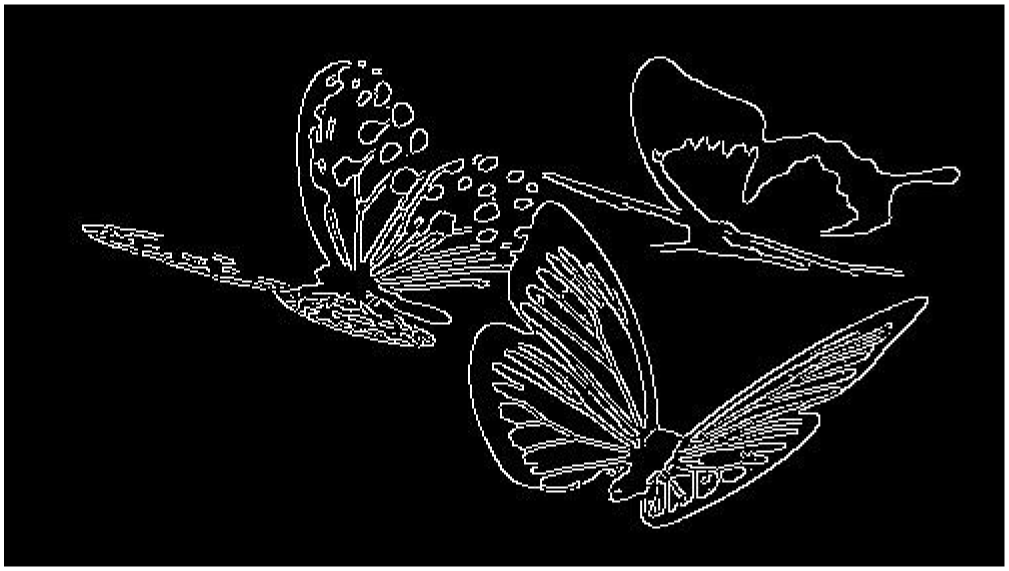
Part 1: Edge Detection

We ran the inbuilt cv2 edge detection method on the butterfly image to get the below result



The parameters we chose finally were canny(100, 150) after some experimentation.

Steps Taken to Improve

The purpose of gaussian blur before edge detection would have been to smooth out any noise that would show up as edges.

Since the image did not have much noise or any artifacts that needed to be smoothed out we only added a small noise variance 0.1

The results we got by fine tuning canny threshold values were just as food as the given file `edges.jpg`

Part 2: Contour Detection

As a first step edge detection was ran on the input image as recommended by the problem statement.

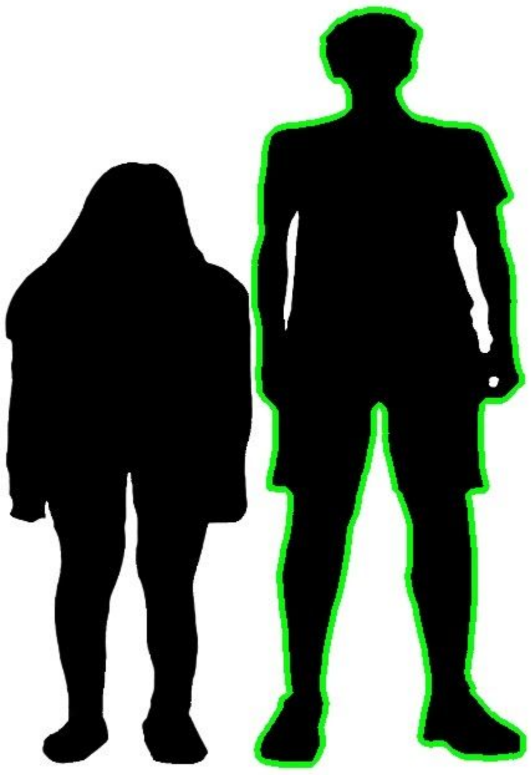
Then the detectContours() method followed by the drawContours() were called with the default parameters suggested in the tutorial.

Finally to detect the contour corresponding to the taller man, the command

c = max(contours, key = cv2.contourArea)

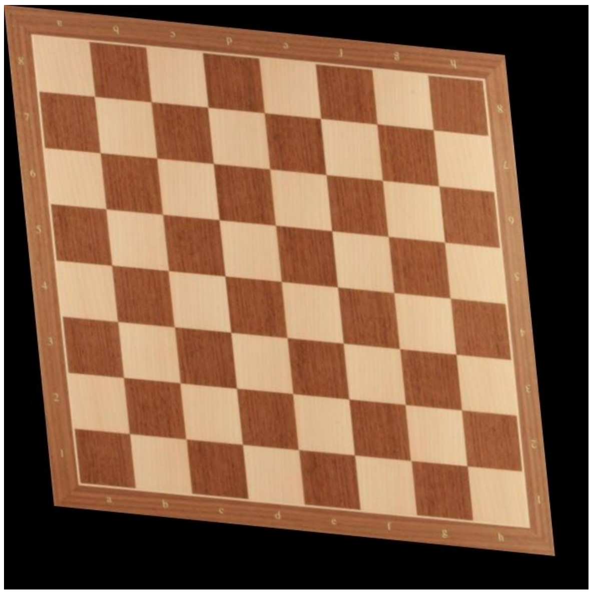
Was used. This command selects the contour bounding the largest area amongst all the contours present in the list `contours`

Result is as shown below:



It was already specified in the question that the transformation that has been applied to obtain the distorted images was a planar axial shear.

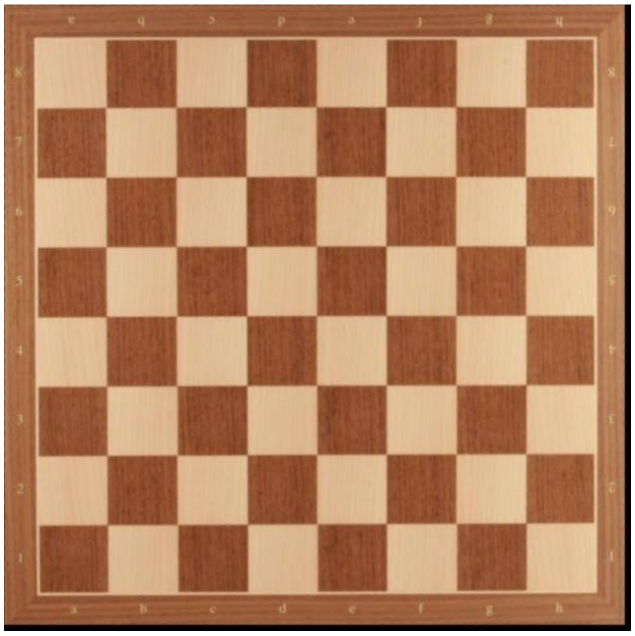
Distorted Image



Just by looking at the above distorted image, it was clear that there is both a shear-x and a shear-y transform involved.
The displacement from the horizontal axis of the upper right corner and the displacement from the vertical axis of the lower left corner were measured and found to be 70px
Since the size of the image (700px, 700px) the shear (or rather anti-shear) matrix M was obtained as below

M = [[1, -70/700, 0], [-70/700, 1, 0]]

Then for the first part, using this M the warpAffine() function was used to recover the below result



The image looks quite close to the original but we notice that some thin black borders have been left over as a result of there being large amounts of black space in the input distorted image.

For the 2nd part

For the second part since the Shear transform matrix is 2x3 it has 6 free parameters that need to be calculated, for this purpose we need to mark 3 points which would give 6 equations,
We chose the below 3 pairs of corresponding points and gave it to getAffineTransform() to obtain the matrix M

```
pts1 = np.float32([[600,58],[662,662],[62,603]])
pts2 = np.float32([[599, 0],[599, 599],[0, 599]])
```

The resulting image is shown below

There are no stark differences between the 2 methods.
This is also reflected in the fact that they have almost identical transformation matrices

```
M_manual = [[ 1.  -0.1  0. ]
            [-0.1  1.   0. ]]

M_automatic = [[ 1.00851308 -0.10352287 -0.10352287]
              [-0.0985137   1.00183419  1.00183419]]
```

Perhaps the automatic method does a slightly worse job since it also captures a small translation between the images. (But there was no translation involved while distorting the original image)

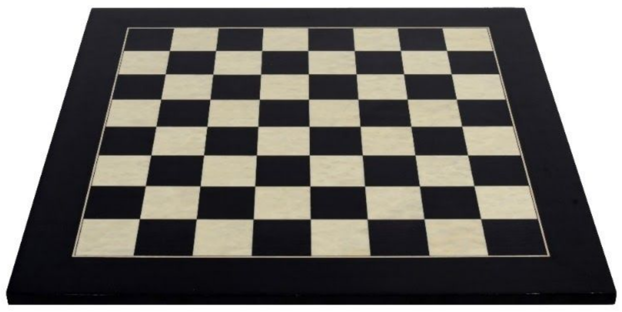
In the 2nd problem we had to generate the orthographic view of a given perspective image of a chess-board (similar to what cam-scanner does)

Since a perspective transformation matrix has 8 free parameters, we needed 4 pairs of corresponding points between the perspective view and the desired orthographic view

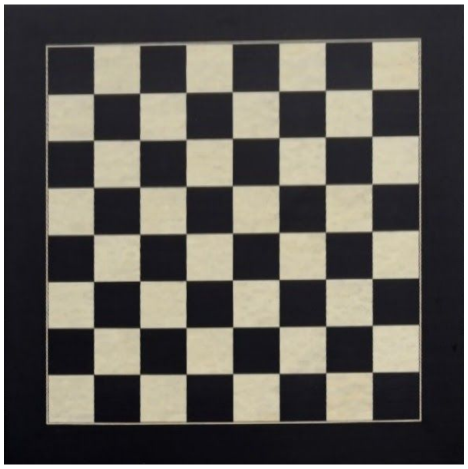
We chose the following points

```
pts1 = np.float32([[134,220],[32,558],[655,224], [764,561]])
pts2 = np.float32([[0, 0],[0, 599],[599, 0], [599,599]])
```

Input Image



Created Orthographic view



We needed a total 4 pairs of points correspondencies

3D Geometry

The role of 3D geometry in the problem comes into using that the 4 points that have been selected lie on a single plane in 3 dimensions.

The 3D representation of the plane is used to solve the problem.

The 3X3 transformation matrix (with 8 free parameters (3, 3) entry is fixed to be 1.0) was obtained as below

```
M =     [[ 1.54299924e+00  4.65638824e-01 -3.09202440e+02]
         [-2.58277014e-02  3.36405810e+00 -7.36631871e+02]
         [-2.67371228e-05  1.61918179e-03  1.00000000e+00]]
```