Computer Vision Project 1
Ayan Agrawal
aka398
N15025000

1) **<u>Details on how you detect parallelograms:</u>**

a) **<u>Edge Detection:</u>**

**1)** Colour Image to Gray Scale Image**:**

Using the luminance formula I converted the RGB image(24 bits) to Gray Scale image(8 bits) where pixel value ranges from 0-255.

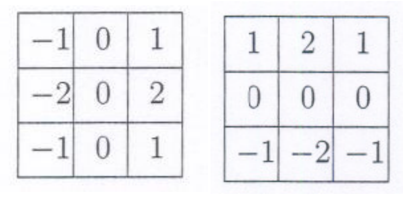
*luminance = 0.30R + 0.59G + 0.11B*

2) Smoothing Filter:

To remove the noise, I applied 3*3 mean filter over the entire image to remove the noise.

3) Sobel Operator:

The smoothed image is passed through the mask of Sobel operator to detect the edges.



| | | |
|---|---|---|
| −1 | 0 | 1 |
| −2 | 0 | 2 |
| −1 | 0 | 1 |

Gx

| | | |
|---|---|---|
| 1 | 2 | 1 |
| 0 | 0 | 0 |
| −1 | −2 | −1 |

Gy

Thus using the above mask I calculated the gradient magnitude and gradient angle to detect the edges. The gradient magnitude is then normalized to give the Normalized Gradient Magnitude image.

4) Non-Maxima Suppression:

To avoid the thick edges in the image and to get the local maxima I have used the concept of quantized angle and non-maxima suppression. The gradient angles obtained from the Sobel Operator lies between $-90^0$ to $+90^0$ because 'math.atan()' function in python gives the values in that range.

5) Thresholding:

6) *The non-maxima suppressed image is then passed through a suitable value called threshold wherein the values above a threshold limit is set to white pixel (black pixel just for display purpose) and below the limit is set to black pixel.*

7) The image thus obtained is in binary form having many curved lines apart from straight lines.

b) **<u>Hough Transform</u>**

8) To detect the straight lines, the image space is converted to parametric space using the equation of rho = x*cos(theta) + y*sin(theta).

9) In this the concept of the accumulator array is used wherein on the x-axis is the rho and on y-axis is theta to represent the 2-D accumulator array.

10) Here the rho ranges from -maximumDistance to +maximumDistance and theta varies from -900 to +900 where maximumDistance = square root(width*width + height*height), where width and height is the size of image.

11) Every detected edge in the binary image is visited and the corresponding (x,y) value is substituted on the above equation to get the value of rho.

12) Subsequently, the values in the accumulator array is incremented by 1. Hence more the value denotes the possibility of more edges passing through that point.

13) The above technique will result in many such values and thus taking only top 150 maximum values. Also to suppress the neighbouring values in the accumulator array using the 3*3 mask.
14) Now using the peak values of (rho, theta), the equation of rho = x*cos(theta) + y*sin(theta) is used over all the pixel of grey image and whichever value of rho gets satisfied will be taken and that pixel value will be assigned as white (value = 255). This will give the relevant straight lines.
15) These straight lines can then be superimposed on the binary image and compared whether both the binary Image and straight line have pixel value of 255, is yes keep that value else discard that part of straight line. This will finally give us the line segments.

**c) <u>Parallelogram detection:</u>**

16) Using the above straight lines, we can find the intersecting points using the equation
rho1 = x*cos(theta1) + y*sin(theta1)
rho2 = x*cos(theta2) + y*sin(theta2)
This will give the value of (x,y) which will be the coordinates where the lines intersect.
17) Also using the theta values from the peak values of the accumulator, if theta value is same or approximately same, they can be treated as parallel lines.
18) Now the above found coordinates we want coordinates which lie on these parallel lines. So again using the formula of rho = x*cos(theta) + y*sin(theta), I find those values of (x,y) which lie on the parallel lines.
19) Consider the pair of parallel lines with parameter (rho1, theta1) representing L1 and (rho2, theta1) representing L2. Let say there are 4 points[(x1,y1), (x2,y2), (x3,y3), (x4,y4)] lying on L1 and 3 points[(x5,y5), (x6,y6), (x7,y7)] lying on L2 . Consider another set of parallel lines (rho3, theta2) and (rho4, theta2).
20) For now, consider 2 points (x1,y1), (x5,y5) to calculate the slope
using the slope formula,
slope = (y5 – y1)/(x5 – x1)
if this slope is equal to theta2 consider the points (x1,y1), (x5,y5) as this is forming the part of parallelogram.
If not, then consider next 2 points (x1,y1), (x6,y6) and calculate the slope using the above formula and compare that with theta2.
Repeat the step for all the combinations of the coordinates. This way all the coordinates obtained will eventually lead to the formation of parallelograms.

**d) <u>Superimpose the Parallelogram:</u>**

*21)* Based on the set of coordinates obtained and using the drawLine function of the built-in python Image library, we can draw the line segments on the original image forming a parallelogram.

**2) <u>Programming Language used and Instruction on how to compile the program:</u>**

Programming Language: Python 2.7

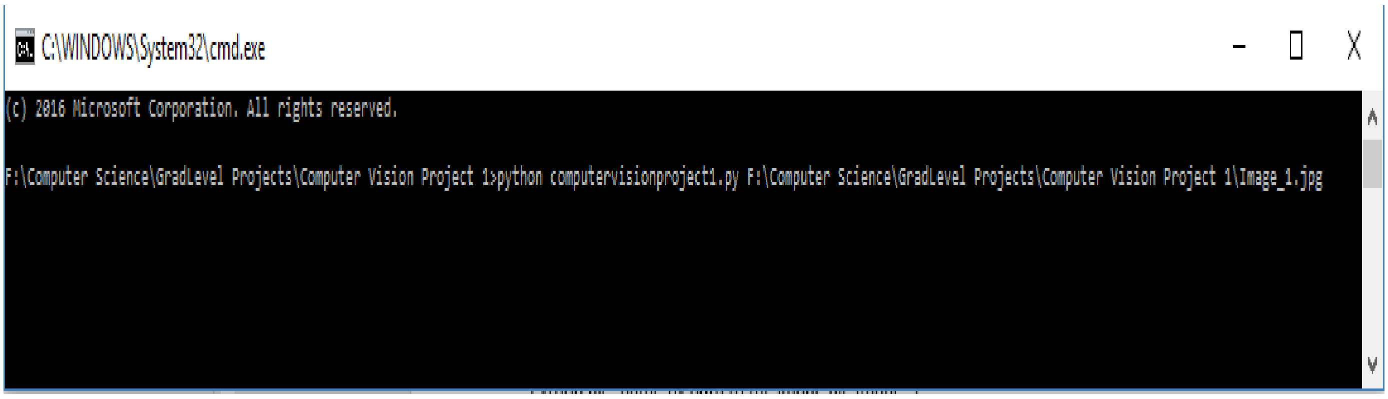
Instruction to compile:

a) Go to command prompt of the desktop
b) Copy and paste the below code in the editor of python and save the file name as *file_name.py*
c) Go to the path where the *file_name.py* file is located
d) Write the below commands

Computer Vision Project 1
Ayan Agrawal
aka398
N15025000

Python file_name.py *path to the image file Image_1*

*See the below screenshot for example*



```
(c) 2016 Microsoft Corporation. All rights reserved.

F:\Computer Science\GradLevel Projects\Computer Vision Project 1>python computervisionproject1.py F:\Computer Science\GradLevel Projects\Computer Vision Project 1\Image_1.jpg
```

**<u>Please note:</u>** Below libraries are required for the code to be compiled

sys

math

PIL → Image, ImageDraw

numpy

3) **<u>Source code of the program with inline comments</u>**

```
#importing the libraries to be used
#PIL is the library used for the built-in functions to read, show an image
#numpy library is used for the matrix related basic operations
#math is used for gradient calculation
import sys
import math
from PIL import Image, ImageDraw
import numpy as np

#convert RGB image to Gray Level Image
#function returns the converted gray image object
def convertToGrayLevel(image_object):
    #width and height of image from the size function of PIL library
    width , height = image_object.size
    #Creating a new Image object for Gray Image. Here 'L' is the mode denoting the 8 bit pixel value
```

Computer Vision Project 1
Ayan Agrawal
aka398
N15025000

```
    grayLevelImage = Image.new("L",(width,height))

    #To convert the image to gray, parse all the pixels of the RGB image and apply the luminance formula to get the gray level pixel value

    for i in range(0,width):

        for j in range(0,height):

            R,G,B = image_object.getpixel((i,j))

            #getpixel fuction is used to get the value of corresponding (i,j) pixel value

            #luminance is calculated as R*0.30 + G*0.59 + B*0.11

            luminance = 0.30 * R + 0.59 * G + 0.11 * B

            #the corresponding pixel in the image would have the graylevel value as determined by luminance

            #putpixel fuction is used to put the value in corresponding (i,j) pixel value

            grayLevelImage.putpixel((i,j),int(luminance))

    #return the object of Gray Image for further processing

    return grayLevelImage




#smooth the image by applying mean filter of mask 3*3

#Take the input as object of gray image obtained above

#function returns smoothed image

def smoothImage(grayLevelImage):

    for i in range(1,grayLevelImage.size[0]-1) :

        for j in range(1,grayLevelImage.size[1]-1):

            #mean filter is the sum of all the 9 (3*3 mask) pixel values divide by 9

            mean_value = (grayLevelImage.getpixel((i-1,j-1))+grayLevelImage.getpixel((i-1,j))+grayLevelImage.getpixel((i-
1,j+1))+grayLevelImage.getpixel((i,j-1))+grayLevelImage.getpixel((i,j))+grayLevelImage.getpixel((i,j+1))+grayLevelImage.getpixel((i+1,j-
1))+grayLevelImage.getpixel((i+1,j))+grayLevelImage.getpixel((i+1,j+1)))/9

            grayLevelImage.putpixel((i,j),int(mean_value))

    return grayLevelImage




#finding the gradient magnitude and gradient angle at each pixel point by applying the sobel operator

#function returns the image after applying sobel operator, gradientAngles

def SobelOperator(smoothedImage):

    #Initialize all the values of 2-D graidientAngles array to 0

    gradientAngles = np.zeros(shape = (smoothedImage.size))

    #Creating the object for Sobel Image

    sobelMagnitudeImage = Image.new('L',(smoothedImage.size))

    #Initialize all the values of 2-D magnitude array to 0
    magnitude = np.zeros((smoothedImage.size[0], smoothedImage.size[1]),dtype = np.uint64)

    min_grad = 0

    max_grad = 0

    #Applying the sobel mask to the smooth image

    for i in range(1,smoothedImage.size[0]-1):

        for j in range(1,smoothedImage.size[1]-1):

            gradient_x = (-1)*smoothedImage.getpixel((i-1,j-1)) + (-2)*smoothedImage.getpixel((i,j-1)) + (-1)*smoothedImage.getpixel((i+1,j-1)) + smoothedImage.getpixel((i-1,j+1)) + (2)*smoothedImage.getpixel((i,j+1)) + smoothedImage.getpixel((i+1,j+1))

            gradient_y = smoothedImage.getpixel((i-1,j-1)) + (2)*smoothedImage.getpixel((i-1,j)) + smoothedImage.getpixel((i-1,j+1)) + (-1)*smoothedImage.getpixel((i+1,j-1)) + (-2)*smoothedImage.getpixel((i+1,j)) + (-1)*smoothedImage.getpixel((i+1,j+1))

            magnitude[i][j] = math.sqrt(gradient_x*gradient_x + gradient_y*gradient_y)

            min_grad = min(min_grad,magnitude[i][j])

            max_grad = max(max_grad,magnitude[i][j])

            if gradient_x == 0:

                gradientAngles[i,j] = 90

            else:

                gradientAngles[i,j] = math.degrees(math.atan(gradient_y/gradient_x))

    #normalizing the Gradient magnitude to obtain the Normalized Gradient Magnitude

    mag_norm=np.zeros((smoothedImage.size[0], smoothedImage.size[1]),dtype = np.uint64)

    for i in range(1,smoothedImage.size[0]):

        for j in range(1,smoothedImage.size[1]):

            mag_norm[i][j]=((magnitude[i][j]-min_grad)/(max_grad-min_grad))*255

            sobelMagnitudeImage.putpixel((i,j),int(mag_norm[i][j]))

    return sobelMagnitudeImage,gradientAngles




#Applying angle quantization method to thin the lines(non maxima suppression)

#taking the input as the gradientAngles obtained above in the sobel

#function returns the quantized angles

def quantizeAngles(gradientAngles):

    for i in range(0,gradientAngles.shape[0]):

        for j in range(0,gradientAngles.shape[1]):

            #defining the sector in which the pixel lies

            if -22.5 <= gradientAngles[i,j] < 22.5 :

                gradientAngles[i,j] = 0

            elif 22.5 <= gradientAngles[i,j] < 67.5 :

                gradientAngles[i,j] = 1
        elif 67.5 <= gradientAngles[i,j] <= 90 or -90 <= gradientAngles[i,j] <=-67.5:

            gradientAngles[i,j] = 2

        elif -67.5 < gradientAngles[i,j] < -22.5 :

            gradientAngles[i,j] = 3

    return gradientAngles




#perform non maxima suppression by using the quantized angles

#if the value of center pixel is greater than the pixel value along the gradient line than keep that value else assign that pixel to value 0

#function returns the image with thinned edges

def nonMaximaSuppression(sobelMagnitudeImage,quantizeAngles):

    for i in range(1,sobelMagnitudeImage.size[0]-1):

        for j in range(1,sobelMagnitudeImage.size[1]-1):

            if quantizeAngles[i,j] == 0:

                if sobelMagnitudeImage.getpixel((i,j)) < sobelMagnitudeImage.getpixel((i,j-1)) or sobelMagnitudeImage.getpixel((i,j)) <
sobelMagnitudeImage.getpixel((i,j+1)):

                    sobelMagnitudeImage.putpixel((i,j),0)

            elif quantizeAngles[i,j] == 1 :

                if sobelMagnitudeImage.getpixel((i,j)) < sobelMagnitudeImage.getpixel((i-1,j+1)) or sobelMagnitudeImage.getpixel((i,j)) <
sobelMagnitudeImage.getpixel((i+1,j-1)):

                    sobelMagnitudeImage.putpixel((i,j),0)

            elif quantizeAngles[i,j] == 2 :

                if sobelMagnitudeImage.getpixel((i,j)) < sobelMagnitudeImage.getpixel((i-1,j)) or sobelMagnitudeImage.getpixel((i,j)) <
sobelMagnitudeImage.getpixel((i+1,j)):

                    sobelMagnitudeImage.putpixel((i,j),0)

            elif quantizeAngles[i,j] == 3 :

                if sobelMagnitudeImage.getpixel((i,j)) < sobelMagnitudeImage.getpixel((i-1,j-1)) or sobelMagnitudeImage.getpixel((i,j)) <
sobelMagnitudeImage.getpixel((i+1,j+1)):

                    sobelMagnitudeImage.putpixel((i,j),0)

    return sobelMagnitudeImage




#Applying thresholding to convert the image to binary form

#Different threshold needs to be used for different image based on the desired output

#Threshold for, 'Test Image 1 = 115' ; 'Test Image 2 = 20'; 'Test Image 3 = 18'

#function returns the binary image

def binaryConversion(sobelMagnitudeImage):
```

Computer Vision Project 1
Ayan Agrawal
aka398
N15025000

```
    edgesAt = []

    #the input is sobelized image

    binaryImage = Image.new('L',(sobelMagnitudeImage.size))

    for i in range(0,sobelMagnitudeImage.size[0]):

        for j in range(0,sobelMagnitudeImage.size[1]):

            #for display purpose we can change the output and assign the foreground to be black (pixel value = 0) and background to be white (pixel value = 255)

            #below the foreground is considered to be white and background is considered to be black

            if sobelMagnitudeImage.getpixel((i,j)) <= 115:

                #identifying as background below threshold

                binaryImage.putpixel((i,j),0)

            else:

                binaryImage.putpixel((i,j),255)

                #identifying as foreground above threshold

                #the foreground pixels are saved in an array to be used for further processing

                edgesAt.append([i,j])

    return binaryImage, edgesAt



#rho ranges from -maximum distance to +maximumDistance with step size of 1. maximumDistance = sqrt(width*width + height*height)

#theta ranges from -90 to +90 and step size is 1

#function returns the accumulator array(parametric space) for hough transform

def formAccumulatorArray(edgesAt,binaryImage):

    maximumDistance = int(math.sqrt(binaryImage.size[0]**2 + binaryImage.size[1]**2))

    #thetas ranging from -90 to +90 and step size is 1

    thetas = [i for i in range(-90,91,1)]

    #Initializing accumulator with 0's

    accumulator = np.zeros((2*maximumDistance,len(thetas)),dtype = np.uint64)

    height = binaryImage.size[1]

    for x,y in edgesAt:

        #print("Edge points :",str(x),str(y))

        for theta in thetas:

            #converting degrees to radians

            radian = math.radians(theta)

            rho = int(x*math.cos(radian) + y*math.sin(radian))

            if -maximumDistance <= rho <= maximumDistance:
```

```
            accumulator[rho+maximumDistance,90+theta] +=1

    return accumulator,maximumDistance


#sorting the accumulator in reverse order

#taking the first 150 values from the accumulator array

#returning the indexes (rho, theta) based on the accumulator value chosen

def maxIndex(accumulator,theta):

    indexes = []

    reverseSorted = np.sort(np.ndarray.flatten(accumulator))[::-1]

    indexes =[]

    for i in range(0,len(reverseSorted[0:150])):

        val = reverseSorted[i]

        index=[(x,y) for x, row in enumerate(accumulator) for y, i in enumerate(row) if i == val]

        indexes.append(index)

    return indexes


#based on the above final_indexes, eliminating if not local maxima

#using 3*3 mask to identify the neighbouring pixels and suppress

#returns the final accumulator array with the suppressed values if not maxima

def changedAccumulator(accumulator,final_indexes):

    for k in range(0,len(final_indexes)):

        indexes = final_indexes[k]

        for l in range(0,len(indexes)):

            index = indexes[l]

            i = index[0]

            j = index[1]

            for x in range(i-1,i+1):

                for y in range(j-1,j+1):

                    if accumulator[i][j] > 0 and accumulator[i][j] > accumulator[x][y]:

                        accumulator[x][y] = 0

    return accumulator



#choosing the values greater than a particular threshold value

#returns the (rho,theta) value above a particular threshold

def getPeakValues(accumulator,maximumDistance):
```

Computer Vision Project 1
Ayan Agrawal
aka398
N15025000

```
    peakList = []

    thetas = [i for i in range(-90,91,1)]

    for i in range(0,accumulator.shape[0]):

        for j in range(0,accumulator.shape[1]):

            x = accumulator[i,j]

            if x > 100:

                peakList.append([i-maximumDistance,thetas[j]])

    return peakList


#eliminating the values which are near in rho's and thetas

#eliminating the redundant rho,theta

def removerepeatedLines(peakValues):

    redundantvalues = []

    for i in range(0,len(peakValues)-1):

        temp_i = peakValues[i]

        p_i = temp_i[0]

        t_i = temp_i[1]

        for j in range(i+1,len(peakValues)):

            temp_j = peakValues[j]

            p_j = temp_j[0]

            t_j = temp_j[1]

            if abs(p_i - p_j) < 15 and abs(t_i - t_j) < 4 :

                redundantvalues.append(peakValues[i])

    for i in range(0,len(redundantvalues)):

        if redundantvalues[i] in peakValues:

            peakValues.remove(redundantvalues[i])


    return peakValues




#it detects the edges of binary image

#whichever is satisfying the (rho, theta) equation take that and plot the line segments on the original colour image

def drawLineSegment(peakValues,binaryImage, image_object_for_line_segment_superimpose):

    pixelsAt = []

    for k in range(0,len(peakValues)):

        for i in range(0,binaryImage.size[0]):
```

```
        for j in range(0,binaryImage.size[1]):

            temp = peakValues[k]

            p = temp[0]

            t = math.radians(temp[1])

            if p == int(i*(math.cos(t)) + (j)*(math.sin(t))):

                image_object_for_line_segment_superimpose.putpixel((i,j),(255,0,0))

                pixelsAt.append([i,j])

    return image_object_for_line_segment_superimpose,pixelsAt



#it finds the coordinates of the intersecting points. These are not the coordinates of the parallelogram

#this function returns all the coordinates where the lines are intersecting

def findCoordinates(pixelsAt,peakValues):

    coordinate_points = []

    coordiantesAt =[]

    for i in range(0,len(pixelsAt)):

        pixel = pixelsAt[i]

        for k in range(0,len(peakValues)-1):

            temp_k = peakValues[k]

            p_k = temp_k[0]

            t_k = math.radians(temp_k[1])

            for l in range(k+1,len(peakValues)):

                temp_l = peakValues[l]

                p_l = temp_l[0]

                t_l = math.radians(temp_l[1])

                if p_k == int(pixel[0]*math.cos(t_k) + pixel[1]*math.sin(t_k)) and p_l == int(pixel[0]*math.cos(t_l) + pixel[1]*math.sin(t_l)):

                    coordinate_points.append(pixelsAt[i])

    for i in range(0,len(coordinate_points)):

        if coordinate_points[i] not in coordiantesAt:

            coordiantesAt.append(coordinate_points[i])

    return coordiantesAt



#finds the parallel lines having the same theta value

#approximation is taken with theta in some range consider that to be equal

#returns the rho,theta value in pair
```

Computer Vision Project 1
Ayan Agrawal
aka398
N15025000

```
def findParallel(peakValues):
    parallelLines = []
    for i in range(0,len(peakValues)-1):
        temp_i = peakValues[i]
        for j in range(i+1,len(peakValues)):
            temp_j = peakValues[j]
            if abs(temp_i[1] - temp_j[1]) < 7:
                parallelLines.append([peakValues[i],peakValues[j]])
    return parallelLines



#combine the theta having similar(nearly equal) values
#this is used in the parallelogram formation
#later, used to compare the theta values
def combineParallelLines(parallelLines):
    parallelTheta = []
    for i in range(len(parallelLines)):
        parallelTheta.append([])
        for j in range(len(parallelLines[i])):
            parallelTheta[i].append(parallelLines[i][j][1])
    return parallelTheta



#the coordinates found in above defination will return all intersecting lines coordinates
#using that find only those values which lie on parallel line
def pointsCorrespondingToParallel(parallelLines, coordinates):
    parallelCoordinateSet = {}
    for subList in parallelLines:
        for lines in subList:
            temp_theta = math.radians(lines[1])
            parallelCoordinateSet[lines[1]] = []
            for points in coordinates:
                #print int(points[0]*math.cos(temp_theta) + points[1]*math.sin(temp_theta))
                if (lines[0] == int(points[0]*math.cos(temp_theta) + points[1]*math.sin(temp_theta))):
                    parallelCoordinateSet[lines[1]].append([points[0], points[1]])
    return parallelCoordinateSet
```

Computer Vision Project 1
Ayan Agrawal
aka398
N15025000

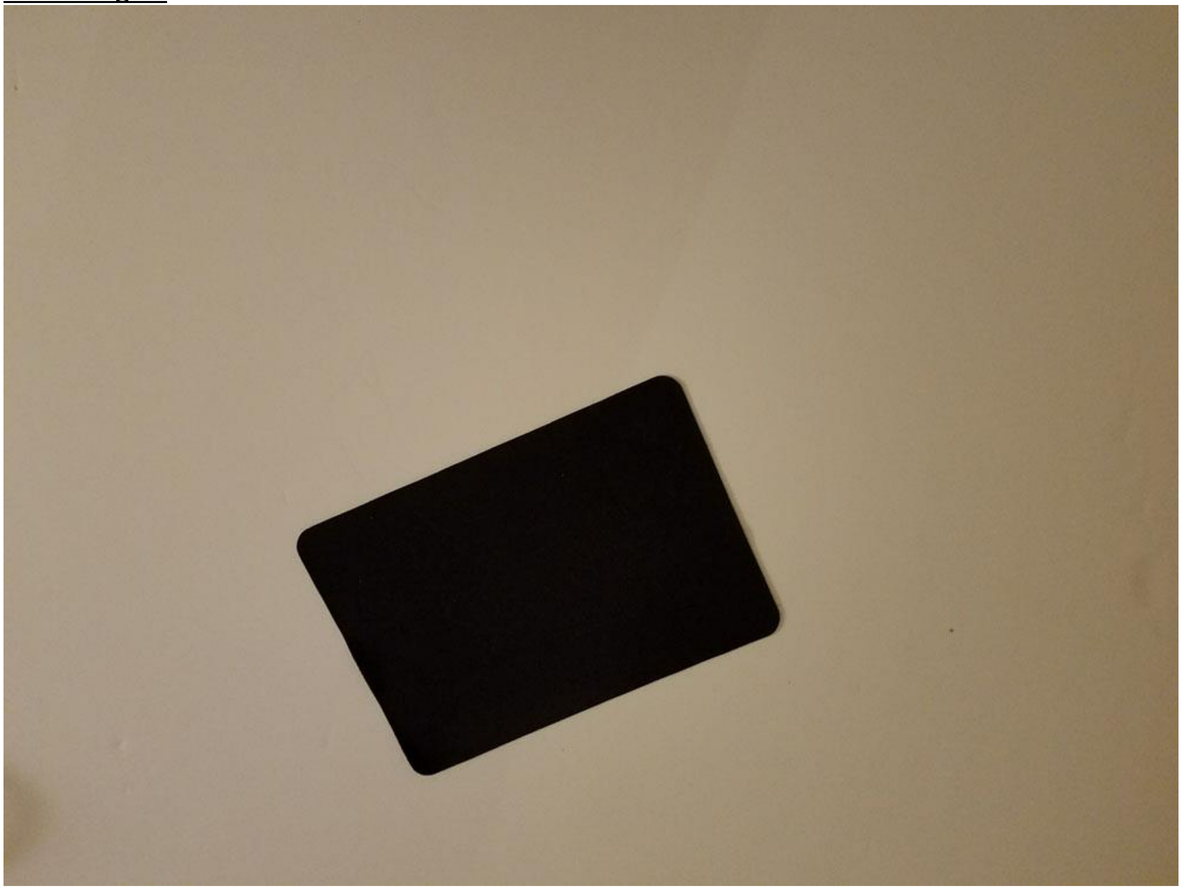
```
def finalCoordinates(pointsCorrespondingToParallel, sameTheta):
    coordinates = []
    for theta in pointsCorrespondingToParallel:
        for i in range(len(pointsCorrespondingToParallel[theta])):
            first_coordinate = pointsCorrespondingToParallel[theta][i]
            x1 = first_coordinate[0]
            y1 = first_coordinate[1]
            for j in range(len(sameTheta)):
                if theta in sameTheta[j]:
                    for other_thetas in sameTheta[j]:
                        if other_thetas!= theta:
                            for k in range(len(pointsCorrespondingToParallel[other_thetas])):
                                second_coordinate = pointsCorrespondingToParallel[other_thetas][k]
                                x2 = second_coordinate[0]
                                y2 = second_coordinate[1]
                                angle = int(math.degrees(math.atan2((y2-y1),(x2-x1))))
                                if angle<0:
                                    angle = angle + 90
                                else:
                                    angle = angle - 90
                                if angle in pointsCorrespondingToParallel:
                                    coordinates.append([first_coordinate,second_coordinate])
    return coordinates


def drawParallelogram(coordinates, image_object_for_parallelogram_superimpose):
    draw = ImageDraw.Draw(image_object_for_parallelogram_superimpose)
    for segmentCoordinate in coordinates:
        draw.line((segmentCoordinate[0][0],segmentCoordinate[0][1],segmentCoordinate[1][0],segmentCoordinate[1][1]), fill = 128, width = 1)
    #image_object_for_parallelogram_superimpose.save("---PATH TO SAVE THE FILE---", "JPEG")
    image_object_for_parallelogram_superimpose.show()
```

Computer Vision Project 1
Ayan Agrawal
aka398
N15025000

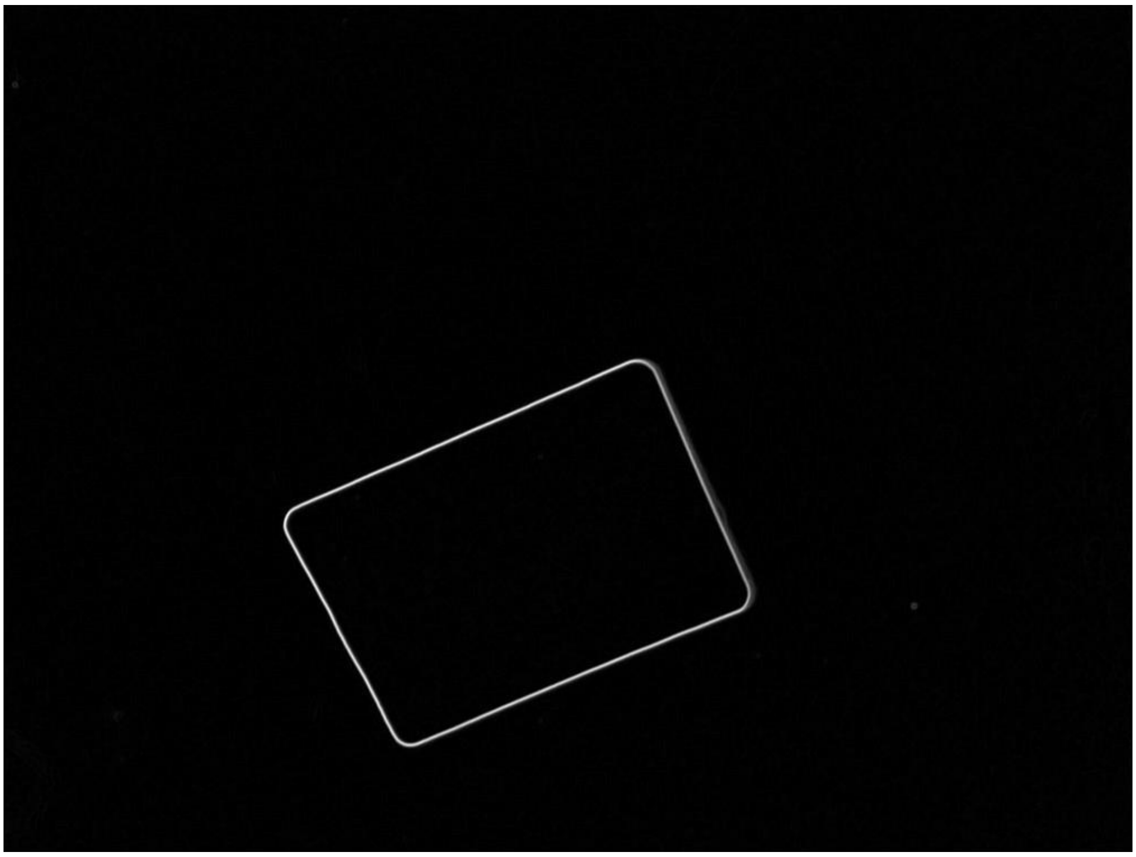
```
if __name__ == "__main__":

    image_object = Image.open(sys.argv[1])

    image_object_for_line_segment_superimpose = Image.open(sys.argv[1])

    image_object_for_parallelogram_superimpose = Image.open(sys.argv[1])

    grayLevelImage = convertToGrayLevel(image_object)

    smoothedImage = smoothImage(grayLevelImage)

    sobelMagnitudeImage,gradientAngles = SobelOperator(smoothedImage)

    #sobelMagnitudeImage.show()

    #sobelMagnitudeImage.save("---PATH TO SAVE THE FILE---", "JPEG")

    quantizedGradientAngles = quantizeAngles(gradientAngles)

    nonMaximasuppressedImage = nonMaximaSuppression(sobelMagnitudeImage,quantizedGradientAngles)

    binaryImage, edgesAt = binaryConversion(nonMaximasuppressedImage)

    #binaryImage.show()

    #binaryImage.save(""---PATH TO SAVE THE FILE---", "JPEG")

    accumulator,maximumDistance = formAccumulatorArray(edgesAt,binaryImage)

    thetas = [i for i in range(-90,91,1)]

    final_indexes=maxIndex(accumulator,thetas)

    accumulator = changedAccumulator(accumulator,final_indexes)

    peakValues = getPeakValues(accumulator,maximumDistance)

    removeTheseValues = peakValues

    peakValues = removerepeatedLines(removeTheseValues)

    drawLineSegmentImage,pixelsAt = drawLineSegment(peakValues,binaryImage, image_object_for_line_segment_superimpose)

    #drawLineSegmentImage.show()

    coordinates = findCoordinates(pixelsAt,peakValues)

    parallelLines = findParallel(peakValues)

    sameTheta = combineParallelLines(parallelLines)

    pointsCorrespondingToParallel = pointsCorrespondingToParallel(parallelLines, coordinates)

    coordinates = finalCoordinates(pointsCorrespondingToParallel, sameTheta)

    #print coordinates

    drawParallelogram(coordinates, image_object_for_parallelogram_superimpose)
```

Computer Vision Project 1
Ayan Agrawal
aka398
N15025000

4) <u>Test Images:</u>

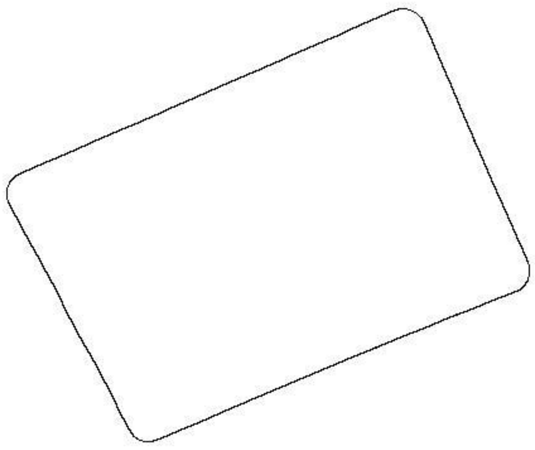

Original Image

Computer Vision Project 1
Ayan Agrawal
aka398
N15025000



Normalised gradient Magnitude Image

Computer Vision Project 1
Ayan Agrawal
aka398
N15025000



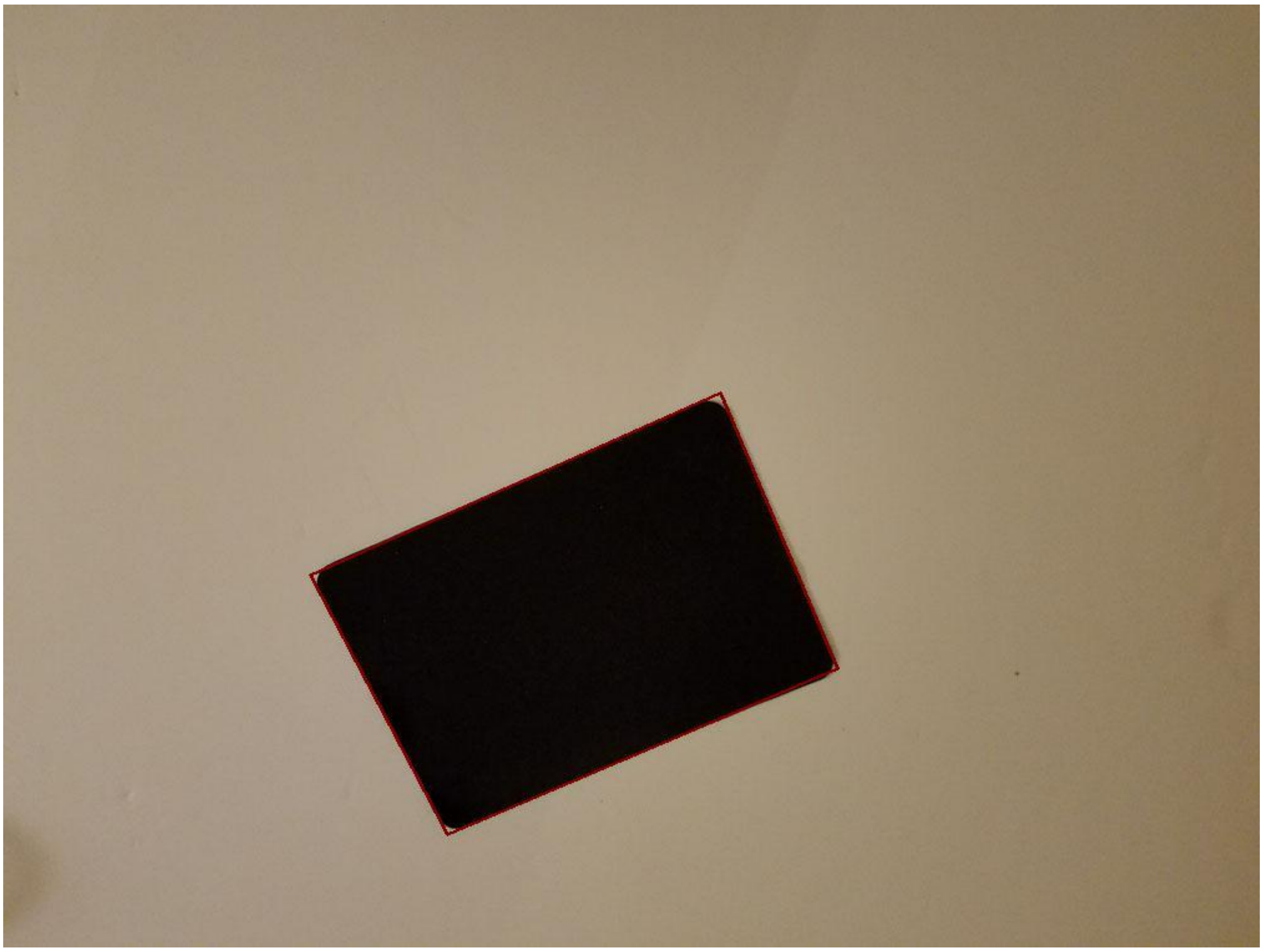
Binary Image

<u>Threshold Value for Image1</u> is 115

<u>Coordinates of Image 1</u>

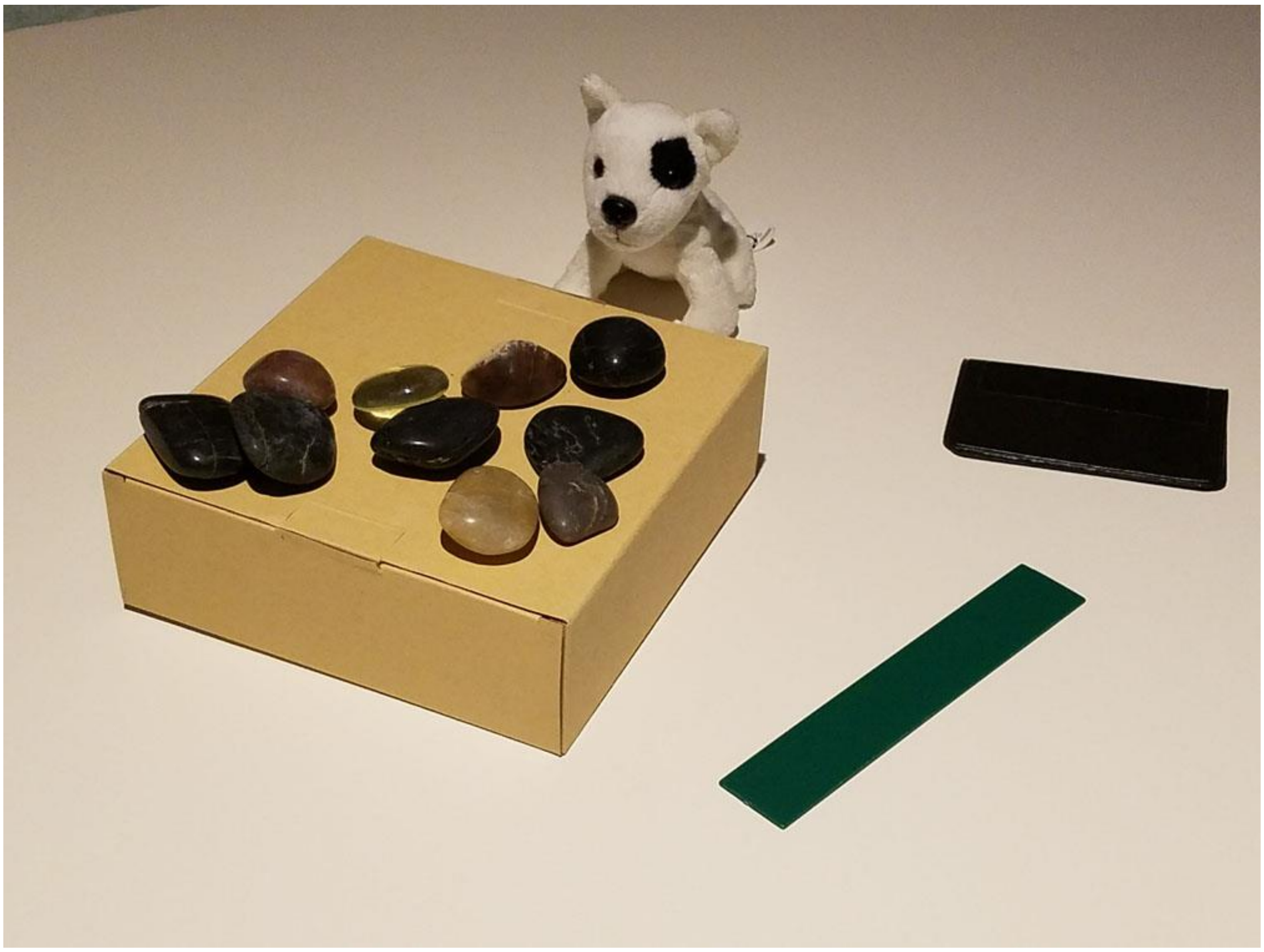
[575, 311] ; [246, 457] ; [669, 532] ; [356, 665]

Computer Vision Project 1
Ayan Agrawal
aka398
N15025000



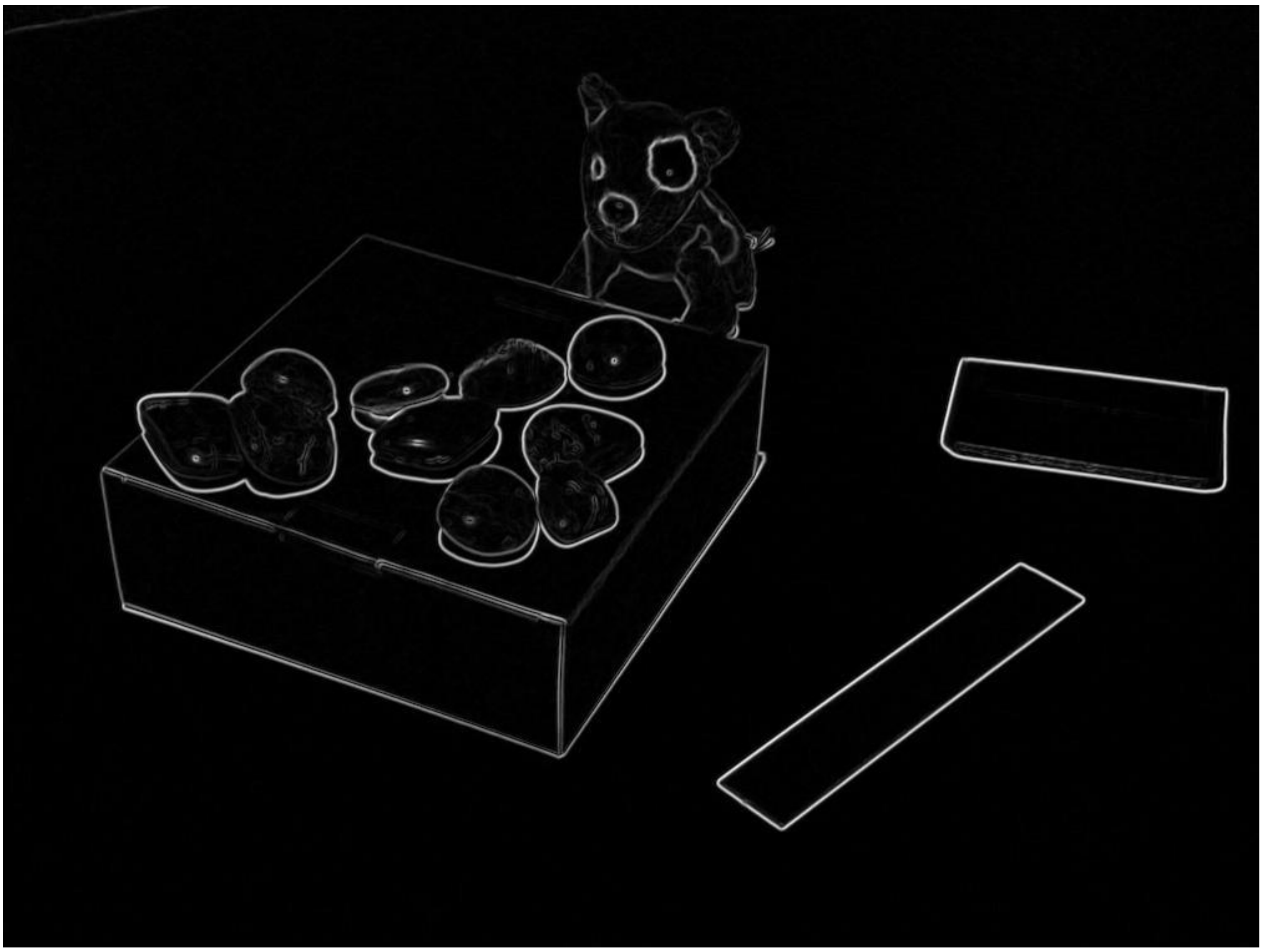
Parallelogram Superimposed Image

Computer Vision Project 1
Ayan Agrawal
aka398
N15025000



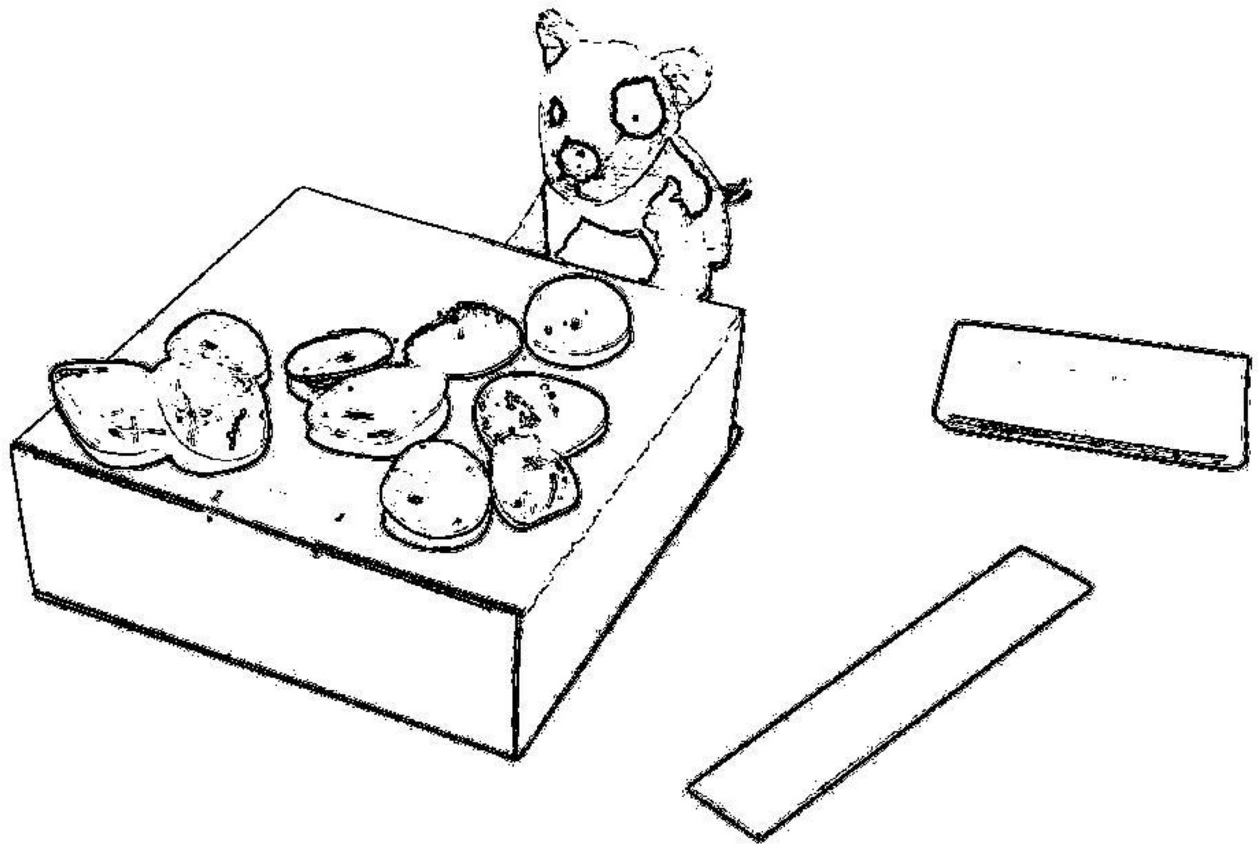
Original Image

Computer Vision Project 1
Ayan Agrawal
aka398
N15025000



Normalised gradient Magnitude Image

Computer Vision Project 1
Ayan Agrawal
aka398
N15025000



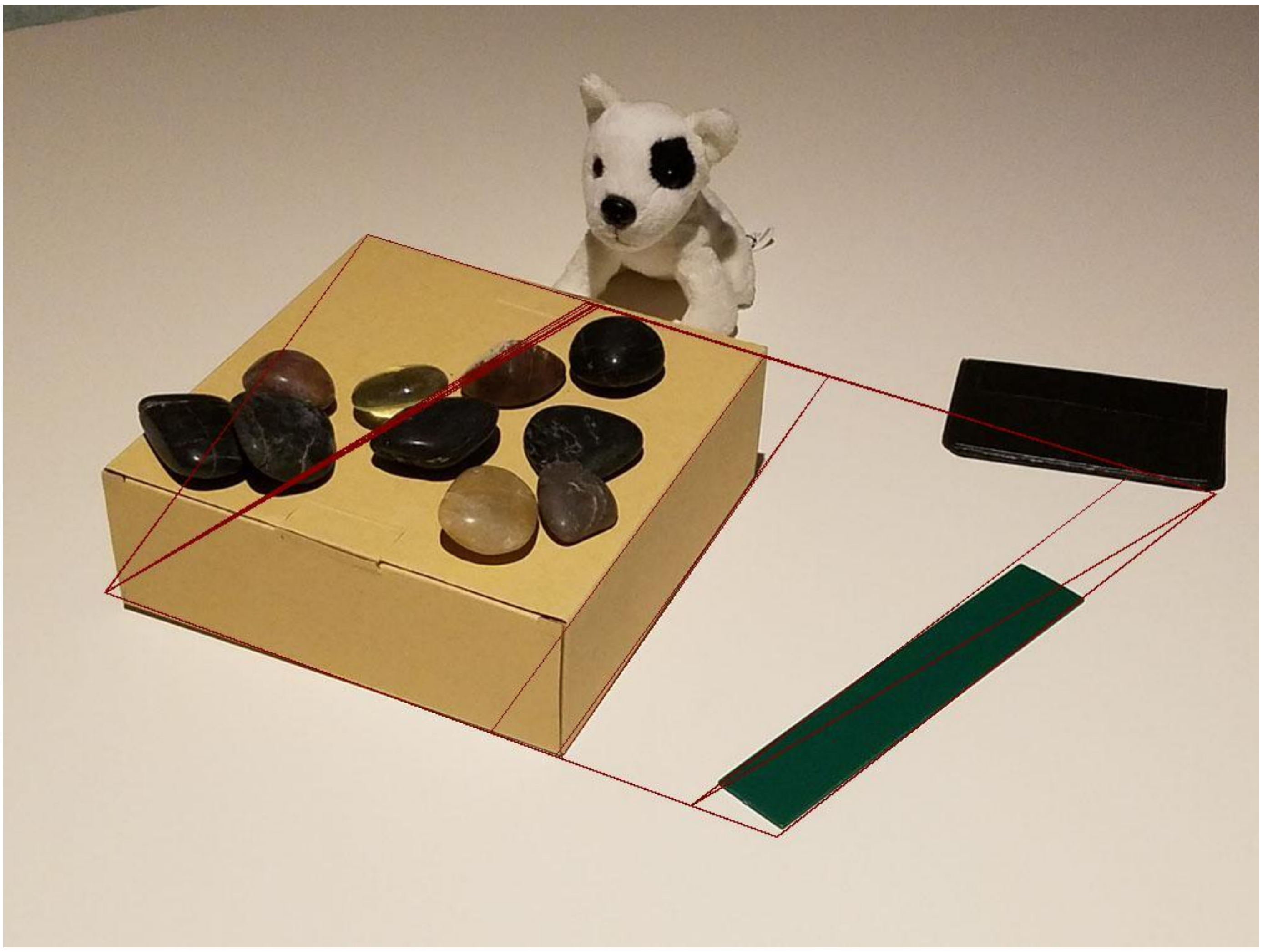
Binary Image

Threshold Value for Image2 is 20

Coordinates of Image 2

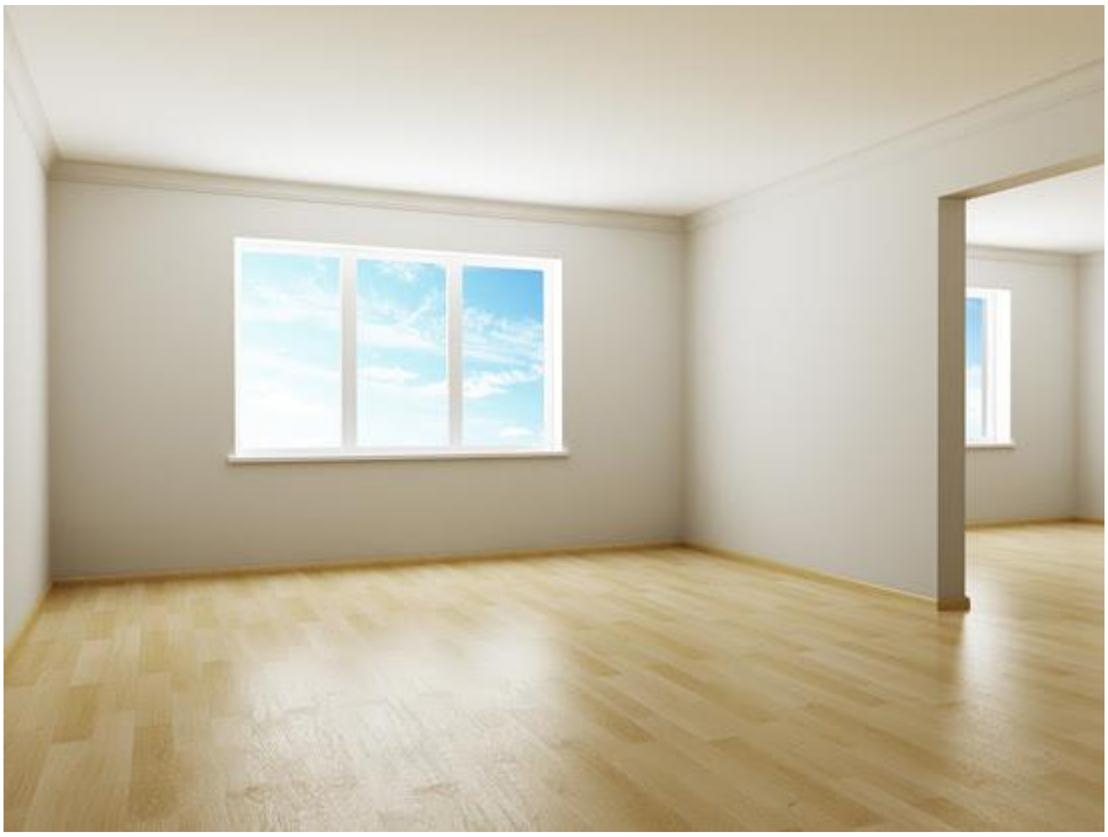
[467, 239], [474, 240], [480, 241], [292, 184], [611, 281], [391, 583], [81, 471], [447, 604], [662, 297], [551, 642], [620, 667], [909, 373], [972, 392], [986, 314], [987, 314], [434, 730]

Computer Vision Project 1
Ayan Agrawal
aka398
N15025000

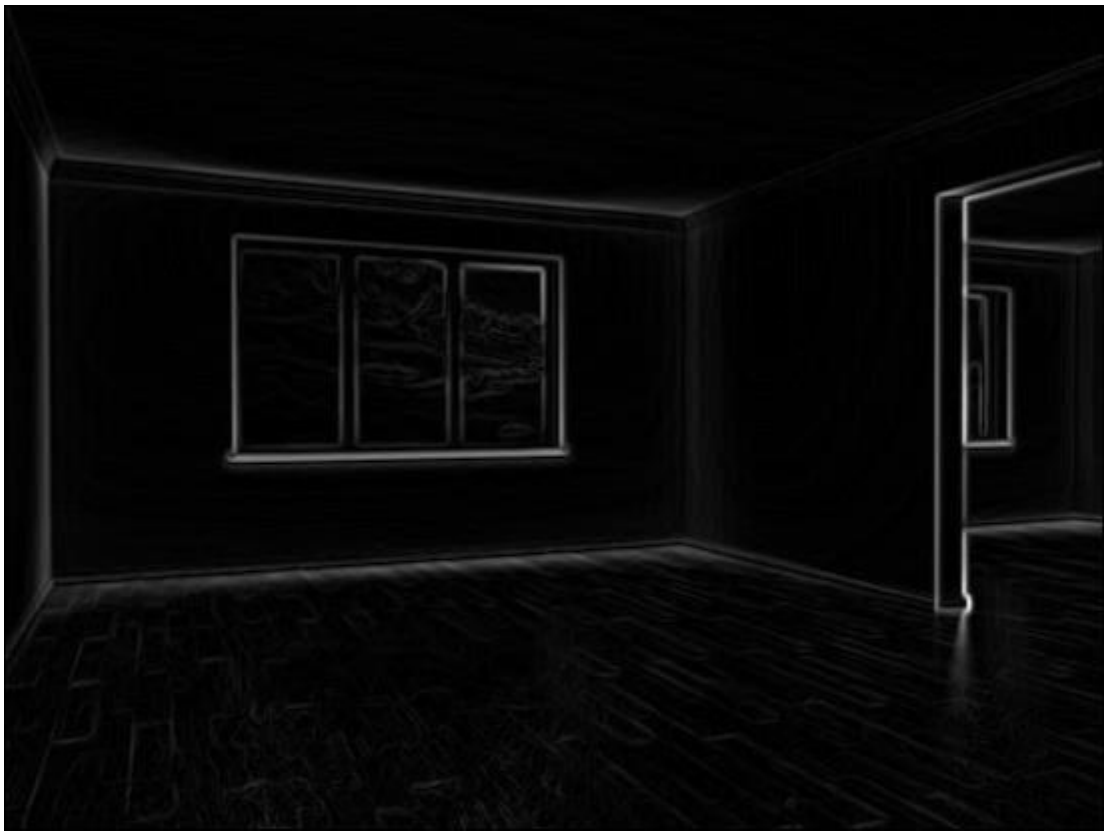


Superimposed Image

Computer Vision Project 1
Ayan Agrawal
aka398
N15025000

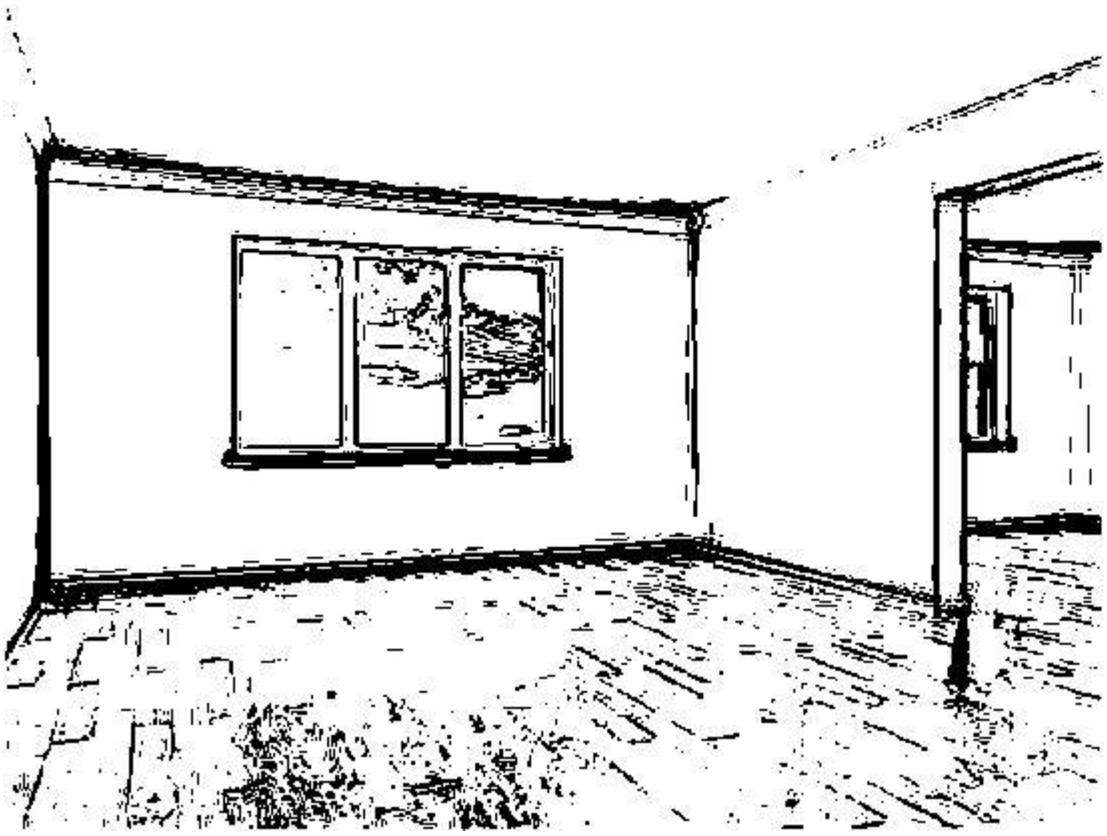


Original Image



Normalized Gradient Magnitude

Computer Vision Project 1
Ayan Agrawal
aka398
N15025000



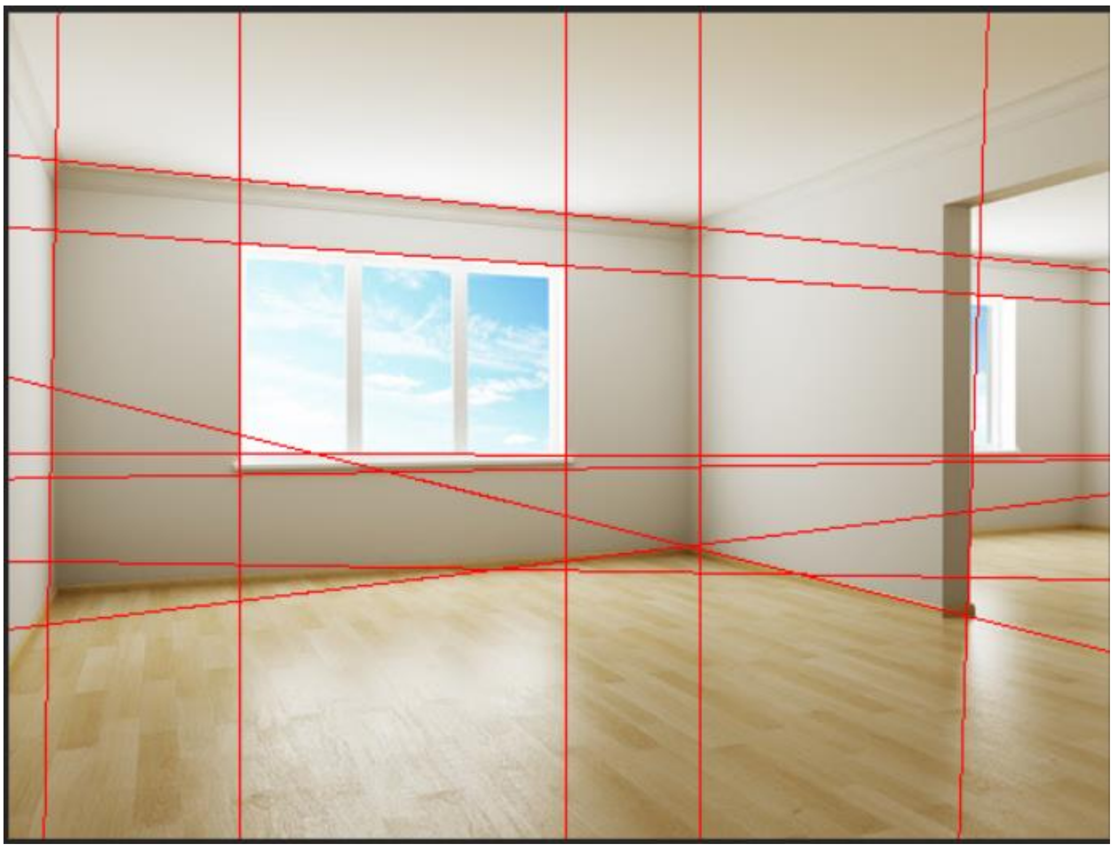
Binary Image

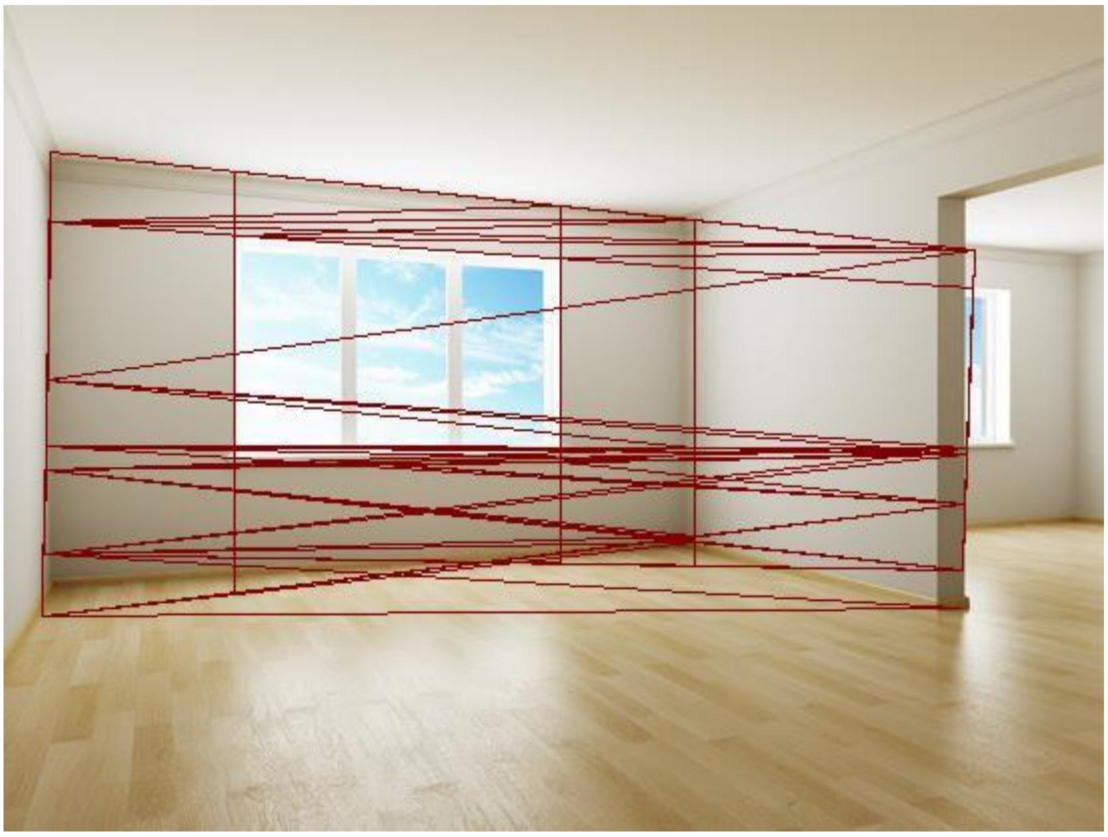
<u>Threshold Value for Image3</u> is 18

<u>Coordinates of Image 3</u>

[21, 187], [21, 187], [345, 268], [345, 227], [345, 268], [345, 268], [345, 268], [478, 301], [478, 301], [20, 232], [21, 2 20], [23, 73], [21, 220], [23, 73], [115, 83], [278, 278], [21, 220], [23, 73], [115, 276], [115, 83], [278, 278], [345, 131] , [345, 221], [345, 131], [484, 141], [481, 221], [23, 73], [20, 274], [23, 73], [21, 220], [23, 73], [115, 83], [115, 276], [ 115, 83], [115, 220], [115, 83], [115, 115], [278, 278], [278, 100], [278, 221], [278, 100], [278, 126], [345, 280], [345, 221], [23, 108], [345, 107], [345, 131], [479, 282], [485, 122], [481, 221], [20, 232], [115, 231], [20, 232], [345, 265], [345, 227], [187, 229], [188, 229], [189, 229], [190, 229], [278, 228], [20, 232], [337, 266], [20, 232], [338, 266], [20, 232], [339, 266], [20, 232], [19, 305], [115, 231], [115, 294], [278, 228], [278, 274], [345, 227], [345, 265], [481, 224] , [480, 249]

Computer Vision Project 1
Ayan Agrawal
aka398
N15025000





Superimposed image