**Kunal Goyal**
**CS558 – Computer Vision**
**Homework2**

## Code:

### Main.py

```python
#main file which will call all other modules
"""
Name: Kunal Goyal
CS558
Computer Vision
"""


import numpy as np
import random
import itertools
import math
import cv2
import filterop
import ransac
import hough



if __name__ == "__main__":

    try:
        """Filtering variable and corrosponding values.
        g: gaussian
        sobelFilHorizontal: Horizontal Sobel Filter values
        sobelFilVerticle: Verticle Sobel Filter values
        """
        g = [[0.077847, 0.123317, 0.077847],
                  [0.123317, 0.195346, 0.123317],
                  [0.077847, 0.123317, 0.077847]]
        sobelFilHorizontal = [[1, 2, 1],
                       [0, 0, 0],
                       [-1, -2, -1]]
        sobelFilVerticle = [[1, 0, -1],
            [2, 0, -2],
            [1, 0, -1]]


        # Get the image
        img = cv2.imread("road.png", 0)
        arr = filterop.updated_arr(img)

        gfilImg = filterop.apply_filter(g, arr)
        h = filterop.apply_filter(sobelFilHorizontal, gfilImg[0])
        v = filterop.apply_filter(sobelFilVerticle, gfilImg[0])
```

**Kunal Goyal**
**CS558 – Computer Vision**
**Homework2**

```
    """Apply filter for Horizontal and Vertical sobel
       Get Key points in the image using a Hessian detector
    cord1_xx: for XX cordinates
    cord1_yy: for YY cordinates
    cord1_xy: for XY cordinates
    cord1_yx: for YX cordinates
    """
    cord1_xx = filterop.apply_filter(sobelFilHorizontal, h[0])
    cord1_yy = filterop.apply_filter(sobelFilVerticle, v[0])
    cord1_xy = filterop.apply_filter(sobelFilVerticle, h[0])
    cord1_yx = filterop.apply_filter(sobelFilHorizontal, v[0])

    edges = filterop.overlay_image(h[0], v[0])
    suppress_edges = filterop.sup_nm(edges[0], h[0], v[0], "edges")
    threshold_edges = filterop.threshold(suppress_edges[0], 175, 60)

    hess_matrix = filterop.hes_matrix(cord1_xx[0], cord1_yy[0], cord1_xy[0], cord1_yx[0], 175000)
    hess_threshold = filterop.sup_nm(hess_matrix[0], h[0], v[0], "corners")
    updated_hess_matrix = filterop.overlay_image(hess_threshold[0], threshold_edges / 4, bg=True)

    colored = (updated_hess_matrix[1]).copy()
    colored = cv2.cvtColor(updated_hess_matrix[1], cv2.COLOR_GRAY2RGB)

    """
    Applying RANSAC
    Run the RANSAC algorithm on the key points to find the 4 best lines
    """
    ransac_image = colored.copy()
    ransac_image = ransac.apply_algo(ransac_image, filterop.corners_to_list(hess_threshold[1]), itr
= 20)

    """
    Hough Transformation with the hess_threshold values
    """
  Hough_transform  = colored.copy()
  Hough_transform  = hough.apply_hough(Hough_transform,
  filterop.corners_to_list(hess_threshold[1]), angle=45)


    """
    Saving all output images
    """
    cv2.imwrite("gaussian_filter.png", gfilImg[1])
    cv2.imwrite("h_sobel_filter.png", h[1])
    cv2.imwrite("v_sobel_filter.png", v[1])
    cv2.imwrite("cord1.png", cord1_xx[1])
    cv2.imwrite("cord2.png", cord1_yy[1])
    cv2.imwrite("cord3.png", cord1_xy[1])
    cv2.imwrite("cord4.png", cord1_yx[1])
    cv2.imwrite("edges_with_no_supress.png", edges[1])
    cv2.imwrite("edges_with_supress.png", suppress_edges[1])
```

```
        cv2.imwrite("edges_threshold.png", threshold_edges)
        cv2.imwrite("corners.png", hess_matrix[1])
        cv2.imwrite("corners_threshold.png", hess_threshold[1])
        cv2.imwrite("updated_corners.png", updated_hess_matrix[1])
        cv2.imwrite("ransac.png", ransac_image)
        cv2.imwrite("hough_image.png", Hough_transform)
    except Exception as e:
        print(e)
```

**filterop.py:**

```
import numpy as np
import math


def apply_filter(filter, arr):
    try:
        """
        This method convolves an image with a given kernel.
        Parameters
        ----------
        filter:array, Required

        arr: array, Required

        return tuple
        """

        ar = updated_arr(arr)
        img = updated_arr(arr)
        for i in range(1, len(arr) - 1):
            for j in range(1, len(arr[i]) - 1):
                pos = 0
                for rows in range(len(filter)):
                    for columns in range(len(filter[rows])):
                        x = rows - (len(filter) // 2)
                        y = columns - (len(filter[rows]) // 2)
                        pos += filter[rows][columns] * arr[i + x][j + y]
                ar[i][j] = pos

                img[i][j] = 0 if pos < 0 else  255 if pos > 255 else  pos
        return (ar, img)
    except Exception as e:
        print("error in Apply_filter fn")
        print(e)


def overlay_image(item1, item2, bg = False):
    try:
        """
```

**Kunal Goyal**
**CS558 – Computer Vision**
**Homework2**

```
        This method used to overlays images based on gradient value.
        Parameters
        ----------
        item1:array , Required

        item2:array, Required

        bg:boolean, Optional

        return tuple of item1 and image
        """
        img = item1.copy()
        for i in range(len(item1)):
            for j in range(len(item1[i])):
                if not bg:
                    val = math.sqrt(item1[i][j] ** 2 + item2[i][j] ** 2)
                    item1[i][j] = val
                    img[i][j] = 0 if val < 0 else 255 if val > 255 else  val
                else:
                    if item2[i][j] > item1[i][j]:
                        item1[i][j] = item2[i][j]

                    img[i][j] = 0 if item1[i][j] < 0 else 255 if item1[i][j] > 255 else item1[i][j]
        return (item1, img)
    except Exception as e:
        print("Error in overlay_image function")
        print(e)

def sup_nm(arr, h, v, mode):
    try:
        """
        This method supresses pixel intensities based on the neighborhood pixel,

        Parameters
        ----------
        arr: array, Required

        h:array, Required

        v:array, Required

        mode : str , Required

        return list
        """

        ar = arr.copy()
        img = arr.copy()
        for i in range(len(arr)):
            for j in range(len(arr[i])):
                if mode == "edges":
                    angle = math.atan2(v[i][j], h[i][j])
```

```
                    ar[i][j] = arr[i][j]
                    if i == 0 or j == 0 or i == len(arr) - 1  or j == len(arr[i]) - 1:
                        ar[i][j] = 0
                    elif (angle >=  -1*math.pi/8 and angle <= math.pi / 8) or (angle > 7*math.pi/8 and 
angle <= -7*math.pi/8):
                        if arr[i][j] <= arr[i][j+1] or arr[i][j] <= arr[i][j-1]:
                            ar[i][j] = 0
                    elif (angle < -1*math.pi/8 and angle >= -3*math.pi/8) or (angle > math.pi/8 and angle 
<= 3*math.pi/8):
                        if arr[i][j] <= arr[i+1][j+1] or arr[i][j] <= arr[i-1][j-1]:
                            ar[i][j] = 0
                    elif (angle < -3*math.pi/8 and angle >= -5*math.pi/8) or (angle > 3*math.pi/8 and 
angle <= 5*math.pi/8):
                        if arr[i][j] <= arr[i+1][j] or arr[i][j] <= arr[i-1][j]:
                            ar[i][j] = 0
                    elif (angle < -5*math.pi/8 and angle >= -7*math.pi/8) or (angle > 5*math.pi/8 and 
angle <= 7*math.pi/8):
                        if arr[i][j] <= arr[i+1][j-1] or arr[i][j] <= arr[i-1][j+1]:
                            ar[i][j] = 0
                    else:
                        ar[i][j] = 0


                    img[i][j] = 0 if ar[i][j] < 0 else 255 if ar[i][j] > 255 else ar[i][j]

                elif mode == "corners":
                    ar[i][j] = arr[i][j]
                    if (i == 0 or j == 0 or i == len(arr) - 1 or j == len(arr[i]) - 1):
                        ar[i][j] = 0
                    elif not (arr[i][j] > arr[i+1][j+1] and arr[i][j] > arr[i-1][j-1] and arr[i][j] > arr[i+1][j-1] 
and arr[i][j] > arr[i-1][j+1] and arr[i][j] > arr[i][j+1] and arr[i][j] > arr[i][j-1] and arr[i][j] > 
arr[i+1][j] and arr[i][j] > arr[i-1][j]):
                        ar[i][j] = 0

                    img[i][j] = 0 if ar[i][j] < 0 else  255 if ar[i][j] > 255 else  ar[i][j]


        return [ar, img]
    except Exception as e:
        print("Error in sup_nm fn")
        print(e)

def threshold(ar,min_value,max_value):
    try:

        """
        This method used to set pixel value to zero when it less than min_value,
        and if it is greater than min_value and less than max_value then set to 125 else 255.

        Parameters
        ----------
        ar: , Required
```

**Kunal Goyal**
**CS558 – Computer Vision**
**Homework2**

```
        min_value:int, Required

        max_value:int, Required

        return ar
        """
        canvas_len=len(ar)
        for i in range(canvas_len):
            ar_item_len=len(ar[i])
            for j in range(ar_item_len):
                ar[i][j] = 0 if ar[i][j] < min_value else  125 if ar[i][j] < max_value else 255
        return ar
    except Exception as e:
        print("Error in threshold function")
        print(e)


def hes_matrix(xxcord, yycord, xycord, yxcord, threshold):
    try:
        """
        This method used to hessian matrix to detect corners

        Parameters
        ----------
        xxcord:array , Required

        yycord:array, Required

        xycord:array , Required

        yxcord:array, Required

        threshold:float, Required

        return list of list item
        """
        arr = xxcord.copy()
        img = xxcord.copy()
        for i in range(len(xxcord)):
            for j in range(len(xxcord[i])):
                det = xxcord[i][j]*yycord[i][j]-xycord[i][j]*yxcord[i][j]
                trace = xxcord[i][j] + yycord[i][j]
                r = det - .06*(trace**2)
                arr[i][j] = r
                if r > threshold:
                    img[i][j] = 255
                else:
                    arr[i][j] = 0
                    img[i][j] = 0
        return [arr, img]
    except Exception as e:
```

```
        print("Error in hes_matrix fn")
        print(e)

def updated_arr(img):
    try:
        """
        This method create a array of a image
        Parameters
        ----------
        img:array, Required

        return list
        """
        arr_item = []
        for i in range(len(img)):
            arr_item += [[]]
            for j in range(len(img[i])):
                arr_item[i] += [img[i][j]]
        arr_item = np.array(arr_item, dtype="float32")
        return arr_item
    except Exception as e:
        print("Error updated_arr in Function")
        print(e)


def corners_to_list(corners):
    try:
        """
        This methods change the list of corners into a list

        Parameters
        ----------
        corners: array, Required

        Return list of corners
        """
        corners_list = []
        for i in range(len(corners)):
            for j in range(len(corners[i])):
                if corners[i][j] > 0:
                    corners_list += [(i, j)]
        return corners_list
    except Exception as e:
        print("Error in Corner_to_list function")
        print(e)
```

**ransac.py:**

```
import random
import itertools
import math
import cv2
```

**Kunal Goyal**
**CS558 – Computer Vision**
**Homework2**

```
def ransac_algo(corners, threshold, inliers, itr):
    try:
        """
        This methods uses the RANSAC algorithm on set of points of image.

        Parameters
        ----------
        corners: array, Required

        threshold: float, Required

        inliers: Required

        itr: int, Required

        Return list of list
        """

        maxpts = []
        passes = []
        success = []
        used = []
        endpoints = []
        for i in range(itr):
            maxpts += [[0, 0]]
            passes += [[(0, 0)]]
            success += [0]
            used += [[]]
            endpoints += [[]]
        for j in range(17):
            items = random.sample(range(len(corners)), 2)
            endpoints[j] = [corners[items[0]], corners[items[1]]]
            passes[j] = [corners[items[0]], corners[items[1]]]
            line_dim = (corners[items[0]][0] - corners[items[1]][0])**2 + (corners[items[0]][1] -
corners[items[1]][1])**2
            try:
                m = (corners[items[0]][0] - corners[items[1]][0]) / (corners[items[0]][1] -
corners[items[1]][1])
            except:
                continue
            if m == 0:
                continue
            b = -m*corners[items[0]][1] + corners[items[0]][0]
            for k in range(len(corners)):
                cornerx = (corners[k][1] / m + corners[k][0] - b)/(m + 1/m)
                cornery = cornerx * m + b
                d = (corners[k][1] - cornerx) ** 2 + (corners[k][0] - cornery) ** 2
                if d <= threshold ** 2:
                    success[j] += 1
                    used[j] += [corners[k]]
                    first_dim = (corners[k][0] - endpoints[j][0][0])**2 + (corners[k][1] -
```

```
endpoints[j][0][1])**2
                second_dim = (corners[k][0] - endpoints[j][1][0])**2 + (corners[k][1] -
endpoints[j][1][1])**2
                if first_dim > line_dim or second_dim > line_dim:
                    if first_dim <= second_dim:
                        if first_dim > maxpts[j][0]:
                            maxpts[j][0] = first_dim
                            passes[j][0] = corners[k]
                    else:
                        if second_dim > maxpts[j][1]:
                            maxpts[j][1] = second_dim
                            passes[j][1] = corners[k]
            if success[j] >= inliers:
                return [passes[j], used[j], endpoints[j]]
        winner = success.index(max(success))
        return [passes[winner] , used[winner], endpoints[winner]]
    except Exception as e:
        print("Error in ransac_algo function")
        print(e)

def apply_algo(img, corners, threshold = math.sqrt(3.84), inliers = 1000, feat = 4, itr = 17):
    try:
        """
        This methods apply the RANSAC algorithm on an image as per iteration value(itr parameter) ,
        and creates an image.

        Parameters
        ----------
        img: array, Required

        corners: array, Required

        threshold: float, Optional

        inliers: int, Optional

        feat: int, Optional

        itr: int, Optional

        Return array
        """
        colors = list(itertools.product([0, 255], repeat = 3))
        color = random.sample(colors[1:-1], feat + 1)
        for i in range(feat):
            winner = ransac_algo(corners, threshold, inliers, itr)
            img = cv2.line(img, (winner[0][0])[::-1], (winner[0][1])[::-1], color[i], 1)
            for j in range(len(winner[1])):
                pt = winner[1][j]
                for row in range(3):
                    for column in range(3):
                        x = row - 1
```

```
                    y = column - 1
                    img[pt[0] + x][pt[1] + y] = color[i] if pt != winner[2][0] and pt != winner[2][1] else
color[-1]
                corners.remove(pt)
            return img
        except Exception as e:
            print("Error in apply_algo function")
            print(e)
```

**hough.py:**

```
import math
import cv2

def usehough(img, corners, angle, rad, ftr):
    try:
        """
        This method apply Hough Transform on a set of points.

        Parameters
        ----------
        img:array, Required

        corners:array, Required

        angle: int, Optional

        rad: Optional

        ftr: int, Optional

        return cordinates, list
        """
        cordinates = []
        for i in range(ftr):
            cordinates += [[(0, 0), 0]]
        hough = []
        for i in range(rad):
            hough += [[]]
            for j in range(angle):
                hough[i] += [0]
        for i in range(len(corners)):
            for j in range(0, angle):
                r = corners[i][1]*math.cos(j * math.pi / angle) + corners[i][0]*math.sin(j * math.pi / angle)
                rr = math.floor(rad/2 / math.sqrt(len(img)**2 + len(img[1])**2) * r + rad/2)
                r = math.floor(r)
                hough[rr][j] += 1
```

**Kunal Goyal**
**CS558 – Computer Vision**
**Homework2**

```
            for k in range(ftr):
                if hough[rr][j] > cordinates[k][1]:
                    if [(r, j * math.pi / angle), hough[rr][j]-1] in cordinates:
                        index = cordinates.index([(r, j * math.pi / angle), hough[rr][j]-1])
                        cordinates = cordinates[:k] + [[(r, j * math.pi / angle), hough[rr][j]]] + cordinates[k:index]
+ cordinates[index+1:]
                    else:
                        cordinates = cordinates[:k] + [[(r, j * math.pi / angle), hough[rr][j]]] + cordinates[k:-1]
                    break
        return cordinates
    except Exception as e:
        print("Error in usehough function")
        print(e)


def apply_hough(img, corners, angle = 180, rad = None, ftr = 4):
    try:
        """
        This method apply Hough Transofrm a specifed amount of times, and creates an image depicting the result.

        Parameters
        ----------
        img:array, Required

        corners:array, Required

        angle: int, Optional

        rad: Optional

        ftr: int, Optional

        return array
        """
        if rad == None:
            rad = math.ceil(2*math.sqrt(len(img)**2 + len(img[1])**2))
        hough = usehough(img, corners, angle, rad, ftr)
        for i in range(ftr):
            pt1 = int(hough[i][0][0]/math.sin(hough[i][0][1]))
            pt2 = int((hough[i][0][0] - len(img[1])*math.cos(hough[i][0][1]))/math.sin(hough[i][0][1]))
            img = cv2.line(img, (0, pt1), (len(img[1]), pt2), (0, 0 , 255), 1)
        return img
    except Exception as e:
        print("Error in apply_hough function")
                print(e)
```

**Kunal Goyal**
**CS558 – Computer Vision**
**Homework2**

## Explanation and Images:

### Pre-Processing Step

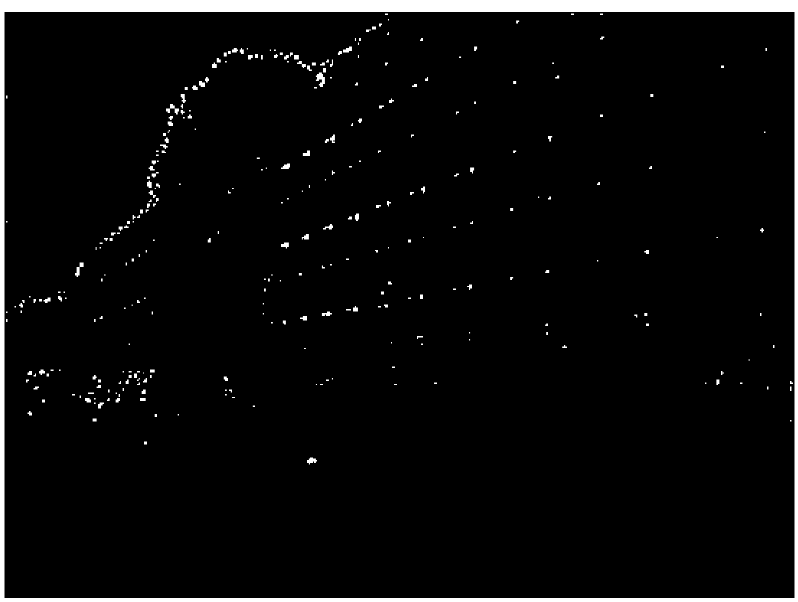
The code reads in the image from the provided file, road.png, converts it to grayscale and then begins the pre-processing part of the code.
The pre-processing starts by first creating a Gaussian filter using the Guassian value equation provided in the slidesThe result of applying the filter to the image can be seen below in Figure 1.

Figure 1: Guassian Filtered Image

**Kunal Goyal**
**CS558 – Computer Vision**
**Homework2**

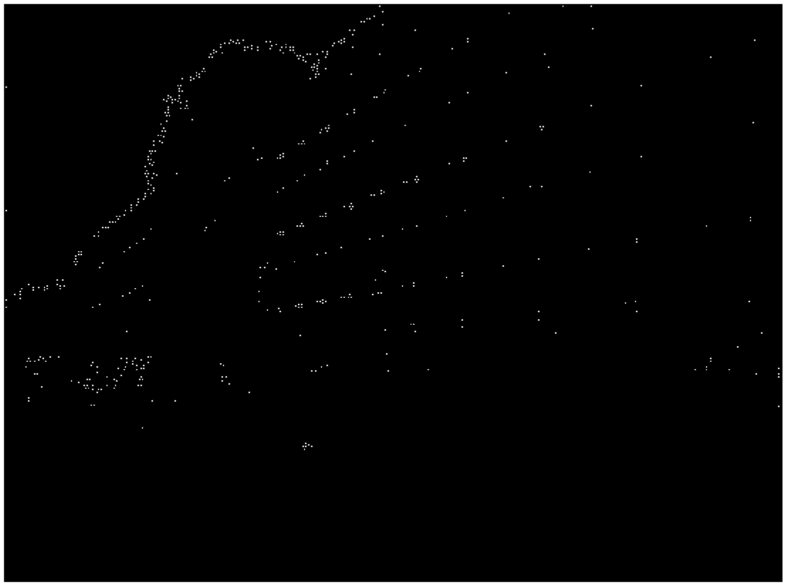
Figure 2: Horizontal Sobel Filtered Image (Left) and Verticle Sobel Filtered Image (Right)

The image is then subjected to Sobel filters as derivatives operators in order to determine the image's second x, second y, and x of y derivatives. The determinant of the Hessian matrix is then calculated using these derivative matrices. The Hessian matrix's determinant was then threshold, with all pixels with values less than 150 set to 0 and all pixels with values more than 150 set to 255. All of the places on the image where the Hessian detector responds significantly were shown as a result. The black tree and the extremely light sky have a significant intensity difference.



Thresholded Hessian Response

**Kunal Goyal**
**CS558 – Computer Vision**
**Homework2**



Non Maximum Suppressed Response

Edges with No supress

**Kunal Goyal**
**CS558 – Computer Vision**
**Homework2**

Edges with supress

The result was then subjected to non-maximum suppression in order to extract examples of the best Hessian detector responses. Figure 2 shows the thresholded Hessian detector response as well as the non-maximum suppressed version. The Hessian detector responds strongly to many of the angles and straight lines that make up the building's characteristics, which is good, but it also responds strongly to the tree in the image's upper left corner, which is not ideal.

## RANSAC:

The RANSAC portion of the code takes within the yield of pre-processing step which ought to be an array of all the zones on the picture where the Hessian detector reacted exceptionally unequivocally after non-maximum suppression was connected. The algorithm first gathers all the points in the image where the locator reacted strongly and stores them as tuples in a list that may be easily accessed later. Two points are then picked at random from the list, and the parameters for the condition of a line, slant m, and y caught c, are discovered using some basic variable-based algebra. The rest of the focuses are at that point circled through in order to decide the focuses that are near sufficient to the line to be inliers for the line. This is often done by calculating the opposite separate between the point and the line and checking in the event that it is underneath a certain threshold.

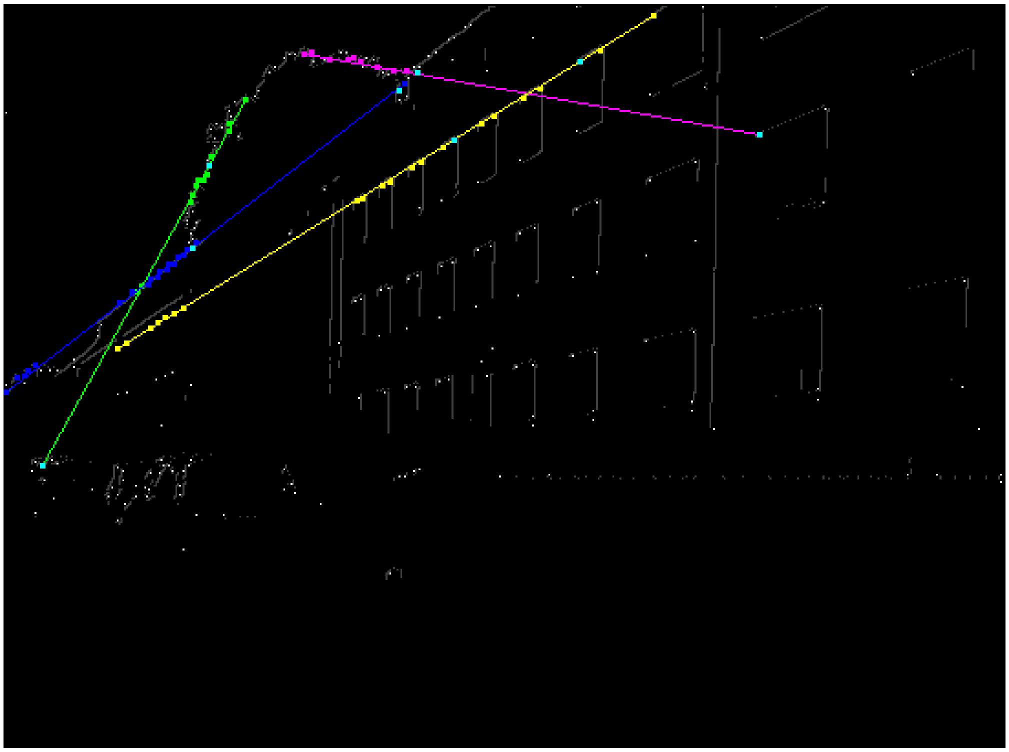
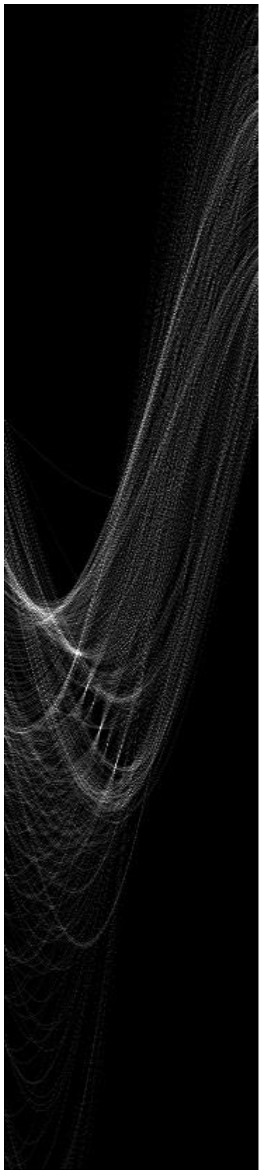


Figure 5: Four Best Lines Produced by RANSAC Plotted

## Hough:

The Hough transform section of the code, like RANSAC, takes in the pre-processing step's output, which should be an array of all the locations on the image where the Hessian detector responded extremely strongly when non-maximum suppression was applied. The algorithm begins by calculating the size that the accumulator will require. To do this, the highest and lowest values for the function $\rho = x \cos \theta$. The algorithm begins by calculating the size that the accumulator will require. To do this, the highest and lowest values for the function $\rho = x \cos \theta$ Because we can't directly express the negative values that rho will take in the accumulator, the range of values is used for the first dimension. To compensate for this, we apply an offset, which is simply the negative of the low value multiplied by the rho given by the equation to produce a correct index to utilise.

**Kunal Goyal**
**CS558 – Computer Vision**
**Homework2**



Visualized Hough Space

**Kunal Goyal**
**CS558 – Computer Vision**
**Homework2**

Suppressed Hough Space

Areas Removed From Hough Space

The algorithm extracts all of the points in the picture where the detector responded strongly and saves them as tuples in a list that may be retrieved later. The line in the image space with the most support based on the number of image space points that voted for it should correspond to the point in the Hough space with the highest value. Non maximum suppression is performed to the Hough space before identifying the points with the highest support in order to prevent near identical lines from being proposed. The model is transformed back into two linear form and two points are created in order to draw a line between them on the picture after the strongest point in suppressed Hough space is discovered.

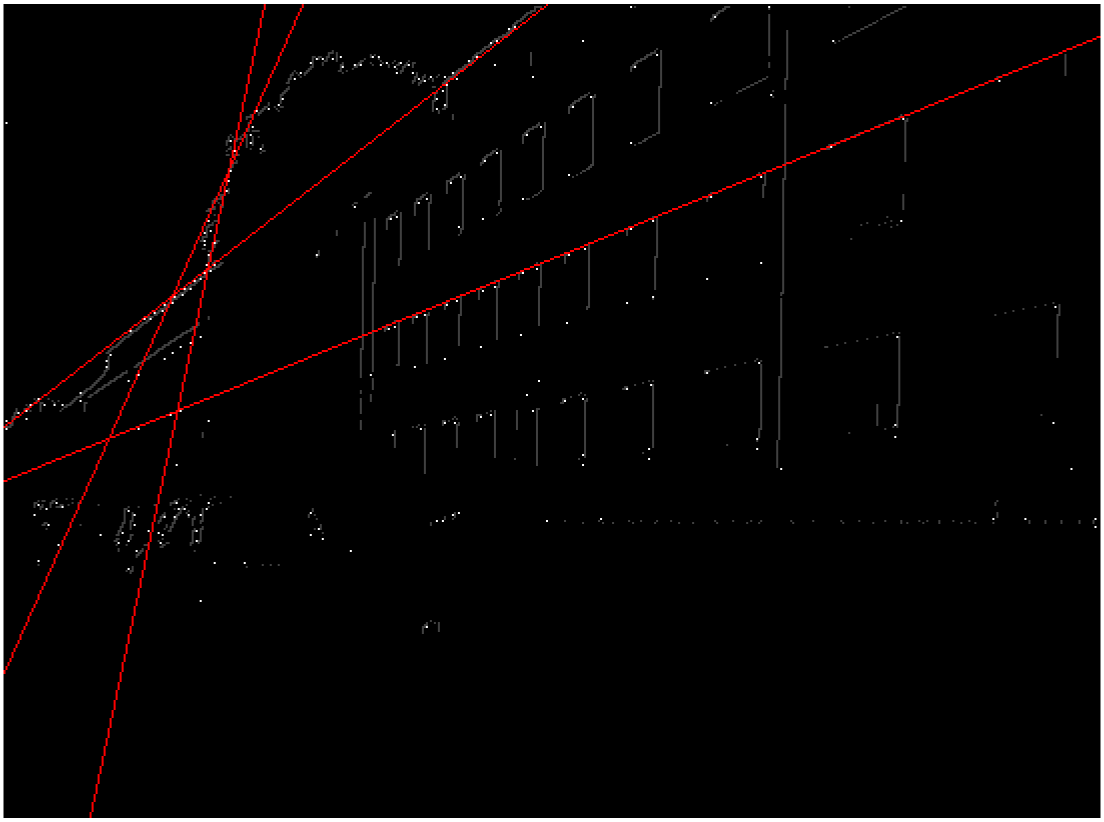


Figure 7: Hough Lines on Original Image