2):
            extremely_final_image[i][j] = (XX[i][j] * YY[i][j]) - (
                    XY[i][j] ** 2)  # this finds the maxima of the determinant formula
= Ixx * Iyy - (Ixy

    old_range = ((np.amax(extremely_final_image)) - (np.amin(extremely_final_image)))

    for i in range(0, val1):
        for j in range(0, val2):
            extremely_final_image[i][j] = (extremely_final_image[i][j] -
np.amin(extremely_final_image)) * (
                    255 / old_range)

    for i in range(0, val1):
        for j in range(0, val2):  # if the pixel value is greater than the threshold ,
then they are cast to maximum
            # luminousity else they are casted as lowest luminousity i.e black
            if extremely_final_image[i][j] >= 125:
                extremely_final_image[i][j] = 255
            else:
                extremely_final_image[i][j] = 0


#####################################################################################
#############
    #
#
```

Assignment 1 – report
Siddhant Barua
Cwid = 10439929

```
    #                           HOUGH TRANSFORMS
#
    #
#

###################################################################################
#############

    Hessian_imgae = add_padd(extremely_final_image, 3)
    row, col = Hessian_imgae.shape
    other_extreme_final_i = suppress(Hessian_imgae, other_extreme_final_i, row, col)

    hough_transform = other_extreme_final_i.copy()
    original_image = input_image.copy()

    hough_img = np.asarray(hough_transform)
    h_row, h_col = hough_img.shape

    # hough transform algorithm works as follows
    # Initialize accumulator H to all zeros
    # For each edge point (x,y)
    # in the image
    #   For θ = 0 to 180
    #   ρ = x cosθ + y sinθ
    #   H(θ,ρ) = H(θ,ρ) + 1
    #   end
    # end
    # Find the value(s) of (θ,ρ) where H(θ,ρ) is a
    # local maximum

    hough_row_space = (int(((h_row + (h_col / 2)) * 2) + 1))
    offset = int((hough_row_space - 1) / 2)
    hough_space = np.zeros((hough_row_space, 181))  # This value can be changed

    for i in range(0, h_row):
        for j in range(0, h_col):
            if hough_img[i][j] != 0:

                for Theta in range(0, 181):
                    p = int(((j * mt.cos(mt.radians(Theta))) + (i *
mt.sin(mt.radians(Theta)))) + offset)
                    hough_space[p][Theta] = hough_space[p][Theta] + 1

    hough_space = hough_space.astype(np.uint8)
    hough_space_img = Image.fromarray(hough_space)
    hough_space = add_padd(hough_space, 3)  # replicate border pixels

    final_hough_img = np.zeros((hough_row_space, 181))
    final_hough_img = suppress(hough_space, final_hough_img, hough_row_space, 181)  #
non maximum supression
    final_hough_img.show()

    Hough_image_space = np.asarray(final_hough_img)
    hough_row_space, hough_col_space = Hough_image_space.shape
    row, col = np.asarray(input_image).shape
    offset = int((hough_row_space - 1) / 2)

    ListofPoints = []

    Maximum_point = 0

    PointNo = 10
```

Assignment 1 – report
Siddhant Barua
Cwid = 10439929

```
    OriginalImg = ImageDraw.Draw(input_image)

    hessianImage = ImageDraw.Draw(other_extreme_final_i)

    Hough_image_space.setflags(write=1)

    while PointNo != 0:
        i = 0
        while i < hough_row_space:
            j = 0
            while j < hough_col_space:  # loop to  check if the hough image space has
pixel value lesser than the
                # other poriions of the image
                if Hough_image_space[i][j] > Maximum_point:
                    Maximum_point = Hough_image_space[i][j]
                    row = i - offset
                    axis_i = i
                    axis_j = j
                j += 1
            i += 1

        Hough_image_space[axis_i][axis_j] = 0

        # we all know the formula y = mx + c
        # this section aims to create the lines from the edges detected in the image

        x0 = int(row * (mt.cos(mt.radians(axis_j))))  # initial x coordinate
        y0 = int(row * (mt.sin(mt.radians(axis_j))))  # initial y coordinate
        m = (-mt.cos(mt.radians(axis_j))) / (mt.sin(mt.radians(axis_j)))  # this is
the slant =  tan(θ)= cos(θ)/sin(θ)

        c = row / (
            mt.sin(mt.radians(axis_j)))  # number of rows in the Input image /
sin(Theta) this gives us the constant

        x1 = x0 + 400  # auxiliary x coordinate
        y1 = (x1 * m) + c  # y = mx + c
        x2 = x0 - 50  # auxiliary y coordinate
        y2 = (x2 * m) + c  # y = mx + c

        if PointNo == 10 or PointNo == 9 or PointNo == 8 or PointNo == 2:  # iterate
through to find the number of
            # points that match
            # this is to draw a line between the points i.e. a pair of xy coordinates
( (x,y), (x1,y1))
            OriginalImg.line((x1 - 1, y1 + 1, x1 + 1, y1 + 1), fill=0, width=3)
            OriginalImg.line((x1 - 1, y1 + 1, x1 - 1, y1 - 1), fill=0, width=3)
            OriginalImg.line((x1 + 1, y1 + 1, x1 + 1, y1 - 1), fill=0, width=3)
            OriginalImg.line((x1 - 1, y1 - 1, x1 + 1, y1 - 1), fill=0, width=3)
            OriginalImg.line((x2 - 1, y2 + 1, x2 + 1, y2 + 1), fill=0, width=3)
            OriginalImg.line((x2 - 1, y2 + 1, x2 - 1, y2 - 1), fill=0, width=3)
            OriginalImg.line((x2 + 1, y2 + 1, x2 + 1, y2 - 1), fill=0, width=3)
            OriginalImg.line((x2 - 1, y2 - 1, x2 + 1, y2 - 1), fill=0, width=3)
            OriginalImg.line((x0, y0, x1, y1), fill=0, width=3)
            OriginalImg.line((x0, y0, x2, y2), fill=0, width=3)
            hessianImage.line((x1 - 1, y1 + 1, x1 + 1, y1 + 1), fill=0, width=3)
            hessianImage.line((x1 - 1, y1 + 1, x1 - 1, y1 - 1), fill=0, width=3)
            hessianImage.line((x1 + 1, y1 + 1, x1 + 1, y1 - 1), fill=0, width=3)
            hessianImage.line((x1 - 1, y1 - 1, x1 + 1, y1 - 1), fill=0, width=3)
            hessianImage.line((x2 - 1, y2 + 1, x2 + 1, y2 + 1), fill=0, width=3)
            hessianImage.line((x2 - 1, y2 + 1, x2 - 1, y2 - 1), fill=0, width=3)
            hessianImage.line((x2 + 1, y2 + 1, x2 + 1, y2 - 1), fill=0, width=3)
            hessianImage.line((x2 - 1, y2 - 1, x2 + 1, y2 - 1), fill=0, width=3)
```

```
            hessianImage.line((x0, y0, x1, y1), fill=0, width=3)
            hessianImage.line((x0, y0, x2, y2), fill=0, width=3)

        Hough_image_space[row + offset][axis_j] = 0
        ListofPoints.append([row + offset, axis_j])
        PointNo = PointNo - 1
        Maximum_point = 0

    input_image.show()


########################################################################################
############################
    # RANSAC
    # https://salzis.wordpress.com/2014/06/10/robust-linear-model-estimation-using-
ransac-python-implementation/

########################################################################################
####

    ##generate the sample space with random numbers

    RANSAC_Iimage = hough_transform.copy()
    RANSAC_Iimage = input_image.copy()

    Huff_img = hough_transform.copy()
    Orig_img = input_image.copy()

    imgArr = np.asarray(Huff_img)
    row, col = imgArr.shape
    point = 0
    space = []

    # this generates the sample space for the ransaac operation
    for i in range(0, row):
        for j in range(0, col):
            if imgArr[i][j] != 0:
                point = point + 1

    for i in range(0, row):
        j = int(col / 3)
        while j < col:
            if imgArr[i][j] != 0:
                space.append([j, i])
            j += 1

    sample_space = list(it.combinations(space, 2))

    Image_array = np.asarray(Huff_img)
    row, col = Image_array.shape
    Iterations = 500
    distance = 2
    no_of_inliers = 50
    Master_ratop = 0.08
    fitModuloOp = []

    # #we then select two points from the generated sample space which allows us to
get the x & y coordinates
    while (Iterations != 0):
        sample_space_length = len(sample_space)
        rand_NUM = rn.randint(0, sample_space_length - 1)
        modulousOP = sample_space[rand_NUM]
        sample_space.remove(modulousOP)
```

Assignment 1 – report
Siddhant Barua
Cwid = 10439929

```
        # we all know y =mx + c this helps us find out the values of m and c from the
given sample space
        x1 = modulousOP[0][0]
        y1 = modulousOP[0][1]
        x2 = modulousOP[1][0]
        y2 = modulousOP[1][1]
        if (x2 - x1) == 0:
            m = mt.inf
        else:
            m = (y2 - y1) / (x2 - x1)
        C = y2 - (m * x2)

        modulousOP = [m, C]
        # we now find the location / coordinates where the line model intesects with
the values X,Y

        in_RAN_inlier = []
        RAN_inliers = 0
        setFlag = 1
        for i in range(0, row):
            for j in range(0, col):
                if Image_array[i][j] != 0:

                    # m = (y_{2} - y_{1}) / (x_{2} - x_{1}), and c = y_{2} - mx_{2}
                    yum = modulousOP[0]
                    see = modulousOP[1]
                    x = (j + (yum * i) - (yum * see)) / (1 + yum ** 2)  # y = mx + c
=> x= y-c/x
                    y = ((yum * j) + ((yum ** 2) * i) - ((yum ** 2) * col)) / (1 + yum
** 2) + see
                    dist = mt.sqrt(((x - j) ** 2) + ((y - i) ** 2))  # distance =
((x2-x1)^2 + (y2-y1)^2)^1/2

                    if dist < distance:
                        in_RAN_inlier.append([j, i])
                        RAN_inliers = RAN_inliers + 1  # # if the distance is nearrer
to the threshold
                        # distance it is considered an inlier and is pushed inside the
Ran_inliers variable

        if RAN_inliers / point > Master_ratop:
            if RAN_inliers > no_of_inliers:
                ratio = RAN_inliers / point  # here we find a ratio where the number
of inliers are averaged out by
                # the no of points that exist
                in_RAN_inlier.sort()
                in_len = len(in_RAN_inlier)  # here we find the length of the inlier
list
                if [ratio, [in_RAN_inlier[0], in_RAN_inlier[in_len - 1]]] not in
fitModuloOp:
                    if fitModuloOp == []:  # in case of an empty list
                        fitModuloOp.append([ratio, [in_RAN_inlier[0],
in_RAN_inlier[in_len - 1]]])
                    for k in fitModuloOp:
                        if in_RAN_inlier[0] in k[1] or in_RAN_inlier[in_len - 1] in
k[1] or ratio == k[0]:  # check
                            # if inliers exist at location [0] in the fitModuloOp
list, or if in location of 1 aor if
                            # the value at fitmodulo list is equal to the ratio
                            setFlag = 0
                            break
                    if (setFlag == 1):
                        fitModuloOp.append([ratio, [in_RAN_inlier[0],
```

Assignment 1 – report
Siddhant Barua
Cwid = 10439929

```
in_RAN_inlier[in_len - 1]]]])  # appends the
                        # ratio and the inliers at locations [0] and [in_len -1] into
the modulo list
        Iterations = Iterations - 1
    fitModuloOp.sort()  # sorts the modulo list

    draw_model = fitModuloOp

    no_of_points = 4  # declaring the number of lines to be drawn by ransaac algorithm
    i = len(fitModuloOp) - 1

    while no_of_points != 0:
        # iterate through to find the number of
        # points that match
        # this is to draw a line between the points i.e. a pair of xy coordinates (
(x,y), (x1,y1))

        point_loc = fitModuloOp[i][1]
        x1 = point_loc[0][0]
        y1 = point_loc[0][1]
        x2 = point_loc[1][0]
        y2 = point_loc[1][1]

        draw_Orig = ImageDraw.Draw(RANSAC_Iimage)
        draw_Orig.line((x1 - 1, y1 + 1, x1 + 1, y1 + 1), fill=255, width=3)
        draw_Orig.line((x1 - 1, y1 + 1, x1 - 1, y1 - 1), fill=255, width=3)
        draw_Orig.line((x1 + 1, y1 + 1, x1 + 1, y1 - 1), fill=255, width=3)
        draw_Orig.line((x1 - 1, y1 - 1, x1 + 1, y1 - 1), fill=255, width=3)
        draw_Orig.line((x2 - 1, y2 + 1, x2 + 1, y2 + 1), fill=255, width=3)
        draw_Orig.line((x2 - 1, y2 + 1, x2 - 1, y2 - 1), fill=255, width=3)
        draw_Orig.line((x2 + 1, y2 + 1, x2 + 1, y2 - 1), fill=255, width=3)
        draw_Orig.line((x2 - 1, y2 - 1, x2 + 1, y2 - 1), fill=255, width=3)

        draw_Orig.line((x1, y1, x2, y2), fill=255, width=3)

        draw_on_hessian = ImageDraw.Draw(RANSAC_Iimage)

        draw_on_hessian.line((x1 - 1, y1 + 1, x1 + 1, y1 + 1), fill=255, width=3)
        draw_on_hessian.line((x1 - 1, y1 + 1, x1 - 1, y1 - 1), fill=255, width=3)
        draw_on_hessian.line((x1 + 1, y1 + 1, x1 + 1, y1 - 1), fill=255, width=3)
        draw_on_hessian.line((x1 - 1, y1 - 1, x1 + 1, y1 - 1), fill=255, width=3)
        draw_on_hessian.line((x2 - 1, y2 + 1, x2 + 1, y2 + 1), fill=255, width=3)
        draw_on_hessian.line((x2 - 1, y2 + 1, x2 - 1, y2 - 1), fill=255, width=3)
        draw_on_hessian.line((x2 + 1, y2 + 1, x2 + 1, y2 - 1), fill=255, width=3)
        draw_on_hessian.line((x2 - 1, y2 - 1, x2 + 1, y2 - 1), fill=255, width=3)
        draw_on_hessian.line((x1, y1, x2, y2), fill=255, width=3)

        i = i - 1
        no_of_points = no_of_points - 1

    RANSAC_Iimage.show()
    RANSAC_Iimage.show()


main()
```

Assignment 1 – report
Siddhant Barua
Cwid = 10439929

<u>Notes on the Program:</u>

We first take the gaussian filter of variable size , in this case it is made out to be 5 X 5 in size, and we take the sigma value to be 1 and now this is convolved with the image, and then we use the sobel filters to threshold the Hessian determinant to get the edges of the image.

We then perform the hough transform, which is basically dependent on the bin space of the accumulator which we select to be a function of rows and columns, which is dependent on the image.

After hough transform we perform the ransac operation, where in we first find the sample space, then we select two points from the generated sample space, we then find the values of m and c in the equation y = mx +c. We then generate the line model best suitable for this case, we then find the location at which the line model intersects with the coordinate x and y.

Now we have a few functions in this program

Suppress: this performs the non-maximum suppression, where in the highest pixel value in the matrix or dimension [size, size] is considered the central value of that matrix and then the rest of the locations in the matrix are set to 0 , this is non maximum suppression

AddPad: this function first adds a padding of 0's on the borders of the image and then replicates the pixels at the left, right, top and bottom positions hence padding the image with replicated values. This makes sure that even if the filter is not able to cover the edge pixels, it is able to handle the edge pixels accordingly.

<u>Inferences and observations</u>

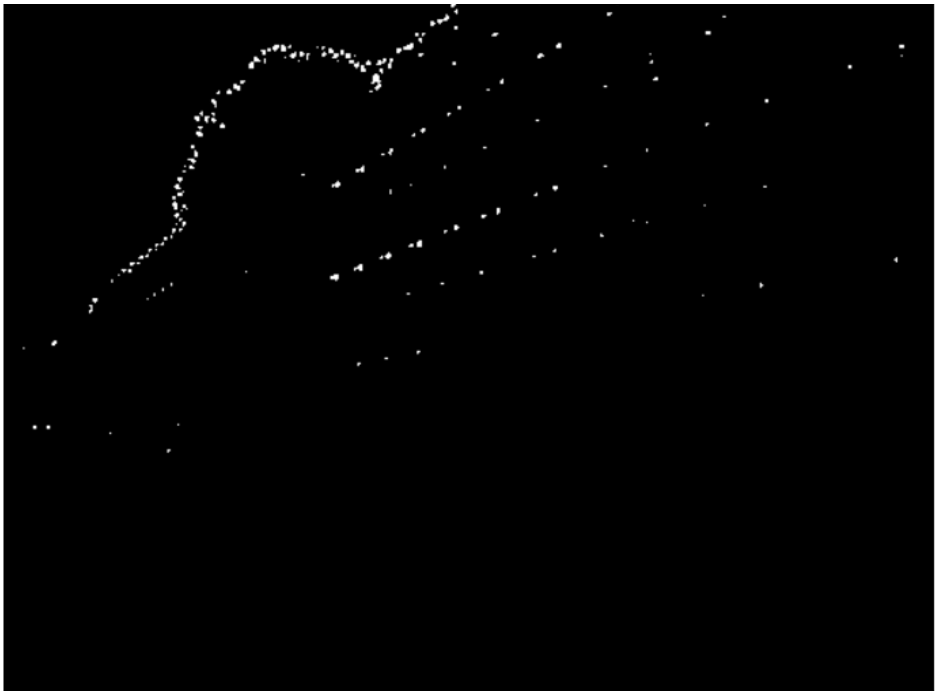
We notice that the hough transform output image is better than the ransac algorithm.
The lines are more desirable in the hough transform algorithm as compared to ransac.

Now lets take a look at the output images

Assignment 1 – report
Siddhant Barua
Cwid = 10439929
Original Image

Gauss_filter_Image

Assignment 1 – report
Siddhant Barua
Cwid = 10439929
Hessian Image



Hough transform Image(s)

Assignment 1 – report
Siddhant Barua
Cwid = 10439929
And

Assignment 1 – report
Siddhant Barua
Cwid = 10439929
Ransac operation Images

LEGEND : In this image the black lines represent the lines obtained through the hough transform and the white lines are the results of the RANSAC operation

Variable setting:

Here we have selected the filter to be of size 5 X 5 and the sigma value to be 1 and the above images are the generated outputs

For the Hough transform, the bin column space is varied based on the image provided and the bin row space is set to be 181 , this is done so that we can iterate from Theta value 0 through 180 in the loops. We then draw the points after we get the x,y coordinates

In the ransac algorithm we set the number of Iterations to be 420, distance threshold is set to 2 and then the number of inliers for line selection is set to 40. This obviously needs to be tuned in order to obtain more favorable lines.

In essence the ransac algorithm will give variable outputs as this is based on the tuning parameters as well as the random sample space that is generated. The RANSAC algorithm plots 4 lines that sometimes overlap , and the hough transform also plots 4 lines on the image , where in the bottom 2 lines in this case intersect, as during the edge detection process, the values on the lower part of the image are blurred out , due to this these points don’t exist for the voting process, hence we are unable to obtain 4 distinct lines. This is in theory.

END

Assignment 1 – report
Siddhant Barua
Cwid = 10439929