OMR_GRADER

Github link :-
https://github.com/ChetanJangid93/OMR_Grader

Introduction

It is a kind of system which can read images of OMR's and proceed on it and finally gives grades that on basis of some answer key.

As of now we are normally giving majority of MCQ based exams on OMR's , it is very necessary to build such a system so that efforts to check and grade OMR's are reduced.

It takes much time for a person to evaluate a sheet whereas a computer can do it in seconds and that's why there is need to develop such a system.

Bubble sheet scanner and test grader using OMR, Python and OpenCV

I will demonstrate how to implement a bubble sheet test scanner and grader
using *strictly* computer vision and image

processing techniques, along with the OpenCV library.

After completion , I'll provide sample results of our test grader on a few example exams.

Implementation

The 7 steps to build a bubble sheet scanner and grader

- **Step #1:** Detect the exam in an image.
- **Step #2:** Apply a perspective transform to extract the top-down, birds-eye-view of the exam.
- **Step #3:** Extract the set of bubbles (i.e., the possible answer choices) from the perspective transformed exam.
- **Step #4:** Sort the questions/bubbles into rows.
- **Step #5:** Determine the marked (i.e., "bubbled in") answer for each row.
- **Step #6:** Lookup the correct answer in our answer key to determine if the user was correct in their choice.
- **Step #7:** Repeat for all questions in the exam.

➢ import required Python packages.

"from imutils.perspective import four_point_transform
from imutils import contours

```
import numpy as np
import argparse
import imutils
import cv2”
```

- Define Answer Key.

```
ANSWER_KEY= {0: 1,1: 4,2: 0,3: 3,4: 1}
```

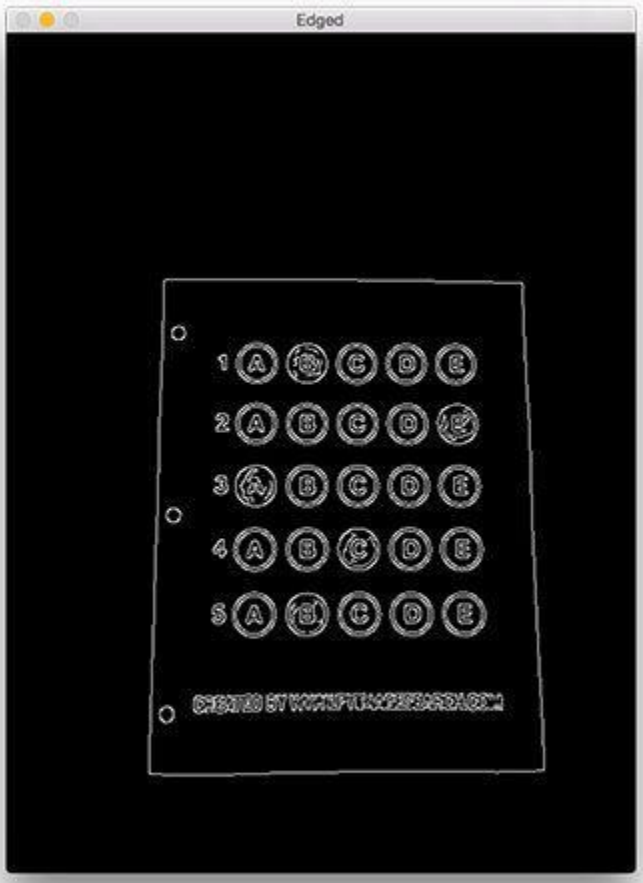
- Let’s Process our input image.

```
image = cv2.imread("test_01.png")
gray = cv2.cvtColor(image,cv2.COLOR_BGR2GRAY)
blurred = cv2.GaussianBlur(gray,(5,5),0)
edged = cv2.Canny(blurred,75,200)
```

we load our image from disk, followed by converting it to grayscale and blurring it to reduce high frequency noise .

- We then apply the Canny edge detector on to find the *edges/outlines* of the exam.

After applying edge detection our exam looks like this:



- Now we have to find outline of the exam to apply perspective transform to the image.

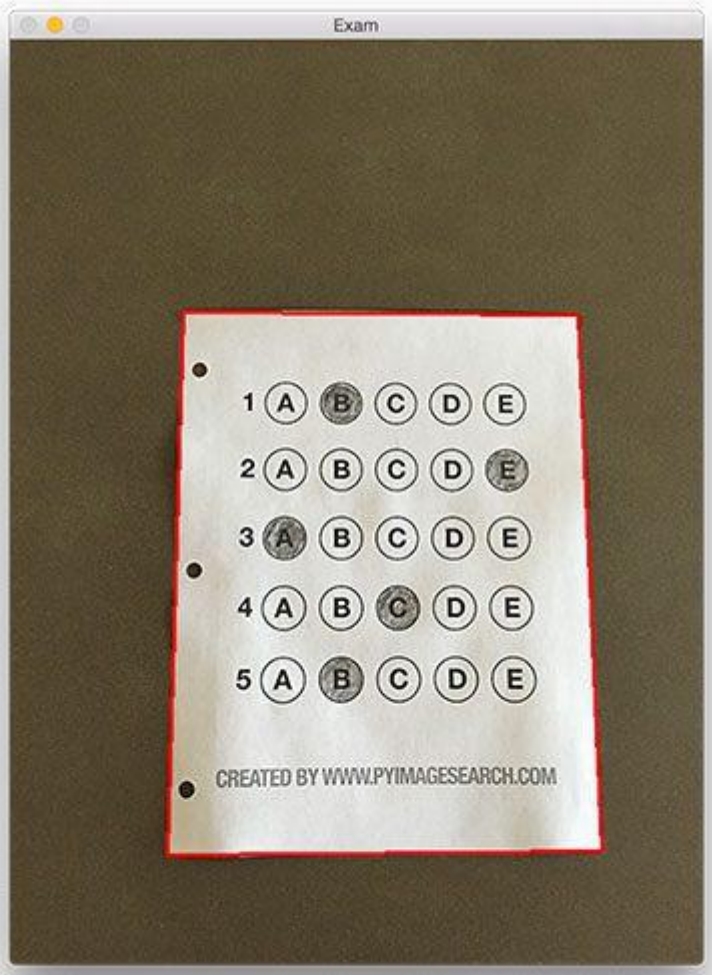
First find all contours and then short them according to the area in descending order.

Now we loop over sorted contours to approximate contour and if approximated contour has 4 points then we suppose that we have found the exam.

cnts = cv2.findContours(edged.copy(),cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
cnts = imutils.grab_contours(cnts)
docCnt = None

```
if len(cnts)>0:
    cnts = sorted(cnts,key=cv2.contourArea,reverse=True)
    for c in cnts:
        peri = cv2.arcLength(c,True)
        approx=cv2.approxPolyDP(c,0.02*peri,True)
        if len(approx) == 4:
            docCnt=approx
            break
```

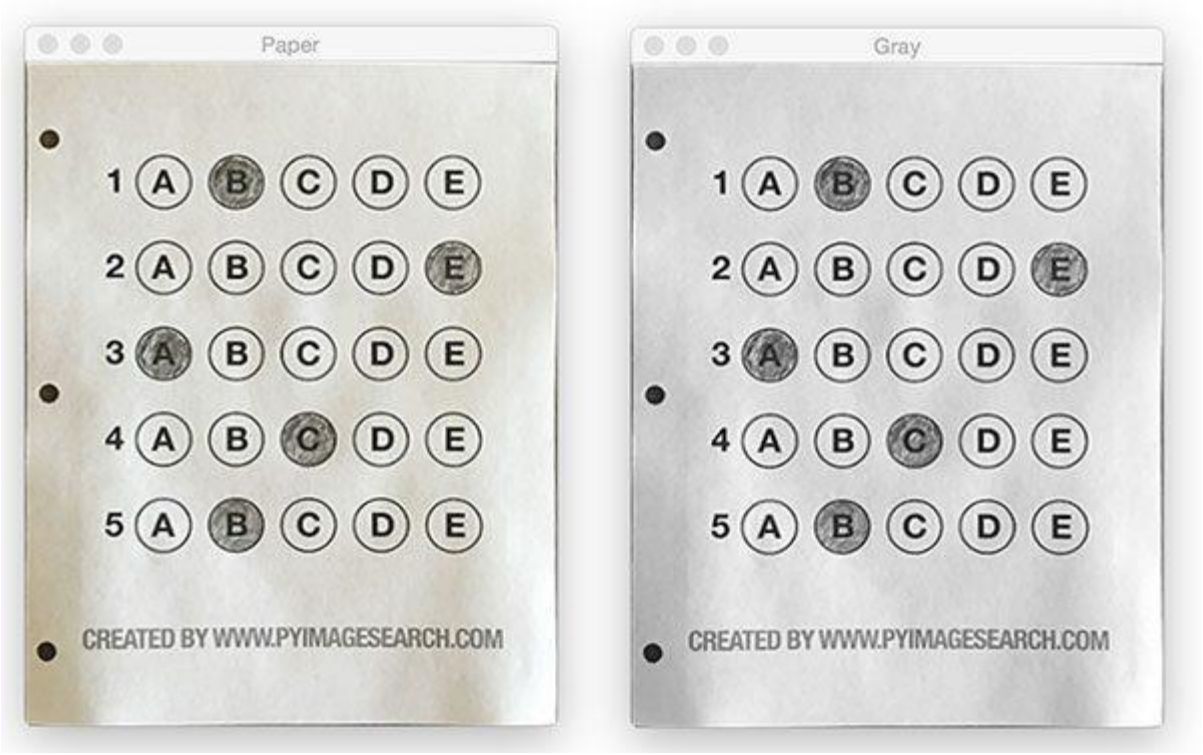
Below image that demonstrates the ‘docCnt’ variable being drawn on the original image:



- Apply perspective transform to obtain a top-down , birds-eye-view of the document.
  In this case we are applying four_point_transform method

paper = four_point_transform(image,docCnt.reshape(4,2))
warped= four_point_transform(gray,docCnt.reshape(4,2)

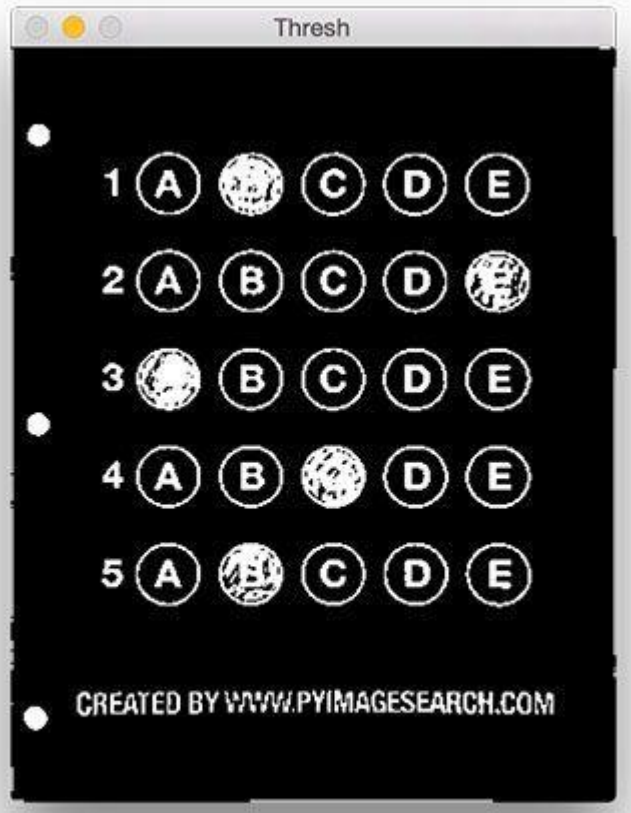
Below images shows this transform.



- We now apply OTSU's Thresholding method which segments the foreground and background of the image , converting it to Binary Image.

```
(T,threshInv)=cv2.threshold(warped,0,255,cv2.THRESH_BINARY_INV|cv2.THRESH_OTSU)
```

Notice how the *background* of the image is *black*, while the *foreground* is *white.*



This binarization will allow us to once again apply contour extraction techniques to find each of the bubbles in the exam:

- Find contours in the ‘thresh’ binary image , then initialize ‘questionCnts’ , a list that corresponds to questions/bubbles in the exam.

We now apply loop on all contours and find question bubbles by applying some conditions on contours , as given in code.

```
cnts = cv2.findContours(threshInv.copy(),cv2.RETR_EXTERNAL,cv2.CHAIN_APPROX_SIMPLE)
cnts = imutils.grab_contours(cnts)
questionCnts = []
for c in cnts:
    (x,y,w,h) = cv2.boundingRect(c)
    ar = w / float(h)
    if w>=20 and h>=20 and ar>=0.9 and ar<=1.1:
        questionCnts.append(c)
```

Below is the image showing output of questionCnts on input image.

Notice how *only* the question regions of the exam are highlighted and nothing else.

- Sort the question bubbles from top to bottom , and then left to right considering 5 bubbles in row per question.

```
questionCnts = contours.sort_contours(questionCnts,method="top-to-bottom")[0]
correct=0
```

Below Image shows each row of questions as separate colour.

- Determine selected bubble in each row.

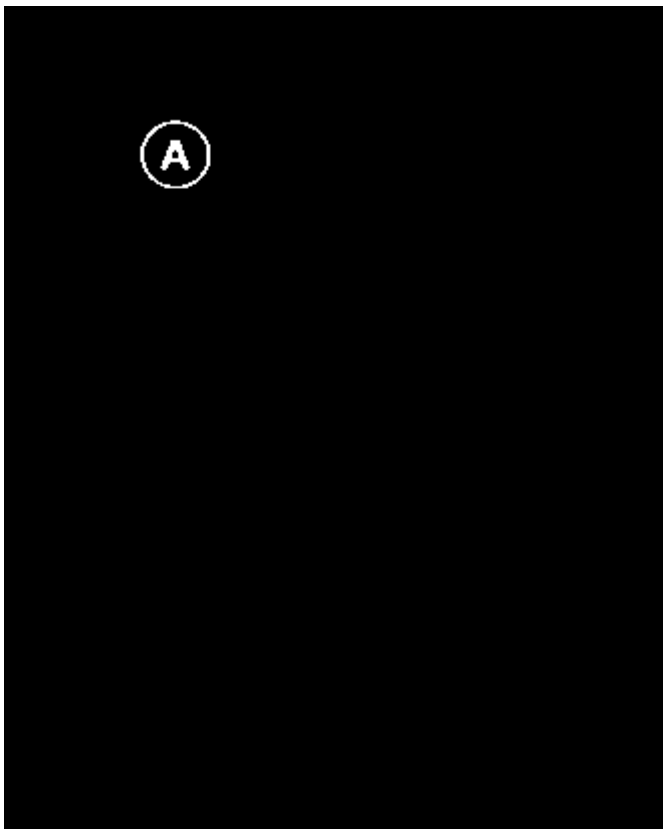
  This can be accomplished by counting no. of non-zero pixels (ie. Foreground pixels) in thresh image for each bubble.
  The bubble having maximum non-zero pixels in thresh image in each row will be selected one.

  Above process can be done by applying mask of bubbles on thresh image as done in code.

```
for (q,i) in enumerate(np.arange(0,len(questionCnts),5)):
    cnts=contours.sort_contours(questionCnts[i: i+5])[0]
```

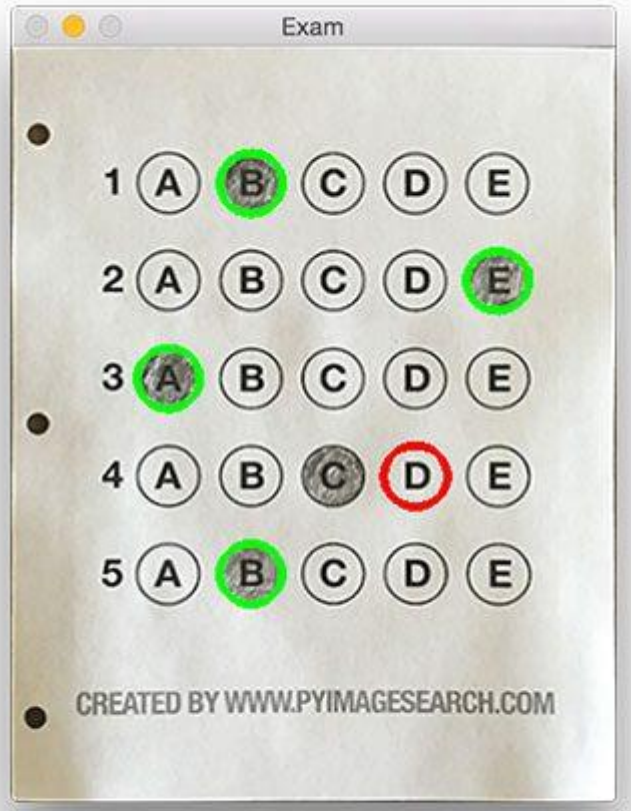
```
    bubbled=None
    for(j,c) in enumerate(cnts):
        mask=np.zeros(threshInv.shape,dtype="uint8")
        cv2.drawContours(mask,[c],-1,255,-1)

        mask=cv2.bitwise_and(threshInv,threshInv,mask=mask)
        total=cv2.countNonZero(mask)
        if bubbled is None or total>bubbled[0]:
            bubbled=(total,j)

    color=(0,0,255)
    k=ANSWER_KEY[q]

    if k==bubbled[1]:
        color=(0,255,0)
        correct+=1

    cv2.drawContours(paper,[cnts[k]],-1,color,3)
```

An example of creating and applying a mask to each question corresponding to a given question is shown below.



- Look for the answer of each question in Answer Key and match it with the marked bubble.
- If the test taker is *correct*, highlight their answer in *green*. However, if the test taker made a mistake and marked an incorrect answer, let them know by highlighting the *correct* answer in *red*:

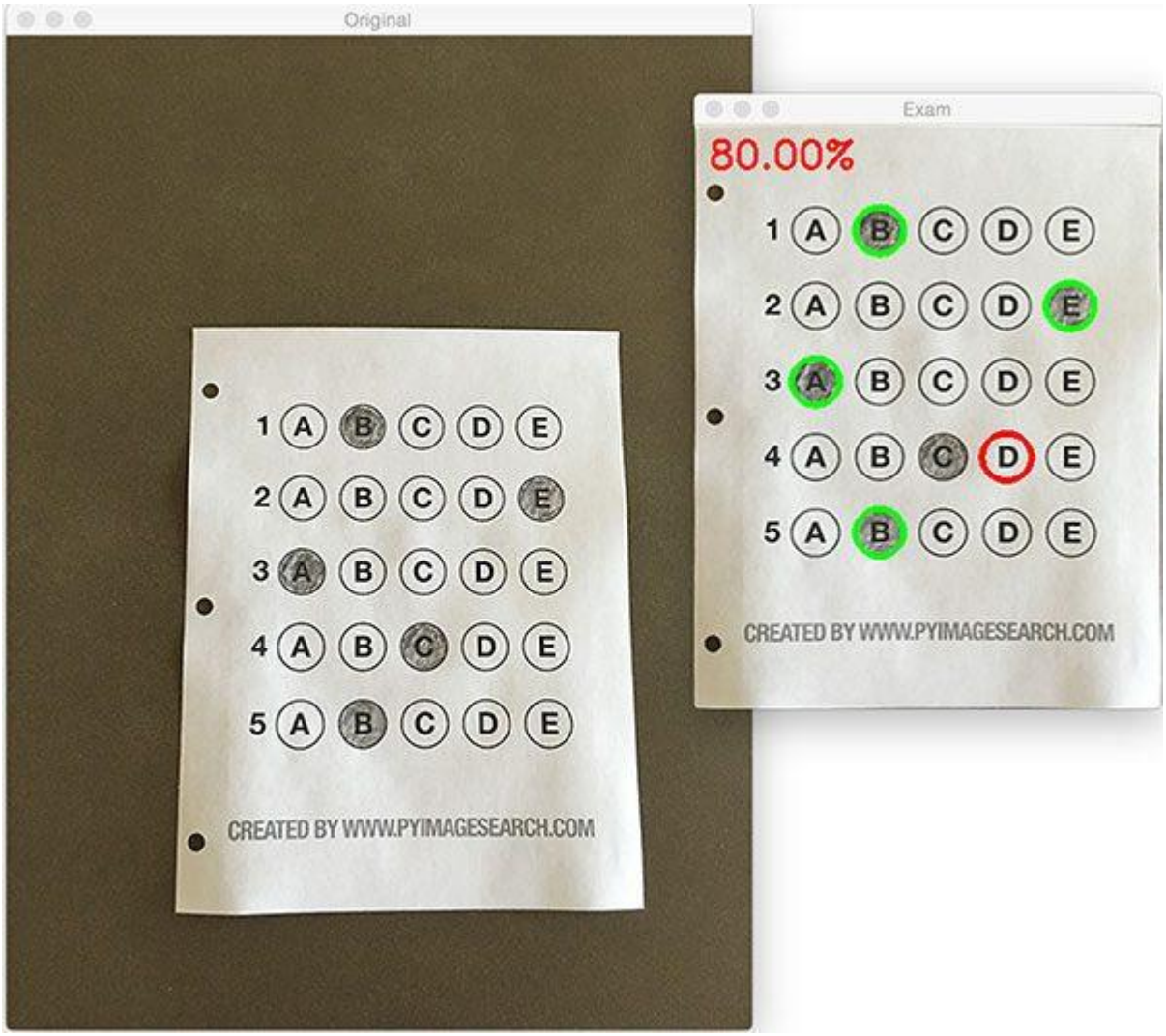
  Below image displays it:



- Finally Score the exam and display the grades on exam input image.

```
score = (correct / 5.0) * 100
print("[INFO] score: {:.2f}%".format(score))
cv2.putText(paper, "{:.2f}%".format(score), (10, 30),
    cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 0, 255), 2)
cv2.imshow("Original", image)
cv2.imshow("Exam", paper)
cv2.waitKey(0)
```

Below is the iamge of input exam and final graded exam.



- Thus we finally got the result.
- I have given 5 more input images and their graded result in the github link.

Experimentation:

First I thought and tried to use circle detection but it did is not that much successful due to user error,

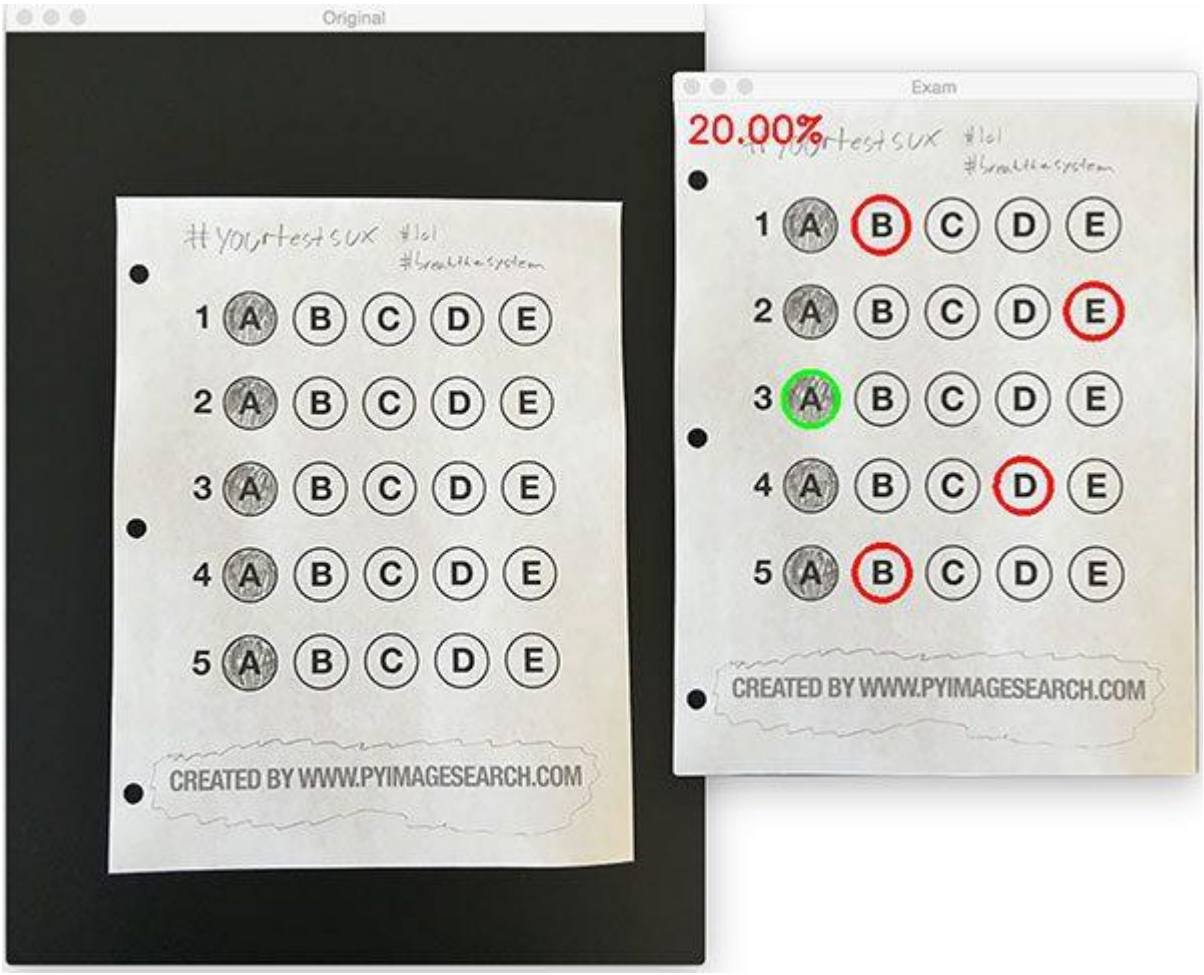
Normally filling OMR we many times makes dark slightly outside the circle which is not handled by circle detection.

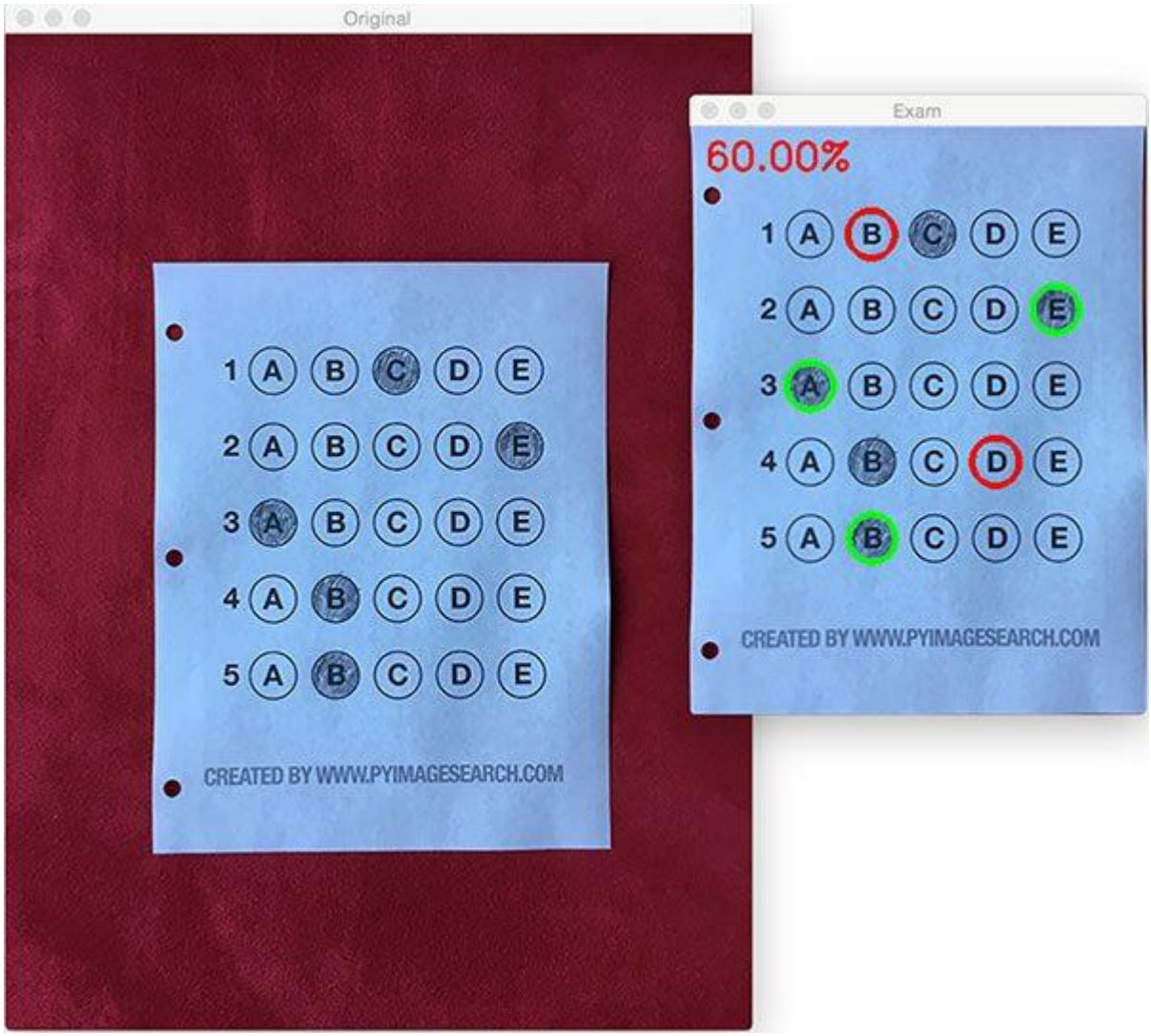
Also I first worked on another file in which I have wriiten extra codes to cross check at each step like whether we are getting image , whether we are able to sort and similarly for all functions.

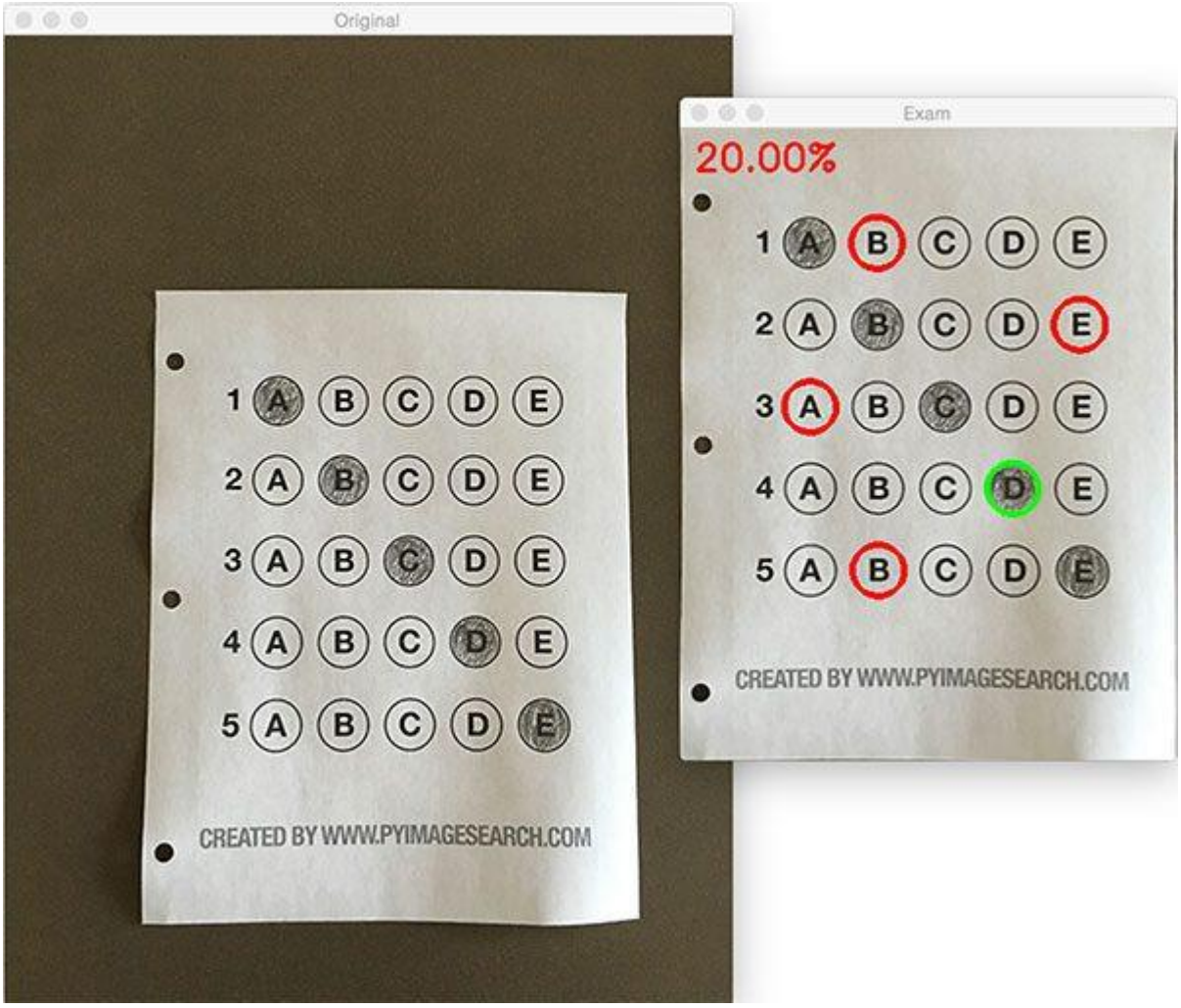
Also seen some videos how we can make it without directly using opencv.

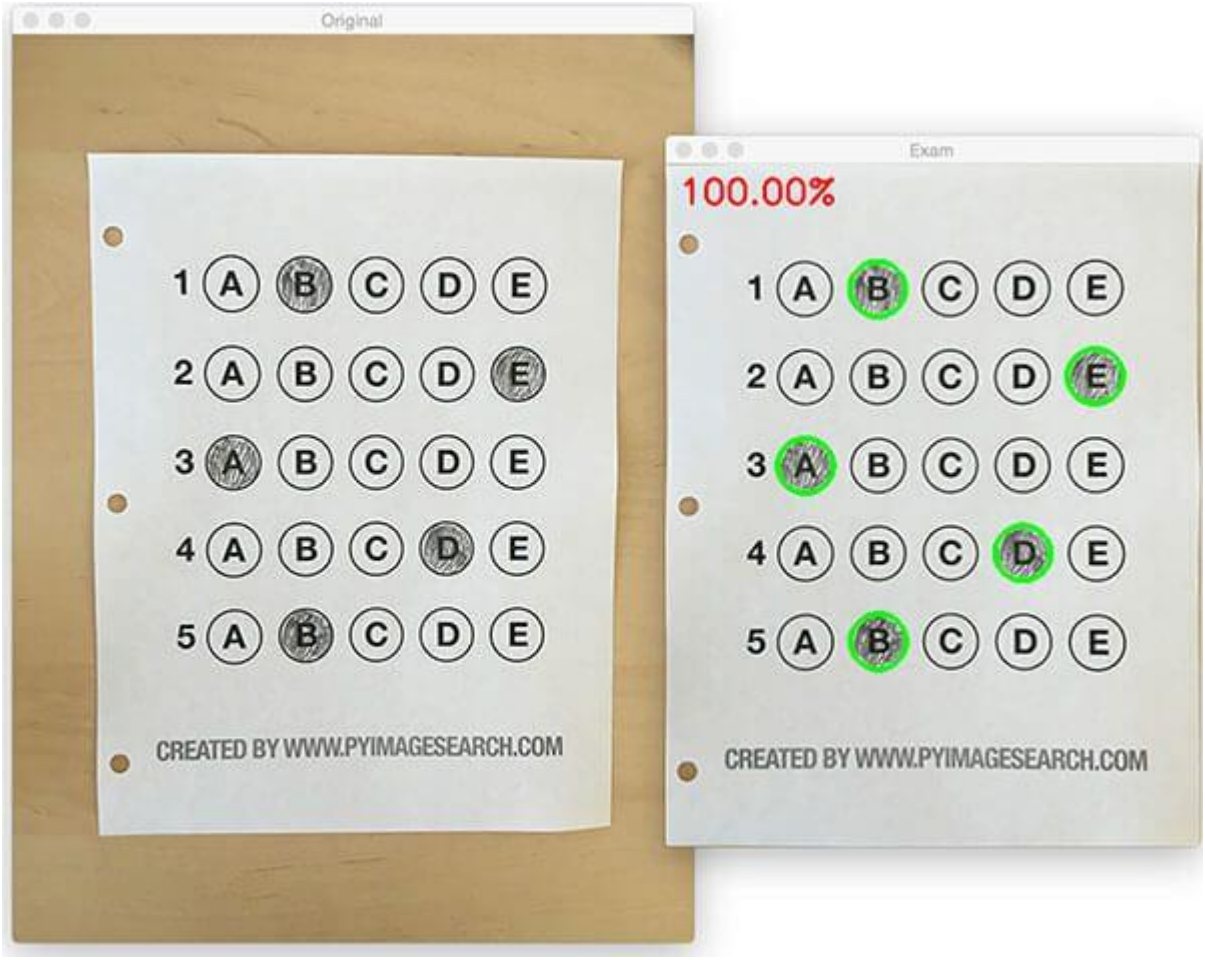
Results:

Below I have shown images of other images on which I have tested this system and it is working fine in all these cases.


Original
#yourtestsux #lol
1 A B C D E
2 A B C D E
3 A B C D E
4 A B C D E
5 A B C D E
CREATED BY WWW.PYIMAGESEARCH.COM
Exam
20.00%
1 A B C D E
2 A B C D E
3 A B C D E
4 A B C D E
5 A B C D E
CREATED BY WWW.PYIMAGESEARCH.COM


Original
1 A B C D E
2 A B C D E
3 A B C D E
4 A B C D E
5 A B C D E
CREATED BY WWW.PYIMAGESEARCH.COM
Exam
60.00%
1 A B C D E
2 A B C D E
3 A B C D E
4 A B C D E
5 A B C D E
CREATED BY WWW.PYIMAGESEARCH.COM


Original
1 A B C D E
2 A B C D E
3 A B C D E
4 A B C D E
5 A B C D E
CREATED BY WWW.PYIMAGESEARCH.COM
Exam
20.00%
1 A B C D E
2 A B C D E
3 A B C D E
4 A B C D E
5 A B C D E
CREATED BY WWW.PYIMAGESEARCH.COM



Conclusion: we have seen how this small project can reduce a lot of efforts and time required to check OMR's like in case of school.

It provides us high accuracy with low efforts and it can be made more useful by updating it further.

We have seen how little applications of python and opencv helps us to easily build this project which looks too difficult without this stuff.

Refrences:

Python Basics : https://youtu.be/vLqTf2b6GZw

Numpy Basics: https://youtu.be/Rbh1rieb3zc

Pandas Basics: https://youtu.be/RhEjmHeDNoA

Opencv playlist: https://youtube.com/playlist?list=PLzMcBGfZo4-lUA8uGjeXhBUUzPYc6vZRn

Other refrences : PyImageSearch - You can master Computer Vision, Deep Learning, and OpenCV.

I have referred python documentation for reading almost all required functions in this project.