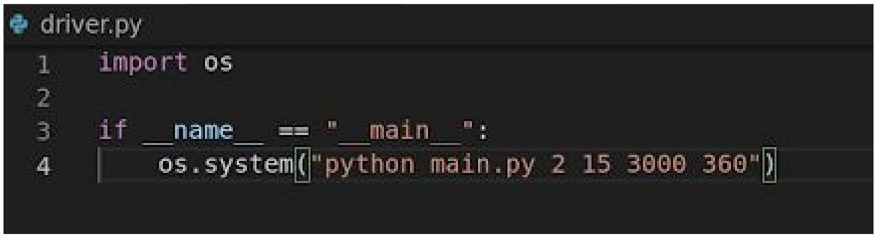
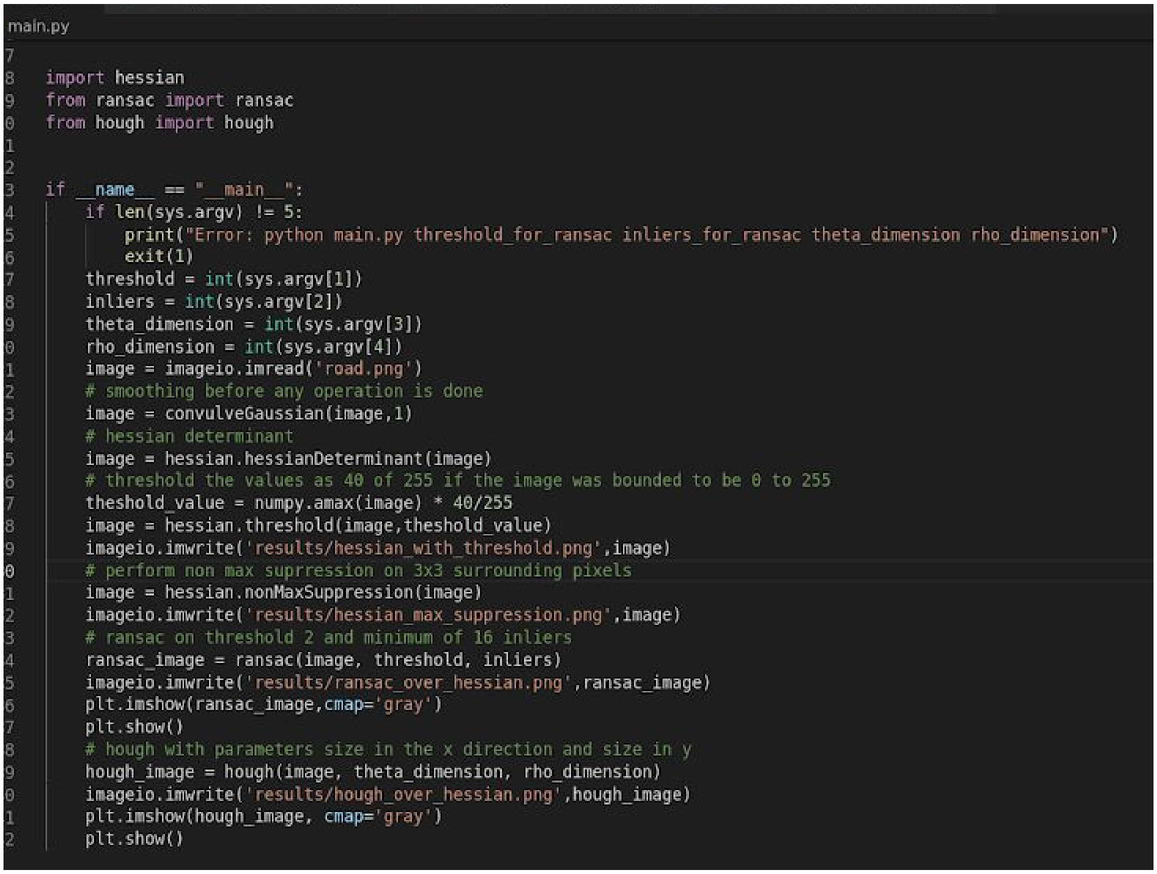
Vincent Lee
I pledge my honor that I have abided by the Stevens Honor System. VL

Driver.py runs the code parameterized with the width of the line for ransac as 2 and number of minimum inliers as 15, the number of dimensions for theta and rho in the hough transform.



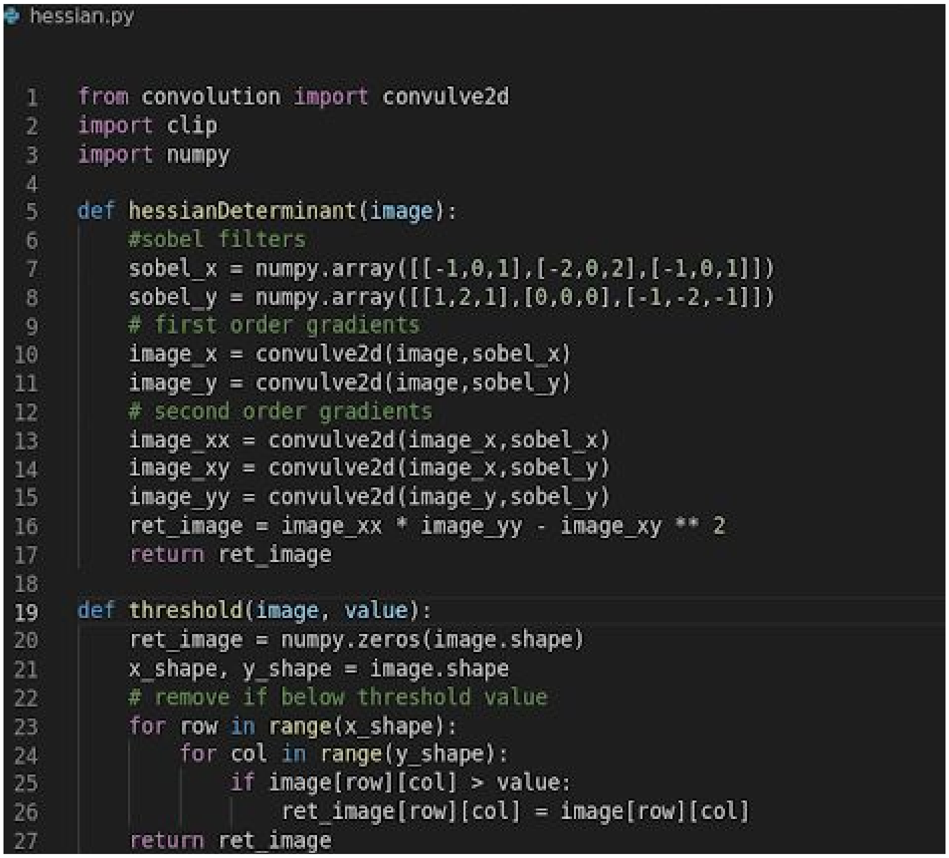
```python
import os

if __name__ == "__main__":
    os.system("python main.py 2 15 3000 360")
```

The main.py file runs all the code. It uses code from hw1 such as gaussian blur and convulution. It reads in the road.png, smoothes it, calls the hessian determinant function, thresholding function(and saves it), non_max_suppression(and saves it), ransac, and hough on the image. It shows the ransac and hough overlayed on the non_max_suppression hessian and saves the other results in the results folder. It thresholds values that are the equivalent of 40/255 in the hessian image since the hessian image has large values.

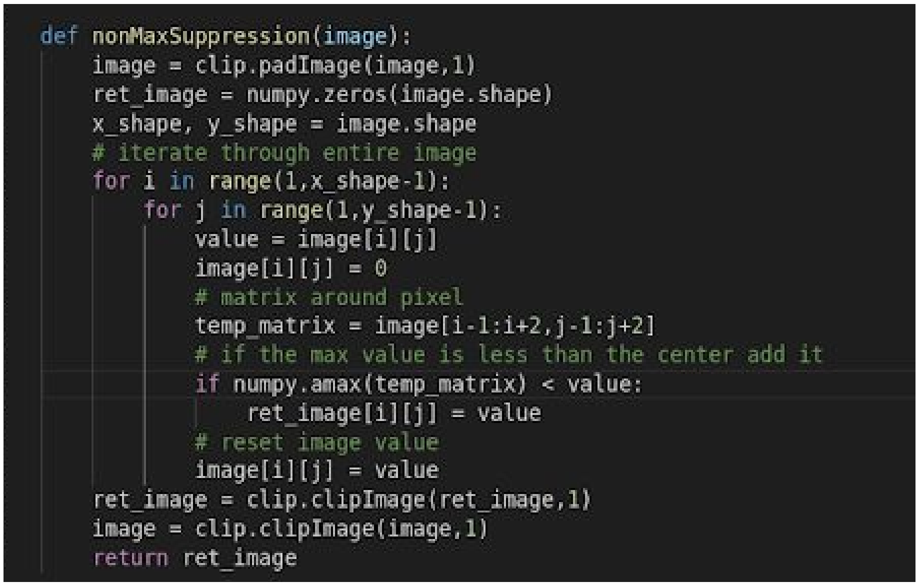


```python

import hessian
from ransac import ransac
from hough import hough


if __name__ == "__main__":
    if len(sys.argv) != 5:
        print("Error: python main.py threshold_for_ransac inliers_for_ransac theta_dimension rho_dimension")
        exit(1)
    threshold = int(sys.argv[1])
    inliers = int(sys.argv[2])
    theta_dimension = int(sys.argv[3])
    rho_dimension = int(sys.argv[4])
    image = imageio.imread('road.png')
    # smoothing before any operation is done
    image = convulveGaussian(image,1)
    # hessian determinant
    image = hessian.hessianDeterminant(image)
    # threshold the values as 40 of 255 if the image was bounded to be 0 to 255
    theshold_value = numpy.amax(image) * 40/255
    image = hessian.threshold(image,theshold_value)
    imageio.imwrite('results/hessian_with_threshold.png',image)
    # perform non max suprression on 3x3 surrounding pixels
    image = hessian.nonMaxSuppression(image)
    imageio.imwrite('results/hessian_max_suppression.png',image)
    # ransac on threshold 2 and minimum of 16 inliers
    ransac_image = ransac(image, threshold, inliers)
    imageio.imwrite('results/ransac_over_hessian.png',ransac_image)
    plt.imshow(ransac_image,cmap='gray')
    plt.show()
    # hough with parameters size in the x direction and size in y
    hough_image = hough(image, theta_dimension, rho_dimension)
    imageio.imwrite('results/hough_over_hessian.png',hough_image)
    plt.imshow(hough_image, cmap='gray')
    plt.show()
```

Hessian.py does the hessian functions. The determinant is straightforward from the notes done through sobel filters. Thresholding is removing pixels less than the specified value. Non max suppression iterates over 3x3 blocks to find the maximum values.



```python
from convolution import convulve2d
import clip
import numpy

def hessianDeterminant(image):
    #sobel filters
    sobel_x = numpy.array([[-1,0,1],[-2,0,2],[-1,0,1]])
    sobel_y = numpy.array([[1,2,1],[0,0,0],[-1,-2,-1]])
    # first order gradients
    image_x = convulve2d(image,sobel_x)
    image_y = convulve2d(image,sobel_y)
    # second order gradients
    image_xx = convulve2d(image_x,sobel_x)
    image_xy = convulve2d(image_x,sobel_y)
    image_yy = convulve2d(image_y,sobel_y)
    ret_image = image_xx * image_yy - image_xy ** 2
    return ret_image

def threshold(image, value):
    ret_image = numpy.zeros(image.shape)
    x_shape, y_shape = image.shape
    # remove if below threshold value
    for row in range(x_shape):
        for col in range(y_shape):
            if image[row][col] > value:
                ret_image[row][col] = image[row][col]
    return ret_image
```

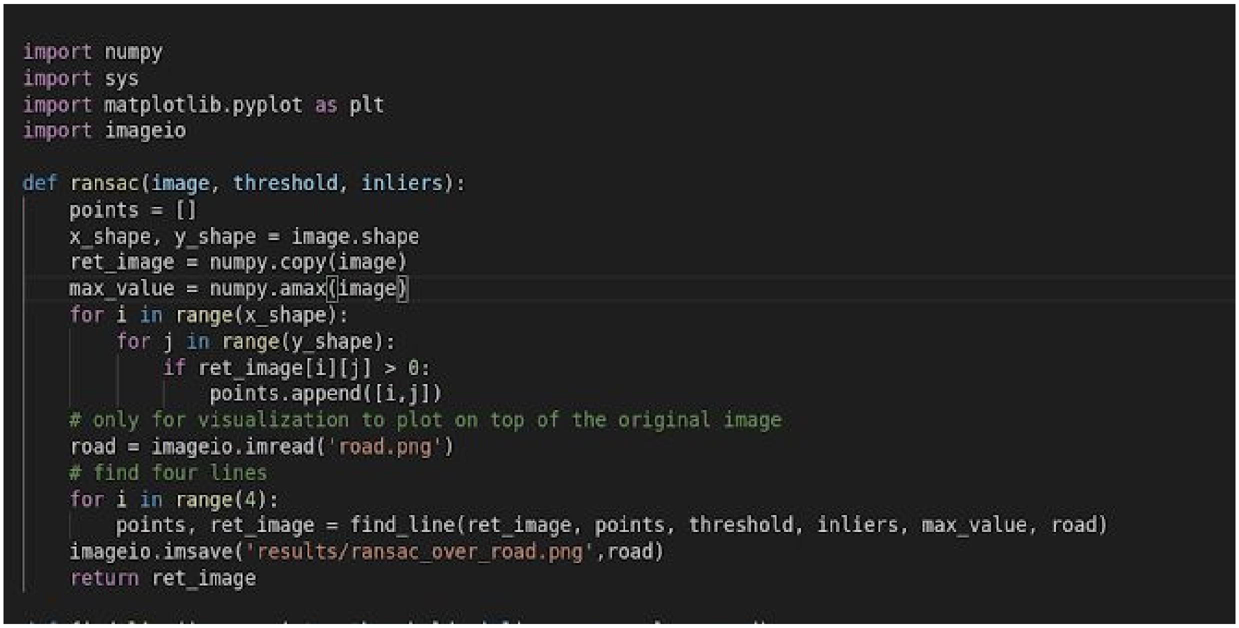


```python
def nonMaxSuppression(image):
    image = clip.padImage(image,1)
    ret_image = numpy.zeros(image.shape)
    x_shape, y_shape = image.shape
    # iterate through entire image
    for i in range(1,x_shape-1):
        for j in range(1,y_shape-1):
            value = image[i][j]
            image[i][j] = 0
            # matrix around pixel
            temp_matrix = image[i-1:i+2,j-1:j+2]
            # if the max value is less than the center add it
            if numpy.amax(temp_matrix) < value:
                ret_image[i][j] = value
            # reset image value
            image[i][j] = value
    ret_image = clip.clipImage(ret_image,1)
    image = clip.clipImage(image,1)
    return ret_image
```

Hessian with thresholding

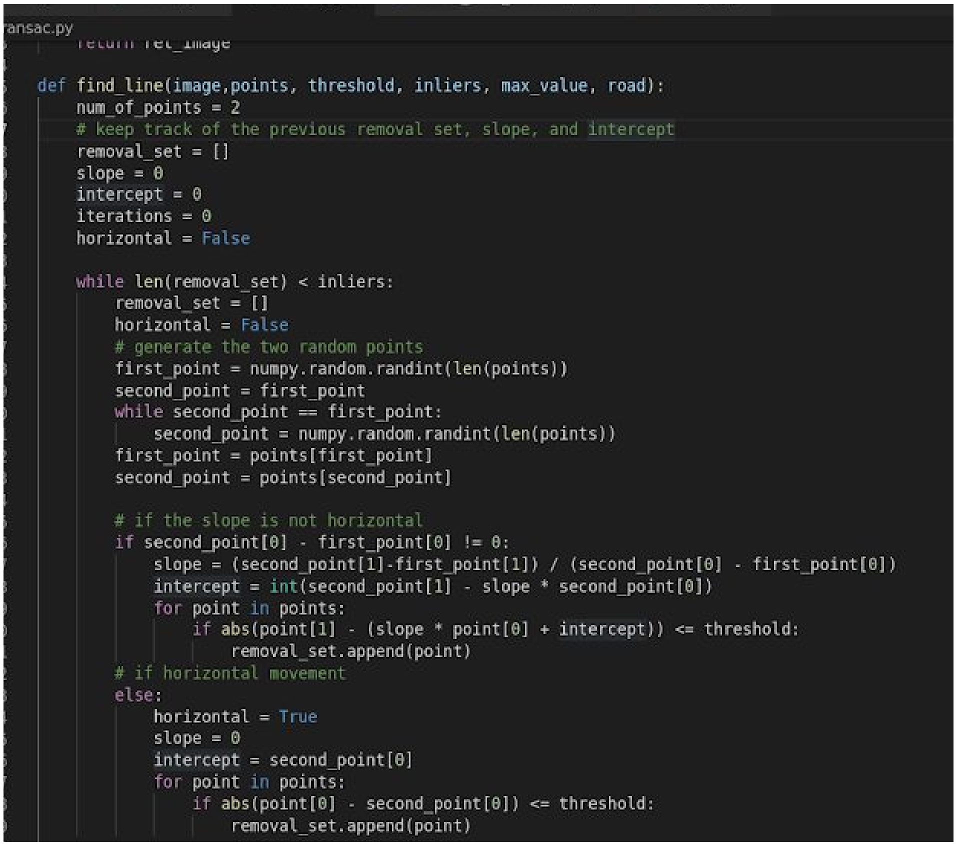
Hessian with non max_suppression

Ransac.py is where ransac is done. First it looks through all of the non_max_suppression image and gets all values greater than 0 as points, it then calls the draw lines function 4 times passing in the points and two images to overlay them on.

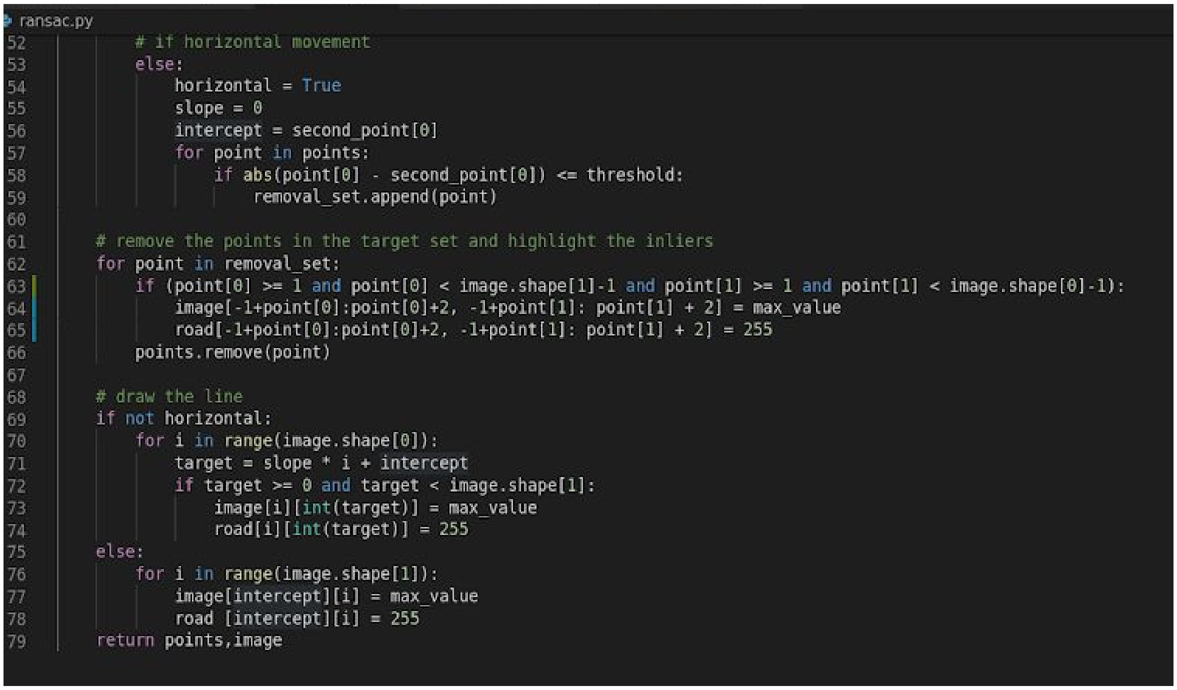


```
import numpy
import sys
import matplotlib.pyplot as plt
import imageio

def ransac(image, threshold, inliers):
    points = []
    x_shape, y_shape = image.shape
    ret_image = numpy.copy(image)
    max_value = numpy.amax(image)
    for i in range(x_shape):
        for j in range(y_shape):
            if ret_image[i][j] > 0:
                points.append([i,j])
    # only for visualization to plot on top of the original image
    road = imageio.imread('road.png')
    # find four lines
    for i in range(4):
        points, ret_image = find_line(ret_image, points, threshold, inliers, max_value, road)
    imageio.imsave('results/ransac_over_road.png',road)
    return ret_image
```

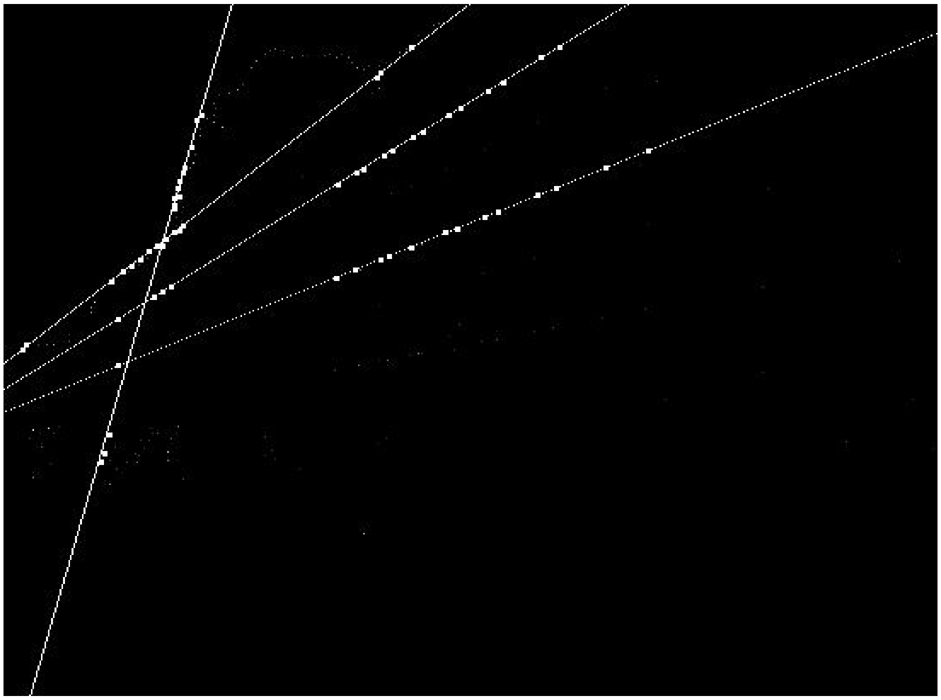
The draw lines function randomly gets two points creates a line through them and sees if a given number of inliers falls in it, if so it removes those from the points the set and draws a line and draws a 3x3 box around the points. If not it gets another set of two points until it does. Note it does not draw a box around the edge pixels that contribute to it as that would make the 3x3 box fall out of bounds. If the change between the vertical dimension is 0 it means the line is horizontal. Ransac draws the box and lines as max_value of the image since the image value is the hessian determinant which is greater than 255.



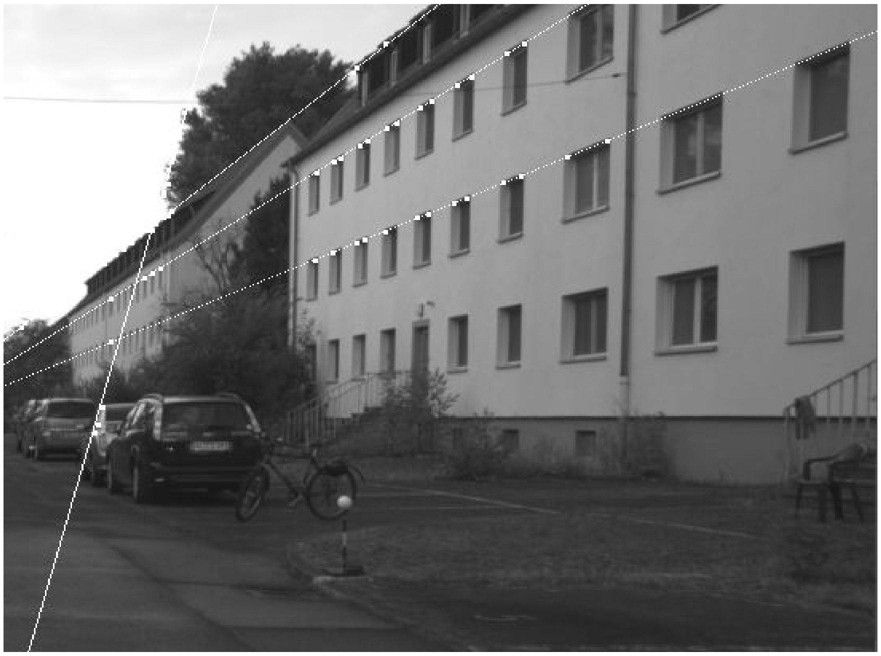
```python
    return ret_image

def find_line(image,points, threshold, inliers, max_value, road):
    num_of_points = 2
    # keep track of the previous removal set, slope, and intercept
    removal_set = []
    slope = 0
    intercept = 0
    iterations = 0
    horizontal = False

    while len(removal_set) < inliers:
        removal_set = []
        horizontal = False
        # generate the two random points
        first_point = numpy.random.randint(len(points))
        second_point = first_point
        while second_point == first_point:
            second_point = numpy.random.randint(len(points))
        first_point = points[first_point]
        second_point = points[second_point]

        # if the slope is not horizontal
        if second_point[0] - first_point[0] != 0:
            slope = (second_point[1]-first_point[1]) / (second_point[0] - first_point[0])
            intercept = int(second_point[1] - slope * second_point[0])
            for point in points:
                if abs(point[1] - (slope * point[0] + intercept)) <= threshold:
                    removal_set.append(point)
        # if horizontal movement
        else:
            horizontal = True
            slope = 0
            intercept = second_point[0]
            for point in points:
                if abs(point[0] - second_point[0]) <= threshold:
                    removal_set.append(point)
```



```python
        # if horizontal movement
        else:
            horizontal = True
            slope = 0
            intercept = second_point[0]
            for point in points:
                if abs(point[0] - second_point[0]) <= threshold:
                    removal_set.append(point)

    # remove the points in the target set and highlight the inliers
    for point in removal_set:
        if (point[0] >= 1 and point[0] < image.shape[1]-1 and point[1] >= 1 and point[1] < image.shape[0]-1):
            image[-1+point[0]:point[0]+2, -1+point[1]: point[1] + 2] = max_value
            road[-1+point[0]:point[0]+2, -1+point[1]: point[1] + 2] = 255
        points.remove(point)

    # draw the line
    if not horizontal:
        for i in range(image.shape[0]):
            target = slope * i + intercept
            if target >= 0 and target < image.shape[1]:
                image[i][int(target)] = max_value
                road[i][int(target)] = 255
    else:
        for i in range(image.shape[1]):
            image[intercept][i] = max_value
            road [intercept][i] = 255
    return points,image
```

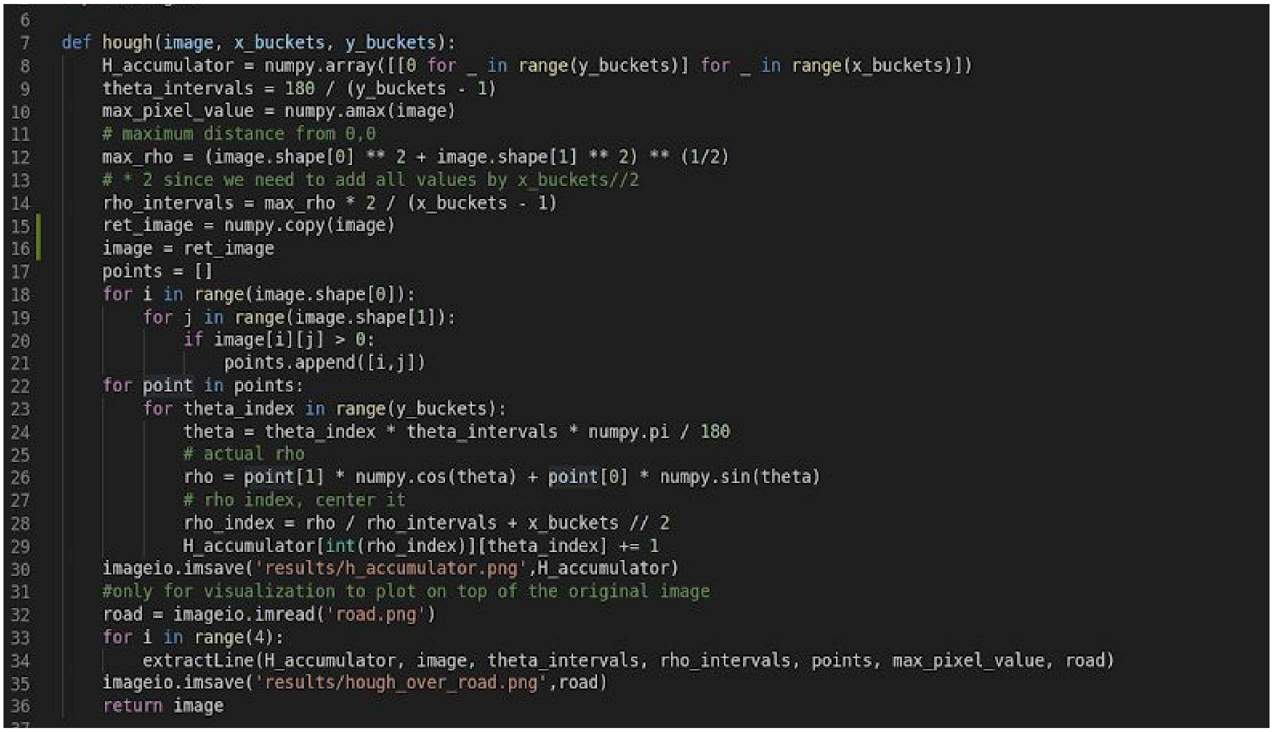


Ransac overlayed on the hessian non_max_suppression image with the inliers highlighted



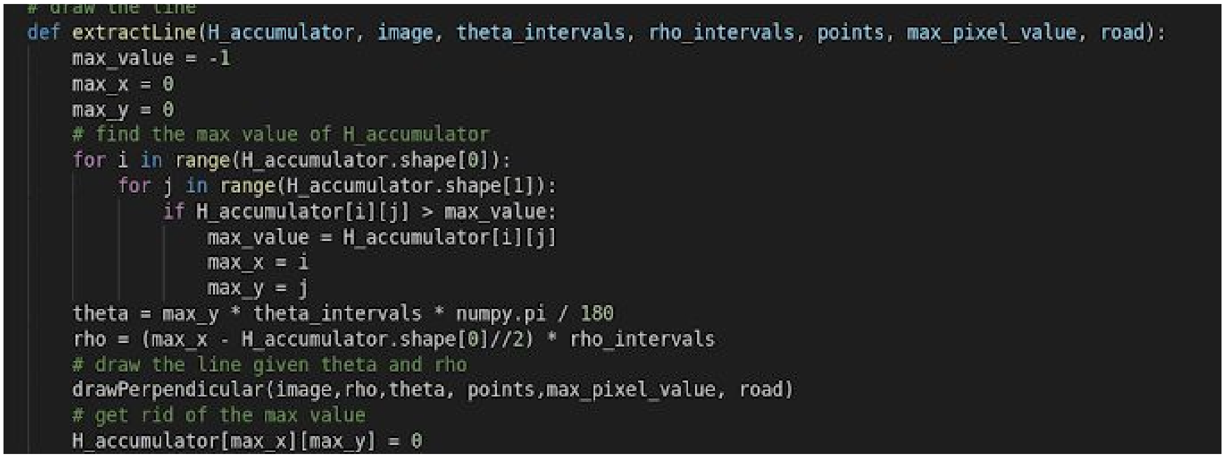
Same lines projected over the actual image. Makes sense. Line goes through what I would expect to be lines. Except the tree because the tree has a lot of points.

Hough.py is where the hough line detection is done. The accumulator matrix dimension is set by the parameters. The theta intervals is the value of 180/ number of indicies I have to represent theta. The max rho is the diagonal of the image because rho is the distance to the perpendicular of the line generated at a point with particular theta. Therefore the furthest distance a line can be is the diagonal. The rho interval is 2 * rho_max / number_of_indicies_for_rho. This is because since I move between 0 to 180 rho can be negative to represent theta values 0 to 360 even though we are bound by 0 to 180. Therefore I center it by adding it by half the number of rho indicies. The algorithm follows the psuedocode given in the slides strictly expect adding the x_indicies/2 to center.



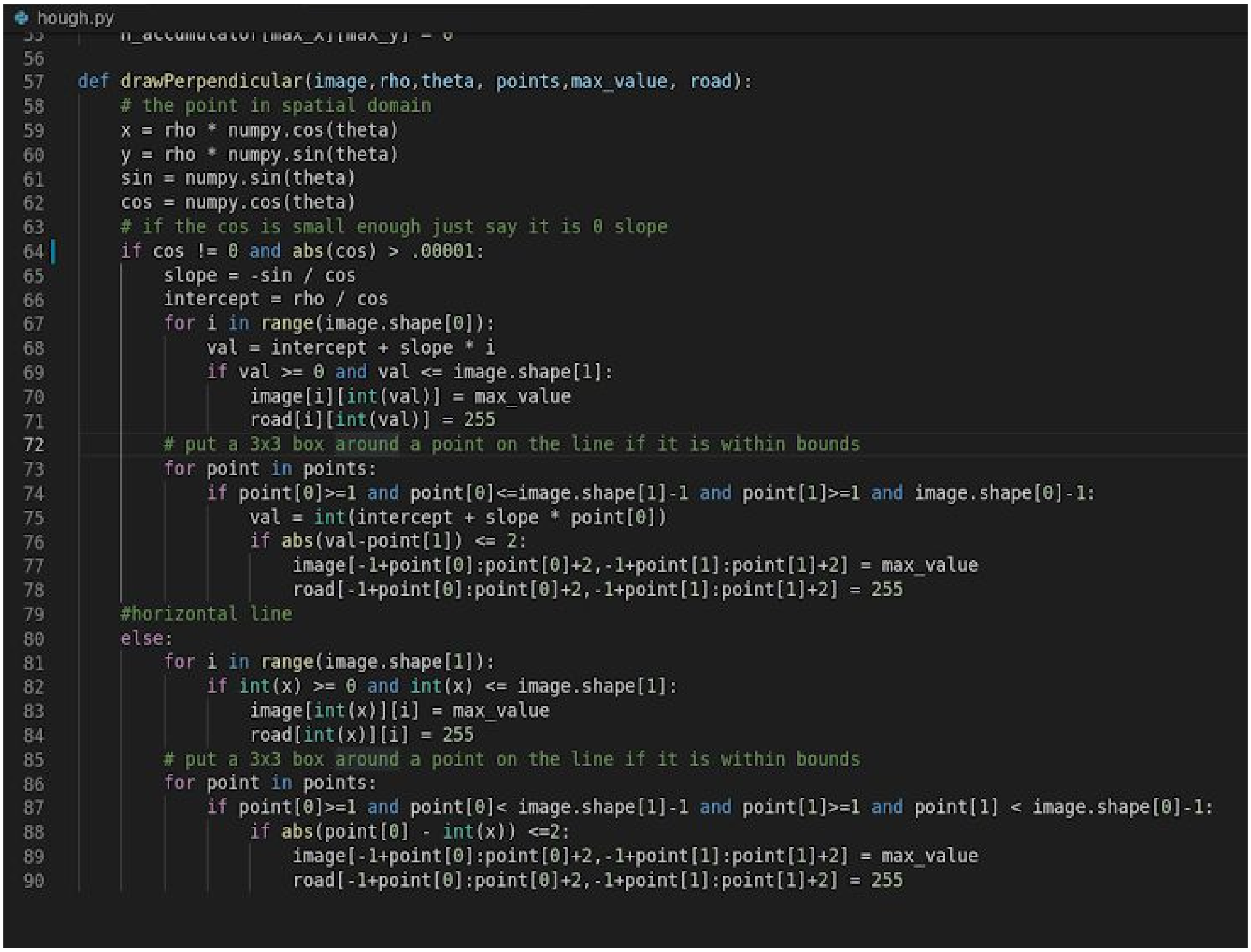
```

def hough(image, x_buckets, y_buckets):
    H_accumulator = numpy.array([[0 for _ in range(y_buckets)] for _ in range(x_buckets)])
    theta_intervals = 180 / (y_buckets - 1)
    max_pixel_value = numpy.amax(image)
    # maximum distance from 0,0
    max_rho = (image.shape[0] ** 2 + image.shape[1] ** 2) ** (1/2)
    # * 2 since we need to add all values by x_buckets//2
    rho_intervals = max_rho * 2 / (x_buckets - 1)
    ret_image = numpy.copy(image)
    image = ret_image
    points = []
    for i in range(image.shape[0]):
        for j in range(image.shape[1]):
            if image[i][j] > 0:
                points.append([i,j])
    for point in points:
        for theta_index in range(y_buckets):
            theta = theta_index * theta_intervals * numpy.pi / 180
            # actual rho
            rho = point[1] * numpy.cos(theta) + point[0] * numpy.sin(theta)
            # rho index, center it
            rho_index = rho / rho_intervals + x_buckets // 2
            H_accumulator[int(rho_index)][theta_index] += 1
    imageio.imsave('results/h_accumulator.png',H_accumulator)
    #only for visualization to plot on top of the original image
    road = imageio.imread('road.png')
    for i in range(4):
        extractLine(H_accumulator, image, theta_intervals, rho_intervals, points, max_pixel_value, road)
    imageio.imsave('results/hough_over_road.png',road)
    return image
```

I then extract lines by finding the max in the accumulator matrix.

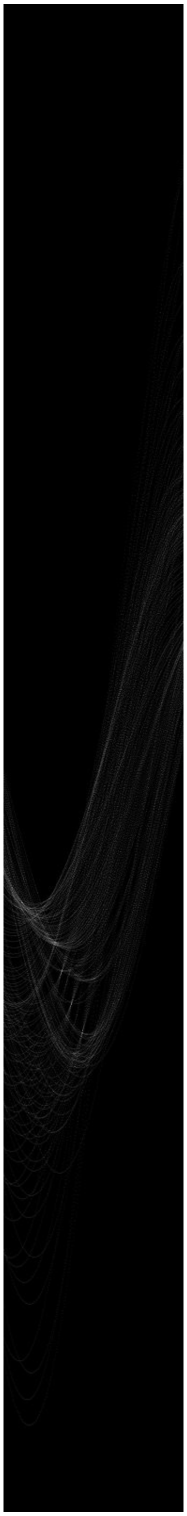


```
# draw the line
def extractLine(H_accumulator, image, theta_intervals, rho_intervals, points, max_pixel_value, road):
    max_value = -1
    max_x = 0
    max_y = 0
    # find the max value of H_accumulator
    for i in range(H_accumulator.shape[0]):
        for j in range(H_accumulator.shape[1]):
            if H_accumulator[i][j] > max_value:
                max_value = H_accumulator[i][j]
                max_x = i
                max_y = j
    theta = max_y * theta_intervals * numpy.pi / 180
    rho = (max_x - H_accumulator.shape[0]//2) * rho_intervals
    # draw the line given theta and rho
    drawPerpendicular(image,rho,theta, points,max_pixel_value, road)
    # get rid of the max value
    H_accumulator[max_x][max_y] = 0
```

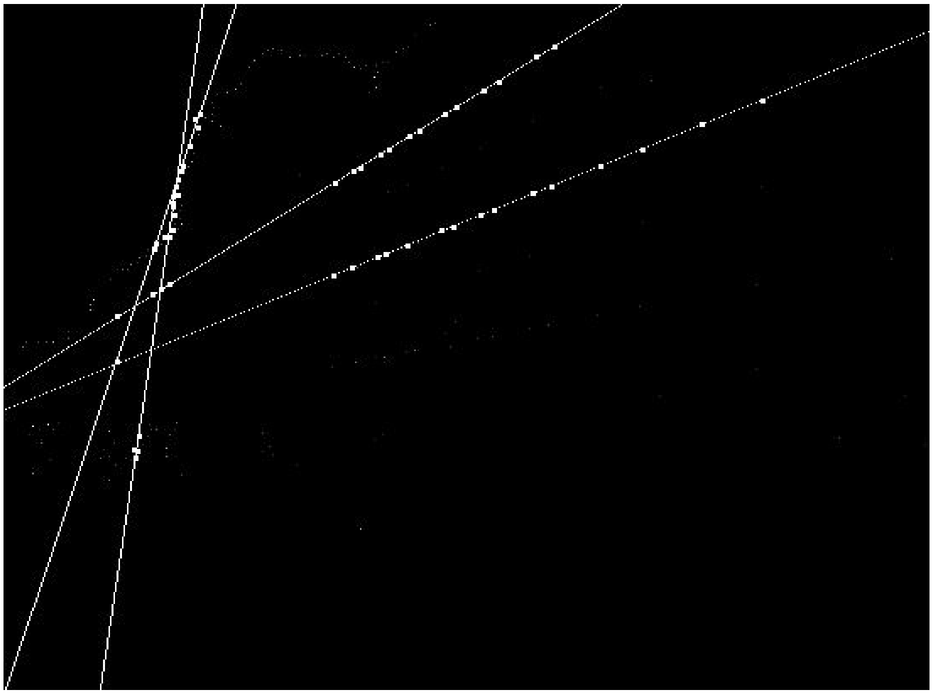
I then draw the perpendicular lines. I get the line equation from the slope = -cos(theta)/sin(theta) and the intercept is r/sin(theta) however I rotate it to switch out axis since images start from the corner left. So it is -sin(theta)/cos(theta) and intercept r/cos(theta). Therefore if cos(theta) is 0 or a very small number, in this case .00001, I say it is horizontal. We must check for very small number because depending on theta intervals because we might not fall exactly on 90 degrees and horizontal lines would become out of the image. Else we calculate slope and draw the line. Then similar to ransac I iterate through the points > 0 and if a point falls between in this case 2 pixels from the line I highlight it with a 3x3 box.



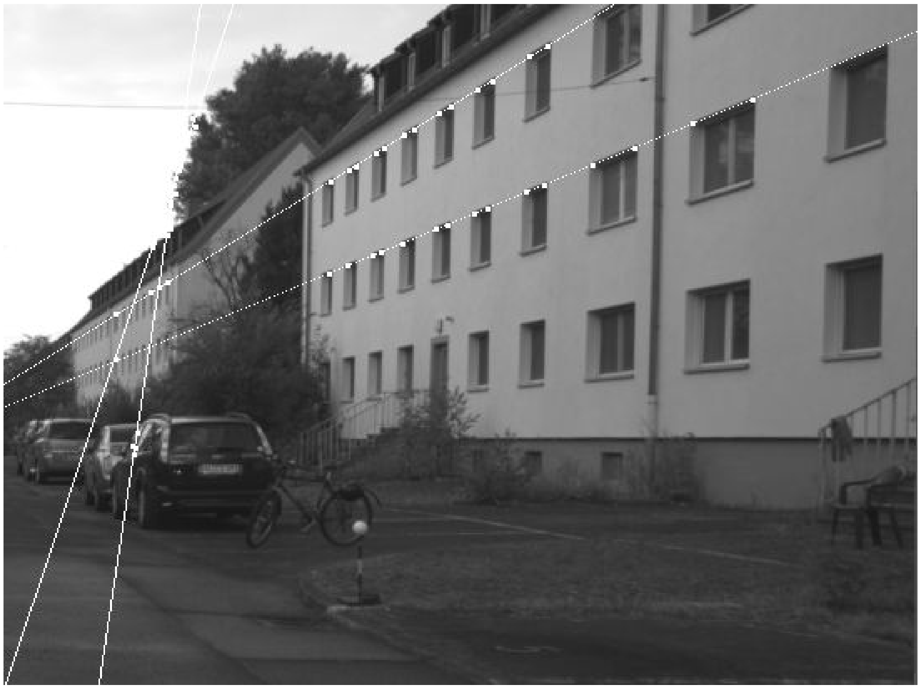
```python
    h_accumulator[max_x][max_y] = 0

def drawPerpendicular(image,rho,theta, points,max_value, road):
    # the point in spatial domain
    x = rho * numpy.cos(theta)
    y = rho * numpy.sin(theta)
    sin = numpy.sin(theta)
    cos = numpy.cos(theta)
    # if the cos is small enough just say it is 0 slope
    if cos != 0 and abs(cos) > .00001:
        slope = -sin / cos
        intercept = rho / cos
        for i in range(image.shape[0]):
            val = intercept + slope * i
            if val >= 0 and val <= image.shape[1]:
                image[i][int(val)] = max_value
                road[i][int(val)] = 255
        # put a 3x3 box around a point on the line if it is within bounds
        for point in points:
            if point[0]>=1 and point[0]<=image.shape[1]-1 and point[1]>=1 and image.shape[0]-1:
                val = int(intercept + slope * point[0])
                if abs(val-point[1]) <= 2:
                    image[-1+point[0]:point[0]+2,-1+point[1]:point[1]+2] = max_value
                    road[-1+point[0]:point[0]+2,-1+point[1]:point[1]+2] = 255
    #horizontal line
    else:
        for i in range(image.shape[1]):
            if int(x) >= 0 and int(x) <= image.shape[1]:
                image[int(x)][i] = max_value
                road[int(x)][i] = 255
        # put a 3x3 box around a point on the line if it is within bounds
        for point in points:
            if point[0]>=1 and point[0]< image.shape[1]-1 and point[1]>=1 and point[1] < image.shape[0]-1:
                if abs(point[0] - int(x)) <=2:
                    image[-1+point[0]:point[0]+2,-1+point[1]:point[1]+2] = max_value
                    road[-1+point[0]:point[0]+2,-1+point[1]:point[1]+2] = 255
```



Hough accumulator to scale. We used 3000 x 360



Hough over non max_suppression



Hough over road with points outlined. Not the tree accumulates a lot of points. Might be hard to see. Pretty similar to ransac, makes sense to me.