**ĐẠI HỌC QUỐC GIA HÀ NỘI**
**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ**


# BÁO CÁO MÔN HỌC
# XỬ LÝ ẢNH

## OPTICAL MARK RECOGNITION:
## MARK THE EXAMS

**Nhóm 14:**

| Họ và tên | Mã sinh viên |
|---|---|
| Cung Văn Thắng* | 21020939 |
| Đinh Thế An | 21020039 |
| Nguyễn Sỹ Hùng | 20020416 |
| Phan Công Tiến | 20020266 |
| Nguyễn Xuân Huy | 17021266 |

*Hà nội, ngày 4 tháng 5 năm 2023*

# MỤC LỤC

# I. Giới thiệu

Công nghệ nhận dạng đánh dấu quang học trích xuất dữ liệu hữu ích từ các trường được đánh dấu như trường điền thông tin và hộp kiểm rất nhanh chóng và có độ chính xác cao. Việc sử dụng phổ biến nhất là trong các văn phòng, học viện và bộ phận nghiên cứu, nơi phải xử lý một số lượng lớn tài liệu điền bằng tay như khảo sát, bảng câu hỏi, bài kiểm tra, phiếu trả lời và phiếu bầu. Khi mà có thể xử lý hàng trăm nghìn tài liệu vật lý mỗi giờ và độ chính xác của nó lên tới 99%. Một ví dụ phổ biến là việc sử dụng các biểu mẫu tiêu chuẩn hóa trong các trường học nơi học sinh phải điền vào một điểm được xác định trước trên trang tính, dùng làm điểm cho thuật toán nhận dạng điểm quang học.

### *Optical Mark Recognition (OMR) là gì?*

Công nghệ nhận dạng đánh dấu quang học là một phương pháp điện tử để thu thập dữ liệu do con người xử lý bằng cách xác định các nhãn hiệu nhất định trên tài liệu. Thông thường, quá trình nhận dạng nhãn hiệu quang học đạt được với sự hỗ trợ của máy quét kiểm tra sự truyền hoặc phản xạ ánh sáng qua giấy; những nơi có đánh dấu sẽ phản chiếu ít ánh sáng hơn so với tờ giấy trắng, dẫn đến khả năng phản xạ tương phản kém hơn.

# II. Kỹ thuật sử dụng

1. **GrayScale**
2. **Canny edge detection**
3. **Gaussian blur**
4. **Threshold**
5. **Homography matrix**

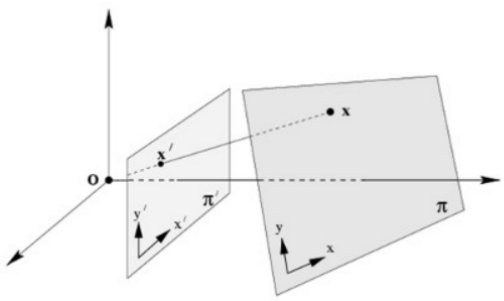
Briefly, the planar homography relates the transformation between two planes (up to a scale factor):

$$s \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \mathbf{H} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

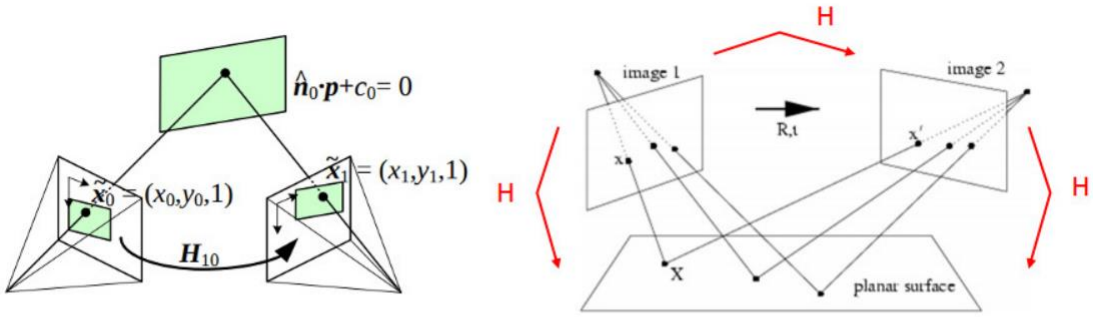
The homography matrix is a `3×3` matrix but with 8 DoF (degrees of freedom) as it is estimated up to a scale. It is generally normalized (see also 1) with $h_{33} = 1$ or $h_{11}^2 + h_{12}^2 + h_{13}^2 + h_{21}^2 + h_{22}^2 + h_{23}^2 + h_{31}^2 + h_{32}^2 + h_{33}^2 = 1$.

The following examples show different kinds of transformation but all relate a transformation between two planes.

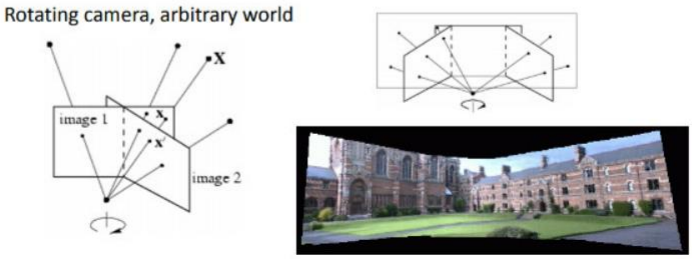
- a planar surface and the image plane (image taken from 2)



- a planar surface viewed by two camera positions (images taken from 3 and 2)

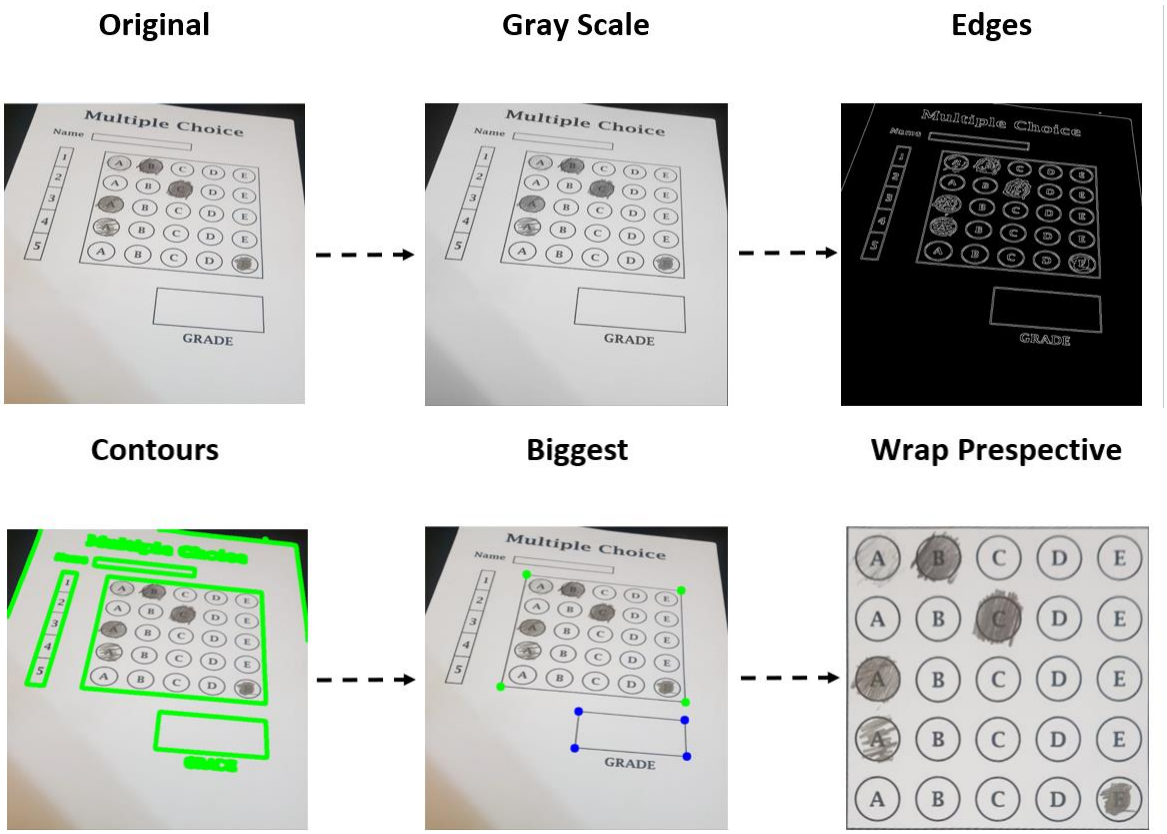


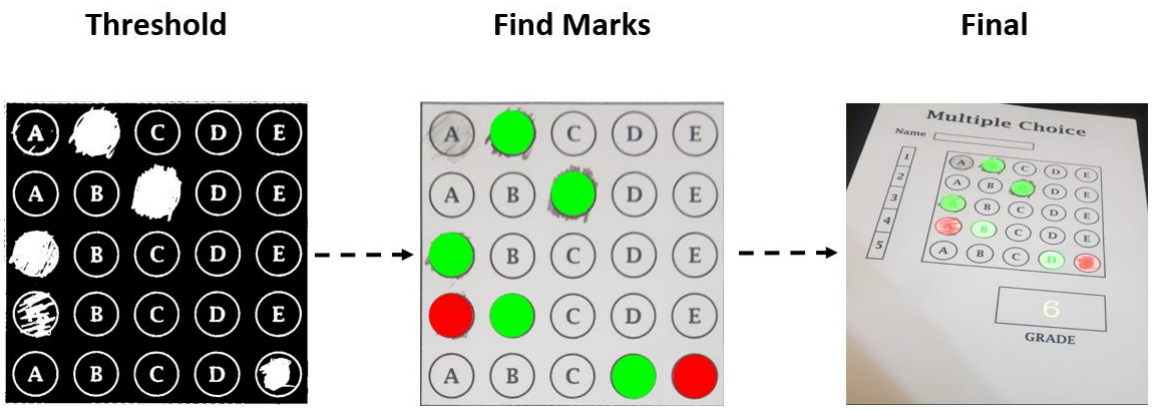

- a rotating camera around its axis of projection, equivalent to consider that the points are on a plane at infinity (image taken from 2)




## III. Tiền xử lý

Source Code: https://github.com/CUNGVANTHANG/Mark-The-Exams






Các thư viện sử dụng:

```
import cv2
import numpy as np
import utlis
```

1. **Đọc và xử lý ảnh đầu vào**

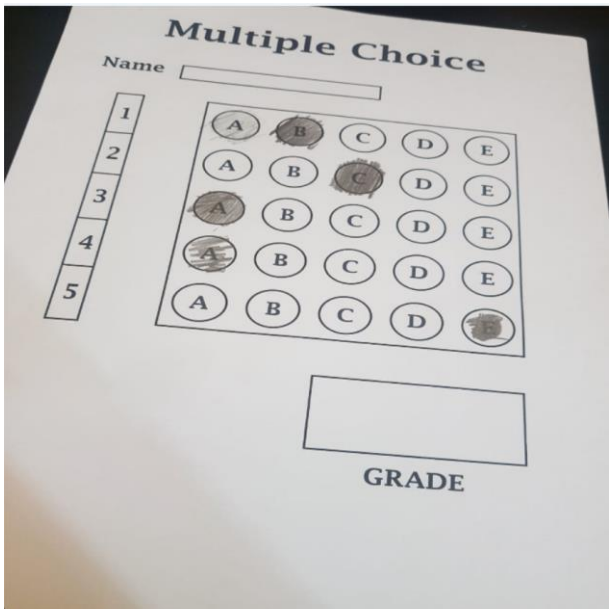
- Đọc ảnh từ đường dẫn được chỉ định

```
path = "input.jpg"
img = cv2.imread(path)
```

- Thay đổi kích thước ảnh đến kích thước mong muốn.

```
widthImg: int = 700
heightImg = 700
img = cv2.resize(img, (widthImg, heightImg))
```

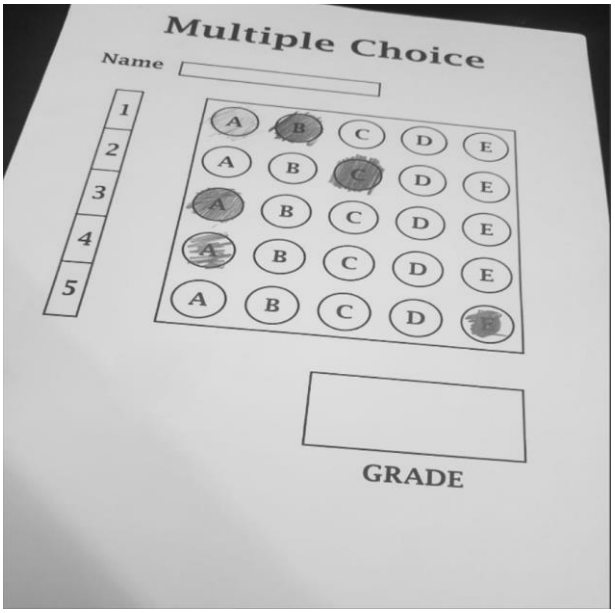
*Kết quả:*




- Chuyển đổi ảnh sang dạng ảnh xám để đơn giản hóa quá trình xử lý và giảm tính toán

```
imgGray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

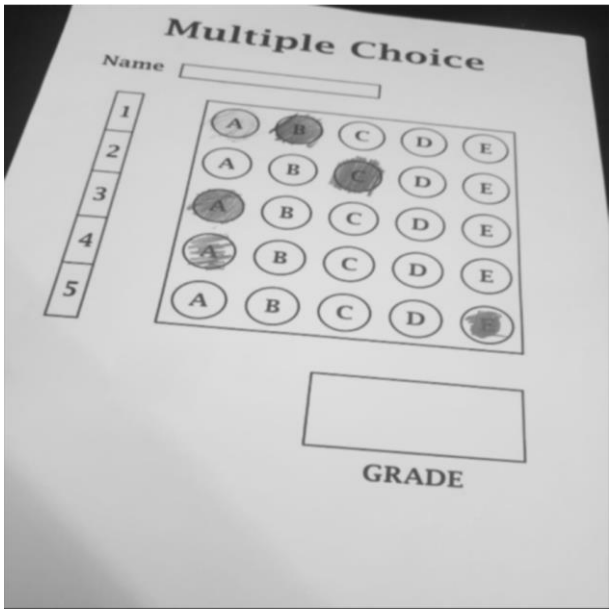
*Kết quả:*




- Làm mờ ảnh xám bằng bộ lọc Gaussian với kenel 5x5 giúp cải thiện chất lượng ảnh, giảm nhiễu

```
imgBlur = cv2.GaussianBlur(imgGray, (5, 5), 1)
```

*Kết quả:*

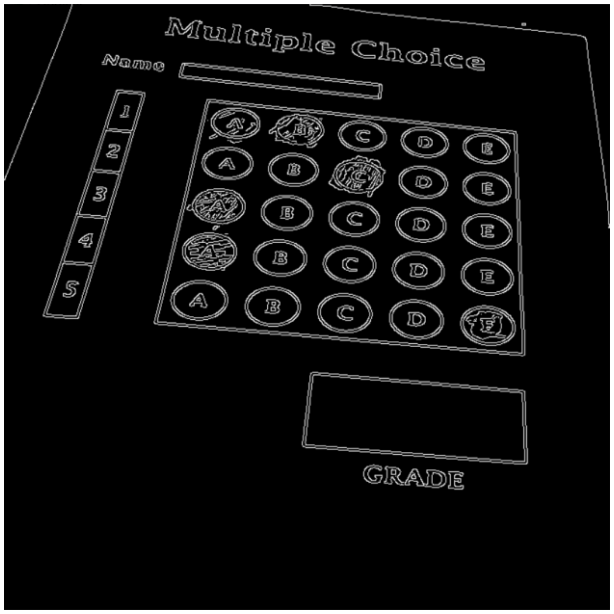



**2. Tìm các đường viền (contours) trong ảnh**

- Sử dụng phép biến đổi Canny để phát hiện biên cạnh của ảnh

```
imgCanny = cv2.Canny(imgBlur, 10, 50)
```

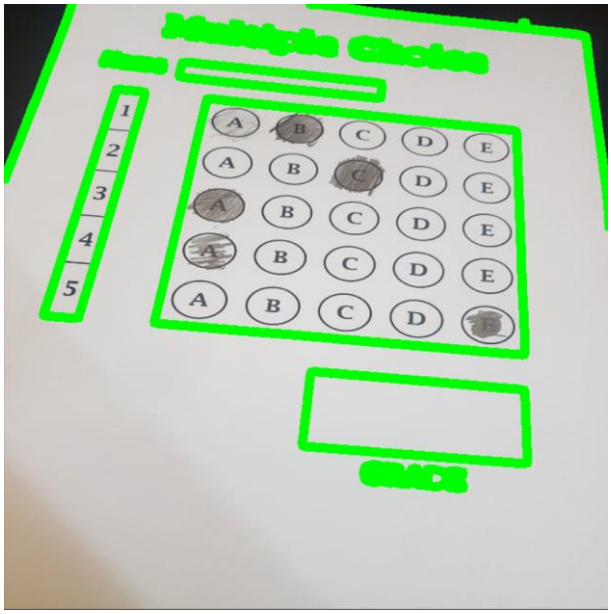
*Kết quả:*




- Tìm và lưu các đường viền ngoài cùng của ảnh

```
imgContours = img.copy()
contours, hierarchy = cv2.findContours(imgCanny,
cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)
cv2.drawContours(imgContours, contours, -1, (0, 255),
10)
```

*Kết quả:*



### 3. Chuyển đổi các đường viền thành hình chữ nhật

- Tìm hình chữ nhật có diện tích lớn nhất và hình chữ nhật thứ hai từ các đường viền tìm được.

```
rectCon = utlis.rectContour(contours)
biggestContour = utlis.getCornerPoints(rectCon[0])
gradePoints = utlis.getCornerPoints(rectCon[1])
```

*Kết quả:*

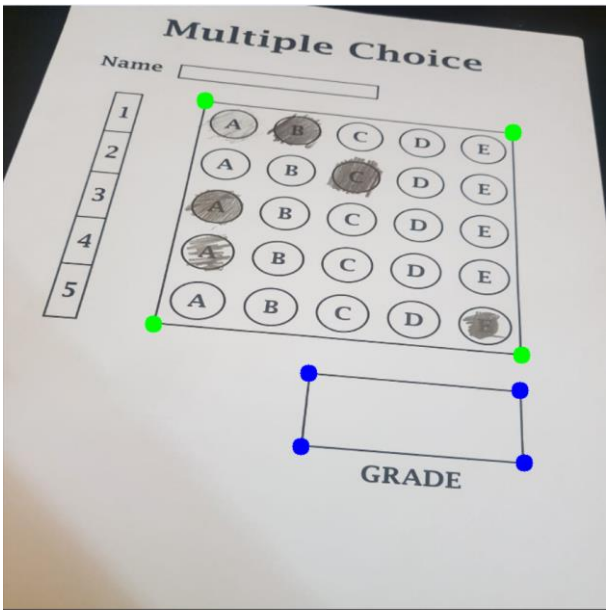
```
biggestContour:  [[[234 110]]

 [[174 369]]

 [[601 405]]

 [[591 147]]]
(4, 1, 2)
```

```
gradePoints:  [[[354 426]]

 [[345 511]]

 [[604 530]]

 [[599 446]]]
(4, 1, 2)
```

- Đánh dấu điểm góc của hình chữ nhật có diện tích lớn nhất và hình chữ nhật thứ hai

```
imgBiggestContours = img.copy()
if biggestContour.size != 0 and gradePoints.size != 0:
    cv2.drawContours(imgBiggestContours, biggestContour,
-1, (0, 255, 0), 20)
    cv2.drawContours(imgBiggestContours, gradePoints, -
1, (255, 0, 0), 20)
```

*Kết quả:*




- Sắp xếp lại các điểm góc trên các hình chữ nhật.

```
    biggestContour = utlis.reorder(biggestContour)
    gradePoints = utlis.reorder(gradePoints)
```

*Kết quả:*

```
biggestContour:  [[[234 110]]

 [[591 147]]

 [[174 369]]

 [[601 405]]]
(4, 1, 2)
```

```
gradePoints:  [[[354 426]]

 [[599 446]]

 [[345 511]]

 [[604 530]]]
(4, 1, 2)
```

**4. Biến đổi góc nhìn**

- Tính toán ma trận biến đổi góc nhìn từ các điểm góc trên các hình chữ nhật.

```
pt1 = np.float32(biggestContour)
pt2 = np.float32([[0, 0], [widthImg, 0], [0, heightImg],
[widthImg, heightImg]])
matrix = cv2.getPerspectiveTransform(pt1, pt2)

pt1G = np.float32(gradePoints)
pt2G = np.float32([[0, 0], [325, 0], [0, 150], [325,
```

```
150]])
matrixG = cv2.getPerspectiveTransform(pt1G, pt2G)
```

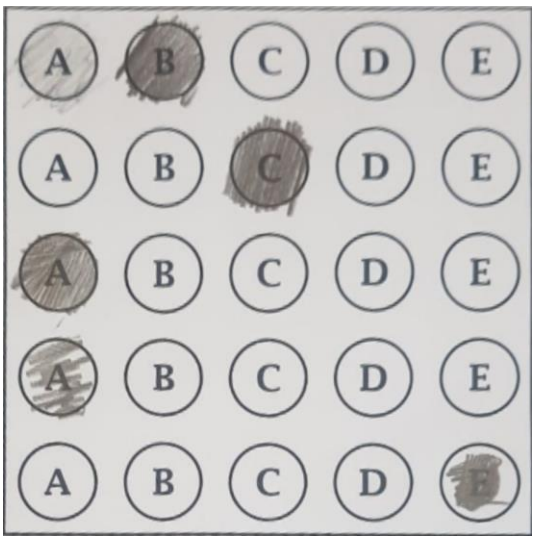
*Kết quả:*

```
matrix:  [[ 1.95290441e+00  4.52410289e-01 -5.06744764e+02]
 [-3.42902975e-01  3.30855033e+00 -2.83701240e+02]
 [-1.55488069e-04  7.60461324e-04  1.00000000e+00]]
matrixG:  [[ 1.95290441e+00  4.52410289e-01 -5.06744764e+02]
 [-3.42902975e-01  3.30855033e+00 -2.83701240e+02]
 [-1.55488069e-04  7.60461324e-04  1.00000000e+00]]
```

- Áp dụng biến đổi góc nhìn cho ảnh gốc và ảnh hiển thị đáp án.

```
imgWrapColored = cv2.warpPerspective(img, matrix,
(widthImg, heightImg))
imgGradeDisplay = cv2.warpPerspective(img, matrixG,
(325, 150))
```

*Kết quả:*

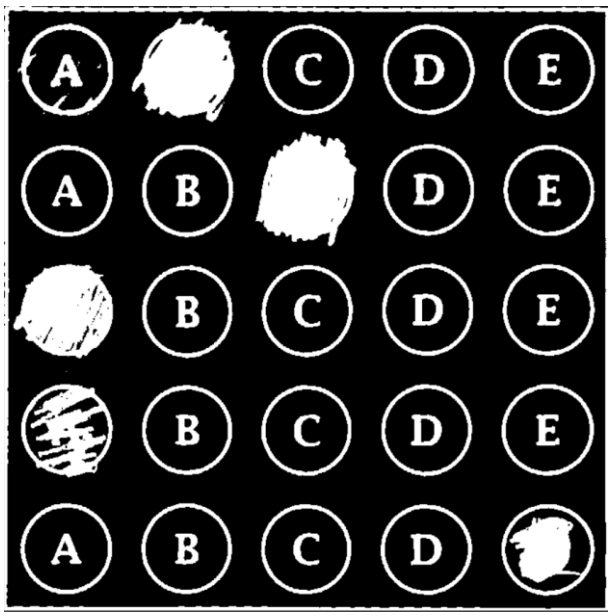



**5. Xử lý ảnh sau biến đổi góc nhìn**

- Chuyển đổi ảnh sau biến đổi góc nhìn sang ảnh xám và áp dụng ngưỡng nhị phân để tách các ô đáp án.

```
imgWrapGray = cv2.cvtColor(imgWrapColored,
cv2.COLOR_BGR2GRAY)
imgThresh = cv2.threshold(imgWrapGray, 170, 255,
cv2.THRESH_BINARY_INV)[1]
```

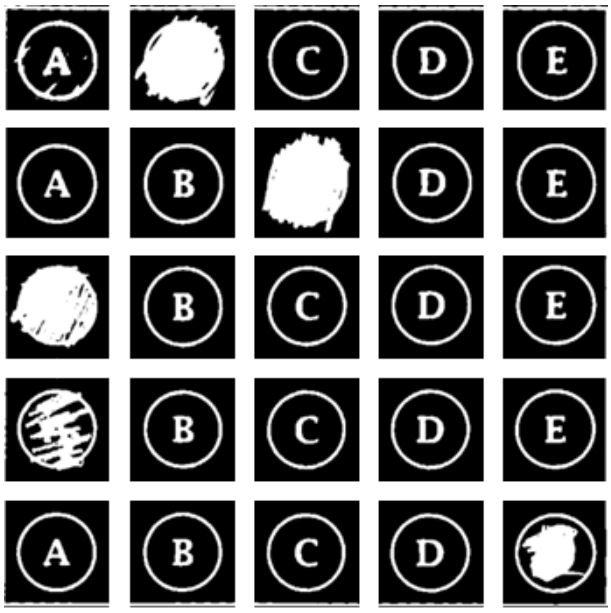
*Kết quả:*



- Tách các ô đáp án thành các ảnh nhỏ độc lập.

```
boxes = utlis.splitBoxes(imgThresh)
```

*Kết quả:*



6. **Xử lý và phân tích từng ô đáp án**

- Đếm số điểm ảnh khác không trong mỗi ô đáp án, tính toán và lưu giá trị pixel của từng ô đáp án.

```
myPixelVal = np.zeros((questions, choices))
countC = 0
countR = 0
for image in boxes:
    totalPixels = cv2.countNonZero(image)
    myPixelVal[countR][countC] = totalPixels
```

```
    countC += 1
    if countC == choices:
        countR += 1
        countC = 0
```

*Kết quả:*

```
myPixelVal:  [[ 4248. 10269.  2689.  2946.  3302.]
 [ 2878.  2301. 10670.  2251.  2620.]
 [ 9553.  2233.  2019.  2224.  2643.]
 [ 6420.  2167.  1946.  2199.  2672.]
 [ 3231.  2719.  2506.  2677.  6349.]]
```

- Xác định vị trí đáp án được chọn trong mỗi câu hỏi.

```
myIndex = []
for x in range(0, questions):
    arr = myPixelVal[x]
    myIndexVal = np.where(arr == np.amax(arr))
    myIndex.append(myIndexVal[0][0])
```

*Kết quả:*

```
arr [ 4248. 10269.  2689.  2946.  3302.]
myIndexVal[0]:  [1]
arr [ 2878.  2301. 10670.  2251.  2620.]
myIndexVal[0]:  [2]
arr [9553. 2233. 2019. 2224. 2643.]
myIndexVal[0]:  [0]
arr [6420. 2167. 1946. 2199. 2672.]
myIndexVal[0]:  [0]
arr [3231. 2719. 2506. 2677. 6349.]
myIndexVal[0]:  [4]
myIndex:  [1, 2, 0, 0, 4]
```

### 7. Tính điểm số

```
grading = []
for x in range(0, questions):
    if ans[x] == myIndex[x]:
        grading.append(1)
    else:
        grading.append(0)
score = (sum(grading) / questions) * 10
```

*Kết quả:*

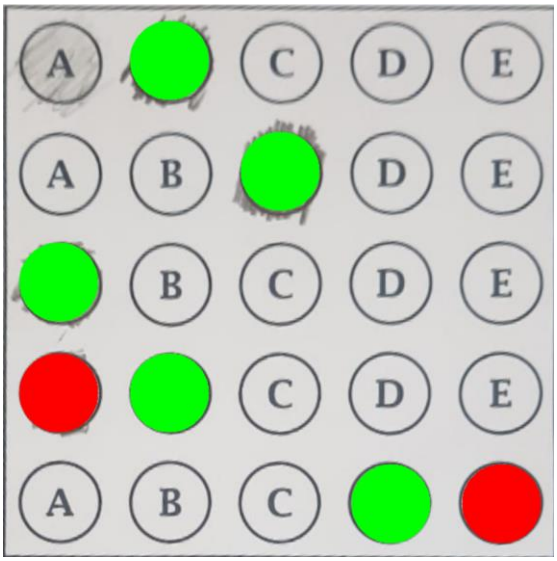
```
grading:  [1, 1, 1, 0, 0]
score:  6.0
```

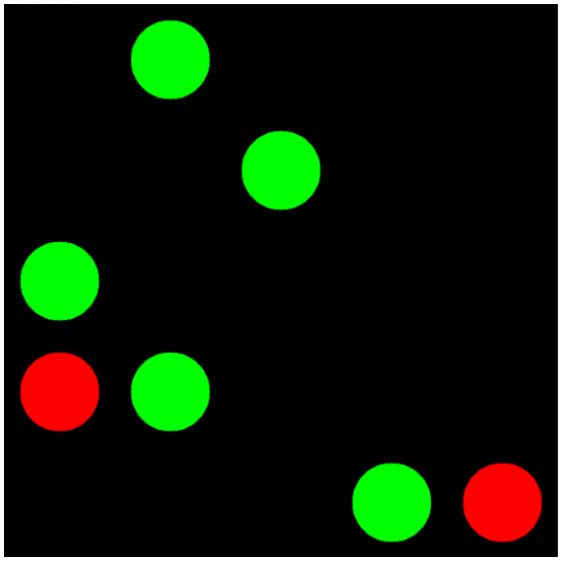
8. **Hiển thị kết quả**

- **Đánh dấu đáp án vào ô đáp án**

```
imgResult = imgWrapColored.copy()
imgResult = utlis.showAnswers(imgResult, myIndex,
grading, ans, questions, choices)
imRawDrawing = np.zeros_like(imgWrapColored)
imRawDrawing = utlis.showAnswers(imRawDrawing, myIndex,
grading, ans, questions, choices)
```

*Kết quả:*



- **Đánh điểm vào ô điểm**

```
imgRawGrade = np.zeros_like(imgGradeDisplay)
text = str(int(score))
font = cv2.FONT_HERSHEY_COMPLEX
fontScale = 3
thinkness = 3
textSize, _ = cv2.getTextSize(text, font, fontScale,
thinkness)
textX = int((imgRawGrade.shape[1] - textSize[0]) / 2)
textY = int((imgRawGrade.shape[0] + textSize[1]) / 2)
cv2.putText(imgRawGrade, text, (textX, textY), font,
fontScale, (0, 255, 255), thinkness)
```

*Kết quả:*


**9. Biến đổi lại góc nhìn**

- Tính ma trận biến đổi góc nhìn

```
invMatrix = cv2.getPerspectiveTransform(pt2, pt1)
invMatrixG = cv2.getPerspectiveTransform(pt2G, pt1G)
```

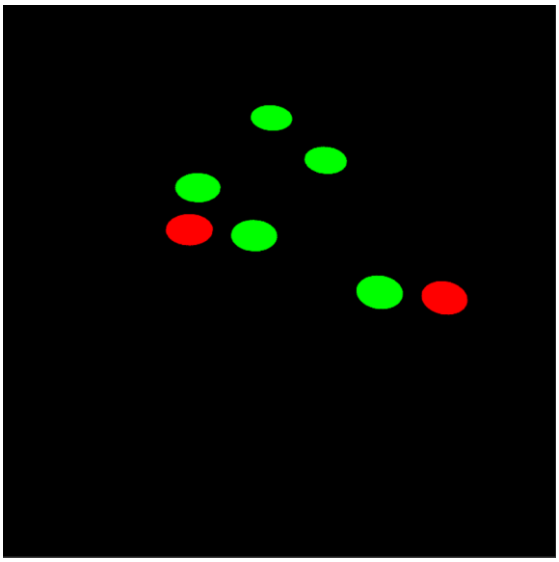
*Kết quả:*

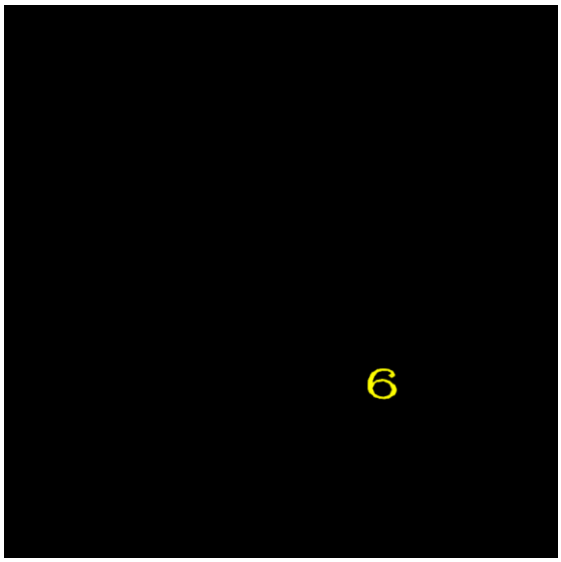
```
invMatrix:  [[ 5.32659144e-01 -1.26619935e-01  2.34000000e+02]
 [ 5.84931737e-02  2.83251813e-01  1.10000000e+02]
 [ 3.83403457e-05 -2.35089938e-04  1.00000000e+00]]
invMatrixG:  [[ 7.98517009e-01 -1.85400489e-01  3.54000000e+02]
 [ 9.47992315e-02  3.80928551e-01  4.26000000e+02]
 [ 7.45757174e-05 -3.63479679e-04  1.00000000e+00]]
```

- Áp dụng biến đổi góc nhìn

```
imgInvWarp = cv2.warpPerspective(imRawDrawing,
invMatrix, (widthImg, heightImg))
imgInvGradeDisplay = cv2.warpPerspective(imgRawGrade,
invMatrixG, (widthImg, heightImg))
```

*Kết quả:*

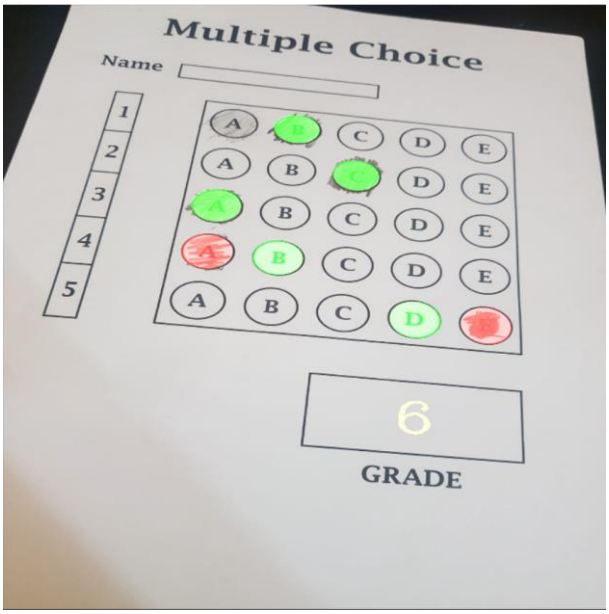




### 10. Final

```
imgFinal = cv2.addWeighted(imgFinal, 1, imgInvWarp, 1,
0)
imgFinal = cv2.addWeighted(imgFinal, 1,
imgInvGradeDisplay, 1, 0)
```

*Kết quả:*




## IV. Kết quả đánh giá

1. **Phụ thuộc vào độ chính xác của đánh dấu:** OMR đòi hỏi đánh dấu chính xác và đồng nhất trên các ô trống hoặc dòng đánh dấu. Nếu có sai sót trong quá trình đánh dấu, chẳng hạn như dấu đánh không đủ đậm, đánh dấu trên ô không đúng, OMR có thể gặp khó khăn trong việc nhận diện và phân tích đúng thông tin.
2. **Số lượng ô giới hạn:** Các hệ thống OMR thường có số lượng ô giới hạn mà nó có thể nhận diện trong một tài liệu. Điều này hạn chế số lượng câu hỏi hoặc thông tin có thể được xử lý trong một tài liệu. Nếu có quá nhiều ô hoặc thông tin cần xử lý, OMR có thể không đáp ứng được yêu cầu.
3. **Độ phức tạp của mẫu:** OMR đòi hỏi các mẫu tài liệu được thiết kế và in ấn đúng quy cách và tiêu chuẩn. Nếu có sự thay đổi trong kích thước, kiểu chữ, độ phân giải hoặc sự biến đổi khác trong tài liệu, OMR có thể gặp khó khăn trong việc nhận diện đúng thông tin.
4. **Độ tin cậy:** Mặc dù OMR có thể đạt được độ chính xác cao, nhưng nó vẫn có thể gặp sai sót trong quá trình nhận diện và phân tích. Các yếu tố như

điều kiện ánh sáng, chất lượng in ấn, vết bẩn, nếp gấp, hoặc nhiễu có thể ảnh hưởng đến kết quả của OMR.

5. **Giới hạn đối với các loại câu hỏi:** OMR thường chỉ áp dụng cho các loại câu hỏi có các lựa chọn đáp án được đúng hoặc sai, hoặc các câu hỏi có các lựa chọn đáp án rời rạc. OMR không phù hợp cho các loại câu hỏi yêu cầu câu trả lời mở, văn bản dài, hay các dạng câu hỏi phức tạp khác.
6. **Khả năng xử lý tài liệu đa dạng:** OMR có thể gặp khó khăn trong việc xử lý tài liệu có định dạng phức tạp, như tài liệu có đồ họa, biểu đồ, hay hình ảnh. OMR tập trung chủ yếu vào việc nhận diện và phân tích các thông tin đánh dấu, do đó không phù hợp cho việc xử lý tài liệu có nhiều yếu tố khác ngoài đánh dấu.
7. **Đòi hỏi thiết bị và phần mềm đặc biệt:** Để triển khai hệ thống OMR, cần có thiết bị quét đặc biệt hoặc máy đọc OMR, cùng với phần mềm nhận diện và phân tích. Điều này có thể tạo ra chi phí đáng kể và đòi hỏi sự đầu tư và bảo trì thêm.
8. **Thời gian xử lý:** Dù OMR có thể hoạt động nhanh chóng trong việc nhận diện và phân tích, thời gian xử lý vẫn phụ thuộc vào kích thước và phức tạp của tài liệu, số lượng ô cần xử lý, và hiệu suất của hệ thống OMR. Đối với tài liệu lớn và phức tạp, thời gian xử lý có thể tăng lên đáng kể.
9. **Phụ thuộc vào đánh dấu chính xác:** Để đảm bảo kết quả chính xác, OMR yêu cầu việc đánh dấu đúng và chính xác trên các ô trống hoặc dòng đánh dấu. Nếu người sử dụng không đánh dấu đúng cách hoặc có sai sót, OMR có thể không nhận diện đúng thông tin.

**Nhận xét phân chia công việc:**

| Họ và tên | Mã sinh viên | Công việc | Đánh giá |
|---|---|---|---|
| Cung Văn Thắng* | 21020939 | Lên ý tưởng, Viết Code, Viết báo cáo, Thiết kế PowerPoint, Thuyết trình | 10 |
| Đinh Thế An | 21020039 | Hỗ trợ làm báo cáo | 7 |
| Nguyễn Sỹ Hùng | 20020416 | Hỗ trợ làm báo cáo | 7 |
| Phan Công Tiến | 20020266 | Hỗ trợ làm báo cáo, viết code | 8 |

| | | | |
|---|---|---|---|
| Nguyễn Xuân Huy | 17021266 | Hỗ trợ làm báo cáo, viết code | 8 |