```
from PIL import Image
import cv2
import copy
import math
from scipy.signal import sepfir2d
import matplotlib.pyplot as plt
import numpy as np
import scipy.stats as st
import random
import operator


sobelkernx_base = [
[-1, 0, 1],
[-2, 0, 2],
[-1, 0, 1]
]

sobelkerny_base = [
[-1, -2, -1],
[0, 0, 0],
[1, 2, 1]
]

truth = [[]]


'''returns a 2d gaussian kernel'''
def gkern(len=5, nsig=1):
lim = len//2 + (len % 2)/2
x = np.linspace(-lim, lim, len+1)
kern1d = np.diff(st.norm.cdf(x))
kern2d = np.outer(kern1d, kern1d)
return kern2d/kern2d.sum()


'''
Turns from width*height*3 to just width*height
'''
def fix_from_png(img):
img_fixed = []
for i in range(0, img.shape[0]):
img_row = []
for j in range(0, img.shape[1]):
img_row.append(img[i][j][0])
img_fixed.append(img_row)
return img_fixed
```

```
def isDifferent(img1, img2):
return not img1 == img2

def divFilt(filt, div):
new_filt = []
for row in filt:
new_row = []
for item in row:
new_row.append(item/div)
new_filt.append(new_row)
return new_filt

def gaussian_filt(img, filt):
height = len(img)
width = len(img[0])
filtered_img = np.zeros((height, width))
half = math.floor(len(filt)/2)
for i in range(0, height-1):
width = len(img[i])
for j in range(0, width-1):
for x in range(-half, half):
for y in range(-half, half):
if(i+x > 0 and i+x < height and j+y > 0 and j+y < width):
new_val = filtered_img[i][j]+filt[x+half][y+half]*img[i+x][j+y]
if new_val < -255:
new_val = -255
if new_val > 255:
new_val = 255
filtered_img[i][j] = new_val
return filtered_img

def sobel(img, kernx, kerny):
rows = len(img)
col = len(img[0])
mag = np.zeros((rows, col))
S1 = np.zeros((rows, col))
S2 = np.zeros((rows, col))


for i in range(1, rows-2):
for j in range(1, col-2):
S1[i][j] += kernx[0][0]*img[i-1][j-1]
S2[i][j] += kerny[0][0]*img[i-1][j-1]

S1[i][j] += kernx[0][1]*img[i-1][j]
S2[i][j] += kerny[0][1]*img[i-1][j]
```

```
S1[i][j] += kernx[0][2]*img[i-1][j+1]
S2[i][j] += kerny[0][2]*img[i-1][j+1]

S1[i][j] += kernx[1][0]*img[i][j-1]
S2[i][j] += kerny[1][0]*img[i][j-1]

S1[i][j] += kernx[1][1]*img[i][j]
S2[i][j] += kerny[1][1]*img[i][j]

S1[i][j] += kernx[1][2]*img[i][j+1]
S2[i][j] += kerny[1][2]*img[i][j+1]

S1[i][j] += kernx[2][0]*img[i+1][j-1]
S2[i][j] += kerny[2][0]*img[i+1][j-1]

S1[i][j] += kernx[2][1]*img[i+1][j]
S2[i][j] += kerny[2][1]*img[i+1][j]

S1[i][j] += kernx[2][2]*img[i+1][j+1]
S2[i][j] += kerny[2][2]*img[i+1][j+1]

mag[i+1][j+1] = math.sqrt(S1[i][j]**2 + S2[i][j]**2)
return mag


def hessian(img):
imgnd = np.asarray(img)
img_grad = np.gradient(img)
hess = np.empty((imgnd.ndim, imgnd.ndim) + imgnd.shape, dtype=imgnd.dtype)
for k, grad_k in enumerate(img_grad):
tmp_grad = np.gradient(grad_k)
for l, grad_kl in enumerate(tmp_grad):
hess[k,l,:,:] = grad_kl
return hess


def percentSame(img1, img2):
height = len(img1)
width = len(img1[0])
div_factor = float(width * height)
cnt = 0
for i in range(height):
for j in range(width) :
if img1[i][j] == img2[i][j]:
cnt=cnt+1
return cnt/div_factor * 100.0
```

```
def sobel_fix(sobel):
for row in sobel:
for member in row:
member = abs(member)

'''
takes in the 4x4 matrix that contains all hessian data
returns hessian determinant values for each pixel
'''
def hesdet(hes):
row = len(hes[0][0])
col = len(hes[0][0][0])
hesd = np.zeros((row,col))
for i in range(row):
for j in range(col):
hesd[i][j] = hes[0][0][i][j]* hes[1][1][i][j]-hes[0][1][i][j]*hes[1][0][i][j]
return hesd


def nonmaxsup(hes):
# print(np.asarray(hes_g).shape)
row = len(hes)
col = len(hes[0])
output = np.zeros((row, col))
#print(hes_g[0][34][433]," ",hes_g[1][34][433])
for i in range(1,row-1):
for j in range(1,col-1):
neighbor = [hes[i-1][j-1], hes[i-1][j], hes[i-1][j+1]  ,
hes[i][j-1], hes[i][j], hes[i][j+1],
hes[i+1][j-1], hes[i+1][j], hes[i+1][j+1]]
if (max(neighbor) == hes[i][j] and hes[i][j] != 0):
output[i][j] = 255
return output

def localize_hess(hes, kern):
row = len(hes[0][0])
col = len(hes[0][0][0])
half = math.floor(len(kern)/2)
hes_new = np.zeros((len(hes),len(hes[:]),row,col))
for i in range(half, row-half):
for j in range(half, col-half):
for x in range(-half, half):
for y in range(-half, half):
hes_new[0][0][i][j] += hes[0][0][i+x][j+y]*kern[x+half][y+half]
hes_new[0][1][i][j] += hes[0][1][i+x][j+y]*kern[x+half][y+half]
hes_new[1][0][i][j] += hes[1][0][i+x][j+y]*kern[x+half][y+half]
```

```
hes_new[1][1][i][j] += hes[1][1][i+x][j+y]*kern[x+half][y+half]
return hes_new

def sobel_companion(sobel, thresh=70):
height = len(sobel)
width = len(sobel[0])
for i in range(height):
for j in range(width):
if(sobel[i][j] <= thresh):
new_sobel = 0
else:
new_sobel = sobel[i][j]
return new_sobel


def toImage(img):
img_arr = np.asarray(img)
return Image.fromarray(img_arr.astype('uint8'))

def twoTruePoints(img):
row = len(img) - 1
col = len(img[0]) - 1
first_pointx = 0
first_pointy = 0
second_pointx = 0
second_pointy = 0
while img[first_pointx][first_pointy] != 255:
first_pointx = random.randint(0, row)
first_pointy = random.randint(0, col)
while img[second_pointx][second_pointy] != 255:
second_pointx = random.randint(0,row)
second_pointy = random.randint(0, col)
return ( (first_pointx, first_pointy), (second_pointx, second_pointy))

def next_pointx(curpoint, slope, intercept, dx):
return (curpoint[0]+1,int(slope*(curpoint[0]+1)+intercept))

def next_pointy(curpoint, slope, intercept, dx):
return (int(slope*(curpoint[1]+1)+intercept),curpoint[1]+1)

def acc_thresh(img,firstpoint, slope, endbounds, thresh_arr,intercept, isx):
offset=math.floor(len(thresh_arr)/2)
cur_point = copy.deepcopy(firstpoint)
acc = 0
print(endbounds)
truth = np.ones((endbounds[0], endbounds[1]))
#offset it so that we can calculate inliers in bounds
```

```
while cur_point[0] < offset+1 and cur_point[1] < offset+1:
if isx:
cur_point = next_pointx(cur_point, slope, intercept, 1)
else:
cur_point = next_pointy(cur_point, slope, intercept, 1)
while cur_point[0]+offset <= endbounds[0] - 1 and cur_point[1]+offset <= 
endbounds[1] - 1 and cur_point[0]+offset >= 0 and cur_point[1]+offset >=0:
i = cur_point[0]
j = cur_point[1]
# print("i = " ,i, " j = ", j)
for x in range(-offset, offset):
for y in range(-offset, offset):
acc+=img[i+x][j+y]*thresh_arr[x][y]*truth[i+x][j+y]
truth[i+x][j+y] = 0
if isx:
cur_point = next_pointx(cur_point, slope, intercept, offset)
else:
cur_point = next_pointy(cur_point, slope, intercept, offset)
return acc
def ransac(img, N, numinlier, threshold):
row = len(img)
col = len(img[0])
thresh_arr = np.ones((threshold, threshold))
line_colletion = {}
while N != 0:
di, df = twoTruePoints(img)
if df[1]-di[1] == 0:
slope = math.inf
else:
slope = float(df[0]-di[0])/float(df[1]-di[1])
b = int(df[0] - slope * df[1] )
if abs(slope) > 1:
b = int(-b/slope)
first_point = (b, 0)
slope = 1 / slope
found_line = acc_thresh(img,first_point, slope, (row, col), thresh_arr, b, 
False)
function_str = "x = " + str(slope)+" y + "+str(b)
else:
first_point = (0, b)
found_line = acc_thresh(img,first_point, slope, (row, col), thresh_arr, b, 
True)
function_str = "y = " + str(slope)+" x + "+str(b)
if found_line > numinlier:
line_colletion[function_str] = found_line
N = N-1
return line_colletion
```

```
def draw_ransac(img, key):
linargs = key.split(" ")
isx = linargs[0] == "y"
slope = float(linargs[2])
intercept = int(linargs[5])
rows = len(img)
cols = len(img[0])
retimg = img
if isx:
cur_point = (0, intercept)
else:
cur_point = (intercept, 0)
while cur_point[0] < rows - 1 and cur_point[1] < cols - 1 and cur_point[0] >= 0
and cur_point[1] >=0:
retimg[cur_point[0]][cur_point[1]] = 255
if isx:
cur_point= next_pointx(cur_point, slope, intercept, 1)
else:
cur_point= next_pointy(cur_point, slope, intercept, 1)
return retimg

def all_feature_points(img):
feature_list = []
row = len(img)
halfrow = row//2
col = len(img[0])
halfcol = row//2
for i in range(row):
for j in range(col):
if img[i][j] == 255:
feature_list.append((i-halfrow, j-halfcol))
return feature_list
def hough(img):
thetas = np.deg2rad(np.arange(-90., 90.))
width, height = img.shape
diag_len = int(np.ceil(np.sqrt(width**2 + height**2)))
rhos = np.linspace(-diag_len, diag_len, diag_len*2)
cos_t = np.cos(thetas)
sin_t = np.sin(thetas)
num_thetas = len(thetas)

acc = np.zeros((2*diag_len, num_thetas), dtype=np.uint64)
y_idxs, x_idxs = np.nonzero(img)

for i in range(len(x_idxs)):
```

```
x = x_idxs[i]
y = y_idxs[i]

for t_idx in range(num_thetas):
rho = round(x*cos_t[t_idx]+y*sin_t[t_idx]) + diag_len
# print("rho = ", rho)
acc[int(rho)][int(t_idx)] += 1

return acc, thetas, rhos

def draw_hough(img,m,firstpoint, isx):
row = len(img)
col = len(img[0])
cur_point = firstpoint
print("First point = ", firstpoint)
retimg = img
if isx:
b = int(first_point[1])
while cur_point[0] < row and cur_point[1] < col and cur_point[0]>=0 and
cur_point[1]>=0:
print("x point = ", cur_point[0], " y point = ", cur_point[1])
retimg[int(cur_point[0])][int(cur_point[1])] = 255
cur_point = next_pointx(cur_point,m,b,1)
else:
b = int(first_point[0])
while cur_point[0] < row and cur_point[1] < col and cur_point[0]>=0 and
cur_point[1]>=0:
print("x point = ", cur_point[0], " y point = ", cur_point[1])
retimg[int(cur_point[0])][int(cur_point[1])] = 255
cur_point = next_pointy(cur_point, m, b, 1)
return retimg



img = cv2.imread('road.png')
truth = np.ones((len(img), len(img[0])))

img = fix_from_png(img)

kernel = gkern()
print(kernel)

img_gauss = gaussian_filt(img, kernel)

img_arr = np.asarray(img)
img_gauss_arr = np.asarray(img_gauss)
```

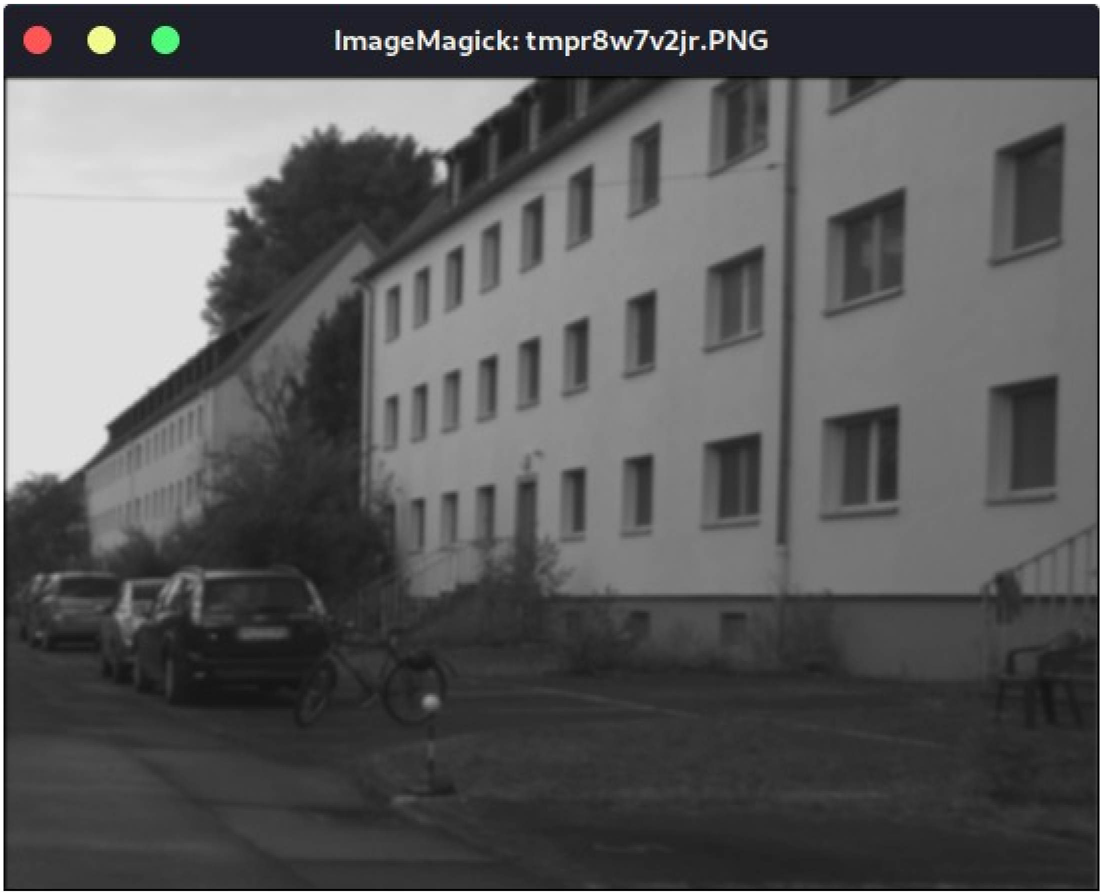
```
ime = Image.fromarray(img_arr)
ime_guass = Image.fromarray(img_gauss_arr)

ime.show()
ime_guass.show()

sobelkernx = sobelkernx_base
sobelkerny = sobelkerny_base

print(sobelkernx)
print(sobelkerny)

sobelimg = sobel(img_gauss, sobelkernx, sobelkerny)

toImage(sobelimg).show()

hes = hessian(sobelimg)
print(hes)
print(hes.shape)

hesl=localize_hess(hes, gkern(len=3))

print(hesl.shape)

hesd = hesdet(hesl)


hesdcop = copy.deepcopy(hesd)
print(hesd)
print(hesd.shape)
thresh = 40.6
super_threshold_indices = abs(hesd) < thresh
hesd[super_threshold_indices] = 0
super_threshold_indices = abs(hesd) > 0
hesd[super_threshold_indices] = 255
hesg = nonmaxsup(hesd)
toImage(hesg).show()
#hesd = copy.deepcopy(hesdcop)

#two_points = twoTruePoints(hesd)
#print(two_points)
#print("first point value = ", hesd[two_points[0][0]][two_points[0][1]], "
second point value = ", hesd[two_points[1][0]][two_points[1][1]])
ransac_dict = ransac(hesg, 1000, 5000, 3)
four_highest = dict(sorted(ransac_dict.items(), key=operator.itemgetter(1),
reverse=True)[:4])
print(four_highest)
```

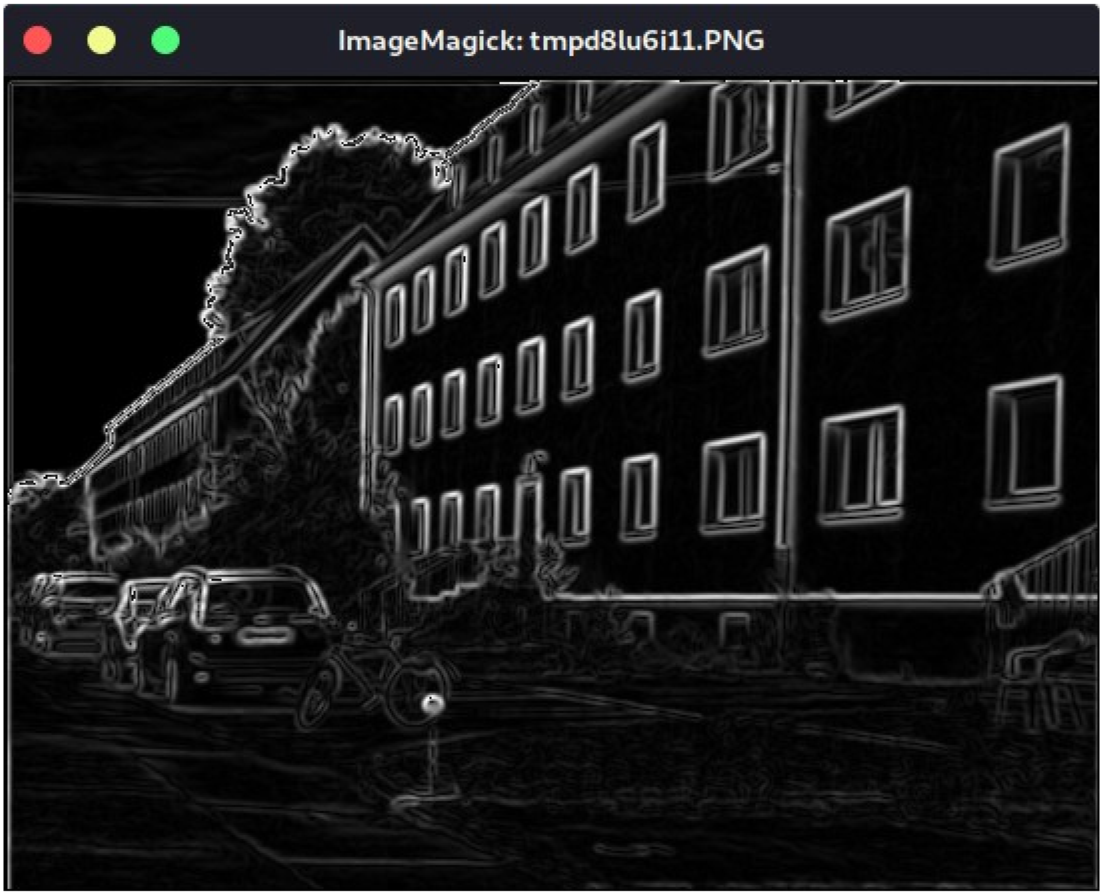
```
randraw = copy.deepcopy(img)
for key in four_highest:
ranimg = draw_ransac(randraw,key)
toImage(ranimg).show()
acc, thetas, rhos = hough(hesg)
houghimg = copy.deepcopy(img)
idx = np.argmax(acc)
print(idx)
rho = rhos[int(idx / acc.shape[1])]
theta = thetas[int(idx % acc.shape[1])]
if np.sin(theta) == 0:
m=math.inf
else:
m = -np.cos(theta) / np.sin(theta)
b = rho / np.sin(theta)
if abs(m) > 1:
b = int(-b/m)
first_point = (b, 0)
outhough = draw_hough(houghimg,m,first_point,False)
else:
first_point = (0, b)
outhough = draw_hough(houghimg,m,first_point,True)
toImage(outhough).show()
```

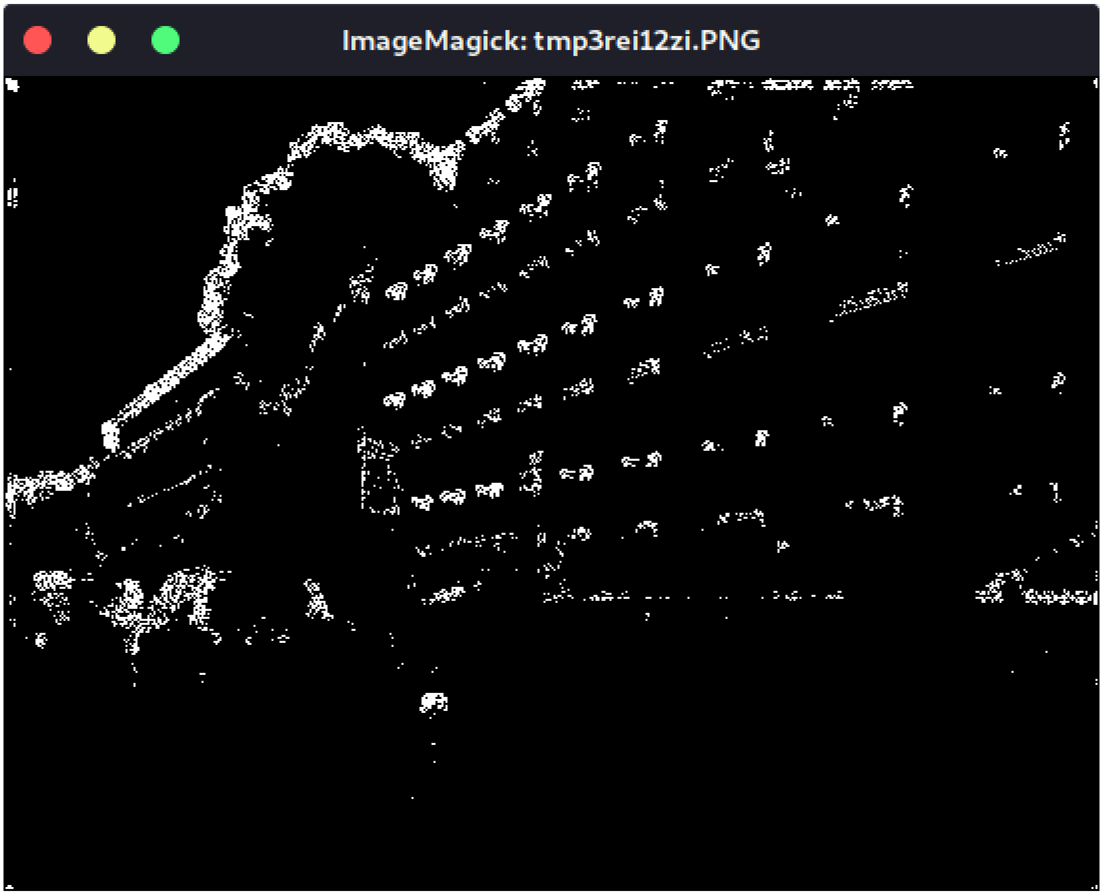
So for a quick description of overview, the first thing that occurs in this python code is a Gaussian filter with a kernel made at runtime in order to take out noise. After that, a Sobel filter with a predefined filter was applied to find the edges for the rest of this. The Hessian was then calculated for every single point on the image, and its determinant calculated for each point. In addition, it also has a gaussian kernel applied to it such that nearby points can be better correlated. After that, a threshold was set at 40.6 (found empirically) to allow points above that number.

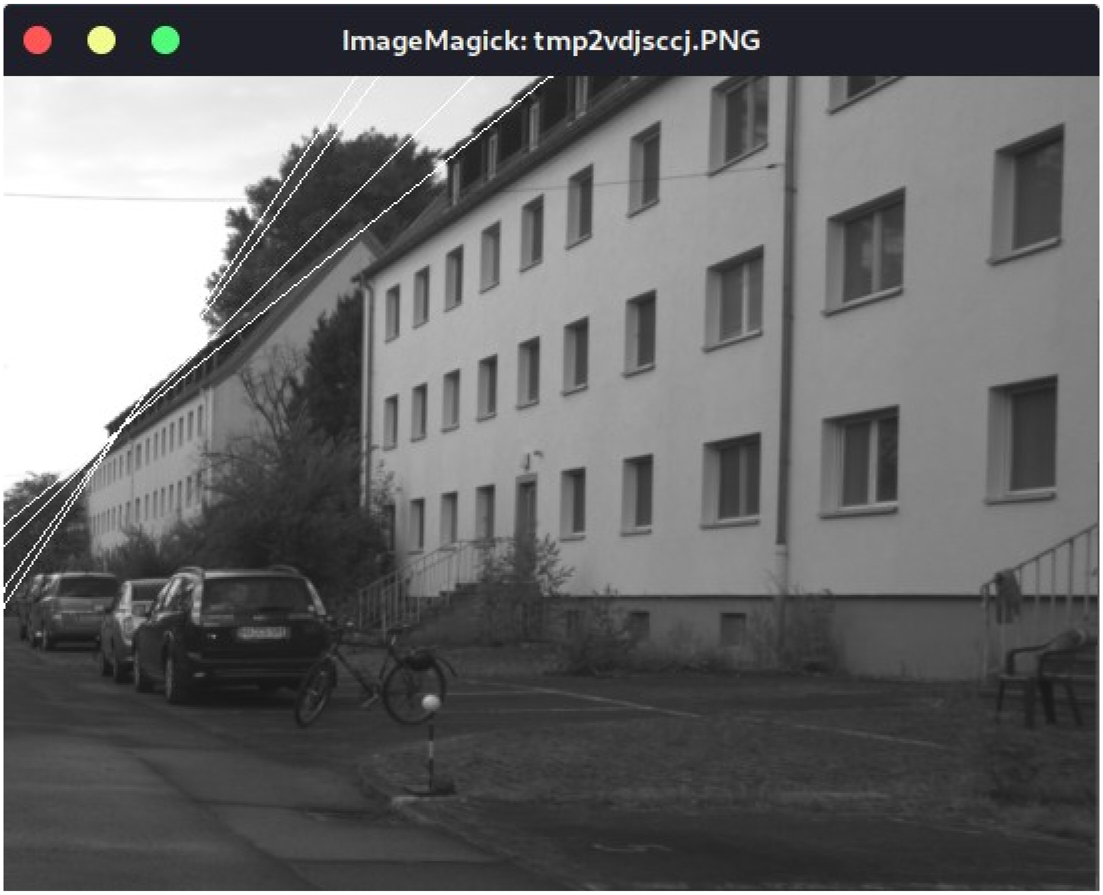
After that, a RANSAC was done, such that two sets of random points were decided, and the number of points along that line within a certain threshold were calculated. After a certain number of iterations were found (1000 in this case), the top 4 were chosen for being the strongest supported lines.

The Hough detector uses the standard hough transform and the transform. It goes through each of the found points in the Hessian, then accumulates whenever a certain theta rho pair is reached. Unfortunately, I ran out of time before being able to plot 4 lines, so as it stands there is still only one line in the final plot with the Hough transform.


ImageMagick: tmpr8w7v2jr.PNG


ImageMagick: tmpd8lu6i11.PNG


ImageMagick: tmp3rei12zi.PNG


ImageMagick: tmp2vdjsccj.PNG