PARALLELOGRAM DETECTION

Project by: SURBHI THOLE
NYU ID: N13263273
NET ID: SST390

Guided by-
Prof. Edward Wong

Index

A) Details on how to detect parallelograms:

The steps to detect parallelogram are as follows:

1. Convert the image to gray scale image
2. Apply edge detection Algorithm
3. Apply a threshold to sharpen the edges
4. Use hough transform to detect straight lines in image
5. Applying threshold to hough space
6. Convert lines detected from hough space to image space
7. Convolve lines in image space with edge image
8. Detect parallelogram (objects) in Image:
9. Detect the coordinate of parallelograms

1. Convert the image to gray scale image:

First separate the RGB matrices from the original image. Calculating these values traverse over each pixel of the image and using the luminance formula: - Gray = 0.30R + 0.59G + 0.11B calculate the gray values of each pixel. Save the gray value of each pixel in the Gray-image.

2. Edge Detection:

The input to the edge detection algorithm is the Gray scale image we converted in first step. We are using a gradient based edge detection algorithm- Sobel’s operator to detect edges in this step.
The sobel operator is applied to each point in the image. At each point, a horizontal gradient(Sx) and vertical gradient(Sy) is applied to detect changes in intensity of this point compared to its direct neighbor points. An edge is recognized by the difference in intensity of the neighboring pixels. On the other side, if a point and its neighbor points have the same grayscale value, the operator value will be zero which means no change in intensity and thus no possible edge.
The sobel operator uses 3*3 kernel to calculate approximations of the gradients.

$$Sx = \begin{matrix}1 & 0 & -1\\ 2 & 0 & -2\\ 1 & 0 & -1\end{matrix}$$

$$Sy = \begin{matrix}1 & 2 & -1\\ 0 & 0 & 0\\ -1 & -2 & -1\end{matrix}$$

Before applying this filter, we have to zero pad the gray Image to avoid out of bound error when kernel is at corner points and border of image.

This is done using:

```
test1 = np.pad(gray1,(1,1),'constant',constant_values=0)
```

Formula for calculating horizontal and vertical gradient:

```
Sx = test1[i + 1, j + 1] + 2 * test1[i, j + 1] + test1[i - 1, j + 1] - test1[i + 1, j -
1] - 2 * test1[i, j - 1] - test1[i - 1, j - 1]
Sy = test1[i + 1, j - 1] + 2 * test1[i + 1, j] + test1[i + 1, j + 1] - test1[i - 1, j -
1] - 2 * test1[i - 1, j] - test1[i - 1, j + 1]
```

The Final gradient value for each pixel is calculated using:

```
    X = np.sqrt(Sx ** 2 + Sy ** 2)
```

3. Apply threshold to detect actual edges:

After having calculated the edges using the sobel's operator, we have to make sure that all the edges detected are part of the edges. For this we apply a threshold to the image in such a way that the pixels having intensity value less than threshold are removed and only the relevant edges are considered. In this project, we have selected the threshold value manually for each image.

4. Use hough transform to detect straight lines in image:

The input image to hough transform is the thresholded edge image. Lines in input image is represented by points in hough space. Therefore, all collinear points in the line is represented by same coordinate in hough space.

Using the normal form each line in image space is represented by:
rho = x cos(theta) + y sin(theta)
where r is the distance between the closest point of the line to the origin. And theta is the angle between the normal vector from the closest point of the line to the origin and the x-axis.

For each pixel in the thresholded image, if the pixel belongs to edge, calculate its rho and theta and add tuple (rho, theta) in a two dimensional matrix which is also called as voting matrix.

```
for each pixel in image space
{
 if pixel is part of the edge
 {
   for each possible angle
   {
     rho = x * cos(theta) + y * sin(theta);
     houghMatrix[theta][rho]++;
   }
 }
}
```

The range of angle is taken from (0-180) . The range of rho is considered based on the maximum length of a line in an image which is equal to sqrt(x^2 + y^2). Rho might also be negative. Therefore the range of rho is taken (-r,r) or (0,2r).

5. Applying threshold to hough space

As earlier shown in step 3, a threshold is applied after converting each line from image space to hough space to remove noise and to get only the valid lines. As the voting matrix is used in hough space, the global maximum value is not known prior to applying threshold.
One approach is to find the actual maximum value in the hough space and then only those values are kept that exceed x% of the global maximum value.

6. Convert lines detected from hough space to image space

In this step, we convert the lines back from hough space to image space. First of all, to convert the lines back to image space we will need the values of rho and theta from the hough space using the formula:
r = x cos (theta) + y sin (theta)

As we have calculated the rho values from (0,2r), and the actual value of rho should be from (-rho,rho), we have to apply the offset to get the actual values.
Same applies to theta, ideally as the image space is in fourth quadrant in the global coordinate system, we need to consider the angle values from (-90,90). And applying the offset to previously calculated angle values (0-180), we get the actual angle values in image space.

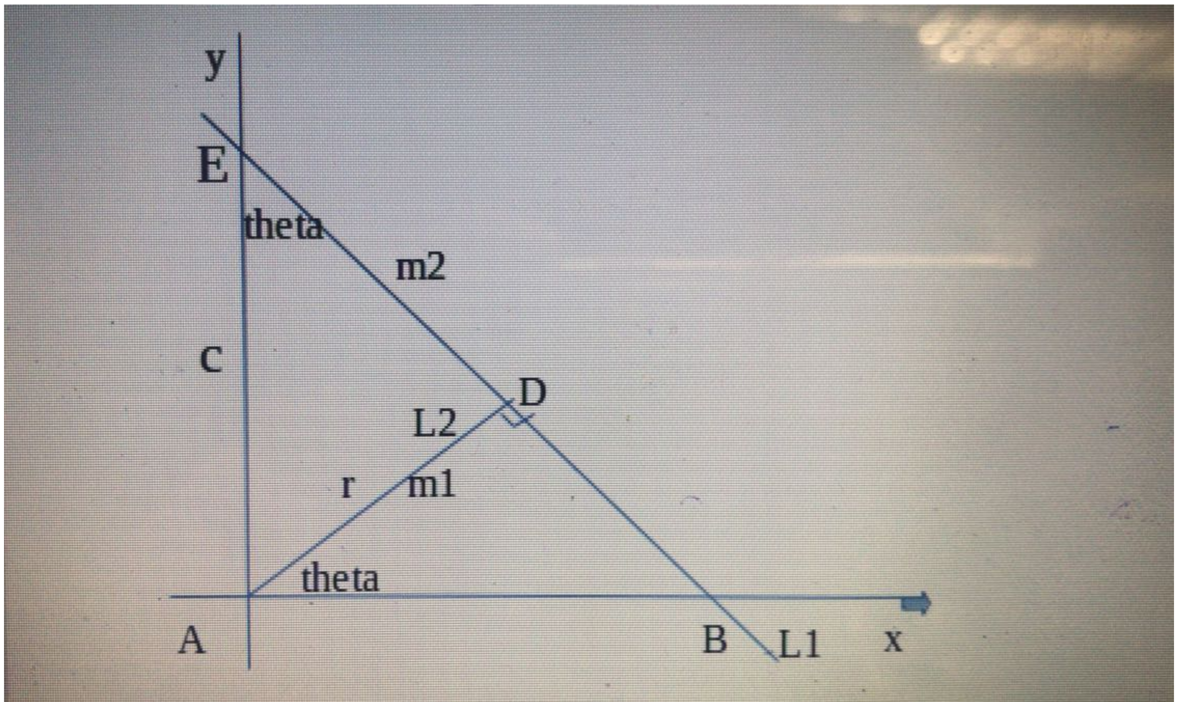


Fig 1

In Fig 1 shown above, L2 is perpendicular to L1.
Slopes of two perpendicular lines: m1*m2 = -1
m2 = -1 / m1
m1 = tan(theta)
therefore, m2 = -1 / tan(theta).....................................(eqn 1)

Now, when x = 0, equation of line becomes y = c
To calculate c = constant in terms of rho and theta,
Consider triangle(AED) in Fig 1 above.

AE = c
angle(AED) = theta
sin(theta) = r / c
therefore, c = r/ sin(theta)……………………………. (eqn 2)

Merging equation 1 and 2, to get the equation of line,
y = (-1 / tan(theta)) * x + (r/ sin(theta)) ……………. (Equation 3)

All the lines in image space are represented by Equation 3.
To get the lines back in image space we need to calculate (x,y) co-ordinates of all the lines detected by hough transform.
This can be obtained by using following pseudocode:

```
set x1=0 ,y1=0, x2=0, y2=0, prev_x=0, prev_y =0
for each rho,theta value:
    for all x in range(Img_col):
        if(theta % 90) == 0:
            continue
        else:
            y = (-1 / tan(theta)) * x + (r/ sin(theta))
            if x1 == 0 and y in range(Img_rows):
                x1 = x
                y1 = y
                prev_x = x
                prev_y = y
                continue
            if x1 != 0:
                if y in range(Img_rows):
                    prev_x = x
                    prev_y = y
                else:
                    x2 = prev_x
                    y2 = prev_y
```

Using the above algorithm we calculate the end points of each line detected by hough transform in image space as x1,y1 and x2,y2.

Using the two end points we draw a line in image space.

7. Convolve lines in image space with edge image:

In step 6, we get multiple lines for a particular edge detected. For this we have to remove duplicate lines. To get the unique lines, the algorithm works as follows:

- Consider the coordinates x1,y1 and x2,y2 of two lines.
- If the absolute value of these co-ordinates is less than dupeLineThresh (by default 20, but can be changed from command line), we remove these lines considering them duplicates.

For displaying the remaining lines, original ones in the gray image we followed the below procedure:

- Create a black image.
- Draw white lines on the black image using coordinates of only those lines which we received as unique lines from the above duplicate lines removal algorithm.
- Then place our black image on edge image from step 3, carry a bit wise "AND" operation on every pixel of the two images. If it returns true, draw a line using those pixels. If it returns false, don't consider the lines from black image as straight lines detected.

In this way we remove the multiple lines detected and convolve the lines with in image space with edge image.

8. Detect parallelogram (objects) in Image:

Now to detect parallelograms in image, consider the objects with straight lines in image in edge image. Suppose there are 4 lines detected with the same angle. The hough transform works by considering all the 4 lines as 1 single line and displays it as a single line. But these lines may be part of different objects.

Therefore to avoid this we apply a threshold in our program such that, consider the lengths of the 4 lines detected. The length of line with maximum length is considered as threshold value. And all the lines below that threshold are removed. In this way we get the edges of the objects of interest.

9. Detect coordinates of parallelograms

To detect the coordinates of parallelogram, we find the intersection point of the two edges of parallelogram. If the distance between the end points of two opposite edges is same, which means that they are parallel, we consider the object to be a parallelogram with the calculated coordinates.

B) Programming Language used:

-Python

C) Instructions on how to run program:

Run the following commands for the testimages in terminal.

1)TestImage1c :

```
python FinalCV.py --testImage TestImage1c.jpg --houghProb 0.7 --
scaleWindows 0.6 --sobelThreshold 125
```

2) TestImage2c :

```
python FinalCV.py --testImage TestImage2c.jpg --houghProb 0.5 --
scaleWindows 0.75 --sobelThreshold 310
```

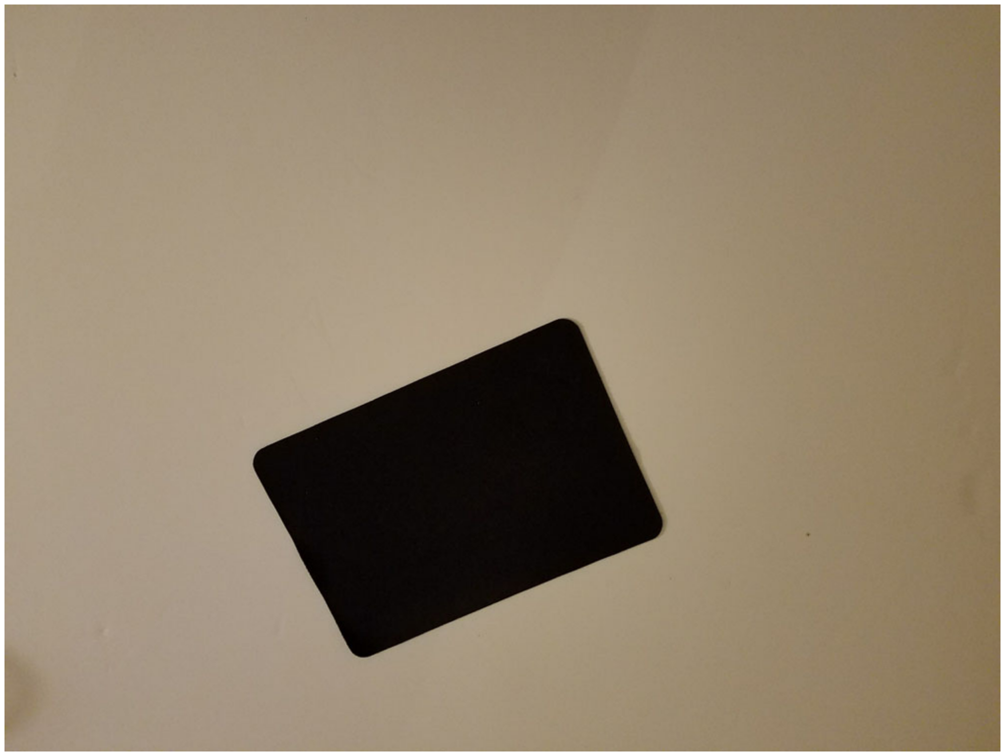
3) TestImage3 :

```
python FinalCV.py --TestImage testImage3.jpg --houghProb 0.4 --
scaleWindows 0.75 --sobelThreshold 110
```

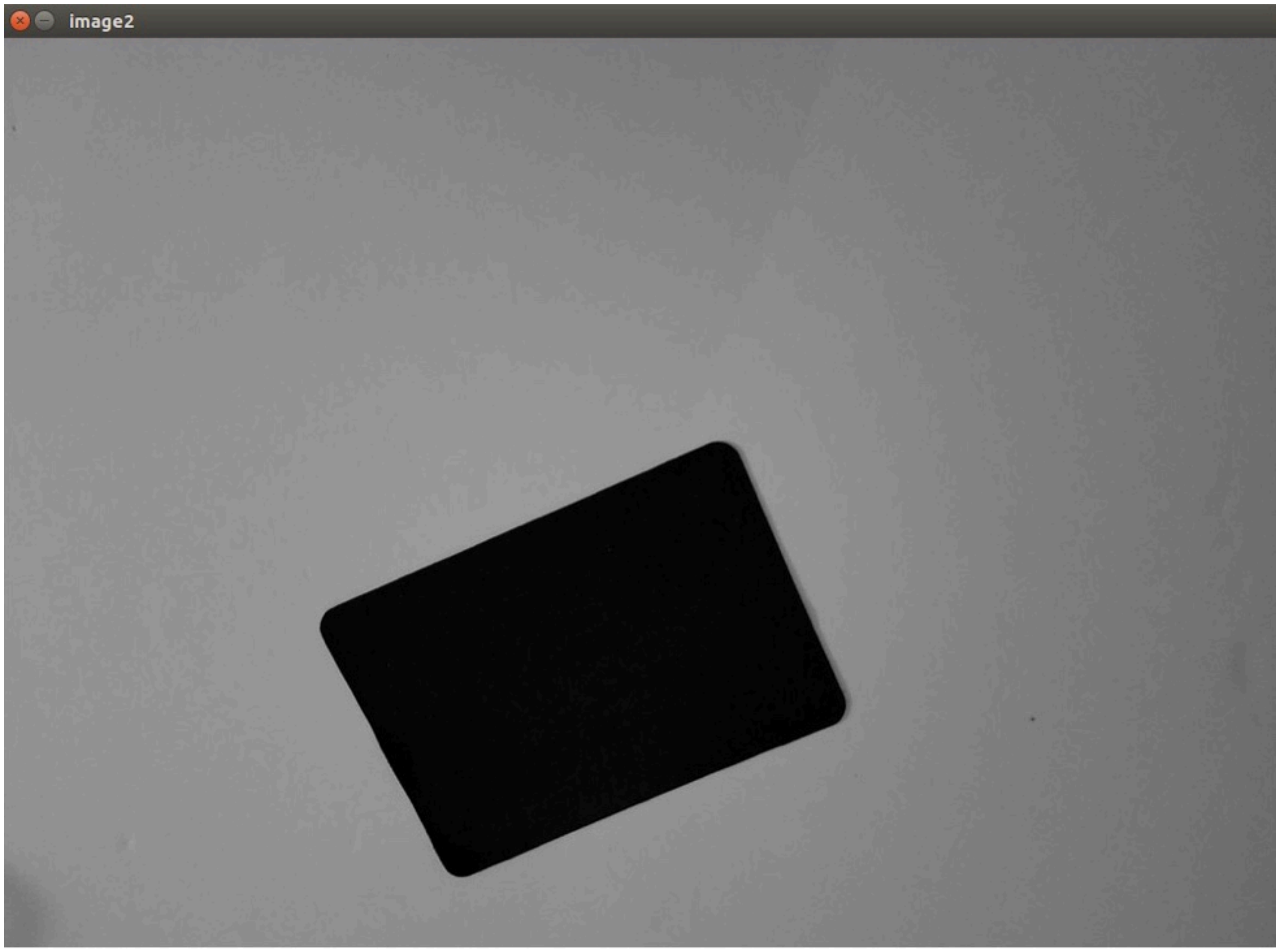
E. Output Images:

A) TestImage1c:

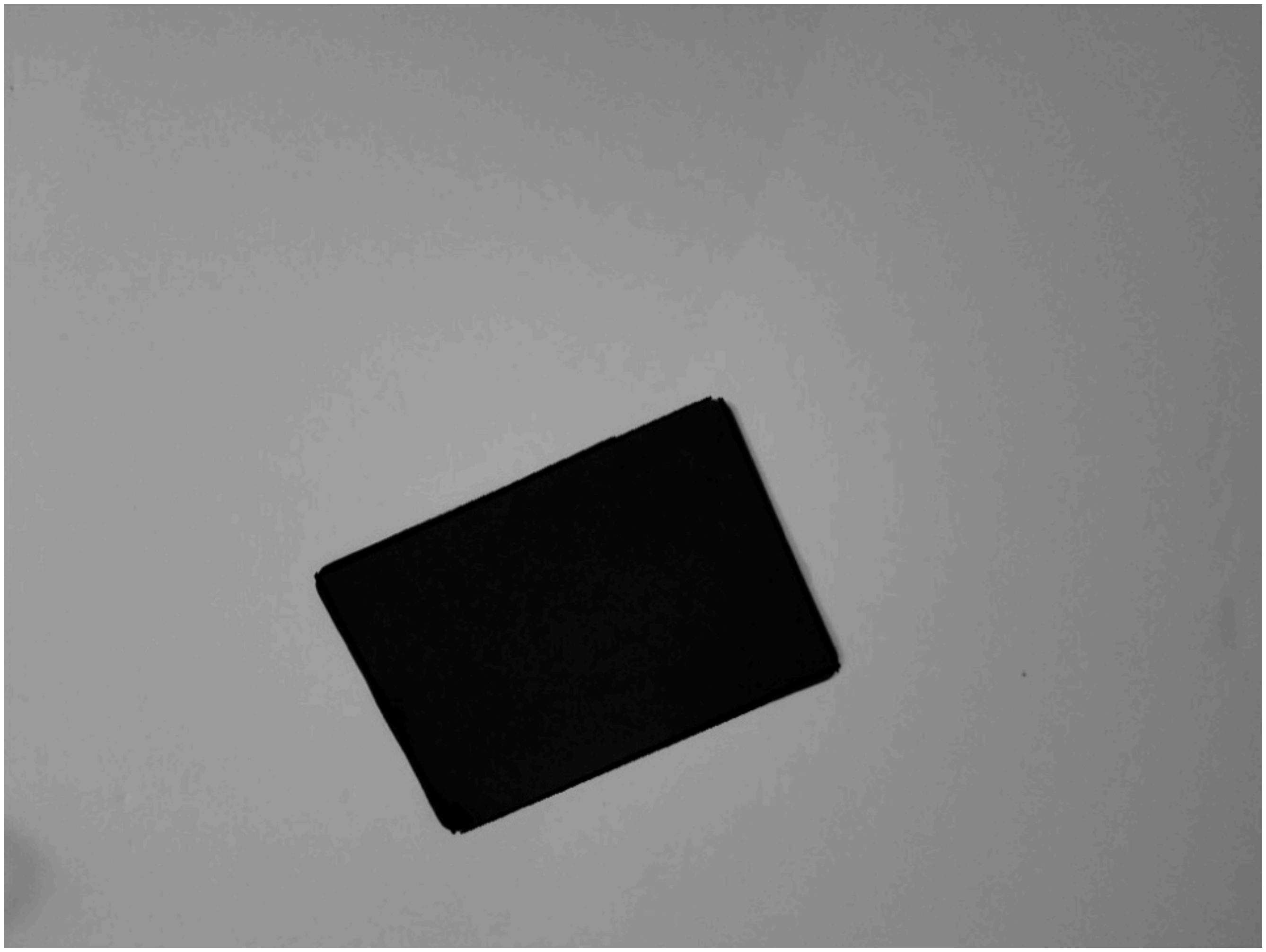
1) Original Image



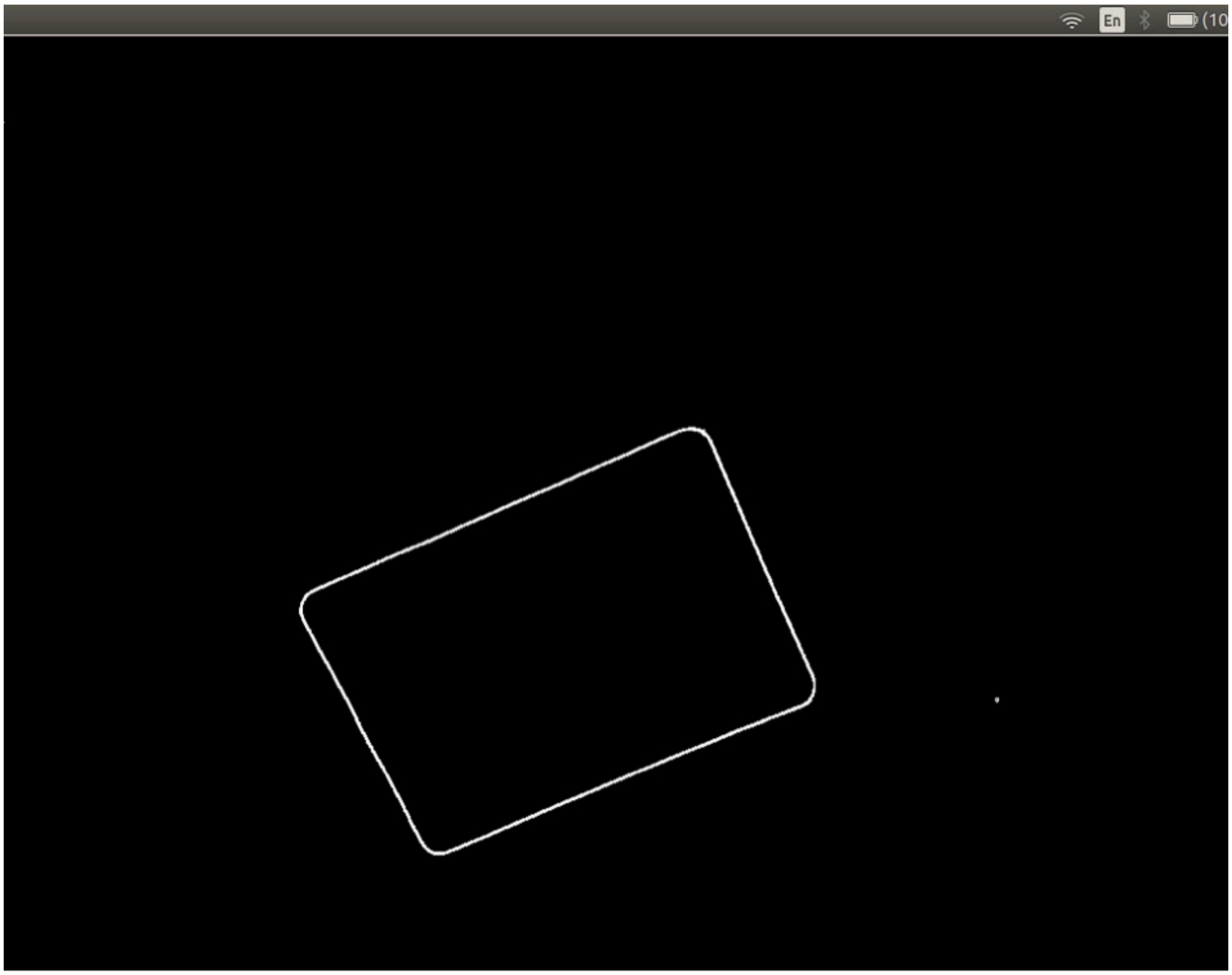
2) Gray Image



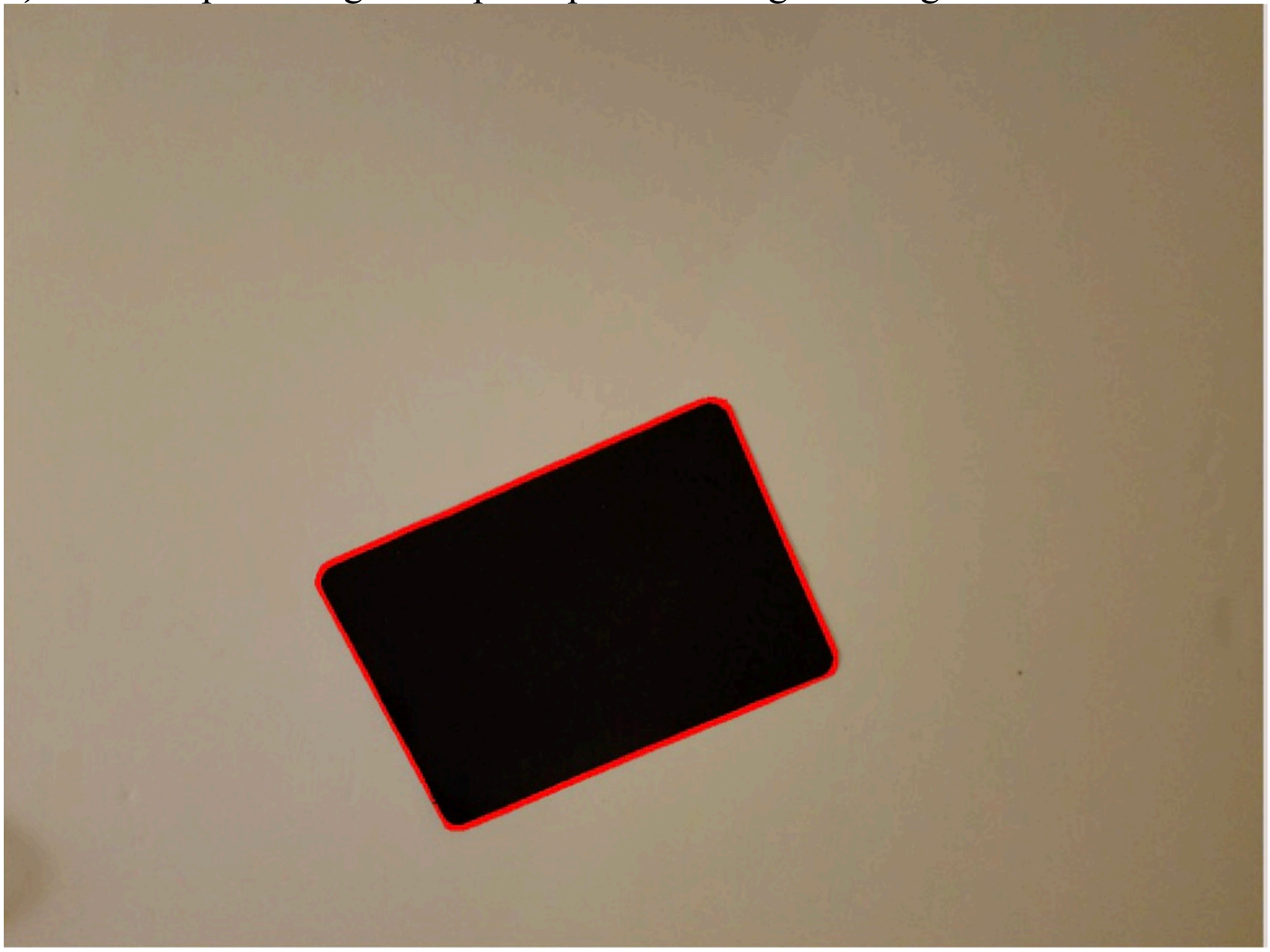
3) Gray scale image with detected lines superimposed.



4) Binary Image



5) Detected parallelogram superimposed on original image:



6) Detected coordinates of parallelogram

(573,317)

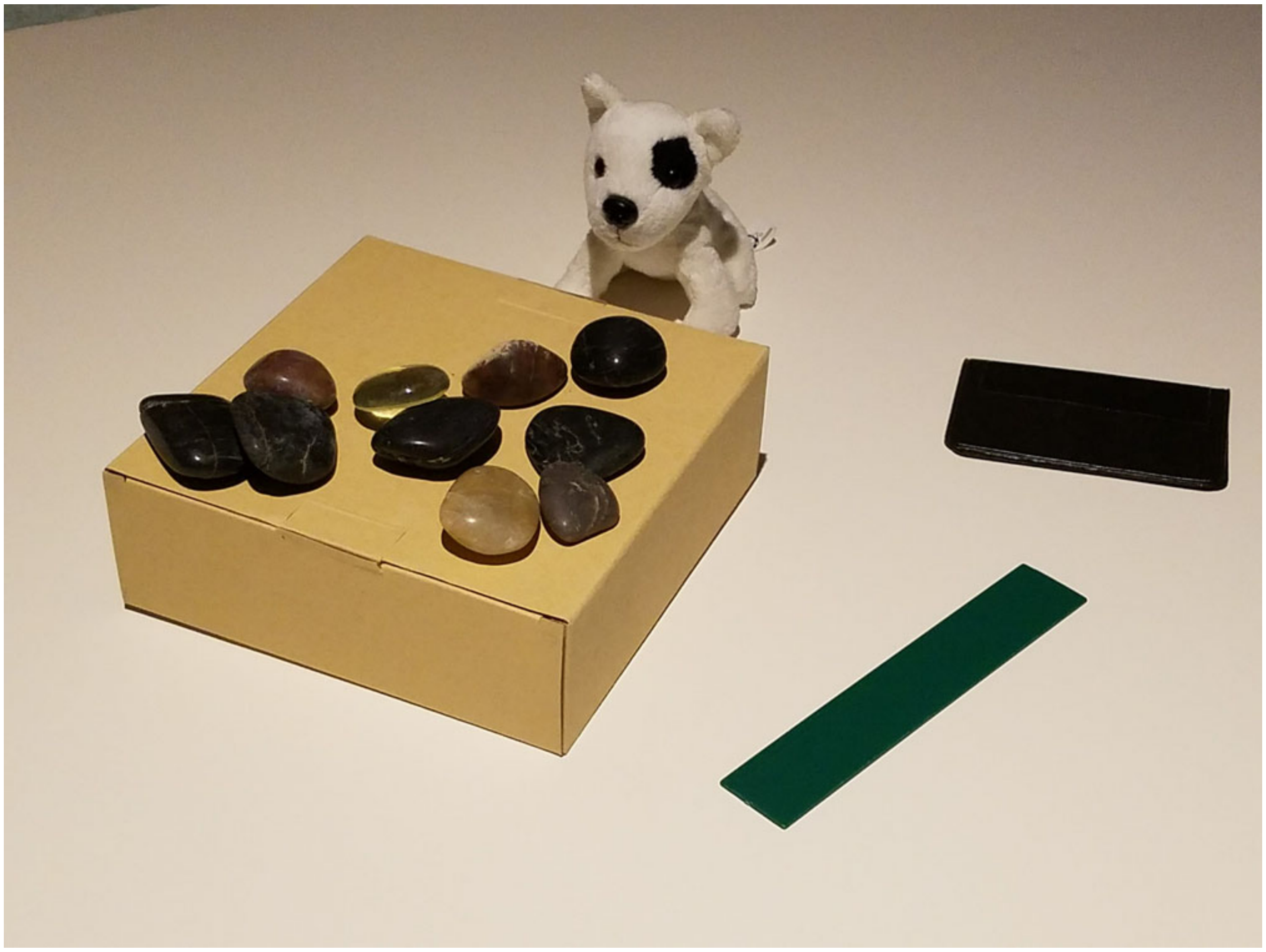
(665,533)

(252,456)

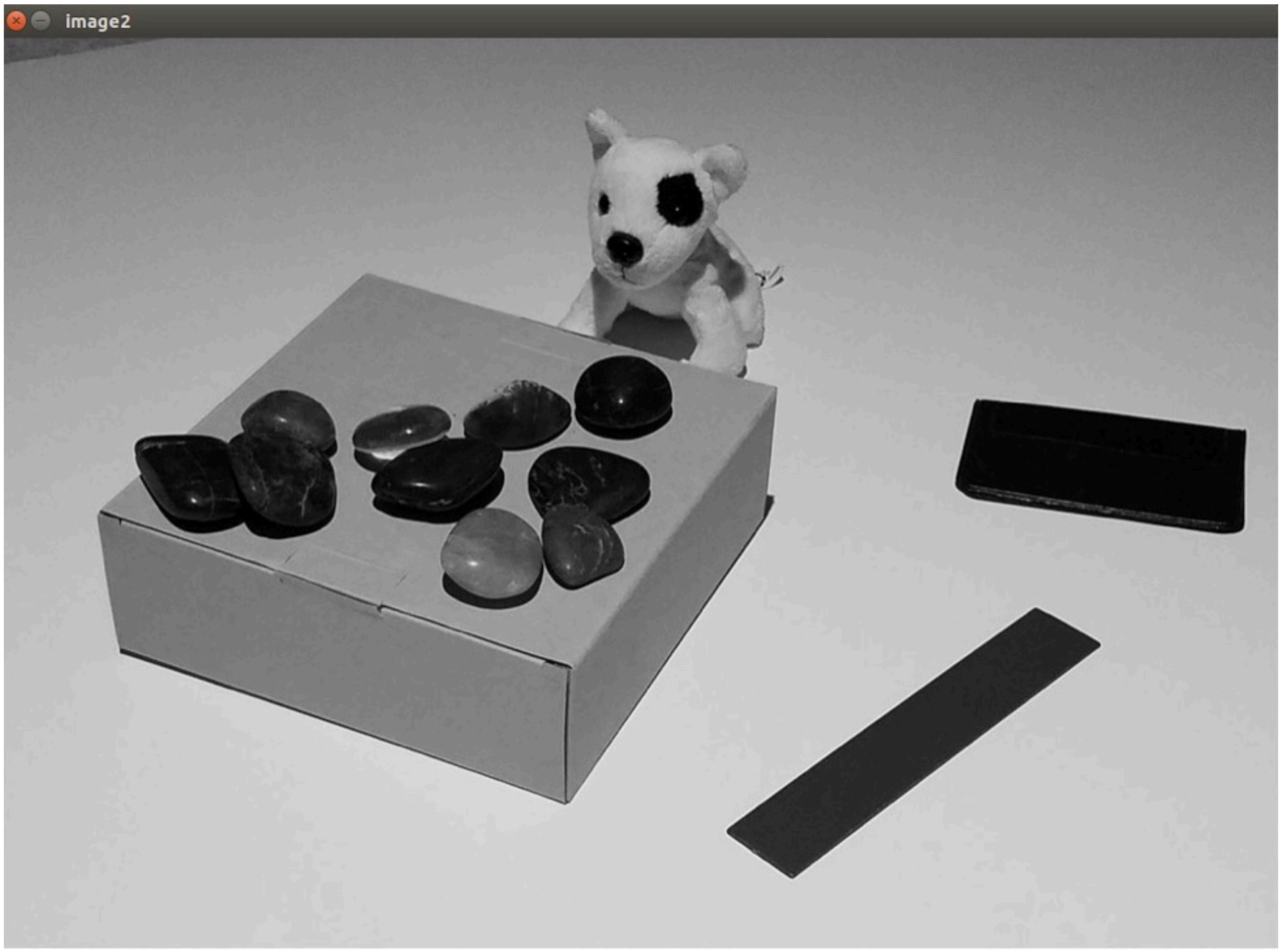
(361,661)

B) TestImage2c

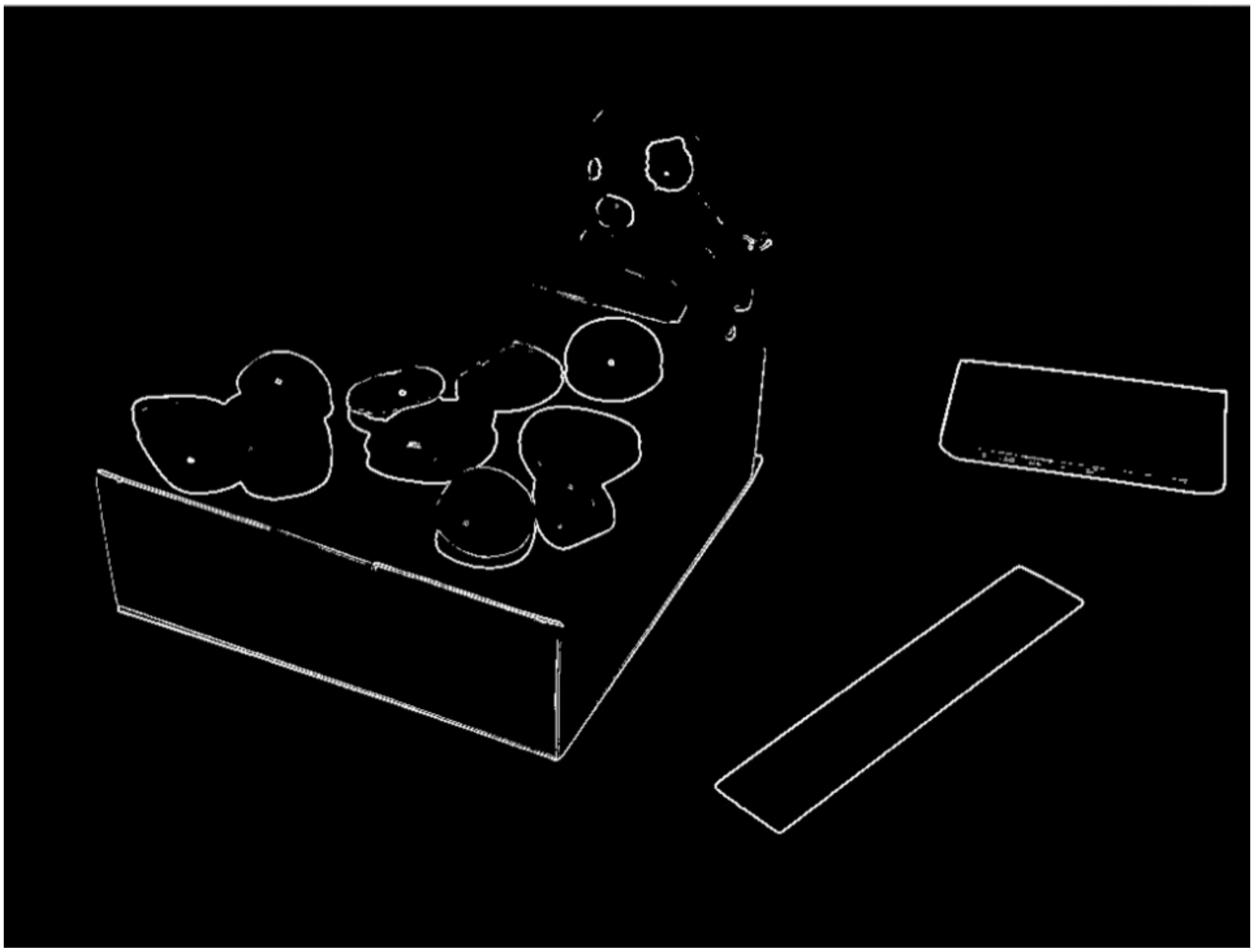
1)Original Image



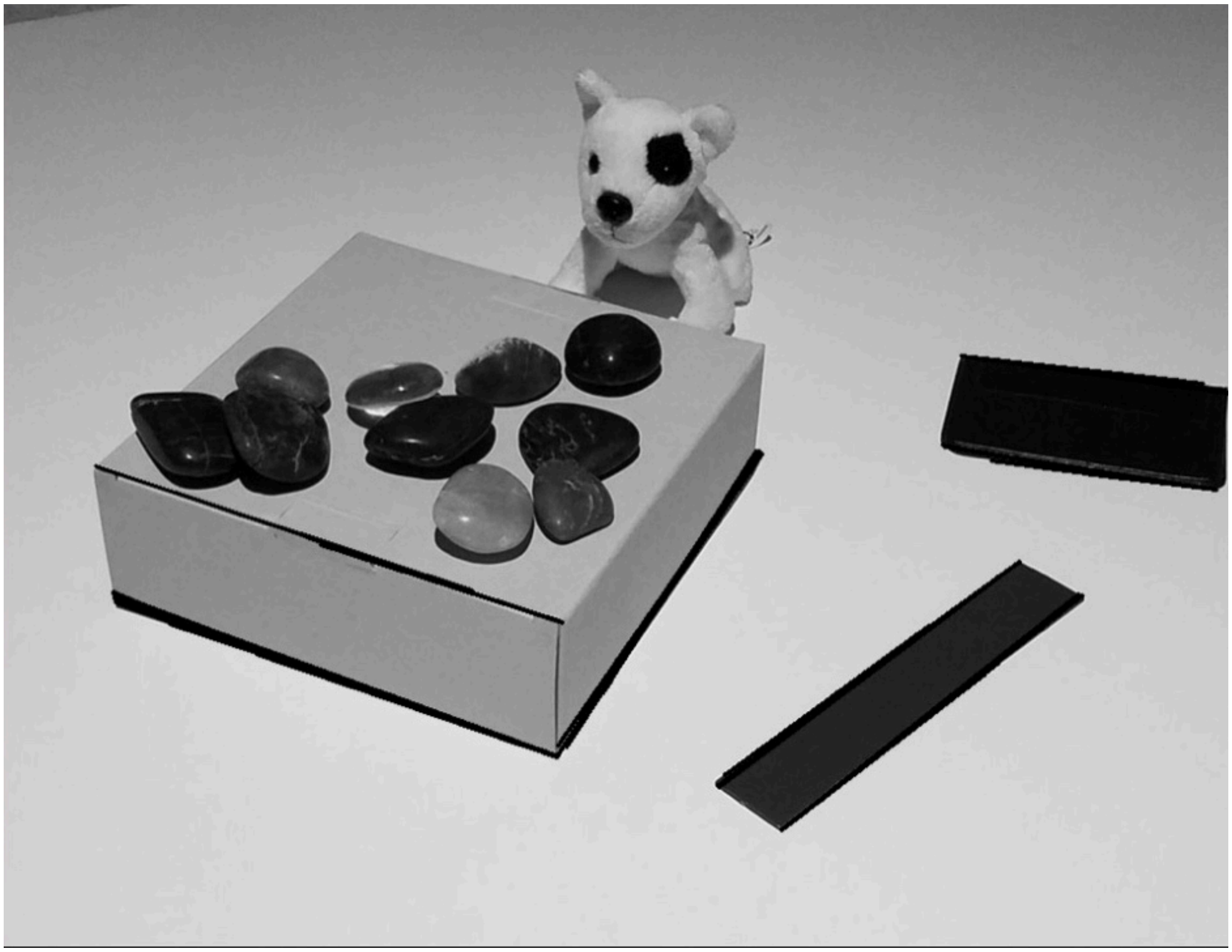
2) Gray Image



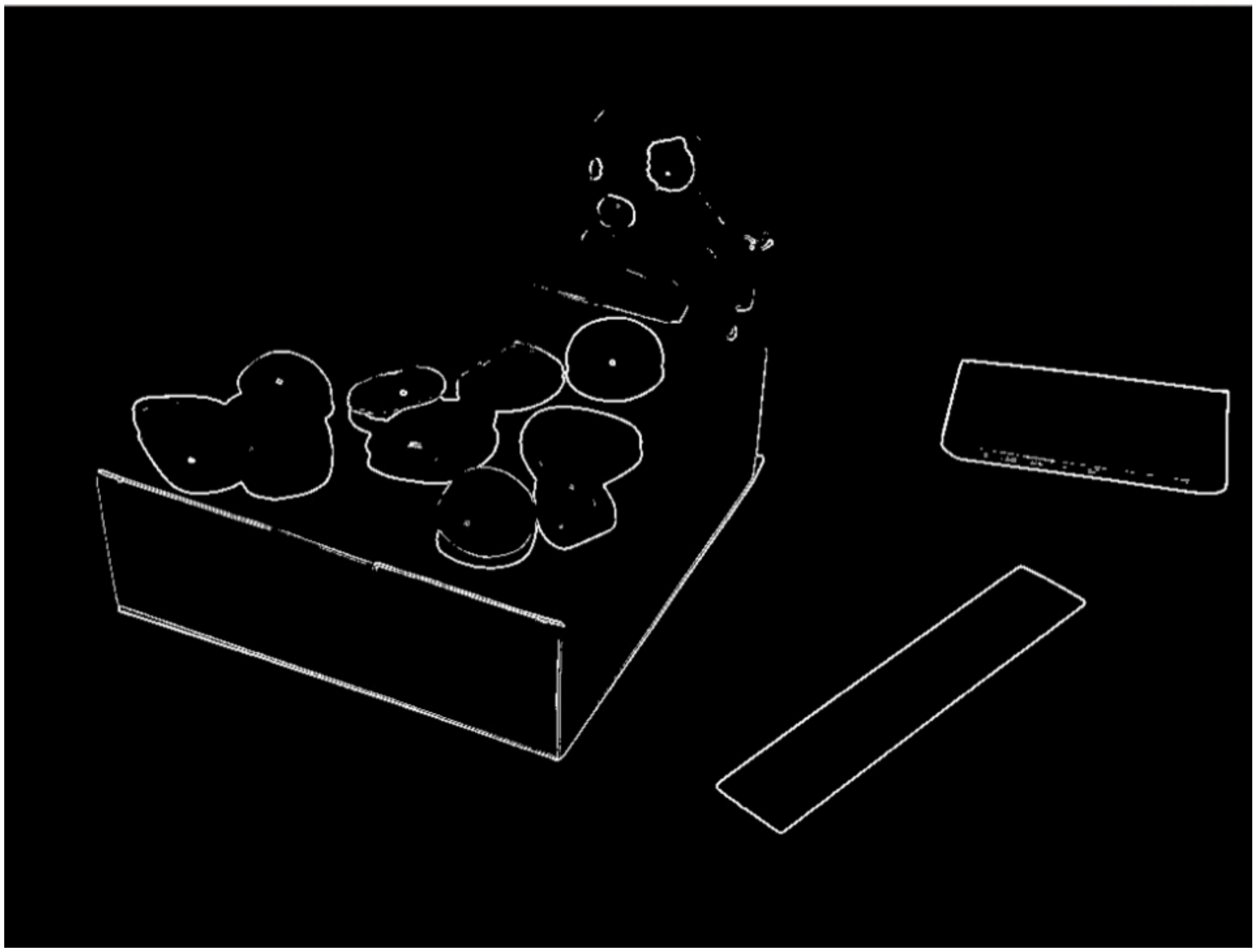
4) Binary Image



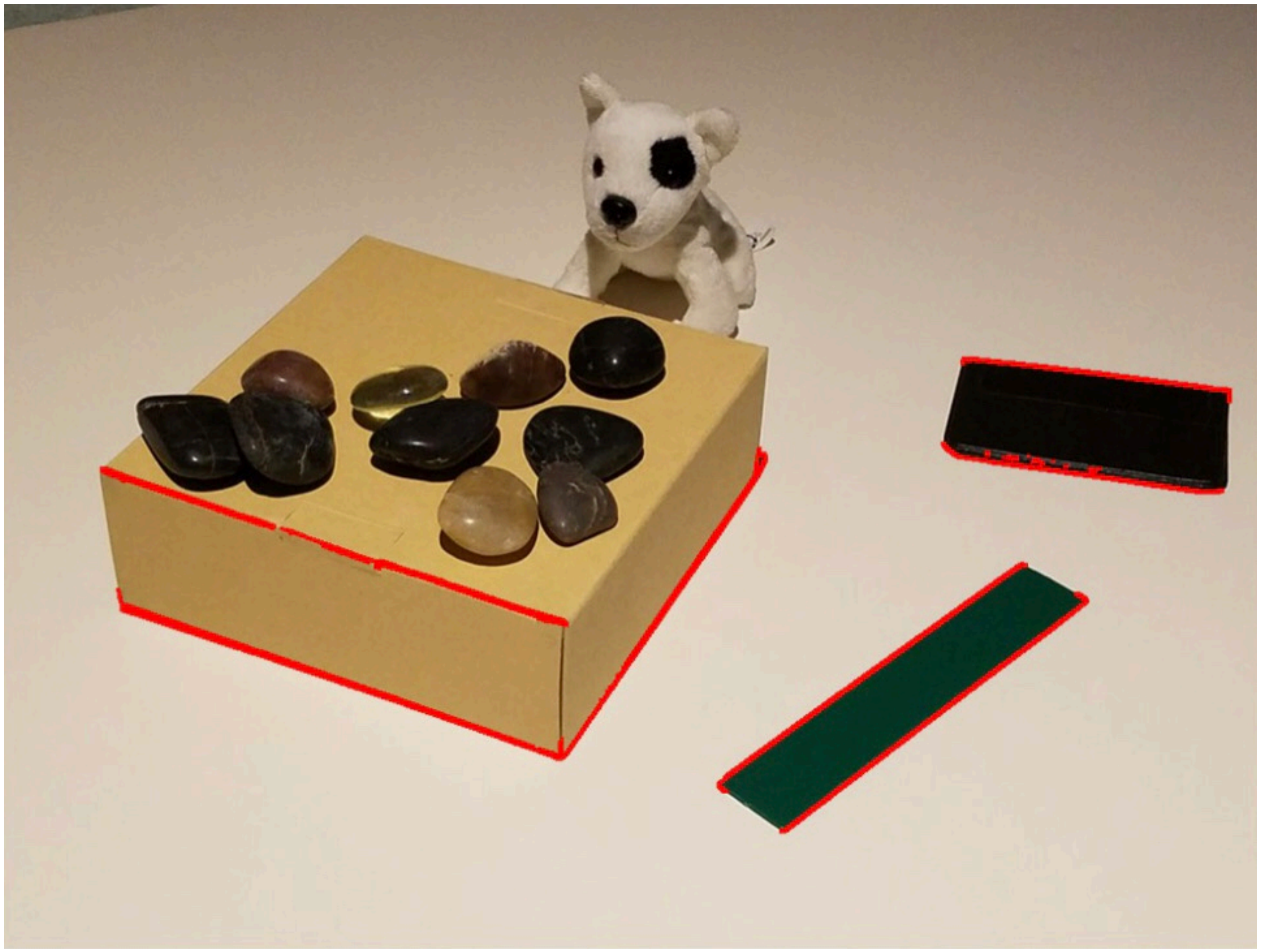
3) Gray Image with detected edges



4) Binary Image



5) detected Lines superimposed on original image

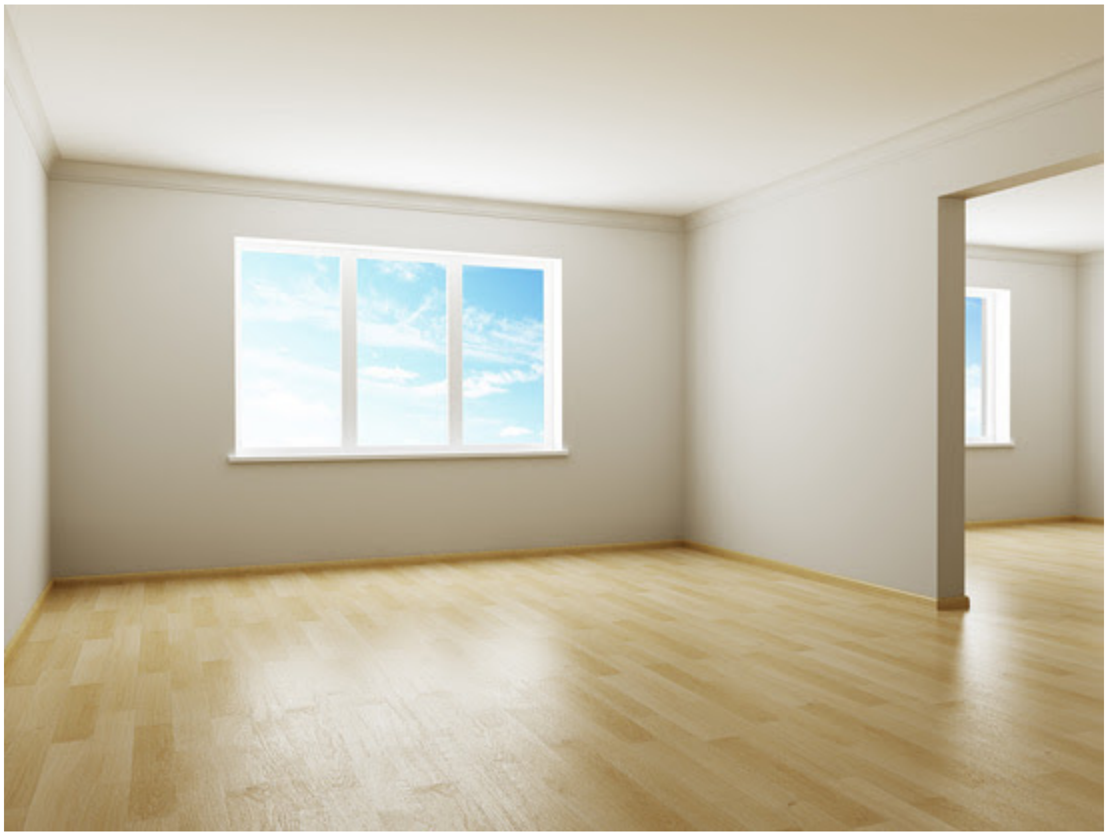


6) Detected coordinates of lines:

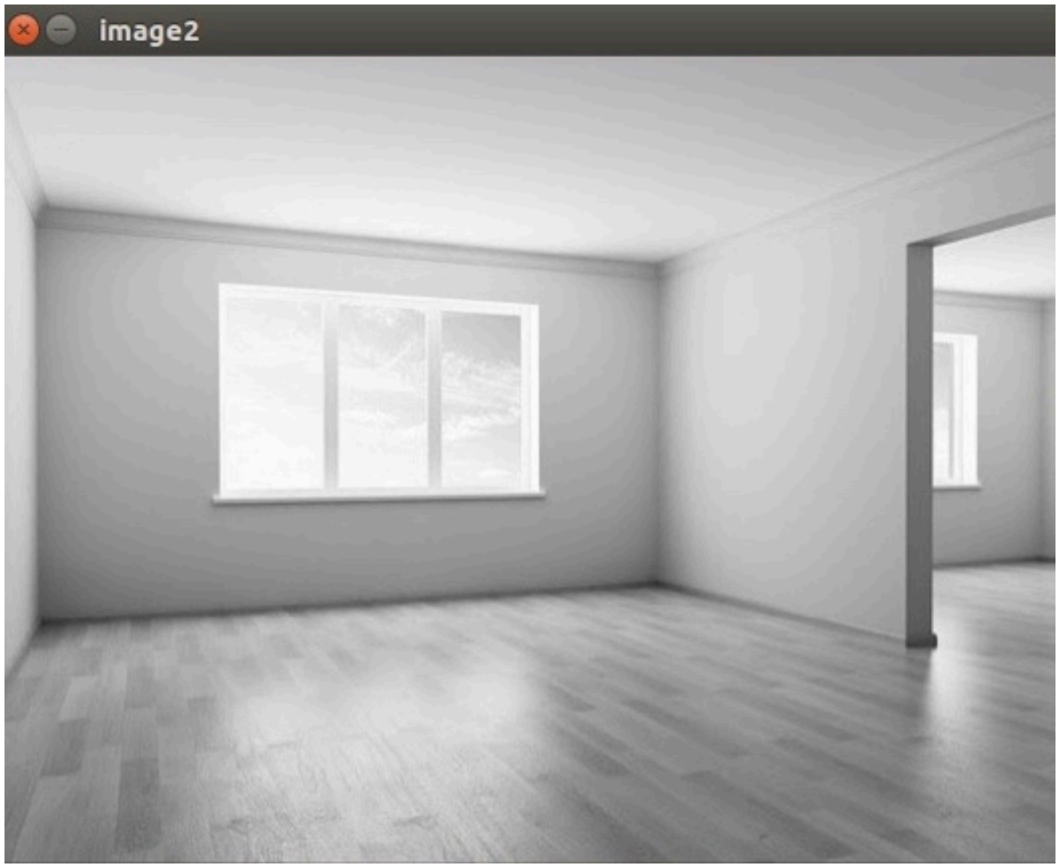
- There is no parallel lines detected. Hence no parallelogram detected in image.

C) TestImage3:

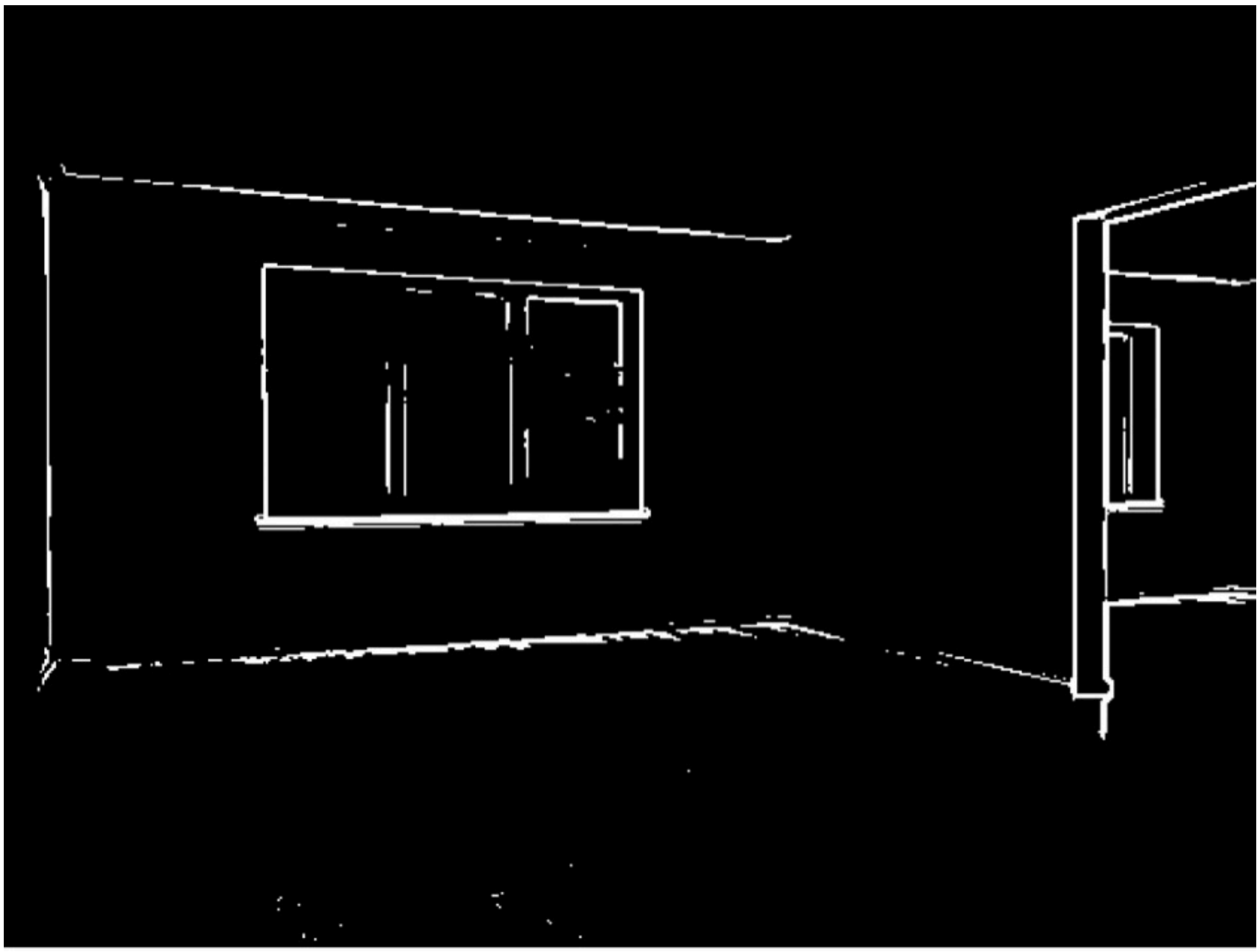
1) Original Image:



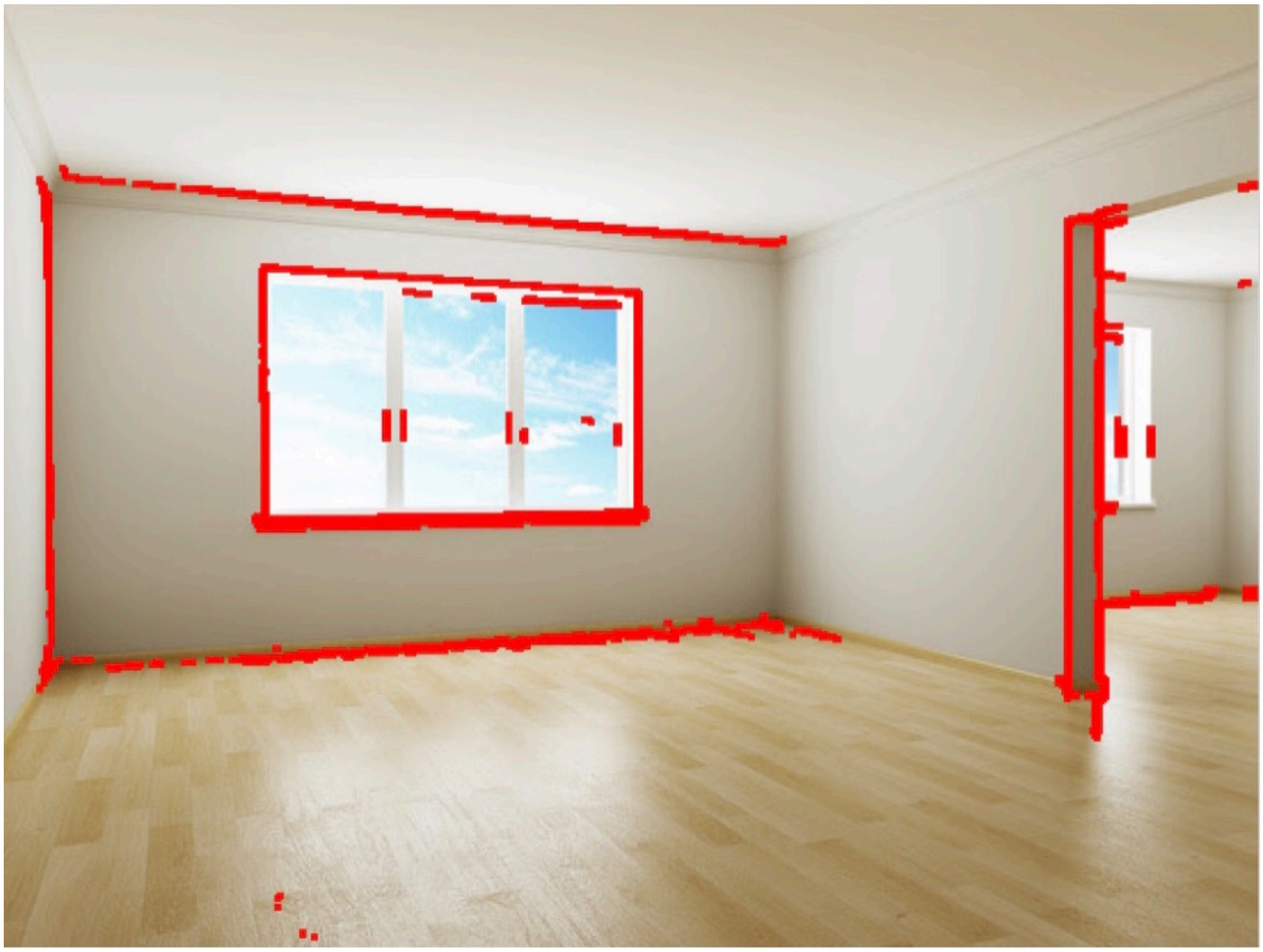
2) Gray Image:



3) Binary Image:



4) Detected parallelogram superimposed on original image:



5)Coordinates of detected parallelograms:

```
LINESTRING (27 72, 208 92)
LINESTRING (546 79, 548 260)
LINESTRING (483 90, 479 321)
LINESTRING (273 99, 281 277)
LINESTRING (472 93, 466 304)
LINESTRING (468 94, 466 304)
LINESTRING (20 113, 279 129)
LINESTRING (112 224, 21 231)
LINESTRING (80 81, 314 102)
LINESTRING (115 223, 278 227)
LINESTRING (112 224, 227 228)
LINESTRING (111 225, 182 229)
LINESTRING (132 407, 137 408)
LINESTRING (21 180, 503 190)
LINESTRING (0 0, 549 0)
LINESTRING (287 277, 24 289)
LINESTRING (287 277, 120 286)
LINESTRING (527 256, 481 264)
LINESTRING (265 277, 21 285)
LINESTRING (265 222, 21 234)
LINESTRING (278 218, 153 228)
LINESTRING (115 222, 21 229)
LINESTRING (278 221, 21 232)
LINESTRING (269 275, 60 290)
LINESTRING (280 274, 51 291)
LINESTRING (294 274, 60 290)
LINESTRING (333 268, 201 282)
LINESTRING (483 90, 479 321)
LINESTRING (17 77, 17 300)
LINESTRING (110 83, 116 288)
LINESTRING (483 90, 479 321)
LINESTRING (483 90, 479 321)
LINESTRING (115 223, 21 231)
LINESTRING (468 94, 466 304)
LINESTRING (108 83, 121 395)
LINESTRING (468 94, 466 304)
LINESTRING (278 220, 132 230)
LINESTRING (88 82, 339 105)
LINESTRING (111 225, 21 231)
LINESTRING (156 89, 339 105)
LINESTRING (293 278, 21 283)
LINESTRING (17 77, 19 297)
LINESTRING (365 278, 21 284)
LINESTRING (177 91, 339 105)
LINESTRING (466 222, 21 227)
LINESTRING (111 226, 21 232)
LINESTRING (265 222, 21 231)
LINESTRING (466 223, 21 228)
LINESTRING (156 89, 308 101)
LINESTRING (466 218, 282 224)
LINESTRING (115 221, 227 228)
LINESTRING (115 223, 182 229)
LINESTRING (306 270, 60 290)
LINESTRING (322 272, 111 286)
LINESTRING (466 225, 21 229)
LINESTRING (278 219, 132 230)
LINESTRING (278 220, 21 229)
LINESTRING (265 222, 21 231)
```

------------------------------SOURCE CODE------------------------------

```
# Import required modules
import numpy as np
import cv2
import random
import pdb
import math
import argparse
from shapely.geometry import LineString

class DetectParallelogram(object):
    def __init__(self):
        '''
        Command Line arguments:

        --testImage <absoulte/relative path> => Ex "TestImage1c.png"
            -> Can be used to specify input image.
        --sobelThreshold <threshold> => Ex "100"
            -> Can be used to specify threshold for sobel edge detection.
        '''
        # Parse command line Arguments
        parser = argparse.ArgumentParser()
        parser.add_argument("--testImage", dest="testImage", required=True,
help="Enter path of image to be processed")
        parser.add_argument("--sobelThreshold", dest="sobelThreshold",
default="100", help="Enter path of image to be processed")
        parser.add_argument("--scaleWindows", dest="scaleWindows",
default="1.0",
                            help="Enter scale for enlarging and contractign
output images")
        parser.add_argument("--houghProb", dest="houghProb", default="0.7",
                            help="Probability of being a line after hough
accumulator voting")
        parser.add_argument("--dupeLineThresh", dest="dupeLineThresh",
default="20",
                            help="Pixels width which will be considered as
single line for hough line mapping")
        self.testArgs = parser.parse_args()

        # Initializing all images that will be used
        self.inpImg = cv2.imread(self.testArgs.testImage)
        self.refRows = self.inpImg.shape[0]
        self.refCols = self.inpImg.shape[1]
        self.grayImg = np.zeros((self.refRows, self.refCols), dtype=np.uint8)
        self.sobelImg = np.zeros((self.refRows, self.refCols), dtype=np.uint8)
        self.edgeImg = np.zeros((self.refRows, self.refCols), dtype=np.uint8)
        self.houghLines = np.zeros((self.refRows, self.refCols),
dtype=np.uint8)

    def convertColorToGray(self):
        '''
        This function converts input colored image to gray scale image.
        '''
        print "Color image to gray scale conversion started"
        # Seperating R, G, B matrices from color image
        redMat = self.inpImg[:, :, 2]
        blueMat = self.inpImg[:, :, 0]
        greenMat = self.inpImg[:, :, 1]

        # Gray = 0.30R + 0.59G + 0.11B
        print "Calculating luminousity using formula: L = 0.3R + 0.59G +
0.11B"
        for row in range(0, self.refRows):
            for col in range(0, self.refCols):
                lum = int(0.30 * redMat[row][col] + 0.59 * greenMat[row][col]
+ 0.11 * blueMat[row][col])
                self.grayImg[row][col] = lum
        print "Conversion from color to gray image completed."

    def detectEdges(self):
        '''
        This function detects edges by calculating sobel and thresholding it
        '''
        print "Detecting edges starting now with threshold:
{0}".format(self.testArgs.sobelThreshold)
        # Zero padded image creation
        zeroPaddedGray = np.pad(self.grayImg, (1, 1), 'constant',
constant_values=0)

        for row in range(0, self.refRows):
            for col in range(0, self.refCols):
                sx = zeroPaddedGray[row, col] * (-1) + zeroPaddedGray[row + 1,
col] * (-2) + zeroPaddedGray[

row + 2, col] * (-1) + \
                     zeroPaddedGray[
                         row, col + 2] * (1) + zeroPaddedGray[row + 1, col +
2] * (2) + zeroPaddedGray[

row + 2, col + 2] * (1)
                sy = zeroPaddedGray[row + 2, col] * (-1) + zeroPaddedGray[row
+ 2, col + 1] * (-2) + zeroPaddedGray[

row + 2, col + 2] * (

-1) + \
                     zeroPaddedGray[
                         row, col] * (1) + zeroPaddedGray[row, col + 1] * (2)
+ zeroPaddedGray[row, col + 2] * (1)
                edgeVal = np.sqrt(sx ** 2 + sy ** 2)
                self.sobelImg[row, col] = edgeVal
                self.edgeImg[row, col] = 0 if edgeVal <
int(self.testArgs.sobelThreshold) else 255
        print "Detecting edges completed"

    def _calSinCosTheta(self, thetaList):
        '''
        This function can be used to cached values for sin(theta) and
cos(theta)
```

```
        Can not be called outside of this class
        :param thetaList: List of Theta in radians
        :return:
            sinThetaList, colThetaList
        '''
        sinThetaList = []
        cosThetaList = []
        for theta in thetaList:
            sinThetaList.append(math.sin(theta))
            cosThetaList.append(math.cos(theta))
        return sinThetaList, cosThetaList

    def calImgHoughTransform(self, drawOnlySpecial=False):
        '''
        This function calculates hough transform of image
        '''
        print "Starting hough transform"
        self.rho = np.sqrt(self.refRows ** 2 + self.refCols ** 2)
        thetas = np.radians(np.arange(-90, 90))
        print "max distance hough line: {0}".format(self.rho)
        self.houghSpaceDisp = np.zeros((len(thetas), int(2 *
np.ceil(self.rho))), dtype=np.uint8)
        self.houghSpace = np.zeros((len(thetas), int(2 * np.ceil(self.rho))),
dtype=np.uint8)
        print "Hough Space dimensions (rows, cols):
{0}".format(self.houghSpaceDisp.shape)

        # why to calculate again n again
        sinThetaDict, cosThetaDict = self._calSinCosTheta(thetaList=thetas)

        for row in range(0, self.refRows):
            for col in range(0, self.refCols):
                if self.edgeImg[row, col] == 255:  # Try hough transform on
only white pixels after edge detection
                    index = 0
                    for theta in thetas:
                        try:
                            rad = round((col * cosThetaDict[index] + row *
sinThetaDict[index]))
                            if self.houghSpaceDisp[int(np.degrees(theta)) +
90, int(rad + int((np.ceil(self.rho))))] < 255:
                                self.houghSpaceDisp[int(np.degrees(theta)) +
90, int(rad + int((np.ceil(self.rho))))] += 1
                            index += 1
                        except Exception as e:
                            print "Exception occured while execution:
{0}".format(e)
                            print "Important info to debug"
                            print "Row: {0}\nCol: {1}".format(row, col)
                            raise Exception(e)

        # Move this image to other image to see hough transform at this point
        for row in range(self.houghSpaceDisp.shape[0]):
            for col in range(self.houghSpaceDisp.shape[1]):
                self.houghSpace[row, col] = self.houghSpaceDisp[row][col]
        maxVote = self.houghSpace.max()
        houghProb = self.houghSpace / float(maxVote)

        self.lineInfo = {}
        lineNo = 1
        for row in range(houghProb.shape[0]):
            for col in range(houghProb.shape[1]):
                if houghProb[row, col] > float(self.testArgs.houghProb):
                    self.houghSpace[row, col] = 255
                    self.lineInfo["".join(("line", str(lineNo)))] = {}
                    self.lineInfo["".join(("line", str(lineNo)))]["rho"] = col
- int((np.ceil(self.rho)))
                    self.lineInfo["".join(("line", str(lineNo)))]["theta"] =
row - 90
                    lineNo += 1
                else:
                    self.houghSpace[row, col] = 0
        drawLineInfo = self._houghToImgSpaceConv()
        # pdb.set_trace()
        if drawOnlySpecial:
            self._drawOnlySpecialLines(drawLineDict=drawLineInfo)
            return
        self._drawHoughLinesOnGray(drawLineDict=drawLineInfo)

    def _houghToImgSpaceConv(self):
        for line in self.lineInfo:
            if self.lineInfo[line]["theta"] == 0 or
self.lineInfo[line]["theta"] == 180:
                self.lineInfo[line]["special"] = "ver"
            elif self.lineInfo[line]["theta"] == 90 or
self.lineInfo[line]["theta"] == -90:
                self.lineInfo[line]["special"] = "hor"
            else:
                self.lineInfo[line]["slope"] = -1 /
(math.tan(np.radians(self.lineInfo[line]["theta"])))
                self.lineInfo[line]["const"] = self.lineInfo[line]["rho"] /
math.sin(
                    np.radians(self.lineInfo[line]["theta"]))

        # Calculating max end points of lines in image
        for line in self.lineInfo:
            fx = 0
            fy = 0
            sx = 0
            sy = 0
            preX = 0
            preY = 0
            for x in range(self.refCols):
                if "special" in self.lineInfo[line].keys():
                    if self.lineInfo[line]["special"] == "hor":
                        fx = 0
                        fy = self.lineInfo[line]["rho"] + self.refRows - 1
                        sx = self.refCols - 1
                        sy = self.lineInfo[line]["rho"] + self.refRows - 1
                    else:
                        fx = self.lineInfo[line]["rho"]
                        fy = 0
                        sx = self.lineInfo[line]["rho"]
                        sy = self.refRows - 1
                    break
                else:
                    y = x * self.lineInfo[line]["slope"] +
self.lineInfo[line]["const"]
                    if fx == 0 and round(y) in range(0, self.refRows):
                        fx = x
                        fy = y
                        preX = x
                        preY = y
                        continue
                    if fx != 0:
                        if round(y) in range(0, self.refRows):
                            preY = y
                            preX = x
                        else:
                            sx = preX
                            sy = preY
                if sx == 0 and sy == 0:
                    y = (self.refCols - 1 - x) * self.lineInfo[line]["slope"]
+ self.lineInfo[line]["const"]
                    if round(y) in range(0, self.refRows):
                        sx = (self.refCols - 1 - x)
                        sy = y


            self.lineInfo[line]["fx"] = int(np.floor(fx))
            self.lineInfo[line]["fy"] = int(np.floor(fy))
            self.lineInfo[line]["sx"] = int(np.floor(sx))
            self.lineInfo[line]["sy"] = int(np.floor(sy))

        uniqueLineInfo = {}
        for line in self.lineInfo:
            if not uniqueLineInfo:
                uniqueLineInfo[line] = {}
                uniqueLineInfo[line] = self.lineInfo[line]
            else:
                remInfo = False
                for prevLine in uniqueLineInfo:
                    if abs(uniqueLineInfo[prevLine]["fx"] -
self.lineInfo[line]["fx"]) < int(
                            self.testArgs.dupeLineThresh) and abs(
                                uniqueLineInfo[prevLine]["fy"] -
self.lineInfo[line]["fy"]) < int(
                            self.testArgs.dupeLineThresh) and abs(
                        uniqueLineInfo[prevLine]["sx"] -
self.lineInfo[line]["sx"]) < int(
                            self.testArgs.dupeLineThresh) and abs(
                        uniqueLineInfo[prevLine]["sy"] -
self.lineInfo[line]["sy"]) < int(
                            self.testArgs.dupeLineThresh):
```

```
                        print "Dupe Line detected....REMOVING!!!!"
                        remInfo = True
                        break
                if not remInfo:
                    uniqueLineInfo[line] = {}
                    uniqueLineInfo[line] = self.lineInfo[line]
        uniqueLineInfo = self._removeIntelligentDupe(lineDict=self.lineInfo)
        return uniqueLineInfo

    def _removeIntelligentDupe(self, lineDict):
        uniqueLineInfo = self.getSlotInfo(lineDict)
        self.warBlack = np.zeros((self.refRows, self.refCols), dtype=np.uint8)
        # interDict = {}
        for line in uniqueLineInfo:
            if "special" in uniqueLineInfo[line].keys():
                cv2.line(self.warBlack, (uniqueLineInfo[line]["fx"],
uniqueLineInfo[line]["fy"]),
                         (uniqueLineInfo[line]["sx"],
uniqueLineInfo[line]["sy"]), (255, 255, 255), 7)

                continue

            # Detecting intersection of lines for detecting coordinates of
parallelogram

            L1 = LineString([(uniqueLineInfo[line]["fx"],
uniqueLineInfo[line]["fy"]),
                             (uniqueLineInfo[line]["sx"],
uniqueLineInfo[line]["sy"])])
            cv2.line(self.warBlack, (uniqueLineInfo[line]["ofx"],
uniqueLineInfo[line]["ofy"]),
                     (uniqueLineInfo[line]["osx"],
uniqueLineInfo[line]["osy"]), (255, 255, 255), 7)
            L2 =
LineString([(uniqueLineInfo[line]["ofx"],uniqueLineInfo[line]["ofy"]),(uniqueL
ineInfo[line]["osx"],uniqueLineInfo[line]["osy"])])

            # print(L2.intersection(L1))


        for row in range(1, self.refRows-1):
            for col in range(1, self.refCols-1):
                if np.bitwise_and(self.warBlack[row,col],
self.edgeImg[row,col]) == 255:
                    self.inpImg[row,col,0] = 0
                    self.inpImg[row, col, 1] = 0
                    self.inpImg[row, col, 2] = 255
                    self.inpImg[row-1, col-1, 0] = 0
                    self.inpImg[row-1, col-1, 1] = 0
                    self.inpImg[row-1, col-1, 2] = 255
                    self.inpImg[row-1, col, 0] = 0
                    self.inpImg[row-1, col, 1] = 0
                    self.inpImg[row-1, col, 2] = 255
                    self.inpImg[row-1, col+1, 0] = 0
                    self.inpImg[row-1, col+1, 1] = 0
```

```
                        self.inpImg[row-1, col+1, 2] = 255
                        self.inpImg[row, col-1, 0] = 0
                        self.inpImg[row, col-1, 1] = 0
                        self.inpImg[row, col-1, 2] = 255
                        self.inpImg[row, col+1, 0] = 0
                        self.inpImg[row, col+1, 1] = 0
                        self.inpImg[row, col+1, 2] = 255
                        self.inpImg[row+1, col-1, 0] = 0
                        self.inpImg[row+1, col-1, 1] = 0
                        self.inpImg[row+1, col-1, 2] = 255
                        self.inpImg[row+1, col, 0] = 0
                        self.inpImg[row+1, col, 1] = 0
                        self.inpImg[row+1, col, 2] = 255
                        self.inpImg[row+1, col+1, 0] = 0
                        self.inpImg[row+1, col+1, 1] = 0
                        self.inpImg[row+1, col+1, 2] = 255
        return uniqueLineInfo



    def _drawHoughLinesOnGray(self, drawLineDict):
        for line in drawLineDict:
            cv2.line(self.grayImg, (drawLineDict[line]["fx"], 
drawLineDict[line]["fy"]),
                     (drawLineDict[line]["sx"], drawLineDict[line]["sy"]), (0, 
0, 0), 2)

    def _drawOnlySpecialLines(self, drawLineDict):
        for line in drawLineDict:
            if drawLineDict[line].has_key("special"):
                cv2.line(self.grayImg, (drawLineDict[line]["fx"], 
drawLineDict[line]["fy"]),
                         (drawLineDict[line]["sx"], drawLineDict[line]["sy"]), 
(0,0,0), 2)

    def getSlotInfo(self, lineDict={}):
        slotDict = {}
        for line in lineDict:
            slotDict[line] = {}
            tempBlack = np.zeros((self.refRows, self.refCols), dtype=np.uint8)
            cv2.line(tempBlack, (lineDict[line]["fx"], lineDict[line]["fy"]),
                     (lineDict[line]["sx"], lineDict[line]["sy"]), (255, 255, 
255), 10)
            index = 1
            discountinuousPixels = 70
            fx = -1
            for row in range(1, self.refRows-1):
                for col in range(1, self.refCols-1):
                    try:

                        if np.bitwise_and(int(tempBlack[row, col]), 
int(self.edgeImg[row, col])) == 255:
                            if "".join(("slot", str(index))) not in 
slotDict[line]:
                                slotDict[line]["".join(("slot", str(index)))]
```

```
= {}
                                slotDict[line]["".join(("slot",
str(index)))]["counter"] = 0
                            if fx == -1:
                                fx = col
                                slotDict[line]["".join(("slot",
str(index)))]["fx"] = col
                                slotDict[line]["".join(("slot",
str(index)))]["fy"] = row
                                slotDict[line]["".join(("slot",
str(index)))]["sx"] = col
                                slotDict[line]["".join(("slot",
str(index)))]["sy"] = row
                                slotDict[line]["".join(("slot",
str(index)))]["counter"] += 1
                            else:
                                slotDict[line]["".join(("slot",
str(index)))]["sx"] = col
                                slotDict[line]["".join(("slot",
str(index)))]["sy"] = row
                                slotDict[line]["".join(("slot",
str(index)))]["counter"] += 1
                        else:
                            if fx == -1:
                                pass
                            else:
                                if 'special' in lineDict[line]:
                                    continue
                                y = col * self.lineInfo[line]["slope"] +
self.lineInfo[line]["const"]
                                if round(y) == row:
                                    discountinuousPixels -= 1
                                    if discountinuousPixels == 0:
                                        discountinuousPixels = 70
                                        index += 1
                                        fx = -1
                                    print "*************{0}".format(index)
                    except Exception as e:
                        pdb.set_trace()
                        print e
        for line in slotDict:
            if slotDict[line]:
                tempArray = []
                for slot in slotDict[line]:
                    tempArray.append(slotDict[line][slot]["counter"])
                try:
                    maxVal = max(tempArray)
                except Exception as e:
                    pdb.set_trace()
                    print e
                for slot in slotDict[line]:
                    if slotDict[line][slot]["counter"] == maxVal:
                        maxSlot = slot
                        break
                slotDict[line]["ofx"] = slotDict[line][maxSlot]["fx"]
                slotDict[line]["ofy"] = slotDict[line][maxSlot]["fy"]
                slotDict[line]["osx"] = slotDict[line][maxSlot]["sx"]
                slotDict[line]["osy"] = slotDict[line][maxSlot]["sy"]
            else:
                try:
                    slotDict[line]["ofx"] = lineDict[line]["fx"]
                    slotDict[line]["ofy"] = lineDict[line]["fy"]
                    slotDict[line]["osx"] = lineDict[line]["sx"]
                    slotDict[line]["osy"] = lineDict[line]["sy"]
                except Exception as e:
                    pdb.set_trace()
                    print e
            slotDict[line]["fx"] = slotDict[line]["ofx"]
            slotDict[line]["sx"] = slotDict[line]["osx"]
            slotDict[line]["fy"] = slotDict[line]["ofy"]
            slotDict[line]["sy"] = slotDict[line]["osy"]


        return slotDict

    def displayAllImages(self):
        '''
        Displays all images
        '''
        print "Listing all images to be displayed"
        print "1. Input Image"
        print "2. Gray Scale Image"
        print "3. Sobel Image"
        print "4. Edge Detected Image"
        print "5. Edges in Hough Space"
        print "6. Grayscale image with hough lines"
        scaledRow = int(self.refRows * float(self.testArgs.scaleWindows))
        scaledCol = int(self.refCols * float(self.testArgs.scaleWindows))
        cv2.namedWindow('Input_image', cv2.WINDOW_NORMAL)
        cv2.resizeWindow('Input_image', scaledCol,scaledRow)
        cv2.namedWindow('Gray_image', cv2.WINDOW_NORMAL)
        cv2.resizeWindow('Gray_image', scaledCol, scaledRow)
        cv2.namedWindow('Sobel_image', cv2.WINDOW_NORMAL)
        cv2.resizeWindow('Sobel_image', scaledCol, scaledRow)
        cv2.namedWindow('Edge_image', cv2.WINDOW_NORMAL)
        cv2.resizeWindow('Edge_image', scaledCol, scaledRow)
        cv2.namedWindow('Hough_space', cv2.WINDOW_NORMAL)
        cv2.resizeWindow('Hough_space', 900, 300)
        # cv2.namedWindow('Gray_hough_image', cv2.WINDOW_NORMAL)
        cv2.resizeWindow('Gray_hough_image', scaledCol, scaledRow)
        cv2.imshow("Input_image", self.inpImg)
        cv2.imshow("Gray_image", self.grayImg)
        cv2.imshow("Sobel_image", self.sobelImg)
        cv2.imshow("Edge_image", self.edgeImg)
        cv2.imshow("Hough_space", self.houghSpaceDisp)
        # cv2.imshow("Gray_hough_image", self.houghLines)
        cv2.imshow("warBlack", self.warBlack)
        cv2.waitKey(0)
```

```
##############################################################################
######################
###################################### EXECUTION FLOW BEGINS
HERE##################################
##############################################################################
######################

step = DetectParallelogram()
step.convertColorToGray()
step.detectEdges()
step.calImgHoughTransform()
step.displayAllImages()
```