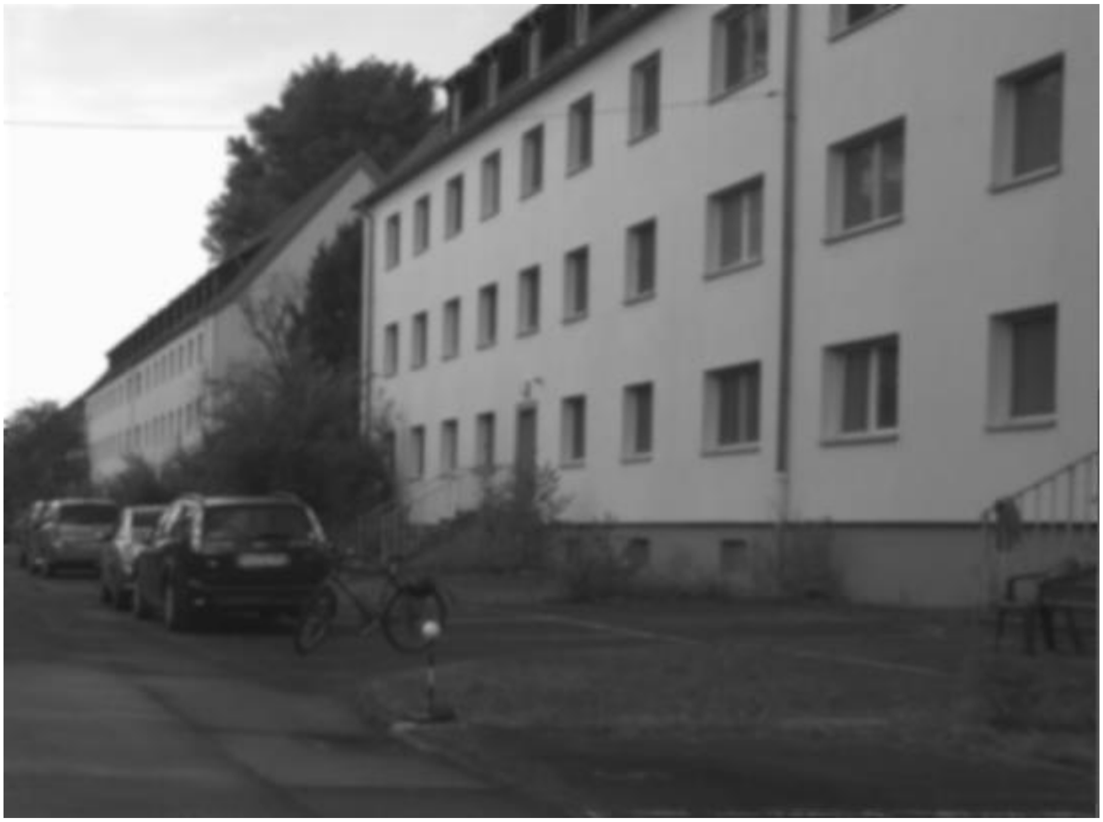
```
from PIL import Image, ImageDraw
import numpy as np
import random
import math

img1 = Image.open("road.png")
img0 = Image.open("road.png")
sig = 1

sobelderX = [1,2,1, 0,0,0, -1,-2,-1]
sobelderY = [1,0,-1, 2,0,-2, 1,0,-1]
def window3x3(pts, x, y):
    return [pts[x-1,y-1],pts[x,y-1],pts[x+1,y-1], pts[x-1,y],pts[x,y],pts[x+1,y], pts[x-1,y+1],pts[x,y+1],pts[x+1,y+1]]
def gaussianFilter(sigma,img,buf):
    #get the dimensions and pixels from our image
    width, height = img.size
    values = img.load()
    #we store the new pixels here
    gaussianImg = buf.load()
    #STUFF
    gaussianMat = np.zeros((5, 5), np.float64)
    for x in range(-2, 3):
        for y in range(-2, 3):
            #the formula
            gaussianMat[2+x, 2+y] = 1 / (np.exp( (x**2+y**2) / (2*sigma**2)) * 2 * np.pi * (sigma**2))
    #apply gaussian blur to pixels from earlier
    for x0 in range(2, width-3):
        for y0 in range(2, height-3):
            temp = 0
            for x1 in range(-2, 3):
                for y1 in range(-2, 3):
                    temp += values[x0+x1, y0+y1]*gaussianMat[x1+2][y1+2]
            gaussianImg[x0, y0] = int(np.ceil(temp))
    return gaussianImg
def sobelFilter(imgv, bufv, w,h):
    #\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/
    for i in range(w-1):
        for j in range(h-1):
            bufv[i,j] = 0
    #/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
    for x in range(2,w-2):
        for y in range(2,h-2):
            # horizontal, vertical = 0.0,0.0
            # horizontal +=(1*imgv[x-1,y-1])+(2*imgv[x,y-1])+(1*imgv[x+1,y-1])
            # horizontal +=(0)+(0)+(0)
            # horizontal +=(1*imgv[x-1,y+1])+(-2*imgv[x,y+1])+(-1*imgv[x+1,y+1])
            # vertical += (1*imgv[x-1,y-1])+(0)+(1*imgv[x+1,y-1])
            # vertical += (2*imgv[x-1,y])+(0)+(-2*imgv[x+1,y])
            # vertical += (1*imgv[x-1,y+1])+(0)+(-1*imgv[x+1,y+1])
            horizontal = sum([a*b for a,b in zip(window3x3(imgv,x,y),sobelderX)])
            vertical = sum([a*b for a,b in zip(window3x3(imgv,x,y),sobelderY)])
            bufv[x,y] = int(np.sqrt(vertical**2+horizontal**2))
    return bufv
def hessian(R, i1vals, i0vals, height, width):
    #hessian feature dectector
    feats = []
    #\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/
    for i in range(width-1):
        for j in range(height-1):
            i0vals[i,j] = 0
    #/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\/\
    for x in range(2, width-5):
        for y in range(2, height-2):
            #Ix = sum([a*b for a,b in zip(window3x3(i1vals,x,y),sobelderX)])
            #Iy = sum([a*b for a,b in zip(window3x3(i1vals,x,y),sobelderY)])
            #Ixx = Ix**2
            #Iyy = Iy**2
            #Ixy = Ix*Iy
            Ixx = i1vals[x+2,y] + i1vals[x,y] - 2*i1vals[x+1,y]
            Iyy = i1vals[x,y+2] + i1vals[x,y] - 2*i1vals[x,y+1]
            Ixy = i1vals[x,y+2] + (i1vals[x,y] - i1vals[x,y+1]) - i1vals[x+1,y]
            det = int((Ixx * Iyy) - (Ixy**2))
            if det > R:
                feats.append((x,y))
                i0vals[x,y] = det
            else:
                i0vals[x,y] = 0
    return feats
def ransac(features, s,t,d):
    inliers = []
    featuresCpy = features.copy()
    imgP = Image.open("hessian.png")
    draw = ImageDraw.Draw(imgP)
    xmax = imgP.size[0]
    for linez in range(1,5):
        maxinliers = []
```

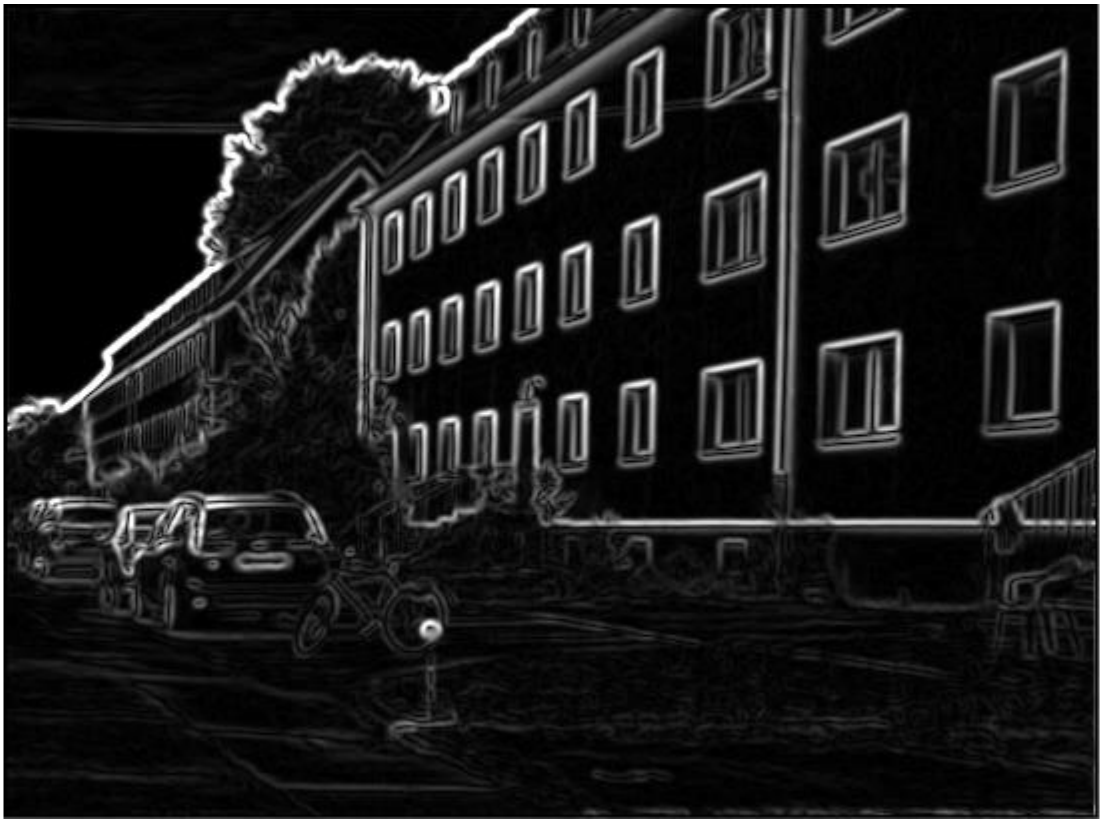
```
subset = []
maxi=10000
while maxi > 0 and len(inliers) < d:
    inliers = []
    if len(featuresCpy)<s:
        featuresCpy = features.copy()
    subset = random.sample(featuresCpy, s)
    ys = np.zeros((s,1),np.float64)
    xs = np.zeros((s,2),np.float64)
    index = 0
    for (theX,theY) in subset:
        xs[index][0] = theX
        xs[index][1] = 1
        ys[index][0] = theY
        index += 1
    try:
        MB = (np.linalg.inv(np.dot(xs.T,xs)).dot(xs.T)).dot(ys)
        for (a1,b1) in featuresCpy:
            if a1*MB[0][0] > b1-(MB[1][0]+t) and a1*MB[0][0] < b1-(MB[1][0]-t):
                inliers.append((a1,b1))
    except np.linalg.LinAlgError:
        #print('bad point pair', end=' ')
        pass
    # finally:
    #     if(maxi+1)%2000 == 0:
    #         print('.')
    if(len(inliers)>len(maxinliers)):
        maxinliers = inliers.copy()
        #print(len(inliers), end=' ')
```

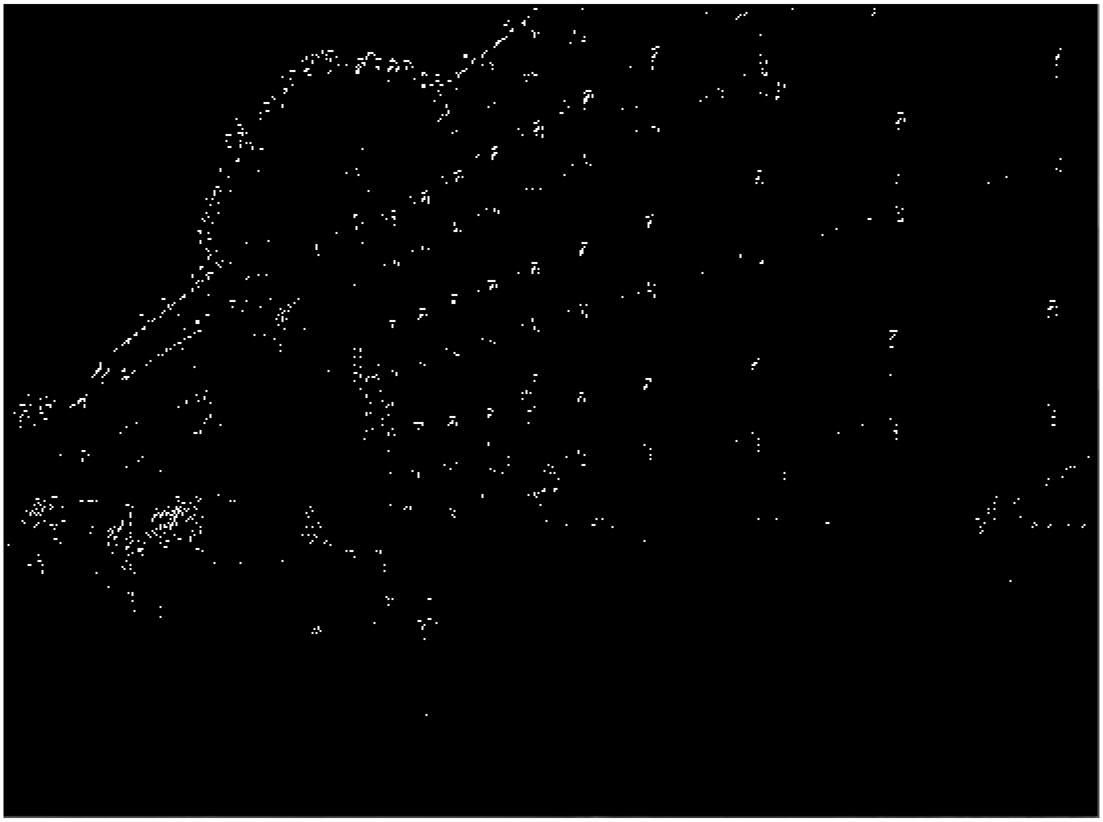
```
            maxi -=1
        #refit the line with all inliers and return it.
        by = np.zeros((len(maxinliers),1),np.float64)
        bx = np.zeros((len(maxinliers),2),np.float64)
        index = 0
        for (i,j) in maxinliers:
            bx[index][0] = i
            bx[index][1] = 1
            by[index][0] = j
            index += 1
        MB = (np.linalg.inv(np.dot(bx.T,bx)).dot(bx.T)).dot(by)
        print('%d/4 for part 2'%linez)
        draw.line((0,MB[1][0], xmax,xmax*MB[0][0]+MB[1][0]), fill=255)
        terminator = []
        for i1 in range(len(featuresCpy)):
            for item in maxinliers:
                if item[0] == featuresCpy[i1][0] and item[1] == featuresCpy[i1][1]:
                    terminator.insert(0,i1)
        for j1 in terminator:
            featuresCpy.pop(j1)
    imgP.save('ransac.png')
    return maxinliers
def hough(features,width,height):
    revp = Image.open('hessian.png')
    draw = ImageDraw.Draw(revp)
    diagonal = int(np.sqrt(width * width + height * height))
    houghResid = Image.fromarray(np.zeros((2*diagonal, 181), np.float64), "L")
    houghResid.save("hough.png")
    houghResid = Image.open("hough.png")
```

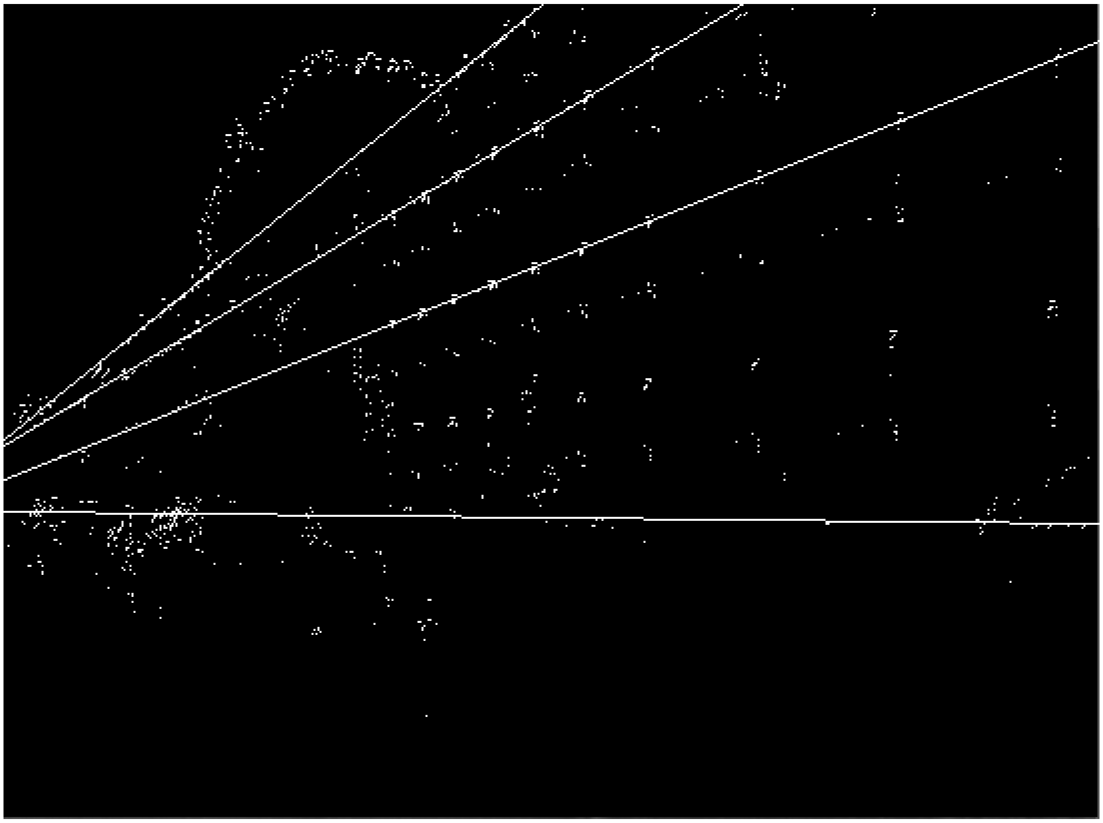
```
Hough = houghResid.load()
for pnt in features:
    x = pnt[0]
    y = pnt[1]
    for theta in range(0, 181):
        rho = x*math.cos(math.radians(theta-90)) + y*math.sin(math.radians(theta-90)) + diagonal
        rho = int(rho)
        Hough[theta,rho] = Hough[theta,rho] + 3
        if Hough[theta,rho] > 255:
            Hough[theta,rho] = 255
linez = [(0,0), (0,0), (0,0), (0,0)]
theta = 2
rho = 2
while theta < 178:
    while rho < 2*diagonal - 2:
        maxV = 0
        pnt = (0,0)
        for i in range(-2, 3):
            for j in range(-2, 3):
                currentV = Hough[theta+i, rho+j]
                if currentV > maxV:
                    maxV = currentV
                    pnt = (theta+i, rho+j)
        if maxV > Hough[linez[0]]:
            linez[3] = linez[2]
            linez[2] = linez[1]
            linez[1] = linez[0]
            linez[0] = pnt
        elif maxV > Hough[linez[1]]:
                linez[3] = linez[2]

                linez[2] = linez[1]

                linez[1] = pnt

            elif maxV > Hough[linez[2]]:

                linez[3] = linez[2]

                linez[2] = pnt

            elif maxV > Hough[linez[3]]:

                linez[3] = pnt

            rho += 5

        theta += 5

        rho = 2

    for lin in linez:

        r = lin[1]

        t = lin[0]

        draw.line((r*math.sin(t),0, width*(-1/math.tan(t))+r*math.sin(t),width), fill=255)

    houghResid.save("polar.png")

    revp.save('hough.png')




#part 1 i.e. the Preprocessing

bufValues = gaussianFilter(sig, img1, img0)

img0.save('gaussianfilter.png')

print('1/3 for part 1')

imgValues = sobelFilter(bufValues, img1.load(), img1.size[0],img1.size[1])

img1.save('sobelfilter.png')

print('2/3 for part 1')

feats = hessian(400, imgValues, bufValues, img1.size[1], img1.size[0])

img0.save('hessian.png')
```

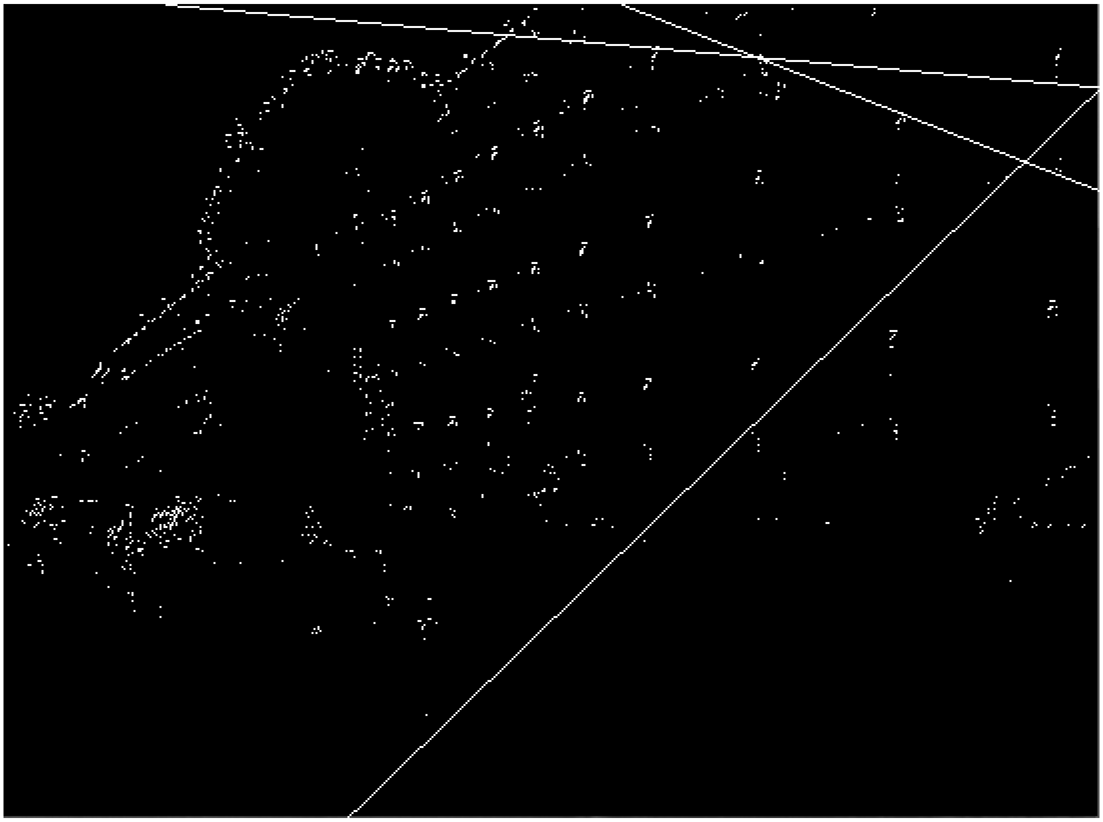
```
print('3/3 for part 1')
#part 2 i.e. the RANSAC
ransac(feats, 2, int(np.ceil(1.95959179423*sig)), 140)
#part 3 i.e. the Hough Transform
hough(feats, img1.size[0],img1.size[1])


print('1/1 for part 3\ndid it!')
'''Submit a zip file containing:
1) a pdf file with the source code (excluding libraries), the resulting images and a brief explanation of the implementation,
2) the code,
3) the output images.'''
```











Brief description =

For part 1 the gaussian filter, I applied gaussian blur then using my sobel derivates and window3x3 function I got the sobel filter which I used to get the hessian by taking determinate which I used for collecting the features.

For part 2 I made a copy of the features then collected the inliers by taking a subset of the copy of the features and least squares line fitting. When I got the line that yielded the maximum number of inliers I removed those inliers and repeated for the next 3 lines then drew the 4 lines ontop of my hessian image.

For part 3 I followed the formulas for the Hough transformation to get the polar transformation and converted rho and theta back into cartesian coordinates to get the lines for my final hough image.