**TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI**

**VIỆN CƠ KHÍ**

**Bộ môn Cơ điện tử**

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**A picture containing logo

Description automatically generated**

**BÀI TẬP LỚN**

**XỬ LÝ ẢNH**

**ĐỀ TÀI:** **Máy quét trắc nghiệm - chấm điểm kiểm tra sử dụng**

**Python và OpenCV**

Giảng viên hướng dẫn: TS. Nguyễn Xuân Hạ

TS. Mạc Thị Thoa

Sinh viên thực hiện: Phạm Quang Long 20184532

Trần Quang Huy 20184489

MỞ ĐẦU

Thi trắc nghiệm là hình thức thi khá phổ biến trong vài năm trở lại đây. Đó là phương pháp dùng để kiểm tra nhanh kiến thức, kỹ năng của người nào đó qua các câu hỏi đúng sai, câu hỏi lựa chọn đáp án A, B, C, D để đánh giá.

Với những ưu điểm nổi bật của việc thi trắc nghiệm là đánh giá kiến thức một cách nhanh và toàn diện. Do bài thi trắc nghiệm có các đáp án cố định nên giúp việc chấm thi trở nên khách quan, trung thực hơn, kết quả bài thi sẽ không phụ thuộc vào yếu tố mang tính chất chủ quan của người chấm. Chuyển sang hình thức thi trắc nghiệm sẽ giúp tiết kiệm được tối đa thời gian và chi phí, kể cả thời gian thi, coi thi và chấm thi, từ đó giảm được tối đa chi phí cho quá trình kiểm tra, thi cử.

Ngoài ra, một lợi ích khác không thể không kể đến đó chính là khả năng tự động hóa việc chấm bài thi. Việc chấm thi bằng máy sẽ cho kết quả nhanh, chính xác đồng thời cũng giảm áp lực cho con người.

Chính vì những lý do như thế này mà chúng em đã chọn đề tài: “**Máy quét trắc nghiệm - chấm điểm kiểm tra sử dụng Python và OpenCV”** với mục tiêu tự động hóa quá trình chấm bài thi trắc nghiệm một cách nhanh và chính xác nhất.

CHƯƠNG 1: CƠ SỞ LÝ THUYẾT

1.1 - **Những khái niệm cơ bản về ảnh kỹ thuật số**

1.1.1 - Pixel

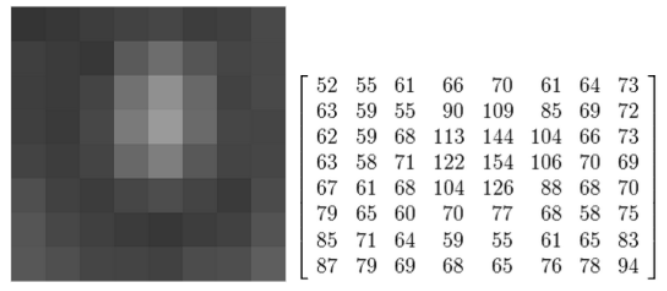
Kỹ thuật số là một kỹ thuật xử lý rời rạc trên máy tính, kỹ thuật này chỉ áp dụng đối với các bit (các con số), chính vì vậy các bức ảnh liên tục về không gian và độ sáng cần phải được “số hóa”. Số hoá ảnh thực chất là sự biến đổi gần đúng một ảnh liên tục thành một tập điểm rời rạc phù hợp với ảnh thật về vị trí (không gian) và độ sáng (mức xám). Khoảng cách giữa các điểm ảnh đó được thiết lập sao cho mắt người không thể phân biệt được ranh giới giữa chúng. Chính vì vậy, khi chúng ta nhìn một bức tranh là tập hợp của rất nhiều điểm rời rạc, chúng ta sẽ có cảm giác đó là một bức ảnh liên tục. Mỗi một điểm như vậy gọi là một điểm ảnh hay gọi tắt là Pixel. Trong khuôn khổ ảnh hsai chiều, mỗi pixel ứng với cặp tọa độ (x,y). Trong giới hạn của môn học, một Pixel là một phần tử của ảnh số tại toạ độ (x, y) với độ xám hoặc màu nhất định.

1.1.2 - Cường độ của pixel (Intensity)

Một ảnh được định nghĩa là một hàm hai chiều f (x, y) và biên độ của f tại bất kỳ cặp tọa độ nào (x, y) được gọi là cường độ hay mức độ xám của ảnh tại điểm đó. Cường độ của một pixel thường được biểu diễn trên dải giá trị từ 0 tới 255.

1.1.3 - Ảnh kỹ thuật số

Trong qua trình sử dụng các thư viện liên quan tới xử lý ảnh, chúng ta coi ảnh là một ma trận của cường độ. Các thư viện từ đó sẽ lưu trữ và xử lý ảnh qua các ma trận đó.



*Hình 1.1: Một hình ảnh được biểu diễn dưới dạng ma trận.*

1.2 - **Những khái niệm cơ bản về xử lý ảnh**

1.2.1 - Phân đoạn ảnh

Phân đoạn ảnh là quá trình phân vùng ảnh kỹ thuật số thành nhiều đoạn ảnh, còn được gọi là vùng ảnh hoặc đối tượng ảnh. Mục tiêu của phân đoạn là đơn giản hóa và hoặc thay đổi hình ảnh đại diện thành một thứ có ý nghĩa hơn và dễ phân tích hơn. Phân đoạn hình ảnh thường được sử dụng để định vị các đối tượng và ranh giới (đường thẳng, đường cong, v.v.) trong hình ảnh. Chính xác hơn, phân đoạn hình ảnh là quá trình gán nhãn cho mọi pixel trong một hình ảnh sao cho các pixel có cùng nhãn chia sẻ các đặc điểm nhất định.

1.2.2 - Chuyển đổi phối cảnh

Như tên gọi, phép biến đổi phối cảnh có liên quan đến sự thay đổi góc nhìn. Kiểu biến đổi này không bảo tồn sự song song, chiều dài và góc. Nhưng các đường thẳng sẽ vẫn thẳng ngay cả sau khi chuyển đổi.

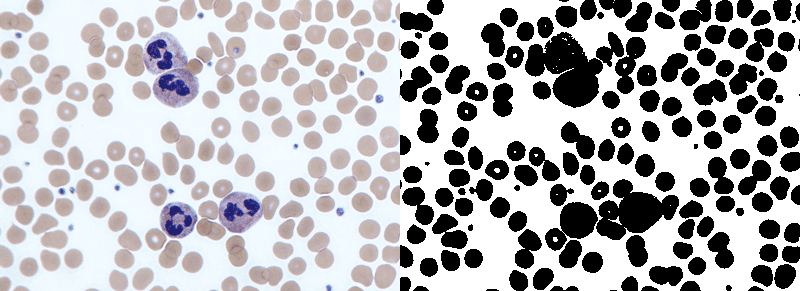
Diagram

Description automatically generated

*Hình 1.2: Một phép biến đổi dưới dạng ma trận.*

1.2.3 - “Nhị phân hóa” (Binarization)

Binarization, hay “Nhị phân hóa”, là phương pháp chuyển đổi bất kỳ ảnh xám (ảnh nhiều tông màu) thành ảnh đen trắng (ảnh hai tông màu). Để thực hiện quy trình mã hóa nhị phân, trước tiên chúng ta chọn một giá trị ngưỡng (threshold) của thang màu xám và kiểm tra xem giá trị xám của pixel. Nếu giá trị màu xám của các pixel lớn hơn ngưỡng, thì các pixel đó sẽ được chuyển thành màu trắng. Tương tự, nếu giá trị màu xám của các pixel nhỏ hơn ngưỡng, thì các pixel đó sẽ được chuyển đổi thành màu đen.



*Hình 1.3: Ví dụ về một bức ảnh được “Nhị phân hóa”.*

Hãy xem xét một hình ảnh chỉ có hai giá trị hình ảnh riêng biệt, điều đó có nghĩa là biểu đồ tần suất trên dải giá trị xám sẽ chỉ bao gồm hai đỉnh. Một ngưỡng tốt sẽ nằm ở giữa hai giá trị đó. Đây chính là cách phương pháp Otsu xác định một giá trị ngưỡng toàn cầu tối ưu từ biểu đồ hình ảnh.

1.2.4 - Viền (Contour) của ảnh

Đường viền trong hình ảnh là một đường viền trên các đối tượng hiện diện trong hình ảnh. Chi tiết hơn, đường viền có thể được giải thích là một đường cong nối tất cả các điểm pixel liên tục (dọc theo đường biên), có cùng màu hoặc cường độ.

Để phát hiện các cạnh của hình ảnh đầu vào, chúng ta sẽ tìm các điểm mà có sự khác biệt đáng kể về cường độ màu, sau đó chỉ cần “bật” các pixel đó lên. Phát hiện đường viền hoạt động tương tự như phát hiện cạnh nhưng với hạn chế là các cạnh được phát hiện phải tạo thành một đường khép kín.

1.2.3 - Làm mờ Gaussian

Làm mờ là làm cho ảnh kém rõ ràng hoặc khác biệt hơn. Điều này có thể được hiểu khá rộng trong bối cảnh phân tích hình ảnh - bất kỳ điều gì làm giảm hoặc làm sai lệch chi tiết của hình ảnh đều có thể áp dụng. Việc áp dụng bộ lọc thông thấp, loại bỏ chi tiết xảy ra ở tần số không gian cao, được coi là hiệu ứng làm mờ. Làm mờ Gaussian là một bộ lọc sử dụng thuật toán Gaussian để tính toán sự chuyển đổi áp dụng cho từng pixel trong hình ảnh.

Các giá trị từ phân phối Gaussian được sử dụng để xây dựng ma trận tích chập áp dụng cho hình ảnh ban đầu. Quá trình tích chập này được minh họa trực quan trong hình dưới. Giá trị mới của mỗi pixel được đặt thành giá trị trung bình có trọng số của vùng lân cận pixel đó. Giá trị của pixel gốc nhận được trọng lượng nặng nhất (có giá trị Gaussian cao nhất) và các pixel lân cận nhận được trọng số nhỏ hơn khi khoảng cách của chúng đến pixel gốc tăng lên. Điều này dẫn đến hiệu ứng làm mờ giúp bảo toàn ranh giới của các cạnh tốt hơn và cũng đồng nhất hơn các bộ lọc làm mờ khác.



*Hình 1.3: Ví dụ về một bức ảnh được làm mờ với Gaussian.*

1.2.3 - Thuật toán phát hiện cạnh Canny

Thuật toán phát hiện cạnh Canny là một toán tử phát hiện cạnh sử dụng thuật toán nhiều giai đoạn để phát hiện một loạt các cạnh trong hình ảnh. Quy trình của thuật toán phát hiện cạnh Canny có thể được chia thành năm bước khác nhau:

1. Áp dụng bộ lọc Gaussian để làm mịn hình ảnh nhằm loại bỏ nhiễu.
2. Tìm độ dốc cường độ (intensity gradients) của hình ảnh.
3. Áp dụng ngưỡng cường độ gradient hoặc triệt tiêu giới hạn dưới để loại bỏ cạnh giả.
4. Áp dụng ngưỡng kép để xác định các cạnh tiềm năng .
5. Hoàn thiện việc phát hiện các cạnh bằng cách loại bỏ tất cả các cạnh khác “yếu” và không được kết nối với các cạnh “mạnh”.

Chúng ta nhận thấy rằng một số cạnh sáng hơn những cạnh khác. Những cạnh sáng, dày hơn có thể được coi là cạnh “mạnh” nhưng những cạnh mảnh, nhạt hơn thực sự cũng có thể là cạnh hoặc chúng có thể là do nhiễu. Để giải quyết vấn đề “cạnh nào thực sự là cạnh và cạnh nào không”, Canny sử dụng ngưỡng trễ (Hysteresis Thresholding). Trong trường hợp này, chúng ta đặt hai ngưỡng cường độ (Intensity) là "Cao" và "Thấp":

1. Bất kỳ cạnh nào có cường độ lớn hơn ‘Cao’ là các cạnh “chắc chắn”.
2. Bất kỳ cạnh nào có cường độ nhỏ hơn ‘Thấp’ chắc chắn không phải là cạnh.
3. Các cạnh giữa ngưỡng "Cao" và "Thấp" chỉ được phân loại là các cạnh nếu chúng được kết nối với một cạnh “chắc chắn”, nếu không thì sẽ bị loại bỏ.

1.2.4 - Moment của ảnh

Trong xử lý ảnh, moment của ảnh được dùng để nêu bật đặc trưng hình dạng của một ảnh. Những moment này ghi lại những thuộc tính của hình ảnh, bao gồm diện tích của đối tượng (the area), trọng tâm (centroid), hướng (orientation) và những thuộc tính liên quan khác.

CHƯƠNG 2: PHÂN TÍCH HỆ THỐNG

2.1 - **Tổng quan về hệ thống**

Hệ thống sẽ bao gồm 2 thành phần chính:

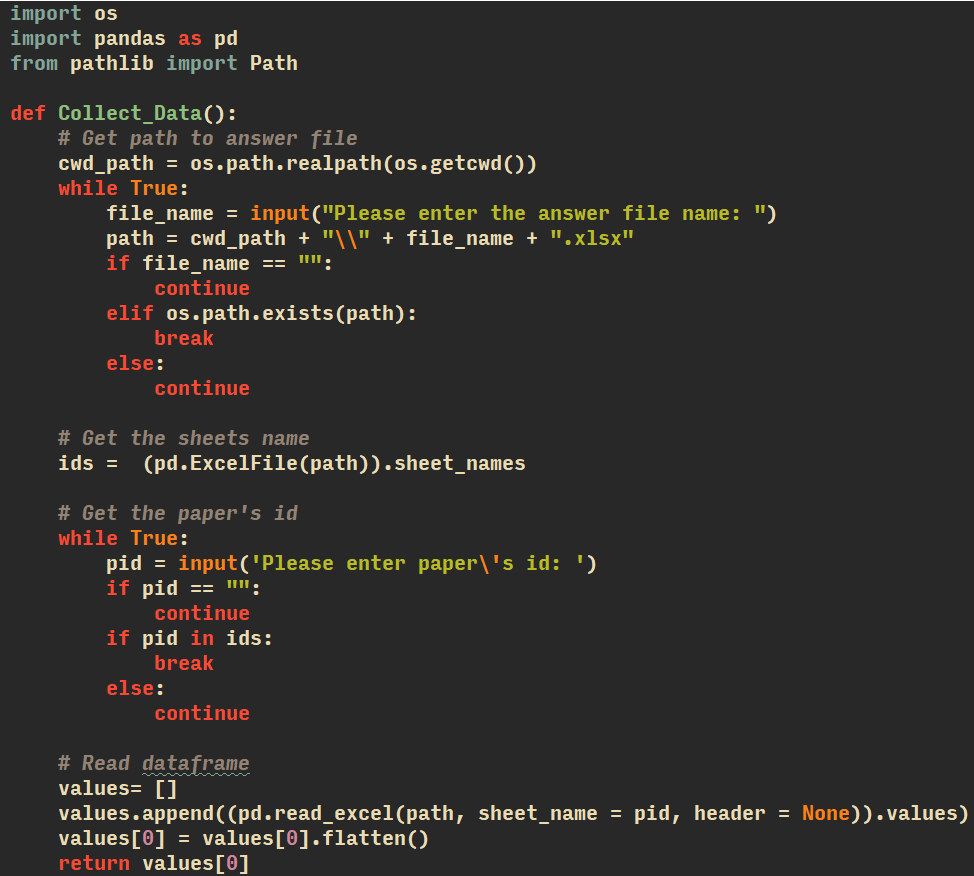
+ Phần cứng: Phương tiện dùng để thu được ảnh bài thi nhằm xử lý. Đó có thể là máy ảnh cầm tay, camera của điện thoại… sao cho ảnh nhận được có độ phân giải đủ lớn để phần mềm xử lý.

+ Phần mềm: Bao gồm những chương trình được viết bằng python kèm theo những thư viện như open cv, imutils… Những file trong phần mềm là:

* Một file excel chứa đáp án. Mỗi một trang tính tương đương với đáp án của một mã đề.
* File collect\_data.py dùng để đọc file excel và trả về một list đáp án tùy vào mã đề.
* File transform.py chứa những hàm dùng để biến đổi để thu được góc nhìn từ trên xuống.
* File scan.py dùng để quét ảnh chụp từ đó lấy phần bài làm trong bức ảnh và sau đó biến đổi ảnh sang ảnh xám (gray scale).
* File test\_grader.py là file dùng để xác định đáp án thí sinh chọn và đáp án đó là đúng hay sai. Từ đó tính toán được điểm của bài thi.

*Hình 2.1: Pipeline của hệ thống.*

2.3 - **Module collect\_data.py**



*Hình 2.2: Module collect\_data.py.*

Module này được tạo ra nhằm mục đích đọc file excel chứa đáp án cho bài kiểm tra. File excel được sử dụng có dạng như hình bên dưới. Trong đó, đáp án đúng được đánh số từ 0 đến 3 tương ứng với A, B, C, D. Ngoài ra các mã đề sẽ được phân loại theo số để thuận tiện.

A picture containing table

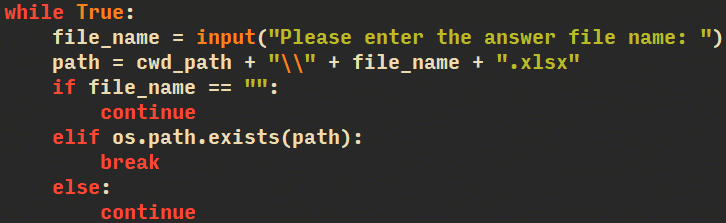
Description automatically generated

*Hình 2.3: File excel ví dụ.*

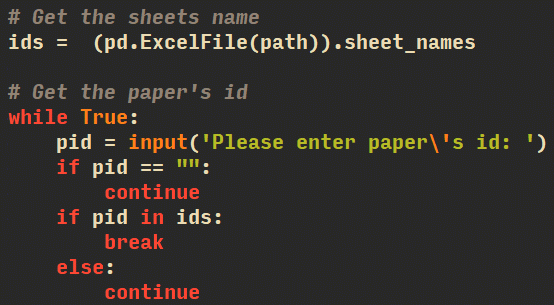
Module này chứa duy nhất một hàm là **Collect\_Data**. Đầu tiên, chúng ta sẽ lấy địa chỉ mà module được gọi (Current Working Directory), từ đó chúng ta xác định vị trí của file excel (Trong trường hợp này, coi mọi module và file liên quan ở trong một folder để thuận tiện cho việc lập trình).

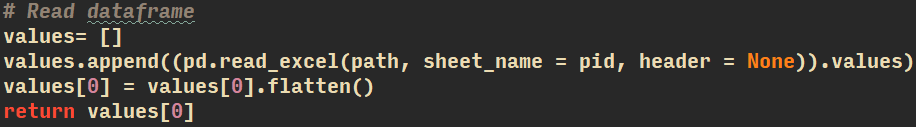


Tiếp theo, chúng ta để người dùng nhập tên của file excel và kiểm tra qua một vòng lặp loop. Nếu tên file tồn tại, chương trình sẽ tiếp tục. Ngược lại, vòng lặp sẽ lặp lại và yêu cầu người dùng nhập tên file thỏa mãn.



Chúng ta cũng làm tương tự như vậy khi chọn mã đề (hay nói cách khác là chọn sheet khác trong file excel). Điều này có thể được thực hiện nhờ method *.sheet\_names* của thư viện pandas.



 Cuối cùng, chúng ta đọc file excel với id và địa chỉ tìm được phía trên. Sau đó, chúng ta biến đổi giá trị đọc được thành một list theo thứ tự từ trái sang phải, từ trên xuống dưới để thuận tiện hơn cho quá trình về sau.

2.2 - **Module transform.py**

Mục tiêu của module này đó chính là để sắp xếp các đỉnh của một vùng ảnh mà chúng ta đã xác định là thuộc về bài làm của thí sinh. Từ đó áp dụng những thuật toán biến đổi trong xử lý ảnh để thu được góc nhìn từ trên cao xuống của vùng ảnh đó. Chương trình này sẽ sử dụng hai thư viện là open cv2 và numpy.

Text

Description automatically generated*Hình 2.2: File transform.py.*

2.2.1 - *Hàm order\_points*

A picture containing diagram

Description automatically generated Giả sử chúng ta xác định được 4 đỉnh của vùng ảnh mà chúng ta quan tâm. Tuy nhiên, sau khi thực hiện thuật toán để tìm ra 4 đỉnh đó, chúng không được ở đúng theo một thứ tự nhất định và có thể khác nhau đối với mỗi hình ảnh. Điều quan trọng là chúng ta phải có thứ tự nhất quán của các điểm. Bản thân việc đặt thứ tự thực tế có thể tùy ý, miễn là nó nhất quán trong suốt quá trình thực hiện.Hàm **order\_points** từ đó sinh ra dùng để sắp xếp các đỉnh của một vùng ảnh mà chúng ta đã xác định là thuộc về bài làm của thí sinh. Hàm có một tham số đầu vào **pts** chính là tọa độ của 4 đỉnh của vùng ảnh mà chúng ta đã xác định. Sau khi qua hàm này, 4 đỉnh đó sẽ được sắp xếp theo thứ tự xác định là “trên cùng bên trái”, “trên cùng bên phải”, “dưới cùng bên phải” và “dưới cùng bên trái” (Hình 2.3).

*Hình 2.3: Ví dụ về bốn đỉnh xác định của bài thi.*

Đầu tiên chúng ta sẽ tạo một ma trận toàn 0 **rect** dưới vai trò là một biến giữ chỗ cho những tọa độ mà sau này chúng ta sẽ sắp xếp. Ma trận **rect** này sẽ được khởi tạo dựa trên method *.zeros* của thư viện numpy. Ma trận này bao gồm 4 hàng và 2 cột, tương ứng với 4 điểm và tọa độ (x, y) của 4 điểm đó với kiểu dữ liệu là float32.



Text

Description automatically generated Tiếp theo đó chúng ta sử dụng method *.sum* trong thư viện numpy để tính tổng của các giá trị trong một ma trận dãy (array). Với parameter *axis = 1*, kết quả trả về sẽ là một list tổng của ma trận dãy theo hàng, trong trường hợp này là tổng của tọa độ x và y của một đỉnh của vùng ảnh. Do đỉnh “trên cùng bên trái” chắc chắn sẽ có tổng tọa độ nhỏ nhất và đỉnh “dưới cùng bên phải” sẽ có tổng tọa độ lớn nhất (gốc tọa độ của thư viện open cv2 nằm ở góc trái trên cùng của ảnh).

Biến **s** sẽ giữ tổng tọa độ của 4 đỉnh đó. Method argmin, argmax của thư viện numpy sẽ trả về thứ tự (index) của giá trị nhỏ nhất và lớn nhất trong list **s**. Từ đó, chúng ta gán cặp tọa độ tương ứng của biến **pts** vào ma trận giữ chỗ **rect**.

Để xác định tọa độ của 2 điểm “trên cùng bên phải” và “dưới cùng bên trái”, chúng ta sẽ sử dụng hàm diff của thư viện numpy. Hàm này sẽ trả về hiệu của 2 tọa độ (y – x). Tọa độ nào có sự khác biệt nhỏ nhất sẽ là điểm “trên cùng bên phải” và ngược lại, tọa độ nào có sự khác biệt lớn nhất sẽ là điểm “dưới cùng bên trái”.

Text

Description automatically generated

Hàm trả về biến **rect** là một ma trận dãy chứa 4 điểm đã được sắp xếp theo thứ tự đã quy định.

2.2.2 - *Hàm four\_point\_transform*

Hàm **four\_point\_transform** có hai parameters là **image** và **pts**. Parameter **image** chứa bức ảnh chúng ta sẽ biến đổi và tương tự, **pts** sẽ là một list bao gồm 4 điểm chứa vùng ảnh cần quan tâm (Region Of Interest - ROI) mà chúng ta sẽ biến đổi.

Đầu tiên, chúng ta gọi lại hàm **order\_points** để sắp xếp lại các điểm. Sau đó, chúng ta gán 4 điểm tọa độ trên vào 4 biến tương ứng để cho việc tính toán về sau được thuận lợi hơn:

+ **tl** (Top-left) tương ứng tọa độ “trên cùng bên trái”,

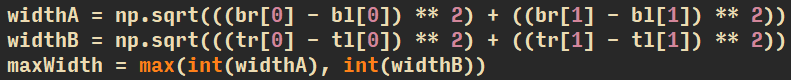
+ **tr** (Top-tight) tương ứng tọa độ “trên cùng bên phải”,

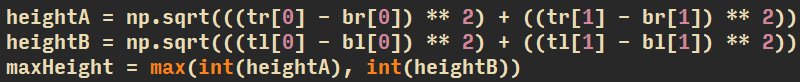
+ **br** (Bottom-right) tương ứng tọa độ “dưới cùng bên phải”,

+ **bl** (Bottom-left) tương ứng tọa độ “dưới cùng bên trái”.



Bây giờ chúng ta cần xác định kích thước của hình ảnh mới của chúng ta. Chiều rộng là khoảng cách lớn nhất giữa tọa độ x dưới cùng bên phải và dưới cùng bên trái hoặc tọa độ x trên cùng bên phải và trên cùng bên trái. Chúng ta sẽ xử dụng công thức Pythagoras để tính trong trường hợp này. Chiều rộng của bức ảnh mới sẽ được gán vào biến **maxWidth**.



Tương tự như vậy chúng ta tính được chiều cao của bức ảnh và gán vào biến **maxHeight**.

Tiếp theo, chúng ta xác định 4 điểm đại diện cho chế độ xem “từ trên xuống”. Tọa độ của điểm “trên cùng bên phải”, “dưới cùng bên phải” và “dưới cùng bên trái” phải trừ đi 1 để nhường 1 pixel cho tọa độ (0, 0) của điểm “trên cùng bên trái”.

Text

Description automatically generated

Để thực sự có được chế độ xem “từ trên xuống” của hình ảnh, chúng ta sẽ sử dụng hàm .*getPerspectiveTransform* của thư viện open cv2. Hàm này sẽ trả về một ma trận biến đổi góc nhìn để chúng ta có thể áp dụng vào bức ảnh gốc, từ đó lấy được vùng ảnh cần quan tâm. Hàm này yêu cầu hai parameter, parameter đầu tiên (được gán là biến **rect**) là ma trận dãy bao gồm tọa độ của 4 điểm chứa vùng ảnh cần quan tâm trong bức ảnh gốc, parameter thứ hai (được gán là biến **dst**) là ma trận dãy bao gồm tọa độ của các điểm sau biến đổi.

Sau đó, chúng ta áp dụng ma trận biến đổi lên hình ảnh gốc nhờ vào hàm *.warpPerspective* cũng của thư viện open cv2. Chúng ta đưa hình ảnh gốc làm parameter đầu tiên, ma trận biến đổi M là parameter thứ hai và cuối cùng chúng ta xác định kích thước của ảnh output.

Text

Description automatically generated

2.2 - **Module scan.py**

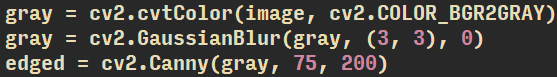
Mục tiêu của file này là quét ảnh chụp (từ người dùng), sau đó lấy phần bài làm trong bức ảnh và cuối cùng là biến đổi ảnh sang ảnh xám (gray scale). Việc quét tài liệu có thể được chia thành ba bước riêng biệt và đơn giản:

1. Bước đầu tiên là áp dụng tính năng phát hiện cạnh.
2. Bước thứ hai là tìm các đường viền trong hình ảnh đại diện cho tài liệu mà chúng ta muốn quét.
3. Và bước cuối cùng là áp dụng phép biến đổi phối cảnh để có được hình ảnh từ trên xuống (sử dụng chương trình transform.py chúng ta đã viết ở trên) và chuyển đổi ảnh sang ảnh xám.

Chương trình này sẽ sử dụng những thư viện sau: open cv2, numpy và imutils. Đây sẽ là chương trình chính trong hệ thống của chúng ta. Imutils là một thư viện bao gồm một loạt các chức năng tiện lợi hơn cho OpenCV. Imutils sẽ giúp chúng ta thực hiện các tác vụ cơ bản như dịch, xoay, thay đổi kích thước và tạo khung nhanh hơn với OpenCV. Ngoài ra, chúng ta cũng sử dụng chương trình transform như một module tự tạo và cuối cùng là hàm threshold\_local của thư viện skimage.

Đầu tiên, như thường lệ, chúng ta sẽ đọc ảnh đầu vào và gán vào biến **image**. Sau đó chúng ta sẽ biến đổi ảnh thành ảnh xám và gán vào biến **gray** nhờ vào hàm *.cvtColor* của open cv2. Tiếp đó chúng ta thực hiện làm mờ với phương thức Gaussian để loại bỏ nhiễu ở tần số cao. Hàm chúng ta sử dụng tương ứng ở đây là hàm *.GaussianBlur* với tham số thứ nhất là ảnh làm input. Tiếp theo sau đó là kích thước của hạt nhân (Kernal) mà chúng ta sẽ dùng để làm mờ. Trong trường hợp này, chúng ta chọn là 3x3 do kích thước này cho kết quả tốt nhất. Tham số cuối cùng chính là giá trị Độ lệch chuẩn. Giá trị này được đặt là 0 để thuật toán tự tính độ lệch chuẩn theo kích thước hạt nhân.

Ngay sau đó chúng ta sẽ phát hiện cạnh, sử dụng hàm *.Canny*. Tương tự như những hàm khác, tham số đầu vào đầu tiên vẫn là ảnh input. Hai tham số tiếp theo lần lượt là ngưỡng thứ nhất và thứ hai của thuật toán Canny. Chúng ta chọn lần lượt là 75 và 200. Kết quả trả về sẽ được gán vào biến **edged**.



Khi sử dụng thuật toán Canny, cạnh trả về sẽ rất mỏng, đặc biệt là phần cạnh giấy của bài kiểm tra. Chính vì vậy, chúng ta sẽ sử dụng thuật toán “Hoạt động hình thái” (Morphological Operations) để làn “nở” các cạnh tìm được, đặc biệt là đường viền. Thuật toán này bao gồm việc trượt một nhân (kernal) B qua một ảnh A. Hạt nhân này có thể có bất kỳ hình dạng hoặc kích thước nào, thường là hình vuông hoặc hình tròn. Hạt nhân B có một điểm neo xác định, thường là trung tâm của hạt nhân. Khi hạt nhân B được quét qua hình ảnh, chúng ta dựa vào giá trị pixel lớn nhất được phủ bởi B và thay thế pixel hình ảnh ở vị trí điểm neo bằng giá trị tối đa đó. Hoạt động tối đa hóa này từ đó sẽ làm cho các vùng sáng trong hình ảnh giãn nở. Hàm *.dilate* sẽ được sử dụng để làm “nở” ảnh. Chúng ta sẽ khởi tạo một mặt nạ vuông đại diện bởi ma trận 1 với kích thước là một số lẻ. Hàm *.dilate* sẽ nhận bức ảnh là đầu vào ở tham số đầu tiên và mặt nạ nhân là tham số thứ hai. Chúng ta sẽ gán ảnh nhận được sau xử lý vào lại biến **edged**.

Text

Description automatically generated

Để tìm viền, chúng ta sẽ sử dụng hàm *.findContours*. Tham số đầu tiên sẽ là mảng hình ảnh nguồn. Tham số tiếp theo để xác định Chế độ truy xuất đường viền (Contour retrieval mode) và cuối cùng là Phương pháp xấp xỉ đường viền (Contour approximation method).

Đôi khi các viền ở các vị trí khác nhau. Nhưng trong một số trường hợp, một số viền nằm bên trong các viền khác. Trong trường hợp này, chúng ta gọi viền bên ngoài là “bố mẹ” và bên trong là “con”. Bằng cách này, các đường viền trong một hình ảnh có một số mối quan hệ với nhau. Và chúng ta có thể chỉ định cách một đường bao được kết nối với nhau như thế nào. Khi chúng ta sử dụng chế độ *RETR\_EXTERNAL*, chúng ta sẽ chỉ lấy ra các đường viền bên ngoài nhất.

Xấp xỉ đường bao, nhằm mục đích đơn giản hóa một đường đa tuyến bằng cách giảm các đỉnh của nó bởi một giá trị ngưỡng. Nói cách khác, chúng ta lấy một đường cong và giảm số đỉnh của nó trong khi vẫn giữ nguyên hình dạng của nó. Cho một điểm đầu và điểm cuối của một đường cong đa tuyến, trước tiên, thuật toán sẽ tìm đỉnh ở khoảng cách tối đa từ đường nối hai điểm tham chiếu. Hãy gọi nó là max\_point. Nếu max\_point   
nằm ở khoảng cách nhỏ hơn ngưỡng, thuật toán sẽ tự động bỏ qua tất cả các đỉnh giữa điểm đầu và điểm cuối và làm cho đường cong thành một đường thẳng. Nếu max\_point lớn hơn ngưỡng, thuật toán sẽ tìm một điểm khác và lặp lại. Trong trường hợp này chúng ta chọn kiểu *CHAIN\_APPROX\_SIMPLE*.



Tùy vào phiên bản open cv mà chúng ta sử dụng, hàm *.findContours* sẽ trả về 2 hoặc 3 giá trị khi chúng ta gán vào biến **cnts**. Nhằm tránh trường hợp chương trình bị lỗi khi lấy giá trị từ hàm đó, chúng ta sẽ sử dụng hàm *.grab\_contours* của thư viện imutils. Thư viện này sẽ tự động xác định giá trị nào trả về sẽ là viền tìm được và gán lại vào **cnts**.



Để xác định được bài kiểm tra, chúng ta cần xác định phân vùng có kích thước lớn nhất và có dạng hình chữ nhật. Chính vì vậy, việc đầu tiên chúng ta cần làm là tìm ra những phân vùng có kích thước lớn nhất. Điều này có thể được thực hiện nhờ hàm *sorted()*. Với tham số thứ là hàm đóng vai trò như một khóa để so sánh sắp xếp, nghĩa là các giá trị cần sắp xếp sẽ được đưa qua hàm này sau đó mới so sánh dựa vào giá trị trả về. Tham số thứ ba được gán bằng true để các giá trị được sắp xếp từ lớn đến bé. Cuối cùng, chúng ta sẽ chỉ lấy 3 giá trị lớn nhất.



Với 3 vùng ảnh chúng ta vừa tìm được, chúng ta sẽ lặp cho tới khi nào tìm được vùng ảnh hình chữ nhật đầu tiên. Hàm quan trọng nhất chúng ta cần quan tâm ở đây chính là hàm *.approxPolyDP* của thư viện open cv. Hàm *.approxPolyDP* sẽ “xấp xỉ” một đường cong hoặc một đa giác bằng một đường cong / đa giác khác có ít đỉnh hơn để khoảng cách giữa chúng nhỏ hơn hoặc bằng độ chính xác được chỉ định. Hàm này sẽ trả về những tọa độ đại diện cho hình đó, nghĩa là những tọa độ (hay đỉnh) khi nối vào sẽ cho ra vùng ảnh “xấp xỉ” vùng ảnh ban đầu. Tham số thứ hai xác định độ chính xác gần đúng. Đây là khoảng cách lớn nhất giữa đường cong gốc và giá trị gần đúng của nó. Để xác định giá trị cho tham số này, chúng ta cần tính chu vi của vùng ảnh với hàm *.arcLength()* và chọn giá trị khoảng tầm 0.02 lần chu vi này. Ngoài ra, khi tham số thứ hai của hàm *.arcLength()* là True, chúng ta xác định viền này là viền đóng. Quay trở lại với hàm *.approxPolyDP*, tham số cuối cùng của hàm này cũng giống với tham số thứ hai của hàm *.arcLength()*.

Do vùng ảnh cần tìm là hình chữ nhật, số đỉnh trả về của hàm *.approxPolyDP* sẽ phải là 4 nếu vùng ảnh đó thỏa mãn. Ngay sau khi chúng ta tìm ra vùng ảnh đó, chúng ta sẽ thoát ra khỏi vòng lặp.

Text

Description automatically generated

Ngay sau khi chúng ta tìm ra vùng ảnh của bài kiểm tra, với sự giúp đỡ của module **transform** chúng ta đã viết ở trên, chúng ta sử dụng hàm *four\_point\_transform* để thu được góc nhìn từ trên xuống.



Cuối cùng, chúng ta xuất ra vùng ảnh bài kiểm tra dưới tên **input.png**.

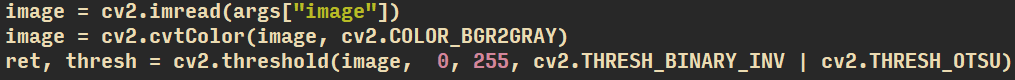


2.3 - **Module test\_grader.py**

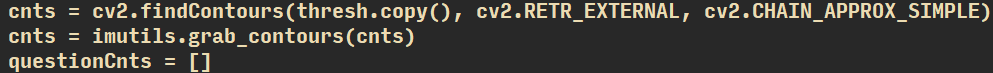
Chương trình này sẽ sử dụng những thư viện sau: open cv2, numpy và imutils. Đây sẽ là chương trình chính trong hệ thống của chúng ta.

Đầu tiên, chúng ta sẽ load ảnh đã được xử lý nhờ vào file **scan.py** và gán vào biến **image** sử dụng method *.imread* của thư viện open cv2. Sau đó, chúng ta sẽ chuyển đổi bức ảnh từ ảnh màu sang ảnh xám với method *.cvtColor*. Tiếp theo, chúng ta sẽ “nhị phân hóa” (Binarization) bức ảnh theo phương pháp Otsu để phân loại lớp ảnh phía trên và ảnh nền. Chúng ta làm việc này nhờ vào hàm *.threshold* của thư viện open cv2.

Như đã nói ở phần lý thuyết bên trên, phương pháp Otsu phân loại nhờ xác định 2 đỉnh của biểu đồ tần suất và phân ngưỡng ở giữa hai giá trị đó. Sau khi đã phân loại, tất cả các pixel nằm bên trái sẽ có cường độ là 0 và ngược lại sẽ có cường độ là 255. Hàm này trả về một bộ gồm 2 giá trị, trong đó giá trị thứ nhất là giá trị ngưỡng đã cho và giá trị thứ hai là mảng hình ảnh đã sửa đổi. Chúng ta sẽ chỉ quan tâm tới giá trị thứ hai đã được gán vào biến **thresh**.



Quá trình nhị phân hóa này sẽ cho phép chúng ta áp dụng kỹ thuật xác định đường viền để tìm từng bong bóng trong bài kiểm tra. Tương tự như trên, chúng ta sử dụng lại hàm *.findContours* để xác định viền nhưng trong trường hợp này là của các bong bóng.



Với những đường viền vừa tìm được, có những đường viền sẽ không phải là của các bong bóng. Để xác định vùng nào của hình ảnh là bong bóng, trước tiên chúng ta lặp qua từng đường viền riêng lẻ. Đối với mỗi đường viền này, chúng ta tính toán hộp giới hạn (Bounding Rectangle), điều này cũng cho phép chúng ta tính toán tỷ lệ khung hình, hoặc đơn giản hơn, tỷ lệ chiều rộng với chiều cao. Để một vùng đường viền được coi là bong bóng, vùng đó phải:

1. Đủ rộng và cao (trong trường hợp này, ít nhất 20 pixel ở cả hai chiều).
2. Có tỷ lệ khung hình gần bằng 1.

Với những đường viền thỏa mãn, chúng ta sẽ thêm nó vào biến **questionCnts** kiểu list đã được khởi tạo ở trên.

Để làm được điều đó, đầu tiên, chúng ta sử dụng hàm *.boundingrect()* để tạo một hình chữ nhật gần đúng bao quanh viền hay nói một cách khác là làm nổi bật khu vực quan tâm sau khi có được hình dạng bên ngoài của hình ảnh. Hàm này sẽ trả về 4 giá trị theo 2 cặp:

1. (x, y) là tọa độ của gốc tọa độ trên cùng bên trái của hình chữ nhật.
2. (w, h) là chiều rộng và chiều cao của nó.

Chúng ta sẽ gán chúng với tên gọi tương tự như trên. Sử dụng giá trị chiều rộng và chiều cao được trả về, chúng ta tính tỷ lệ **ar**.

Text

Description automatically generated

Áp dụng những điều kiện đã nói đến ở trên, sử dụng method *.append*, chúng ta thêm những viền thuộc về bong bóng vào biến **questionCnts.**



Tiếp theo, chúng ta sẽ sắp xếp các viền đã tìm được với sự giúp đỡ của hàm *.sort\_contours* trong thư viện imutils. Chúng ta sẽ sắp xếp chúng từ trên xuống dưới trước.



Sau đó chúng ta khởi tạo biến giữ chỗ **correct** để chứa số lượng câu đúng trong bài làm.



Để sắp xếp các viền theo thứ tự từ trái sang phải, đầu tiên chúng ta cần xác định trọng tâm của các vùng ảnh được giới hạn bởi viền. Việc đó được thực hiện bởi hàm **x\_cord\_contour**.

Text

Description automatically generated

Hàm *.moments()* cung cấp một từ điển của tất cả các giá trị moment được tính. Chúng ta sẽ chỉ tính moment cho những vùng ảnh có diện tích lớn hơn 10. Sử dụng phương trình như hình phía trên, chúng ta tính được tọa độ theo trục x của trọng tâm các vùng ảnh.

Tiếp theo, chúng ta sử dụng hàm *Collect\_Data* trong module **collect\_data** để thu được một list **correct\_answer\_list** chứa những đáp án đúng của mã đề.



Cuối cùng, chúng ta đi đến vòng lặp quan trọng nhất của chương trình.Chúng ta sẽ nhìn vào vòng lặp for đầu tiên. Ở đây chúng ta sử dụng 2 biến đếm **q** và **i**. Do bản chất chúng ta lặp từng dòng, chính vì vậy biến **q** sẽ dùng để lặp số dòng và biến **i** sẽ dùng để tượng trưng cho số bong bóng được lặp trong 1 lần (trong trường hợp này là 16). Để làm được như vậy chúng ta sẽ sử dụng hai hàm *enumerate* và hàm *.arange*. Hàm *.arange* sẽ trả về các giá trị cách đều nhau trong một khoảng nhất định. Trong trường hợp này, các giá trị sẽ bắt đầu từ 0, kết thúc bằng số giá trị của viền chúng ta tìm được và khoảng quy định ở đây là 16. Sau đó hàm *enumerate* sẽ gộp số lần lặp và giá trị của hàm *.arange* thành một cặp (tuple).



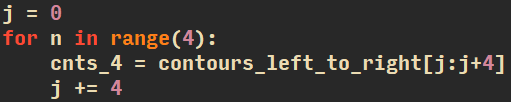
Tiếp theo đó, chúng ta lấy 16 vùng ảnh của bong bóng và gán vào biến **cnts**.



Ở đoạn code phía trên, chúng ta chỉ mới sắp xếp đoạn những viền ảnh theo thứ tự từ trên xuống dưới. Để sắp xếp chúng theo thứ tự từ trái sang phải, chúng ta sử dụng lại hàm *sorted*. Chúng ta sẽ sử dụng key chính là hàm tìm tọa độ của vùng ảnh *x\_cord­\_contour*. Dựa vào tọa độ của vùng ảnh theo trục x, chúng ta có thể sắp xếp những viền ảnh thể hiện bong bóng từ trái sang phải.



Ở vòng lặp tiếp theo, chúng ta sẽ lặp qua 4 bong bóng một lần, tương ứng với 1 câu hỏi. Chúng ta sử dụng biến đếm **j** để ghi nhớ thứ tự bong bóng chúng ta đang lặp.



Tiếp theo, chúng ta sẽ dùng một vòng lặp for khác để xác định bong bóng nào đã được tô đen. Trong trường hợp này, chúng ta sử dụng 2 biến khác là **k** và **l**. Biến đếm **k** sẽ lưu số thứ tự của bong bóng trong bộ 4 bong bóng đang xét và biến **l** sẽ dùng để chứa vùng ảnh của bong bóng chúng ta sẽ xét trong vòng lặp.



Sau đó, chúng ta tạo một mặt nạ để xác định vùng cho bong bóng hiện tại trong ảnh bằng cách tạo một mảng ma trận ảnh **mask** bằng 0 có hình dạng (shape) giống với hình ảnh được gán ở biến **thresh** phía trên. Sau đó chúng ta sẽ vẽ vùng ảnh được khoanh tròn lên trên mảng ma trận vừa tạo ra đó nhờ hàm *.drawContours*. Hàm này có 5 tham số:

1. Tham số đầu tiên là ảnh đầu vào, trong trường hợp này là ảnh **mask**.
2. Tham số thứ hai chính là vùng ảnh của bong bóng.
3. Tham số thứ ba được gán giá trị -1 để vẽ toàn bộ viền ảnh (trong trường hợp này chỉ có vùng ảnh duy nhất đã được gán vào biến **l**)
4. Tham số thứ ba là màu dùng để vẽ (chúng ta gán 255 để cho màu trắng)
5. Tham số cuối cùng được gán giá trị -1 để điền đầy màu vào bên trong vùng ảnh giới hạn bởi viền.



Sau đó chúng ta sử dụng hàm *.bitwise­\_­and* để thu về vùng ảnh cần quan tâm, chính là vùng ảnh chứa bong bóng trả lời và gán vào biến **mask**.



Khi thu được chỉ riêng vùng ảnh của bong bóng trả lời trên nền đen, chúng ta sẽ đếm số pixel khác 0 nhờ vào hàm *.countNonZero*. Chúng ta đếm được càng nhiều pixel khác 0, thì càng có nhiều pixel ở phía trước, và do đó bong bóng có số lượng pixel khác 0 tối đa sẽ là bong bóng mà thí sinh đã khoanh vào. Chúng ta xác định bong bóng có số lượng pixel khác 0 tối đa qua vòng lặp for bên trong. Biến **bubbled** được sử dụng để chứa số lượng pixel của bong bóng có số pixel khác 0 tối đa và vị trí của bong bóng đó trong dãy 4 bong bóng (chính là biến **k**).

 Graphical user interface, application

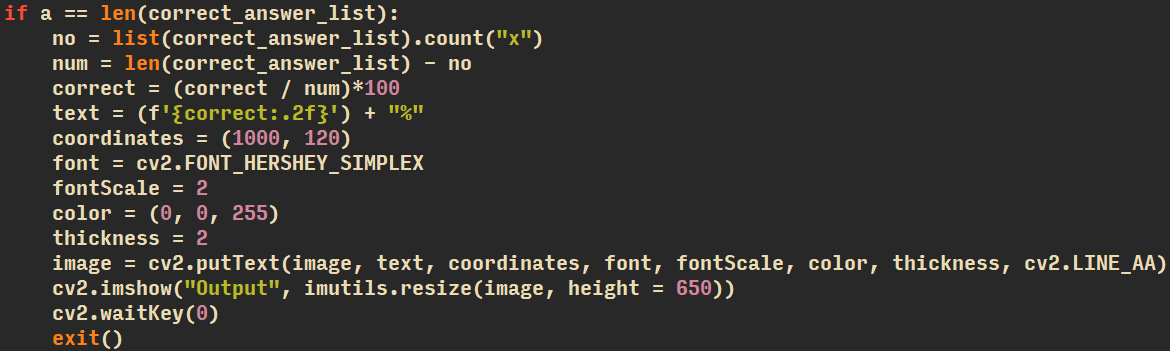
Description automatically generated

Trong trường hợp cả 4 bong đóng đều không được khoanh (Tức là thí sinh đã bỏ qua câu đó hoặc câu đó không nằm trong số câu hỏi) chúng ta sẽ gán giá trị thứ 2 của cặp giá trị thuộc về biến bubbled bằng “x” để đánh dấu.



Khi đã xác định được vị trí của bong bóng thí sinh khoanh vào, chúng ta sẽ so sánh với đáp án chứa ở list **correct\_answer\_list**. Chúng ta sẽ chia làm hai trường hợp.

1. Trong trường hợp thí sinh khoanh đúng: Biến **correct** sẽ được cộng thêm một.
2. Trong trường hợp thí sinh khoanh sai: Chúng ta sẽ pass.

Cuối cùng khi chúng ta đã đi qua đủ số lượng đáp án đúng chứa trong biến **correct\_answer\_list**, chúng ta sẽ in ra số điểm của bài kiểm tra và thoát chương trình nhờ hàm *exit()*.