By: Ibrahim Abou Zahr – Mohammad Abdel Rahman – Dana Mrad

**Optical Mark Recognition**

**Contents**

[1) **Introduction:** **1**](#_Toc154092192)

[1.1) Overview 1](#_Toc154092193)

[1.2) Purpose: 1](#_Toc154092194)

[2) **Classes:** **1**](#_Toc154092195)

[2.1) Main Class: 1](#_Toc154092196)

[2.2) Utilis Class 6](#_Toc154092197)

[3) **Conclusion:** **9**](#_Toc154092198)

# Introduction:

## Overview

Optical Mark Recognition is an increasingly in demand method for extracting information and marks. It offers a streamlined approach to digitizing and interpreting marked data from paper based forms.

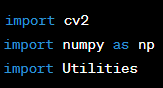
## Purpose:

The main purpose of this project is to create and implement an Optical Mark Recognition Code that addresses the need for a robust and versatile solution for automatic processing of marked documents.

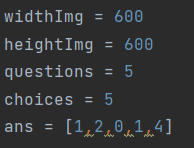
# Classes:

## 2.1) Main Class:

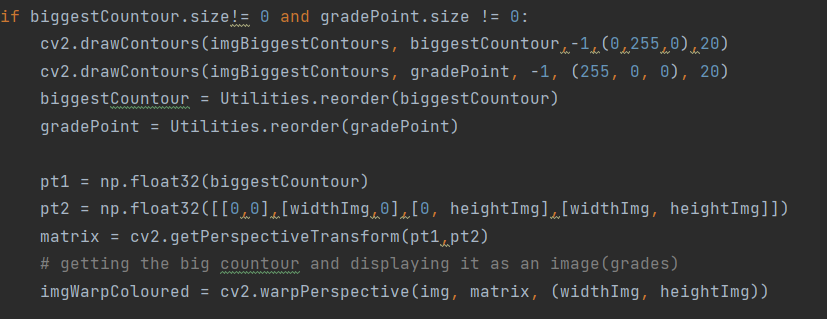
This project utilizes the OpenCV library to implement an Optical Mark Recognition system. The code used is designed to extract relevant information and grade the answers. Here is a brief breakdown of the main class code:



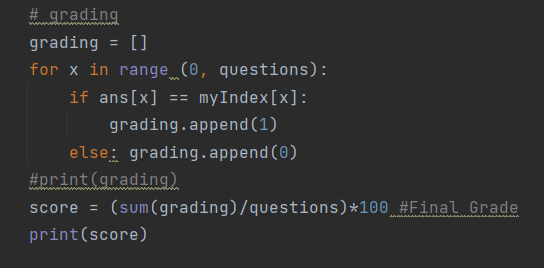
The library used here is OpenCV, numpy is imported as np and Utitlis is a custom class that contains the functions used.



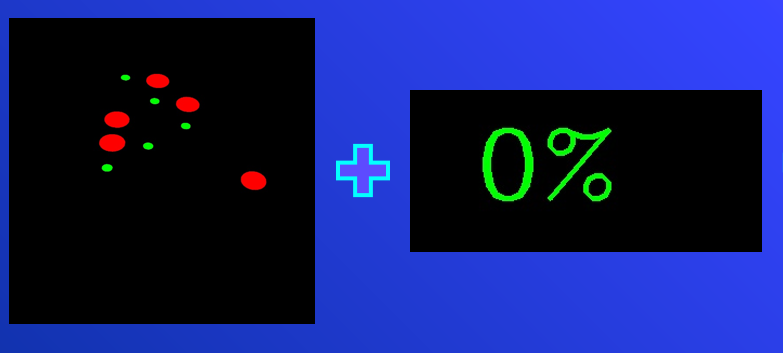
Parameters for image size (**widthImg** and **heightImg**), the number of questions and choices, correct answers (**ans**), and the file path of the input image (**path**) are defined.

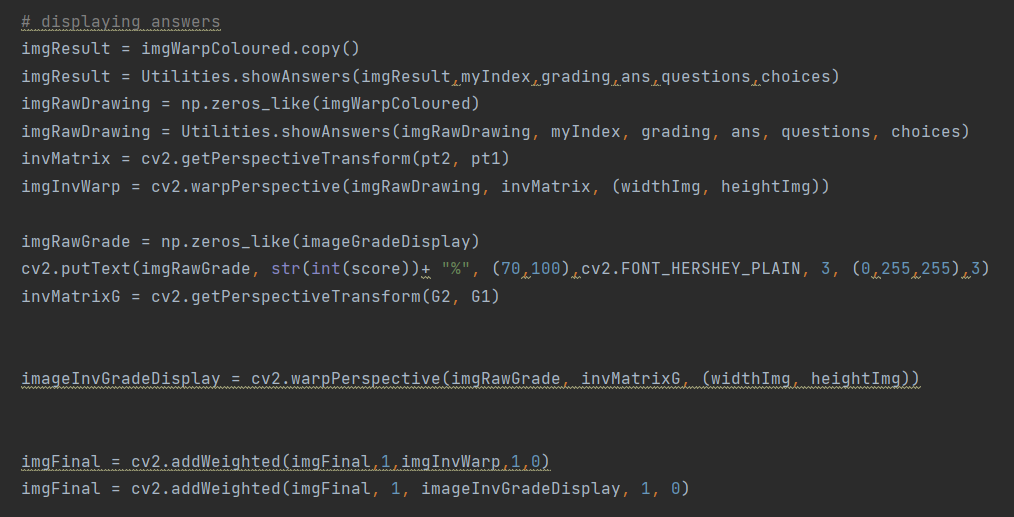


The above code performs several preprocessing steps on the input image. Contours of the image are found, and rectangles are identified using custom **utilis** functions.

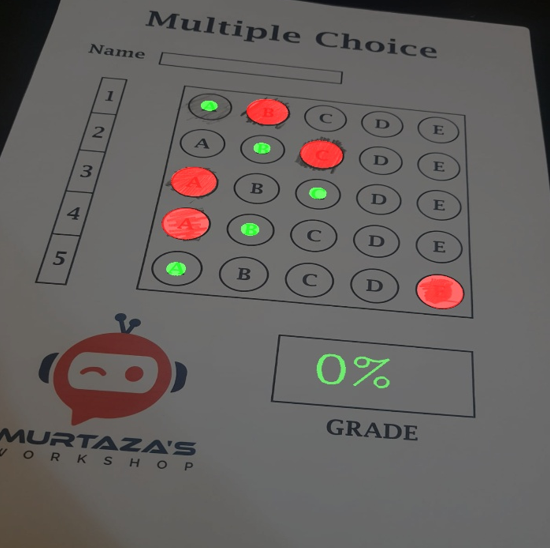


The marked answers are compared with the correct answers and a grading score is calculated.





The code displays the final results, including the graded image and a stacked view of all processing steps used.



**The code for the Main Class:**

import cv2  
import numpy as np  
import Utilities  
  
widthImg = 600  
heightImg = 600  
questions = 5  
choices = 5  
ans = [1,2,0,1,4]  
  
path = "1.jpg"  
img = cv2.imread(path)  
  
# Preprocessing  
  
# change image size  
img = cv2.resize(img,(widthImg, heightImg))  
imgContours = img.copy()  
imgFinal = img.copy()  
imgBiggestContours = img.copy()  
# Grayscale image  
imgGray = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)  
# Make image blurry  
imgBlur = cv2.GaussianBlur(imgGray, (5,5),1)  
# Canny edge detector  
imgCanny = cv2.Canny(imgBlur, 10, 50)  
# finding all countours  
countours, hierarchy = cv2.findContours(imgCanny, cv2.RETR\_EXTERNAL,cv2.CHAIN\_APPROX\_NONE)  
cv2.drawContours(imgContours,countours, -1,(0,255,0),10)  
# find rectangles  
rectCon = Utilities.rectCountour(countours)  
biggestCountour = Utilities.getCornerPoints(rectCon[0])  
print(biggestCountour.shape)  
gradePoint = Utilities.getCornerPoints(rectCon[1]) # second biggest countour  
#print(biggestCountour)  
  
if biggestCountour.size!= 0 and gradePoint.size != 0:  
 cv2.drawContours(imgBiggestContours, biggestCountour,-1,(0,255,0),20)  
 cv2.drawContours(imgBiggestContours, gradePoint, -1, (255, 0, 0), 20)  
 biggestCountour = Utilities.reorder(biggestCountour)  
 gradePoint = Utilities.reorder(gradePoint)  
  
 pt1 = np.float32(biggestCountour)  
 pt2 = np.float32([[0,0],[widthImg,0],[0, heightImg],[widthImg, heightImg]])  
 matrix = cv2.getPerspectiveTransform(pt1,pt2)  
 # getting the big countour and displaying it as an image(grades)  
 imgWarpColoured = cv2.warpPerspective(img, matrix, (widthImg, heightImg))  
  
 G1 = np.float32(gradePoint)  
 G2 = np.float32([[0, 0], [325, 0], [0, 150], [325, 150]])  
 matrixG = cv2.getPerspectiveTransform(G1, G2)  
 # getting the second big countour and displaying it as an image(final mark bar)  
 imageGradeDisplay = cv2.warpPerspective(img, matrixG, (325, 150))  
  
 # Apply Threshold  
 imgWarpGray = cv2.cvtColor(imgWarpColoured,cv2.COLOR\_BGR2GRAY)  
 imgThresh = cv2.threshold(imgWarpGray,170,255,cv2.THRESH\_BINARY\_INV)[1]  
  
 boxes = Utilities.splitBoxes(imgThresh)  
 # cv2.imshow("Test", boxes[2])  
 # print(cv2.countNonZero(boxes[1]),cv2.countNonZero(boxes[2]))  
 # get pixel non zero values of each box  
 myPixelVal = np.zeros((questions, choices))  
 countC = 0  
 countR= 0  
  
 for image in boxes:  
 totalPixels = cv2.countNonZero(image)  
 myPixelVal[countR][countC] = totalPixels  
 # to iterate through all marks  
 countC += 1  
 if(countC == choices): countR += 1; countC = 0  
 #print(myPixelVal)  
  
 # finding index values of the markings  
 myIndex = []  
 for x in range(0,questions):  
 arr = myPixelVal[x]  
 myIndexVAl = np.where(arr == np.amax(arr))  
 myIndex.append(myIndexVAl[0][0])  
  
  
 # grading  
 grading = []  
 for x in range (0, questions):  
 if ans[x] == myIndex[x]:  
 grading.append(1)  
 else: grading.append(0)  
 #print(grading)  
 score = (sum(grading)/questions)\*100 #Final Grade  
 print(score)  
  
  
 # displaying answers  
 imgResult = imgWarpColoured.copy()  
 imgResult = Utilities.showAnswers(imgResult,myIndex,grading,ans,questions,choices)  
 imgRawDrawing = np.zeros\_like(imgWarpColoured)  
 imgRawDrawing = Utilities.showAnswers(imgRawDrawing, myIndex, grading, ans, questions, choices)  
 invMatrix = cv2.getPerspectiveTransform(pt2, pt1)  
 imgInvWarp = cv2.warpPerspective(imgRawDrawing, invMatrix, (widthImg, heightImg))  
  
 imgRawGrade = np.zeros\_like(imageGradeDisplay)  
 cv2.putText(imgRawGrade, str(int(score))+ "%", (70,100),cv2.FONT\_HERSHEY\_PLAIN, 3, (0,255,255),3)  
 invMatrixG = cv2.getPerspectiveTransform(G2, G1)  
  
  
 imageInvGradeDisplay = cv2.warpPerspective(imgRawGrade, invMatrixG, (widthImg, heightImg))  
  
  
 imgFinal = cv2.addWeighted(imgFinal,1,imgInvWarp,1,0)  
 imgFinal = cv2.addWeighted(imgFinal, 1, imageInvGradeDisplay, 1, 0)  
  
  
imgBlank = np.zeros\_like(img)  
imgArray = ([img,imgGray,imgBlur,imgCanny],  
 [imgContours,imgBiggestContours,imgWarpColoured,imgThresh],  
 [imgResult,imgRawDrawing,imgInvWarp,imgFinal])  
lables = [["Original", "Gray", "Blur", "Canny"],  
 ["Contours","Biggest Con", "Warp", "Threshold"],  
 ["Result", "Raw Drawing", "Inv Warp", "Final"]]  
imageStacked = Utilities.stackImages(imgArray,0.3,lables)  
cv2.imshow("Final Result" , imgFinal)  
cv2.imshow("Stacked Images ", imageStacked)  
cv2.waitKey(0)

## 2.2) Utilis Class

The Utilis Class consists of all the utility functions for the Optical Mark Recognition project. The functions are meant to be used with the main script for processing images. Here is a brief explanation of the following functions:

**stackImages Function:**

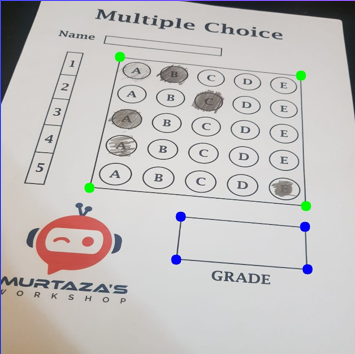


This function takes a 2D array of images and it stacks the images horizontally or vertically and resizes them if necessary.

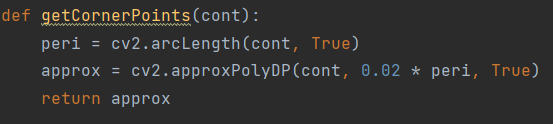
**rectCountour Function:**



This function takes list of contours as input, identifies rectangular contours from the input given, and sorts the contours in descending order.



**getCornerPoints Function:**



Takes a contour as input and approximates the contour to obtain corner points.

**reorder Function:**



Above function takes the corner points as input, changes the shape of the points and reorders them to represent the four corners of a rectangle.

**splitBoxes Function:**



Takes an image as input and splits it into boxes and vertically splits the image into rows.

**showAnswers Function:**



Takes an image, indices of marked answers, correct answers and details about the number of questions available. Draws circles to represent the marked answers, it also highlights the correct answer on the image.

**The code for the Utilis Class:**

import cv2  
import numpy as np  
  
## To stack all images in one window  
def stackImages(imgArray,scale,lables=[]):  
 rows = len(imgArray)  
 cols = len(imgArray[0])  
 rowsAvailable = isinstance(imgArray[0], list)  
 width = imgArray[0][0].shape[1]  
 height = imgArray[0][0].shape[0]  
 if rowsAvailable:  
 for x in range(0, rows):  
 for y in range(0,cols):  
 imgArray[x][y] = cv2.resize(imgArray[x][y], (0,0),None,scale,scale)  
 if len(imgArray[x][y].shape)==2: imgArray[x][y] = cv2.cvtColor(imgArray[x][y], cv2.COLOR\_GRAY2BGR)  
 imageBlank = np.zeros((height, width, 3), np.uint8)  
 hor = [imageBlank]\*rows  
 hor\_con = [imageBlank]\*rows  
 for x in range(0, rows):  
 hor[x] = np.hstack(imgArray[x])  
 hor\_con[x] = np.concatenate(imgArray[x])  
 ver = np.vstack(hor)  
 ver\_con = np.concatenate(hor)  
 else:  
 for x in range(0,rows):  
 imgArray[x] = cv2.resize(imgArray[x], (0, 0), None, scale, scale)  
 if len(imgArray[x].shape) == 2: imgArray[x] = cv2.cvtColor(imgArray[x], cv2.COLOR\_GRAY2BGR)  
 hor = np.hstack(imgArray)  
 hor\_con = np.concatenate(imgArray)  
 ver = hor  
 if len(lables) != 0:  
 eachImgWidth = int(ver.shape[1] / cols)  
 eachImgHeight = int(ver.shape[0] / rows)  
 # print(eachImgHeight)  
 for d in range(0, rows):  
 for c in range(0, cols):  
 cv2.rectangle(ver, (c \* eachImgWidth, eachImgHeight \* d),  
 (c \* eachImgWidth + len(lables[d][c]) \* 13 + 27, 30 + eachImgHeight \* d), (255, 255, 255),  
 cv2.FILLED)  
 cv2.putText(ver, lables[d][c], (eachImgWidth \* c + 10, eachImgHeight \* d + 20),  
 cv2.FONT\_HERSHEY\_COMPLEX, 0.7, (255, 0, 255), 2)  
 return ver  
  
def rectCountour(countours):  
 rectCon = []  
  
 for i in countours:  
 area = cv2.contourArea(i)  
  
 if area > 50:  
 peri = cv2.arcLength(i,True)  
 approx = cv2.approxPolyDP(i,0.02\*peri,True)  
 #print("Corner Points ", len(approx))  
 if len(approx) == 4:  
 rectCon.append(i)  
 rectCon = sorted(rectCon,key = cv2.contourArea, reverse = True)  
  
 return rectCon  
  
  
def getCornerPoints(cont):  
 peri = cv2.arcLength(cont, True)  
 approx = cv2.approxPolyDP(cont, 0.02 \* peri, True)  
 return approx  
  
def reorder(myPoints):  
  
 myPoints = myPoints.reshape((4,2))  
 myPointsNew = np.zeros((4,1,2),np.int32)  
  
 add = myPoints.sum(1)  
 # print(myPoints)  
 # print(add)  
 myPointsNew[0] = myPoints[np.argmin(add)] # [0,0]  
 myPointsNew[3] = myPoints[np.argmax(add)]  
 diff = np.diff(myPoints,axis = 1)  
 myPointsNew[1] = myPoints[np.argmin(diff)] # [width, 0]  
 myPointsNew[2] = myPoints[np.argmax(diff)] # [0, height]  
 # print(diff)  
  
 return myPointsNew  
  
def splitBoxes(img):  
 rows = np.vsplit(img,5)  
 # to get row of grades  
 boxes = []  
 for r in rows:  
 cols = np.hsplit(r,5)  
 for box in cols:  
 boxes.append(box)  
 #cv2.imshow("Split", box)  
 return boxes  
  
def showAnswers(img,myIndex,grading,answers,questions,choices):  
 secW = int(img.shape[1]/questions)  
 secH = int(img.shape[1]/choices)  
  
  
 for x in range(0, questions):  
 myAns = myIndex[x]  
 cX = (myAns\*secW) + secW//2  
 cY = (x\*secH) + secH//2  
  
 if grading[x] == 1:  
 myColor = (0,255,0)  
 else:  
 myColor = (0,0,255)  
 correctAns = answers[x]  
 cv2.circle(img, ((correctAns\* secW) + secW//2,(x\*secH)+secH//2), 20, (0,255,0), cv2.FILLED)  
  
 cv2.circle(img,(cX,cY),50,myColor,cv2.FILLED)  
 return img

# Conclusion:

In conclusion, the implementation of the Optical Mark Recognition program has successfully addresses the difficulties with automated processing of marked paper forms. Therefore, this project represents a significant step forward in analyzing marked documents and information on a revolutionary scale.