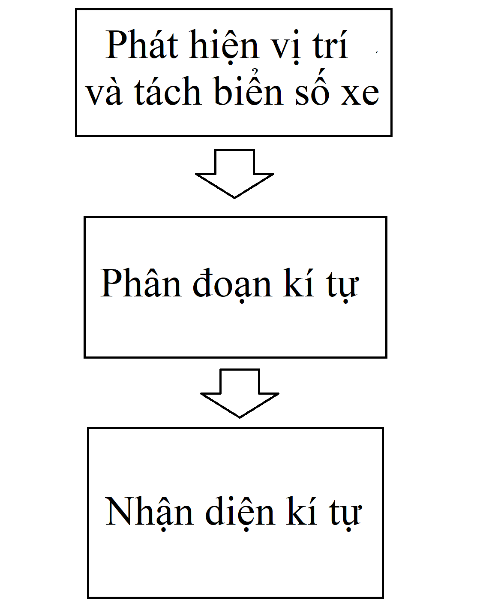
## Hướng giải quyết

Hiện nay trên thế giới đã có rất nhiều cách tiếp cận khác nhau với việc nhận dạng biển số xe, tuy nhiên trong phạm vi đồ án này em sẽ giải quyết vấn đề theo 3 bước chính:

* 1. Phát hiện vị trí và tách biển số xe từ một hình ảnh có sẵn từ đầu vào là camera
  2. Phân đoạn các kí tự có trong biển số xe
  3. Nhận diện các kí tự đó rồi đưa về mã ASCII

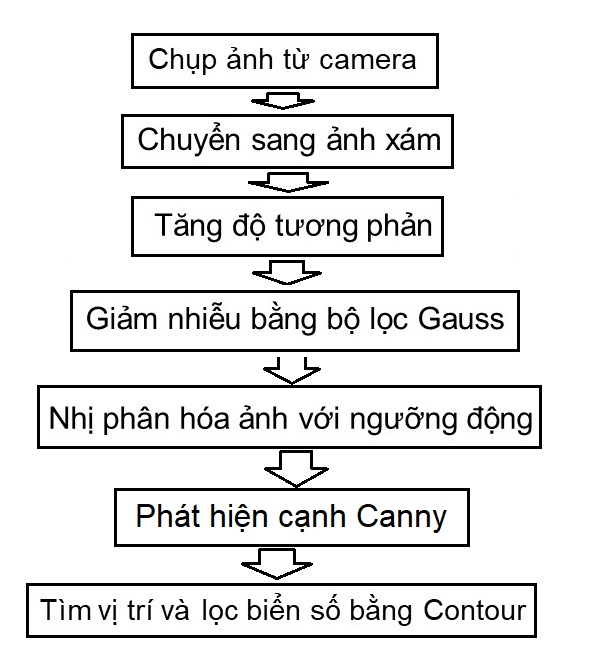


Hình 2.3 - 1 Các bước chính trong nhận dạng biển số xe

# PHÁT HIỆN VỊ TRÍ VÀ TÁCH BIỂN SỐ XE

## Hướng giải quyết

Sơ đồ dưới đây sẽ tóm gọn các bước để xác định và tách biển số xe từ clip:



Hình 3.1 - 1 Xác định và tách biển số xe

Đầu tiên từ clip ta sẽ cắt từng frame ảnh ra từ clip đầu vào để xử lý, tách biển số. Ở phạm vi đồ án này, ý tưởng chủ yếu là nhận diện được biển số từ sự thay đổi đột ngột về cường độ ánh sáng giữa biển số và môi trường xung quanh nên ta sẽ loại bỏ các dữ liệu màu sắc RGB bằng cách chuyển sang ảnh xám. Tiếp theo ta tăng độ tương phản với hai phép toán hình thái học Top Hat và Black Hat để làm nổi bật thêm biển số giữa phông nền, hỗ trợ cho việc xử lý nhị phân sau này. Sau đó, ta giảm nhiễu bằng bộ lọc Gauss để loại bỏ những chi tiết nhiễu có thể gây ảnh hưởng đến quá trình nhận diện, đồng thời làm tăng tốc độ xử lý.

Việc lấy ngưỡng sẽ giúp ta tách được thông tin biển số và thông tin nền, ở đây em chọn lấy ngưỡng động (Adaptive Threshold). Tiếp đó ta sử dụng thuật toán phát hiện cạnh Canny để trích xuất những chi tiết cạnh của biển số. Trong quá trình xử lý máy tính có thể nhầm lẫn biển số với những chi tiết nhiễu, việc lọc lần cuối bằng các tỉ lệ cao/rộng hay diện tích của biển số sẽ giúp xác định được đúng biển số. Cuối cùng, ta sẽ xác định vị trí của biển số trong ảnh bằng cách vẽ Contour bao quanh.

## Chuyển ảnh xám

Ảnh xám (Gray Scale) đơn giản là một hình ảnh trong đó các màu là các sắc thái của màu xám với 256 cấp độ xám biến thiên từ màu đen đến màu trắng, nằm trong giải giá trị từ 0 đến 255, nghĩa là cần 8 bits hay 1 byte để biểu diễn mỗi điểm ảnh này. Lý do cần phải phân biệt giữa ảnh xám và các ảnh khác nằm ở việc ảnh xám cung cấp ít thông tin hơn cho mỗi pixel. Với ảnh thông thường thì mỗi pixel thường được cung cấp 3 trường thông tin trong khi với ảnh xám chỉ có 1 trường thông tin, việc giảm khối lượng thông tin giúp tăng tốc độ xử lý, đơn giản hóa giải thuật nhưng vẫn đảm bảo các tác vụ cần thiết

Ở bài này em sẽ chuyển ảnh xám từ hệ màu HSV thay vì RGB vì với không gian màu HSV ta có ba giá trị chính là: Vùng màu (Hue), độ bão hòa (Saturation), cường độ sáng (Value). Vì lý do đó không gian màu HSV thích nghi tốt hơn đối với sự thay đổi ánh sáng từ môi trường ngoài. Khi chuyển đổi, ảnh xám ta cần là ma trận các giá trị cường độ sáng tách ra từ hệ màu HSV.

*def* extractValue(*imgOriginal*):

    height, width, numChannels = *imgOriginal*.shape

    imgHSV = np.zeros((height, width, 3), np.uint8)

    imgHSV = cv2.cvtColor(*imgOriginal*, cv2.COLOR\_BGR2HSV)

    imgHue, imgSaturation, imgValue = cv2.split(imgHSV)

## Tăng độ tương phản

### Phép toán hình thái học

Hình thái học toán học là một lý thuyết và kỹ thuật để phân tích và xử lý cấu trúc hình học, hình ảnh đầu ra được xác định chủ yếu dựa vào phần từ cấu trúc (structuring elements/kernel)

Ảnh có chứa trò chơi ô chữ

Mô tả được tạo tự động

Hình 3.3 - 1 Ví dụ về phần tử cấu trúc

Hình thái học toán học đã được phát triển cho hình ảnh nhị phân, và sau đó được mở rộng cho ảnh xám,... Đây là một trong những kỹ thuật được áp dụng trong giai đoạn tiền xử lý. Hai phép toán thường dùng là phép giãn nở (Dilation) và phép co (Erosion) . Từ hai phép toán cơ bản này người ta phát triển thành một số phép toán như phép đóng (Closing) và phép mở (Opening) và phép Top Hat, Black Hat.

1. Phép co

Phép toán co có ứng dụng trong việc giảm kích thước của đối tượng, tách rời các đối tượng gần nhau, làm mảnh và tìm xương của đối tượng.

|  |  |
| --- | --- |
|  |  |
| Ảnh gốc | Ảnh sau khi dùng phép co |

Hình 3.3 - 2 Phép co

1. Phép giãn nở

Phép toán này có tác dụng làm cho đối tượng ban đầu trong ảnh tăng lên về kích thước (giãn nở ra). Ứng dụng của phép giãn nở là làm cho đối tượng trong ảnh được tăng lên về kích thước, các lỗ nhỏ trong ảnh được lấp đầy, nối liền đường biên ảnh đối với những đoạn rời nhỏ.

|  |  |
| --- | --- |
|  |  |
| Ảnh gốc | Ảnh sau khi dùng phép giãn nở |

Hình 3.3 - 3 Phép giãn nở

1. Phép mở

Là thực hiện phép co trước sau đó mới thực hiện phép giãn nở. Phép toán mở được ứng dụng trong việc loại bỏ các phần lồi lõm và làm cho đường bao các đối tượng trong ảnh trở nên mượt mà hơn.

|  |  |
| --- | --- |
| Ảnh có chứa văn bản  Mô tả được tạo tự động |  |
| Ảnh gốc | Ảnh sau khi dùng phép mở |

Hình 3.3 - 4 Phép mở

1. Phép đóng

Thực hiện phép giãn nở trước sau đó mới thực hiện phép co. Phép toán đóng được dùng trong ứng dụng làm trơn đường bao các đối tượng, lấp đầy các khoảng trống biên và loại bỏ những hố nhỏ.

|  |  |
| --- | --- |
| Ảnh có chứa văn bản  Mô tả được tạo tự động |  |
| Ảnh gốc | Ảnh sau khi dùng phép đóng |

Hình 3.3 - 5 Phép đóng

1. Phép Top Hat

Phép Top Hat là kết quả của phép trừ ảnh của ảnh ban đầu với ảnh sau khi thực hiện phép mở, dùng để làm nổi bật nhưng chi tiết trắng trong nền tối

|  |  |
| --- | --- |
| Ảnh có chứa văn bản, ngoài trời, con đường, mặt đất  Mô tả được tạo tự động | Ảnh có chứa văn bản, ngoài trời  Mô tả được tạo tự động |
| Ảnh gốc | Ảnh sau khi dùng phép Top Hat |

Hình 3.3 - 6 Phép Top Hat

1. Phép Black Hat

Phép Black Hat là kết quả của phép trừ ảnh của ảnh sau khi thực hiện phép đóng với ảnh ban đầu. Dùng làm nổi bật chi tiết tối trong nền trắng.

|  |  |
| --- | --- |
| Ảnh có chứa văn bản, ngoài trời, con đường, mặt đất  Mô tả được tạo tự động | Ảnh có chứa văn bản, ngoài trời, đen, trắng  Mô tả được tạo tự động |
| Ảnh gốc | Ảnh sau khi dùng phép Black Hat |

Hình 3.3 - 7 Phép Black Hat

### Tăng độ tương phản

Để làm tăng độ tương phản của biển số, em sử dụng chủ yếu hai phép Top Hat và Black Hat. Ý tưởng chung là ảnh đầu ra sẽ là ảnh gốc cộng thêm ảnh qua phép Top Hat và trừ đi ảnh qua phép Black Hat. Những chi tiết đã sáng sẽ sáng hơn và những chi tiết tối lại càng tối hơn, từ đó sẽ làm tăng độ tương phản cho biển số.

   imgTopHat = np.zeros((height, width, 1), np.uint8)

   imgBlackHat = np.zeros((height, width, 1), np.uint8)

    structuringElement = cv2.getStructuringElement(cv2.MORPH\_RECT, (3, 3)) #tạo bộ lọc kernel

    imgTopHat = cv2.morphologyEx(*imgGrayscale*, cv2.MORPH\_TOPHAT, structuringElement, *iterations* = 10) #nổi bật chi tiết sáng trong nền tối

    #cv2.imwrite("tophat.jpg",imgTopHat)

    imgBlackHat = cv2.morphologyEx(*imgGrayscale*, cv2.MORPH\_BLACKHAT, structuringElement, *iterations* = 10) #Nổi bật chi tiết tối trong nền sáng

    #cv2.imwrite("blackhat.jpg",imgBlackHat)

    imgGrayscalePlusTopHat = cv2.add(*imgGrayscale*, imgTopHat)

    imgGrayscalePlusTopHatMinusBlackHat = cv2.subtract(imgGrayscalePlusTopHat, imgBlackHat)

    #cv2.imshow("imgGrayscalePlusTopHatMinusBlackHat",imgGrayscalePlusTopHatMinusBlackHat)

    #Kết quả cuối là ảnh đã tăng độ tương phản

    return imgGrayscalePlusTopHatMinusBlackHat

|  |  |
| --- | --- |
| Ảnh có chứa văn bản, ngoài trời, con đường, mặt đất  Mô tả được tạo tự động | Ảnh có chứa văn bản, ngoài trời, mặt đất, lề đường  Mô tả được tạo tự động |
| Ảnh gốc | Ảnh sau khi tăng độ tương phản |

Hình 3.3 - 8 Ảnh sau khi tăng độ tương phản

## Giảm nhiễu bằng bộ lọc Gauss

### Nhiễu

Noise được hiểu cơ bản là các dạng chấm hạt nhỏ phân bố trên hình ảnh. Noise có thể làm biến dạng các chi tiết trong ảnh khiến cho chất lượng ảnh thấp.

Trên thực tế có nhiều loại nhiễu, nhưng người ta thường chia làm ba loại: nhiễu cộng, nhiễu nhân và nhiễu xung. Bản chất của nhiễu thường tương ứng với tần số cao và cơ sở lý thuyết của bộ lọc là chỉ cho những tín hiệu có tần số nhất định đi qua, nên người ta thường sử dụng bộ lọc thông thấp hay trung bình.

Ảnh có chứa văn bản

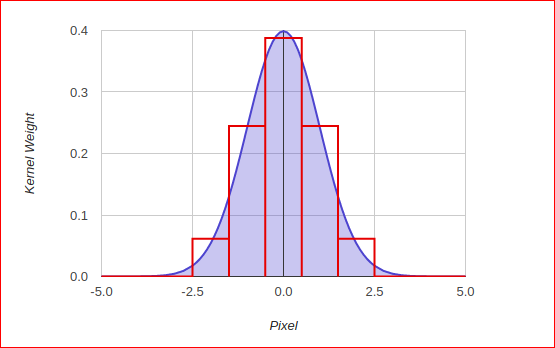
Mô tả được tạo tự động

Hình 3.4 - 1 Nhiễu

### Bộ lọc Gauss (Gauss filter)

Bộ lọc Gauss được cho là bộ lọc hữu ích nhất, được thực hiện bằng cách nhân chập ảnh đầu vào với một ma trận lọc Gauss sau đó cộng chúng lại để tạo thành ảnh đầu ra.

Ý tưởng chung là giá trị mỗi điểm ảnh sẽ phụ thuộc nhiều vào các điểm ảnh ở gần hơn là các điểm ảnh ở xa. Trọng số của sự phụ thuộc được lấy theo hàm Gauss (cũng được sử dụng trong quy luật phân phối chuẩn).



Hình 3.4 - 2 Ma trận lọc Gauss

Giả sử ảnh là một chiều. Điểm ảnh ở trung tâm sẽ có trọng số lớn nhất. Các điểm ảnh ở càng xa trung tâm sẽ có trọng số giảm dần khi khoảng cách từ chúng tới điểm trung tâm tăng lên. Như vậy điểm càng gần trung tâm sẽ càng đóng góp nhiều hơn vào giá trị điểm trung tâm.

|  |  |
| --- | --- |
| Ảnh có chứa văn bản, ngoài trời, mặt đất  Mô tả được tạo tự động | Ảnh có chứa văn bản, ngoài trời, con đường, đường phố  Mô tả được tạo tự động |
| Ảnh gốc | Ảnh sau khi làm mờ, giảm nhiễu |

Hình 3.4 - 3 Kết quả sử dụng bộ lọc Gauss

## Nhị phân hóa với ngưỡng động (Adaptive Threshold)

### Ảnh nhị phân

Là ảnh mà giá trị của các điểm ảnh chỉ được biểu diễn bằng hai giá trị là 0 (Đen) và 255 (Trắng)

### Nhị phân hóa

Là quá trình biến đổi một ảnh xám thành ảnh nhị phân.

* + Gọi giá trị cường độ sáng tại một điểm ảnh là I(x,y) .
  + INP(x,y) là cường độ sáng của điểm ảnh trên ảnh nhị phân .
  + (Với 0 < x < image.width) và (0 < y < image.height).

Để biến đổi ảnh xám thành ảnh nhị  phân. Ta so sánh giá trị cường độ sáng của điểm ảnh với một ngưỡng nhị phân **T**.

* + Nếu I(x,y) > **T** thì INP(x, y) = 0.
  + Nếu I(x,y) > **T** thì INP(x, y) = 255.

### Nhị phân hóa với ngưỡng động

Việc nhị phân hóa ảnh với ngưỡng toàn cục như thông thường sẽ rất khó khăn khi phải tự tính toán và chọn mức ngưỡng phù hợp cho từng ảnh khác nhau. Nhị phân hóa ảnh ngưỡng động sẽ giúp tính toán ngưỡng cho phù hợp với từng ảnh, lợi thế thứ hai chính là nó rất phù hợp khi ảnh có vùng bị quá chói hoặc quá tối dẫn đến có thể mất luôn hình ảnh tại vùng đó nếu sử dụng ngưỡng toàn cục.

Về ý tưởng chính sẽ theo 3 bước sau:

1. Chia tấm ảnh thành nhiều khu vực, cửa sổ (Region) khác nhau.
2. Dùng một thuật toán để tìm một giá trị **T** phù hợp với từng cửa sổ.
3. Áp dụng phương pháp nhị phân hóa cho từng khu vực, cửa sổ với ngưỡng **T** phù hợp.

Có rất nhiều phương pháp để tìm **T**, ở đây em sử dụng một kiểu thuật toán mà thư viện OpenCV hỗ trợ là ADAPTIVE\_THRESH\_GAUSSIAN\_C tức lấy trung bình các giá trị xung quanh điểm ngưỡng động đang xét **T**(x,y) với phân phối Gauss rồi trừ đi hằng số C.

imgThresh = cv2.adaptiveThreshold(imgBlurred, 255.0, cv2.ADAPTIVE\_THRESH\_GAUSSIAN\_C, cv2.THRESH\_BINARY\_INV, ADAPTIVE\_THRESH\_BLOCK\_SIZE, ADAPTIVE\_THRESH\_WEIGHT)

Ảnh có chứa văn bản

Mô tả được tạo tự động

Hình 3.5 - 1 Nhị phân hóa ảnh ngưỡng động

## Phát hiện cạnh Canny (Canny Edge Detection)

Trong hình ảnh, thường tồn tại các thành phần như: vùng trơn, góc/cạnh và nhiễu. Cạnh trong ảnh mang đặc trưng quan trọng, thường là thuộc đối tượng trong ảnh. Do đó, để phát hiện cạnh trong ảnh, có nhiều giải thuật khác nhau như toán tử Sobel, toán tử Prewitt, Zero crossing .... nhưng ở đây em chọn giải thuật Canny vì hương pháp này hơn hẳn các phương pháp khác do ít bị tác động của nhiễu và cho khả năng phát hiện các biên yếu. Phương pháp này đi theo 4 bước chính:

1. Giảm nhiễu (Noise reduction)
2. Tính toán Gradient (Gradient calculation)
3. Loại bỏ những điểm không phải là cực đại (Non-maximum suppression)
4. Lọc ngưỡng (Double threshold)
5. Giảm nhiễu

Làm mờ ảnh, giảm nhiễu dùng bộ lọc Gauss kích thước 5x5. Kích thước 5x5 thường hoạt động tốt cho giải thuật Canny

    imgBlurred = cv2.GaussianBlur(imgGray, (5,5), 0)

1. Tính toán Gradient

Ta dùng 2 bộ lọc Sobel X và Sobel Y (3x3) để tính đạo hàm Gx và Gy

|  |  |
| --- | --- |
|  |  |

Tìm gradient và hướng được làm tròn về 4 hướng: hướng ngang (0 độ), hướng chéo bên phải (45 độ), hướng dọc (90 độ) và hướng chéo trái (135 độ).

|  |  |
| --- | --- |
|  |  |

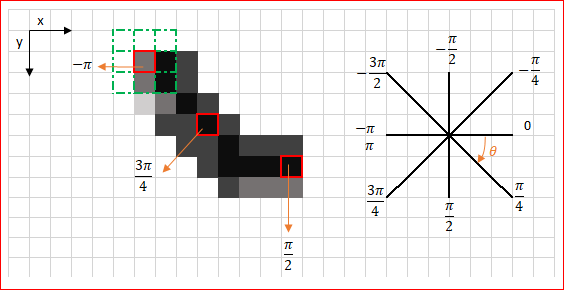
1. Loại bỏ những điểm không phải là cực đại

Ở bước này, dùng một filter 3x3 lần lượt chạy qua các pixel trên ảnh gradient. Trong quá trình lọc, xem xét xem độ lớn gradient của pixel trung tâm có phải là cực đại so với các gradient ở các pixel xung quanh. Nếu là cực đại, ta sẽ ghi nhận sẽ giữ pixel đó lại. Còn nếu pixel tại đó không phải là cực đại lân cận, ta sẽ set độ lớn gradient của nó về zero. Ta chỉ so sánh pixel trung tâm với 2 pixel lân cận theo **hướng gradient.** Ví dụ: nếu hướng gradient đang là 0 độ, ta sẽ so pixel trung tâm với pixel liền trái và liền phải nó. Trường hợp khác nếu hướng gradient là 45 độ, ta sẽ so sánh với 2 pixel hàng xóm là góc trên bên phải và góc dưới bên trái của pixel trung tâm

    canny\_image = cv2.Canny(imgThreshplate,250,255)

    kernel = np.ones((3,3), np.uint8)

kerel3 = cv2.getStructuringElement(cv2.MORPH\_RECT,(3,3))



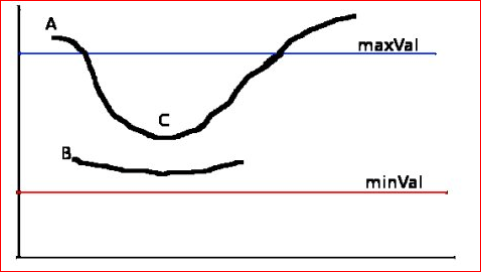
Hình 3.6 - 1 Loại bỏ những điểm không phải cực đại

1. Lọc ngưỡng

**Lọc ngưỡng: ta sẽ xét các pixel dương trên mặt nạ nhị phân kết quả của bước trước. Nếu giá trị gradient vượt ngưỡng max\_val thì pixel đó chắc chắn là cạnh. Các pixel có độ lớn gradient nhỏ hơn ngưỡng min\_val sẽ bị loại bỏ. Còn các pixel nằm trong khoảng 2 ngưỡng trên sẽ được xem xét rằng nó có nằm liền kề với những pixel được cho là "chắc chắn là cạnh" hay không. Nếu liền kề thì ta giữ, còn không liền kề bất cứ pixel cạnh nào thì ta loại. Sau bước này ta có thể áp dụng thêm bước hậu xử lý loại bỏ nhiễu (tức những pixel cạnh rời rạc hay cạnh ngắn) nếu muốn**

            mask = np.zeros(imgGrayscaleplate.shape, np.uint8)

            new\_image = cv2.drawContours(mask, [screenCnt], 0, 255, -1, )



Hình 3.6 - 2 Lọc ngưỡng

1. Kết quả

Sau khi sử dụng phát hiện biên canny, dù đã trích xuất được những chi tiết cạnh của biển số, tuy nhiên vẫn còn quá nhiều chi tiết thừa trong hình ảnh, từ đây chúng ta sẽ vẽ contour, áp dụng nhưng đặc điểm của biển số để lọc lấy ra biển số chính xác.

Ảnh có chứa văn bản

Mô tả được tạo tự động

Hình 3.6 - 3 Ảnh sau khi phát hiện biên Canny

## Lọc biển số với contour

### Một số phương pháp tìm contour

Có thể hiểu Contour là tập hợp các điểm tạo thành đường cong kín bao quanh một đối tượng nào đó. Thường dùng để xác định vị trí, đặc điểm của đối tượng. Có 4 thuật toán Contour Tracing chung nhất. Hai trong số đó có tên là: Square Tracing algorithm và Moore – Neighbor Tracing là dễ để thực hiện và thường xuyên được dùng để dò tìm contour của một mẫu. Với thư viện OpenCV người ta áp dụng thuật toán Suzuki’s Contour tracing. Dưới đây em sẽ trình bày kĩ hơn về 3 phương pháp trên:

1. Thuật toán Square Tracing

Duyệt từ pixel ngoài cùng bên trái phía dưới, đi lên cho tới khi gặp pixel có giá trị bằng 255 (pixel này sẽ được gọi là pixel start) thì bắt đầu di chuyển theo quy tắc sau:

* Nếu gặp Pixel có giá trị bằng 255 thì rẽ trái.
* Nếu gặp Pixel có giá trị bằng 0 thì rẽ phải.
* Di chuyển cho tới khi quay lại pixel start thì dừng lại.

Hình sau mô tả cách hoạt động của thuật toán:

Ảnh có chứa bảng trắng

Mô tả được tạo tự động

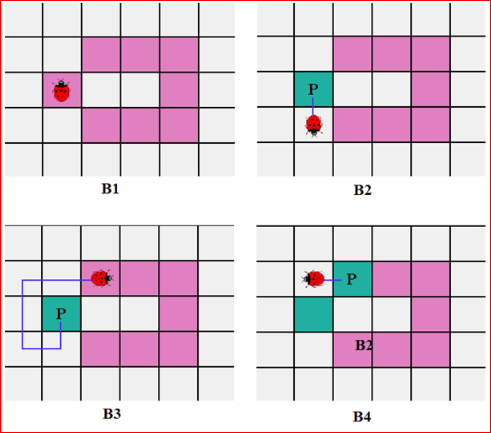
Hình 3.7 - 1 Thuật toán Square Tracing

Thuật toán sẽ kết thúc đúng khi di chuyển vào pixel start lần thứ 2 sau khi đi qua n pixel khác và theo đúng hướng đi vào pixel start lần đầu tiên. Và sai khi di chuyển vào pixel start mà không đúng hướng ban đầu. Vậy thuật toán này chỉ chạy đúng trên đối tượng   
4 - connected.

|  |  |
| --- | --- |
| Ảnh có chứa cửa shoji, tòa nhà  Mô tả được tạo tự động  Hình 3.7 - 2 Thuật toán Square Tracing  chạy đúng | Ảnh có chứa cửa shoji, trò chơi ô chữ, tòa nhà, mẫu họa  Mô tả được tạo tự động  Hình 3.7 - 3 Thuật toán Square Tracing  chạy sai |

1. Thuật toán Moore – Neighbor

Thuật toán này có chút khác biệt so với thuật toán Square Tracking, cụ thể: khi gặp pixel có giá trị bằng 255 đầu tiên (pixel start) thì ta sẽ quay lại pixel trước đó, sau đó đi vòng qua các pixel thuộc 8-connected theo chiều kim đồng hồ cho tới khi gặp pixel khác có giá trị bằng 255. Và điệu kiện kết thúc cũng giống như thuật toán Square Tracking



Hình 3.7 - 4 Thuật toán Moore - Neighbor.

1. Thuật toán Suzuki’s Tracing

Đây là thuật toán được thư viện OpenCV sử dụng, ngoài khả năng xác định được biên của vật thể như hai phương pháp trên. Phương pháp Suzuki’s Tracing còn có khả năng phân biệt được đường biên ngoài (Outer) và đường biên trong (Hole) của vật thể.

Hàm trong OpenCV được biểu diễn như sau:

**findContours**(InputOutputArray image, OutputArrayOfArrays contours, OutputArray hierarchy, int mode, int method, Point offset=Point())

Các tham số:   
**image** : hình ảnh cần tìm biên, là ảnh nhị phân.  
**contours** : lưu trữ các đường biên tìm được, mỗi đường biên được lưu trữ dưới dạng một vector của các điểm.  
**hierarchy** :  chứa thông tin về hình ảnh như số đường viền, xếp hạng các đường viền theo kích thước, trong ngoài, ..  
**mode** :  
CV\_RETR\_EXTERNAL : khi sử dựng cờ này nó chỉ lấy ra những đường biên bên ngoài, nhưng biên bên trong của vật thể bị loại bỏ.  
CV\_RETR\_LIST : Khi sử dụng cờ này nó lấy ra tất cả các đường viền tìm được.  
CV\_RETR\_CCOMP : khi sử dụng cờ này nó lấy tất cả những đường biên và chia nó làm 2 level, những đường biên bên ngoài đối tượng, và những đường biên bên trong đối tượng.  
CV\_RETR\_TREE : khi sử dụng cờ này nó lấy tất cả các đường biên và tạo ra một hệ thống phân cấp đầy đủ của những đường lồng nhau.  
**method** :  
CV\_CHAIN\_APPROX\_NONE : sử dụng cờ này sẽ lưu trữ tất cả các điểm của đường viền .  
CV\_CHAIN\_APPROX\_SIMPLE : Ví dụ : một hình chữ nhật sẽ được mã hoá bằng toạ độ của 4 đỉnh.  
CV\_CHAIN\_APPROX\_TC89\_L1 or CV\_CHAIN\_APPROX\_TC89\_KCOS :  Áp dụng thuật toán xấp xỉ Tech-Chin.

Ảnh có chứa ngoài trời, mặt đất, màu hồng, đã sơn

Mô tả được tạo tự động

Hình 3.7 - 5 Vẽ Contour với OpenCV

imgContours, npaContours, npaHierarchy = cv2.findContours(imgThreshCopy, cv2.RETR\_EXTERNAL, cv2.CHAIN\_APPROX\_SIMPLE)

Trong ảnh, những đường màu hồng là đường contour bao quanh vật thể, nhưng vì có quá nhiều đường bao quanh các vật thể không phải biển số nên chúng ta sẽ áp dụng những đặc trưng riêng về tỉ lệ cao/rộng, diện tích trong khung hình cố định như ở mục 2.1 để lọc ra đúng biển số cần tìm.

contours, hierarchy = cv2.findContours(dilated\_image, cv2.RETR\_LIST, cv2.CHAIN\_APPROX\_SIMPLE)

### Lọc biển số

approx = cv2.approxPolyDP(c, 0.06 \* peri, True)

[x, y, w, h] = cv2.boundingRect(approx.copy())

ratio = w/h

    if (len(approx) == 4) and (0.8 <= ratio <= 1.5 or 4.5 <= ratio <= 6.5):

            screenCnt.append(approx)

    if screenCnt is None:

        detected = 0

        print ("No plate detected")

    else:

        detected = 1

Đầu tiên ta làm xấp xỉ contour thành một hình đa giác và chỉ lấy những đa giác nào chỉ có 4 cạnh. Nghĩa là lúc xấp xỉ contour bộ nhớ chỉ ghi nhớ vị trí các đỉnh của đa giác đó thành một mảng. Số cạnh của đa giác sẽ bằng số đỉnh và bằng chiều dài của mảng đó.

|  |  |
| --- | --- |
| Ảnh có chứa văn bản, mẫu họa  Mô tả được tạo tự động  Hình 3.7 - 6 Contour chưa xấp xỉ đa giác | Ảnh có chứa văn bản, đồng hồ, thiết bị đo  Mô tả được tạo tự động  Hình 3.7 - 7 Contour đã xấp xỉ đa giác |

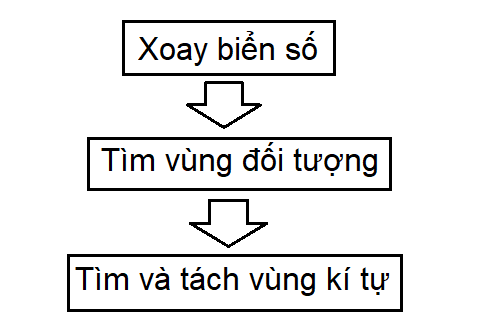
Tiếp theo ta tính toán tỉ lệ cao/rộng và diện tích của biển số phù hợp, sau đó ta lưu tất cả những biển số có trong hình dưới dạng tọa độ các đỉnh

Từ đây, ta cắt hình ảnh biển số từ các tọa độ vị trí đã biết để phục vụ cho mục đích tiếp theo “Tách các kí tự trong biển số”. Lưu ý ở đây ta cắt từ ảnh nhị phân luôn để máy tính xử lý nhanh hơn, tốn ít thời gian hơn.

# PHÂN ĐOẠN KÍ TỰ

## Hướng giải quyết

Ở giai đoạn này có những bước chính sau: Xoay biển số để tăng khả năng nhận diện, Tìm tất cả các vùng kín cho là kí tự và lọc ra những kí tự đúng. Tách hình ảnh nhưng kí tự đó ra và đưa vào bộ nhận diện



Hình 4.1 - 1 Các bước chính trong phân đoạn kí tự

## Xoay biển số

Khi chụp ảnh đầu vào, không phải lúc nào biển số cũng ở chính diện, có thể bị méo sang trái, sang phải, nghiêng góc dẫn đến nếu cứ sử dụng ảnh biển số đã cắt mà không điều chỉnh góc độ dẫn đến ảnh kí tự được cắt ra đưa vào bộ nhận diện rất dễ bị sai. Ví dụ giữa số 1 và số 7, số 2 và chữ Z, chữ B và số 8,...

(x1,y1) = screenCnt[0,0]

(x2,y2) = screenCnt[1,0]

(x3,y3) = screenCnt[2,0]

(x4,y4) = screenCnt[3,0]

array = [[x1, y1], [x2,y2], [x3,y3], [x4,y4]]

sorted\_array = array.sort(*reverse*=True, *key*=*lambda* *x*:*x*[1])

(x1,y1) = array[0]

(x2,y2) = array[1]

doi = abs(y1 - y2)

ke = abs (x1 - x2)

angle = math.atan(doi/ke) \* (180.0 / math.pi)

# Masking the part other than the number plate

mask = np.zeros(imgGrayscaleplate.shape, np.uint8)

            new\_image = cv2.drawContours(mask, [screenCnt], 0, 255, -1, )

Ảnh có chứa văn bản, sinh nhật, xanh lục, nói

Mô tả được tạo tự động

Hình 4.2 - 1 Ảnh biển số chưa xoay

Phương pháp xoay ảnh em sử dụng ở đây là:

1. Lọc ra tọa độ 2 đỉnh A,B nằm dưới cùng của biển số
2. Từ 2 đỉnh có tọa độ lần lượt là A(x1, y1) và B(x2,y2) ta có thể tính được cạnh đối và cạnh kề của tam giác ABC
3. Ta tính được góc quay
4. Xoay ảnh theo góc quay đã tính. Nếu ngược lại điểm A nằm cao hơn điểm B ta cho góc quay âm

# Now crop

(x, y) = np.where(mask == 255)

(topx, topy) = (np.min(x), np.min(y))

(bottomx, bottomy) = (np.max(x), np.max(y))

roi = img[topx:bottomx + 1, topy:bottomy + 1]

imgThresh = imgThreshplate[topx:bottomx + 1, topy:bottomy + 1]

ptPlateCenter = (bottomx - topx)/2, (bottomy - topy)/2

if x1 < x2:

rotationMatrix = cv2.getRotationMatrix2D(ptPlateCenter, -angle, 1.0)

else:

rotationMatrix = cv2.getRotationMatrix2D(ptPlateCenter, angle, 1.0)

roi = cv2.warpAffine(roi, rotationMatrix, (bottomy - topy, bottomx - topx ))

imgThresh = cv2.warpAffine(imgThresh, rotationMatrix, (bottomy - topy, bottomx - topx ))

roi = cv2.resize(roi,(0,0),*fx* = 3, *fy* = 3)

imgThresh = cv2.resize(imgThresh,(0,0),*fx* = 3, *fy* = 3)

Ảnh có chứa văn bản

Mô tả được tạo tự động

Hình 4.2 - 2 Ảnh biển số đã xoay

## Tìm vùng đối tượng

Từ ảnh nhị phân, ta lại tìm contour cho các kí tự (phần màu trắng). Sau đó vẽ những hình chữ nhật bao quanh các kí tự đó. Tuy nhiên việc tìm contour này cũng bị nhiễu dẫn đến việc máy xử lý sai mà tìm ra những hình ảnh không phải kí tự. Ta sẽ áp dụng các đặc điểm về tỉ lệ chiều cao/rộng của kí tự, diện tích của kí tự so với biển số

Ảnh có chứa văn bản

Mô tả được tạo tự động

Hình 4.3 - 1 Ảnh nhị phân

Ảnh có chứa văn bản

Mô tả được tạo tự động

Hình 4.3 - 2 Tìm vùng đối tượng

Trong ảnh 4.3 - 2 những đường màu vàng là đường contour và nếu so sánh với ảnh nhị phân 4.3 -1 thì có rất nhiều đường nhiễu như đường viền biển số, dấu gạch, dấu chấm... Sau khi đã áp dụng các điều kiện thì sẽ vẽ ra những hình chữ nhật màu xanh bao quanh các kí tự.

## Tìm và tách kí tự

Sau khi đã nhận dạng từng kí tự bằng hình chữ nhật và cũng đã có tọa độ vị trí 4 đỉnh của hình đó, ta lúc này có thể cắt hình ảnh kí tự đó ra phục vụ cho giai đoạn sau “Nhận diện kí tự”. Lưu ý ở đây ta cắt ảnh nhị phân chứ không cắt từ ảnh gốc.

# cắt kí tự ra khỏi hình

  imgROI = thre\_mor[y:y+h,x:x+w]

# resize lại hình ảnh

  imgROIResized = cv2.resize(imgROI, (RESIZED\_IMAGE\_WIDTH, RESIZED\_IMAGE\_HEIGHT))

Ảnh có chứa văn bản, vũ khí

Mô tả được tạo tự động

Hình 4.4 - 1 Ảnh kí tự sau khi cắt

# NHẬN DIỆN KÍ TỰ

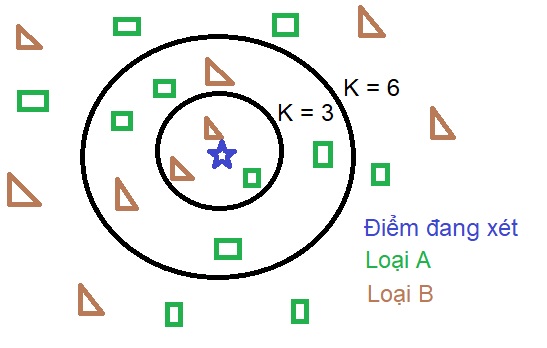
Thực chất quá trình nhận diện kí tự chính là quá trình chuyển đổi từ một hình ảnh là ma trận giá trị các điểm ảnh về một dạng thông tin khác như trong đề tài này là mã ASCII để có thể giao tiếp với người dùng. Để hiểu hơn về nhận diện, chúng ta cần đi ngược lại về ngành khoa học trí tuệ nhân tạo (Artificial Intelligent) hay còn gọi là AI

### Thuật toán KNN (K - Nearest Neighbor)

KNN là một trong những thuật toán học có giám sát đơn giản nhất trong Machine Learning, có thể sử dụng cho cả bài toán phân loại và hồi quy. Về ý tưởng là gán kết quả với dữ liệu training gần giống với mẫu nhất. Ví dụ như khi đi câu cá, ta không biết cá câu lên là cá rô hay cá chép, nhưng khi so sánh những đặc điểm về mắt, mang, vây,... từ những con cá rô, cá chép đã thấy thì cuối cùng có thể quyết định xem con cá mình câu được thuộc nhóm cá nào.

KNN hoạt động theo quy trình gồm 4 bước chính:

1. Xác định tham số K (số láng giềng gần nhất).
2. Tính khoảng cách từ điểm đang xét đến tất cả các điểm trong tập dữ liệu cho trước
3. Sắp xếp các khoảng cách đó theo thứ tự tăng dần
4. Xét trong tập K điểm gần nhất với điểm đang xét, nếu số lượng điểm của loại nào cao hơn thì coi như điểm đang xét thuộc loại đó



Hình 5.1 - 3 Ví dụ về KNN

Việc điểm đang xét thuộc loại nào còn phụ thuộc vào hệ số K hay trọng số khoảng cách... mà người dùng đặt sao cho phù hợp với bài toán đang xét. Chẳng hạn ở hình trên nếu ta xét K = 3 thì điểm đang xét sẽ thuộc loại B, ngược lại nếu K = 6 thì nó thuộc loại A. Ngoài ra người ta có thể để trọng số cao hơn cho những điểm gần hơn hay ít khi sử dụng K = 1 để đảm bảo kết quả đầu ra được tối ưu.

Thông thường việc tính khoảng cách đến các điểm sẽ theo công thức Euclid:

Khi thực hiện so sánh có thể bỏ qua dấu căn bậc 2. Ngoài ra nếu như khoảng cách giữa các biến quá lớn như biến x lớn hơn xấp xỉ 1000000 lần thì ta cũng cần chuẩn hóa lại dữ liệu theo công thức:

## Hướng giải quyết

Ở giai đoạn cuối này được thực hiện theo những bước sau:

1. Tạo tập dữ liệu để huấn luyện
2. Huấn luyện mô hình KNN
3. Đưa hình ảnh từ bước “Phân đoạn kí tự” vào mô hình KNN đã tạo để đưa ra kết quả
4. In ra kết quả biển số

Bước 1 và 2 ta sẽ tạo ra mô hình KNN riêng biệt với code chính. Để khi cần nhận diện kí tự ta không cần phải làm lại các bước từ đầu. Đầu tiên em tạo tập dữ liệu (tập hình ảnh của các chữ số và kí tự) để train từ phần mềm paint. Trong phần mềm Paint ta viết các chữ số và kí tự (trừ kí tự O, I, J) với phông chữ “Biển số xe Việt Nam”, có thể xoay các kí tự này lần lượt với các góc . Kết quả có dạng như sau:

Ảnh có chứa văn bản

Mô tả được tạo tự động

Hình 5.2 - 1 Tập dữ liệu huấn luyện

Tiếp theo ta lấy ngưỡng, vẽ contour và cắt từng kí tự. Vì mỗi kí tự có kích thước khác nhau xử lý phức tạp nên cần chuẩn hóa hình ảnh lại với kích thước cao:rộng là 30:20 pixels. Thay vì mỗi kí tự đưa vào mô hình để máy nhận diện thì những kí tự này sẽ được ta gắn nhãn bằng những phím bấm trên máy tính. Sau khi gắn nhãn hết các kí tự ta sẽ lưu hai file .txt là classifications.txt và flattened\_images.txt. File classifications.txt có nhiệm vụ lưu các mã ASCII của các kí tự đó và file flattened\_images.txt sẽ lưu giá trị các điểm ảnh có trong hình ảnh kí tự (hình 20x30 pixel có tổng cộng 600 điểm ảnh có giá trị 0 hoặc 255)

RESIZED\_IMAGE\_WIDTH = 20

RESIZED\_IMAGE\_HEIGHT = 30

if intChar == 27:

   sys.exit()

elif intChar in intValidChars:

intClassifications.append(intChar)

npaFlattenedImage = imgROIResized.reshape((1, RESIZED\_IMAGE\_WIDTH \* RESIZED\_IMAGE\_HEIGHT))

npaFlattenedImages = np.append(npaFlattenedImages, npaFlattenedImage, 0)

fltClassifications = np.array(intClassifications, np.float32)

npaClassifications = fltClassifications.reshape((fltClassifications.size, 1))

    print ("\n\ntraining complete !!\n")

    np.savetxt("classifications.txt", npaClassifications)

    np.savetxt("flattened\_images.txt", npaFlattenedImages)

    cv2.destroyAllWindows()

    return

Bước 3 và 4. Ta thực hiện đưa ảnh đang xét vào và tính khoảng cách đến tất cả các điểm trong mẫu, kết quả sẽ là mã ASCII đại điện cho hình ảnh đó. Cuối cùng ta in biển số xe ra hình. Tuy nhiên ở Việt Nam có hai loại biển số là biển một hàng và biển hai hàng. Về ý tưởng chung để phân biệt hai hàng này ta dựa vào vị trí của hình ảnh kí tự, nếu vị trí nằm thấp 1/3 chiều cao của biển số thì kí tự sẽ được xếp vào hàng một. Ngược lại sẽ được xếp vào hàng hai.

# Biển số 1 hay 2 hàng

     if (y < height/3):

       first\_line = first\_line + strCurrentChar

     else:

       second\_line = second\_line + strCurrentChar

       strFinalString = first\_line + second\_line

print ("\n License Plate " +str(n)+ " is: " + first\_line + " - " + second\_line + "\n")

|  |  |
| --- | --- |
| Ảnh có chứa văn bản, đĩa, bộ đồ ăn, đĩa lót  Mô tả được tạo tự động  Hình 5.2 - 2 Biển số trước khi nhận diện | Hình 5.2 - 3 Biển số sau khi nhận diện |

Ảnh có chứa văn bản, ngoài trời, mặt đất, bãi đậu xe

Mô tả được tạo tự động

Hình 5.2 - 4 Biển số xe được in ra trên hình gốc

cv2.putText(img, strFinalString, (topy ,topx),cv2.FONT\_HERSHEY\_DUPLEX, 1, (0, 255, 255), 1)