TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM
TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG
KHOA CÔNG NGHỆ THÔNG TIN

BÀI TẬP LỚN CUỐI KÌ NHẬP MÔN XỬ LÍ ẢNH SỐ

TRUY XUẤT THỜI GIAN TỪ ẢNH CHỤP ĐỒNG HỒ

Người hướng dẫn: **TRẦN LƯƠNG QUỐC ĐẠI**

Người thực hiện: **HOÀNG TRUNG KIÊN – 52100903**

Lớp **: 21050301**

Khoá **: 25**

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2023

TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM
TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG
KHOA CÔNG NGHỆ THÔNG TIN

BÀI TẬP LỚN CUỐI KÌ NHẬP MÔN XỬ LÍ ẢNH SỐ

TRUY XUẤT THỜI GIAN TỪ ẢNH CHỤP ĐỒNG HỒ

Người hướng dẫn: **TRẦN LƯƠNG QUỐC ĐẠI**
Người thực hiện: **HOÀNG TRUNG KIÊN**
Lớp **: 21050301**
Khoá **: 25**

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2023

LỜI CẢM ƠN

Để có thể hoàn thành bài báo cáo này, tụi em xin trân trọng gửi lời cảm ơn đến những sự trợ giúp dành cho tụi em.

Lời cảm ơn đầu tiên, tụi em xin dành cho Trường Đại học Tôn Đức Thắng vì đã hỗ trợ tụi em xuyên suốt quá trình học tập tại trường.

Lời cảm ơn thứ hai, tụi em xin phép dành cho Khoa Công Nghệ Thông Tin vì những môn học bổ ích mà Khoa đã dành cho tụi em.

Lời cảm ơn cuối cùng chính là lời dành cho thầy Trần Lương Quốc Đại vì những kiến thức bổ ích mà thầy dạy cho tụi em để có thể hoàn thành bài báo cáo này.

Xin trân trọng cảm ơn.

ĐỒ ÁN ĐƯỢC HOÀN THÀNH TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG

Tôi xin cam đoan đây là sản phẩm đồ án của riêng tôi / chúng tôi và được sự hướng dẫn của Thầy Trần Lương Quốc Đại;. Các nội dung nghiên cứu, kết quả trong đề tài này là trung thực và chưa công bố dưới bất kỳ hình thức nào trước đây. Những số liệu trong các bảng biểu phục vụ cho việc phân tích, nhận xét, đánh giá được chính tác giả thu thập từ các nguồn khác nhau có ghi rõ trong phần tài liệu tham khảo.

Ngoài ra, trong đồ án còn sử dụng một số nhận xét, đánh giá cũng như số liệu của các tác giả khác, cơ quan tổ chức khác đều có trích dẫn và chú thích nguồn gốc.

Nếu phát hiện có bất kỳ sự gian lận nào tôi xin hoàn toàn chịu trách nhiệm về nội dung đồ án của mình. Trường đại học Tôn Đức Thắng không liên quan đến những vi phạm tác quyền, bản quyền do tôi gây ra trong quá trình thực hiện (nếu có).

TP. Hồ Chí Minh, ngày tháng năm

Tác giả

(ký tên và ghi rõ họ tên)

Kiên

Hoàng Trung Kiên

PHẦN XÁC NHẬN VÀ ĐÁNH GIÁ CỦA GIẢNG VIÊN

Phần xác nhận của GV hướng dẫn

_____

_____

_____

_____

_____

_____

_____

Tp. Hồ Chí Minh, ngày tháng năm
(kí và ghi họ tên)

Phần đánh giá của GV chấm bài

_____

_____

_____

_____

_____

_____

_____

Tp. Hồ Chí Minh, ngày tháng năm
(kí và ghi họ tên)

TÓM TẮT

Bài báo cáo này sẽ chú trọng về việc xử lí những bức ảnh chụp đồng hồ và từ đó trích xuất thời gian từ đó.

Mục Lục

DANH MỤC CÁC BẢNG BIỂU, HÌNH VẼ, ĐỒ THỊ

DANH MỤC HÌNH

DANH MỤC BẢNG

CHƯƠNG 1 – GIỚI THIỆU ĐỀ TÀI

1. Lí do chọn đề tài

Lí do chọn đề tài về việc xây dựng một thuật toán để truy xuất thời gian từ đồng hồ gồm những điều sau

Đầu tiên đó chính là việc muốn xây dựng một thuật toán để có thể hỗ trợ người dùng lấy thời gian từ đồng hồ kim khiến cho người dùng tiết kiệm thời gian.

Thứ hai đó chính là việc thuật toán được xây dựng dựa trên Python, vì thể có nhiều thư viện để hỗ trợ trong việc xây dựng như OpenCV, Scikit-Image, …

Và cuối cùng là Python có thể chạy đa nền tảng nên việc mở rộng trở nên dễ dàng hơn, không bị bó buộc mỗi Window nói riêng hay tất cả nền tảng nói chung.

2. Mục tiêu thực hiện đề tài

Mục tiêu thực hiện đề tài là tạo ra một thuật toán để lấy những thông tin của đồng hồ kim, từ đó trả về thông tin thời gian cụ thể lúc đó giúp những người dùng có thể biết được bây giờ là mấy giờ mấy phút hay mấy giây.

CHƯƠNG 2 – KIẾN THỨC NỀN TẢNG

1. Công thức tính khoảng cách Euclide

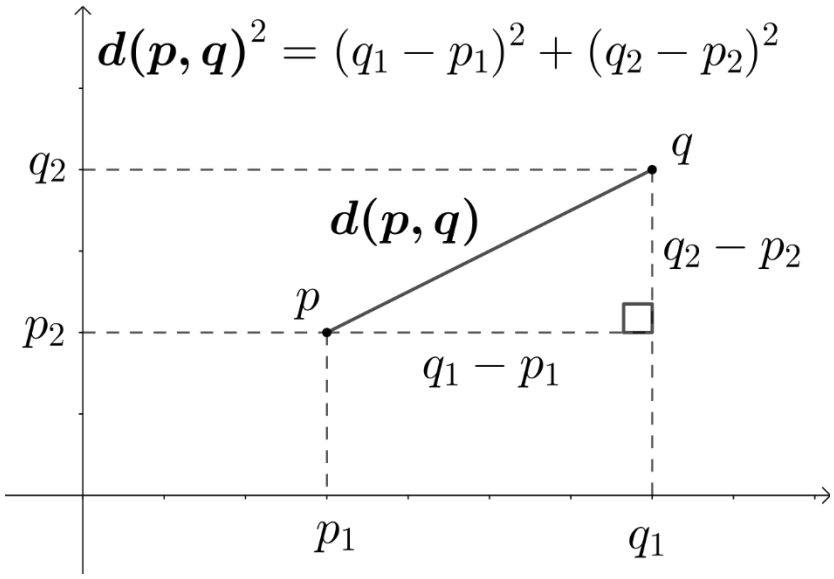
1.1 Tổng quan

Công thức tính khoảng cách Euclide là một thuật toán nền tảng xuyên suốt bài toán này vì nó sẽ ảnh hưởng đến cách mà thuật toán xử lí trong bài toán nhận diện kim đồng hồ.

1.2 Cách hoạt động

Cách hoạt động của công thức là tính khoảng cách giữa 2 điểm trong mặt phẳng 2 chiều với điều kiên đã biết được tọa độ giữa 2 điểm

1.3 Công thức



Hình 2.1: Công thức Euclide trong mặt phẳng 2 chiều

2. Thuật toán tính ngưỡng

2.1 Tổng quan

Thuật toán tính ngưỡng là một kiểu của phân đoạn ảnh, tại đó chúng ta sẽ thay đổi các giá trị của pixel dựa theo một ngưỡng nhất định để có thể phân tích ảnh một

cách dễ hơn. Nếu các giá trị pixel thấp hoặc các hơn ngưỡng, các giá trị đó sẽ được thay đổi về một giá trị khác

Thuật toán tính ngưỡng còn là một công cụ phổ biến để có thể phân tách nền trước và sau của một bức ảnh. Ngưỡng chính là giá trị có hai vùng kế nó, vùng dưới ngưỡng và vùng trên ngưỡng

2.2 Các loại ngưỡng

THRESH_BINARY	$\texttt{dst}(x,y) = \begin{cases} \texttt{maxval} & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ 0 & \text{otherwise} \end{cases}$
THRESH_BINARY_INV	$\texttt{dst}(x,y) = \begin{cases} 0 & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{maxval} & \text{otherwise} \end{cases}$
THRESH_TRUNC	$\texttt{dst}(x,y) = \begin{cases} \texttt{threshold} & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{src}(x,y) & \text{otherwise} \end{cases}$
THRESH_TOZERO	$\texttt{dst}(x,y) = \begin{cases} \texttt{src}(x,y) & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ 0 & \text{otherwise} \end{cases}$
THRESH_TOZERO_INV	$\texttt{dst}(x,y) = \begin{cases} 0 & \text{if } \texttt{src}(x,y) > \texttt{thresh} \\ \texttt{src}(x,y) & \text{otherwise} \end{cases}$

Bảng 2.1: Các loại ngưỡng trong thuật toán

Để có thể hiểu hơn về loại ngưỡng trên thì sau đây là lời giải ngắn gọn cho các loại trên:

- THRESH_BINARY: Nếu giá trị của pixel lớn hơn giá trị ngưỡng, giá trị sẽ được đưa về max value mà ta chuyền vào.
- THRESH_BINARY_INV: Là nghịch đảo của thuật toán ở trên.
- THRESH_TRUNC: Nếu giá trị của pixel lớn hơn giá trị ngưỡng, thì nó sẽ bị cắt giảm xuống giá trị ngưỡng. Còn các giá trị khác giữ nguyên

- THRESH_TOZERO: Các giá trị pixel nhỏ hơn giá trị ngưỡng thì nó sẽ chuyển về 0, còn lại giữ nguyên
- THRESH_TOZERO_INV: Nghịch đảo của THRESH_TOZERO

Ngoài các loại ngưỡng trên ra chứng ta còn có những loại khác như THRESH_OTSU, THRESH_MASK, THRESH_TRIANGLE.

2.3 Công thức chung

Đây là công thức chung áp dụng khi ta sử dụng Python để thực hiện:

```
Python:
    cv.threshold( src, thresh, maxval, type[, dst] ) -> retval, dst
```

Hình 2.2: Công thức ngưỡng thích ứng

Tham số truyền vào gồm:

- src: mảng truyền vào (8-bit hoặc 32-bit số thập phân)
- dst: một mảng khác cùng kiểu và chiều với mảng truyền vào
- thresh: giá trị ngưỡng
- maxval: giá trị tối đa khi sử dụng với các loại tính ngưỡng THRESH_BINARY và THRESH_BINARY_INV
- type: loại ngưỡng mà chúng ta sẽ sử dụng

2.4 Minh họa

Giả sử chúng ta chúng ta truyền vào một tấm hình như sau:

Hình 2.3: Ảnh gốc chưa xử lí ngưỡng

Thì dựa trên 5 thuật toán như trên ta có thể suy ra kết quả như sau:

Hình 2.4: Hình ảnh sau khi đã qua xử lí ngưỡng

3. Thu gọn vùng ảnh (skeletonize)

3.1 Tổng quan

Thu gọn vùng ảnh hay còn gọi là skeletonize là một cách để c có thể thu gọn những vùng ảnh lớn lại trrong ảnh nhị phân. Mục tiêu của thuật toán này là để tạo một biểu diễn đơn giản hóa của hình ảnh đễ dễ dàng xử lí

3.2 Cách hoạt động

Thuật toán hoạt động bằng cách thực hiện các lần truyền hình ảnh liên tiếp. Ở những lần đó, các pixel ở viền sẽ được xác định và loại bỏ với điều kiện chúng không làm đứt đoạn hình ảnh của đối tượng

3.3 Công thức

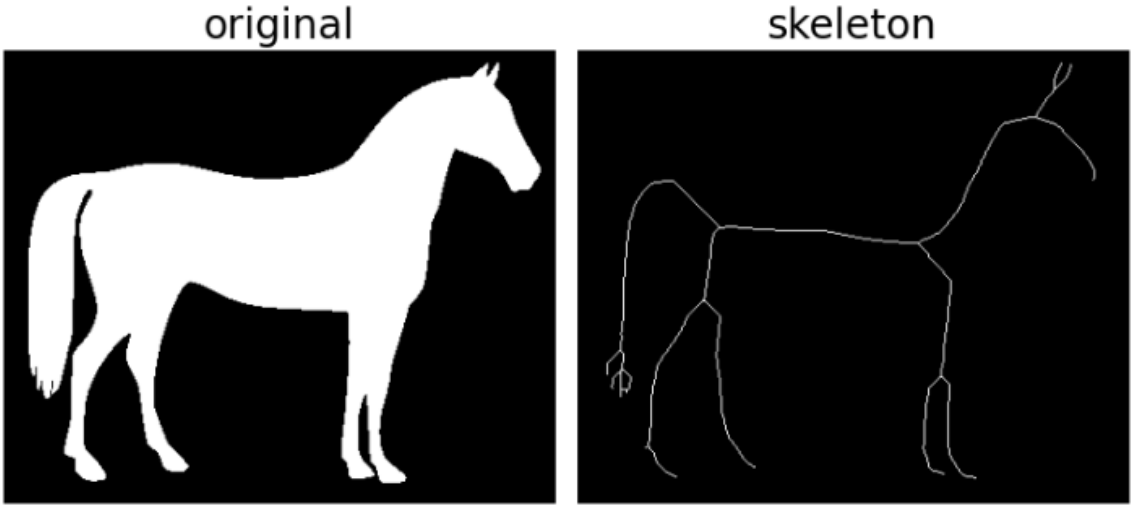
```
skimage.morphology.skeletonize(image, *, method=None) #
```

Hình 2.5: Công thức skeletonize

Tham số truyền vào:

- image: (ndarray, 2D hay 3D) hình chứa các vật thể cần thu gọn, giá trị 0 là nền còn khác 0 là mặt trước.
- method: (zhang hoặc là lee) Thuật toán để sử lí. Thuật toán Zhang là chỉ dành cho ảnh 2D và mặc định là ảnh 2D. Thuật toán của Lee dùng cho cả ảnh 2D và 3D nhưng thường mặc định là ảnh 3D.
- Kết quả trả về: Một bức ảnh đã được thu gọn

3.4 Minh họa



Hình 2.6: Ảnh đã qua xử lí skeletonize

4. Hough Line Transformation

4.1 Tổng quan

Hough Line Transform là một kỹ thuật rất quan trọng và đóng vai trò thiết yếu xuyên suốt bài toán này. Thuật toán này được xây dựng để nhận diện các đường thẳng trong ảnh dưới dạng tham số.

4.2 Cách hoạt động

Như đã biết thì đường thẳng có thể biểu diễn dưới dạng hai giá trị. Ví dụ

- Trong hệ trục tọa độ Cartesian, tham số là $(\boldsymbol{m}, \boldsymbol{b})$

- Trong hệ trục tọa độ Polar, tham số là $(\boldsymbol{r},\ \boldsymbol{\theta})$

Với Hough Transform, chúng ta sẽ biểu diễn nó theo hệ Polar. Vì vậy, chúng ta có đẳng thức sau.

$$\boldsymbol{y} = \left(-\frac{\boldsymbol{cos\theta}}{\boldsymbol{sin\theta}}\right)\boldsymbol{x} + \left(\frac{\boldsymbol{r}}{\boldsymbol{sin\theta}}\right)$$

Điều chỉnh hàm $\boldsymbol{r = xcos\theta + ysin\theta}$

Đầu tiên, với mỗi điểm $(\boldsymbol{x_0, y_0})$, ta có thể xác định được các đường đi qua điểm đó với công thức như sau

$$\boldsymbol{r_\theta = x_0 \times cos\theta + y_0 \times sin\theta}$$

Có nghĩa là đối với mỗi cặp $(\boldsymbol{r_\theta, \theta})$ sẽ tượng trưng cho một đường thẳng đi qua điểm đó

Và với mỗi điểm như vậy ta sẽ có một đồ thị hình sin. Như vậy khi ta lập lại việc này cho toàn bộ điểm trên ảnh nếu những đường cong của các điểm đó giao nhau trên mặt phẳng $\boldsymbol{\theta - r}$, có nghĩa là các điểm đó thuộc về một đường thẳng. Điều đó có nghĩa là các đường thẳng có thể được xác định bằng cách tìm giao điểm giữa những đường cong đó. Càng nhiều đường cong giao nhau có nghĩa là đường thẳng của những giao điểm đó chứa nhiều điểm. Ngoài ra, ta còn có thể định nghĩa một ngưỡng cho số lượng giao điểm cần có để tạo thành một line.

Ngoài ra, OpenCV còn hỗ trợ hai loại Hough Line Transformation

- Hough Transform bình thường
- Hough Transform Probabilistic

4.3 Công thức

Do OpenCV có hỗ trợ hai loại Hough Line Transform khác nhau nên ta có hai công thức khác nhau dựa trên đó.

Standard Hough Line Transform

Python:

cv.HoughLines(image, rho, theta, threshold[, lines[, srn[, stn[, min_theta[, max_theta]]]]]) -> lines

Hình 2.7: Công thức Standard Hough Line Transform

Tham số truyền vào gồm:

- image: ảnh nhị phân một kênh màu và 8 bit
- lines: một chuỗi vector chứa các đường
- rho: độ phân giải khoảng cách của bộ tích lũy tính bằng vector
- theta: độ phân giải góc của bộ tích lũy tính bằng gradian
- threshold: tham số ngưỡng tích lũy, chỉ những đường đủ tiêu chuẩn mới được trả về (> threshold)
- srn: đối với phép biến đổi Hough đa tỷ lệ, nó là ước số cho độ phân giải khoảng cách rho. Độ phân giải khoảng cách tích lũy thô là rho và độ phân giải chính xác của bộ tích lũy là rho/srn. Nếu cả srn=0 và stn=0 thì phép biến đổi Hough cổ điển được sử dụng. Ngược lại, cả hai tham số này phải dương.
- stn: đối với phép biến đổi Hough đa tỷ lệ, nó là ước số cho độ phân giải khoảng cách theta
- min_theta: Đối với phép biến đổi Hough tiêu chuẩn và đa tỷ lệ, góc tối thiểu để kiểm tra các đường. Phải nằm trong khoảng từ 0 đến max_theta.
- max_theta: Đối với phép biến đổi Hough tiêu chuẩn và đa tỷ lệ, giới hạn trên của góc. Phải nằm trong khoảng min_theta và CV_PI. Góc tối đa thực tế trong bộ tích lũy có thể nhỏ hơn max_theta một chút, tùy thuộc vào tham số min_theta và theta.

Probabilistic Hough Line Transform

Python:

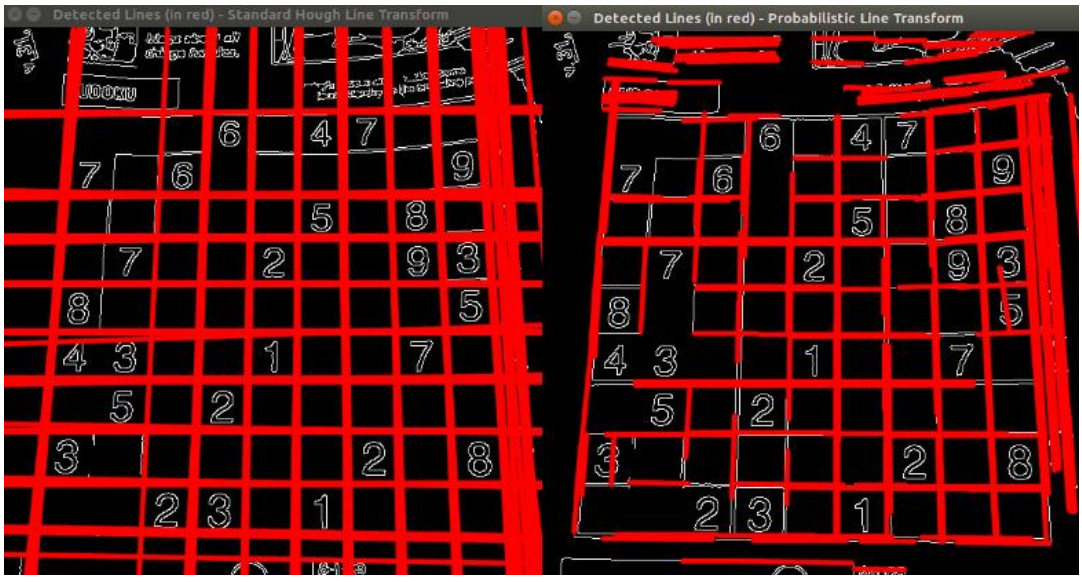
cv.HoughLinesP(image, rho, theta, threshold[, lines[, minLineLength[, maxLineGap]]]) -> lines

Hình 2.8: Công thức Probabilistic Hough Line Transform

Tham số truyền vào gồm:

- image: ảnh nhị phân một kênh màu và 8 bit
- lines: một chuỗi vector chứa các đường
- rho: độ phân giải khoảng cách của bộ tích lũy tính bằng vector
- theta: độ phân giải góc của bộ tích lũy tính bằng gradian
- threshold: tham số ngưỡng tích lũy, chỉ những đường đủ tiêu chuẩn mới được trả về (> threshold)
- minLineLength: khoảng cách tối thiểu của một đường. Những đường nhỏ hơn sẽ bị loại bỏ
- maxLineGap: Khoảng cách tối đa chho phép giữa các điểm trên cùng một đường để liên kết chúng

4.4 Minh họa



Hình 2.9: Ảnh xử lí bằng 2 phương thức Hough Line khác nhau

CHƯƠNG 3 – XÂY DỰNG THUẬT TOÁN

1. Mô hình và các bước thực hiện

Bài toán được xây dựng dựa trên các bước như sau sau:

Bước 1: Lấy ảnh tham số đầu vào

Bước 2: Chuyển ảnh sang kênh màu xám

Bước 3: Tính ngưỡng của ảnh và đưa ảnh về ảnh nhị phân

Bước 4: Đảo ngược màu ảnh

Bước 5: Tìm các đường nét của ảnh và lưu khu vực của nó sau đó tiến hành sắp xếp nó theo khu vực

Bước 6: Lấy ba cái đường nét lớn nhất để thu gọn nó bằng skeletonize

Bước 7: Lấy Probabilistic Hough Line của những đường nét đã qua xử lí

Bước 8: Gộp những đường gần như trùng nhau do một số sai sót trong hough line

Bước 9: Gán tên cho các đường thẳng đã lấy được theo độ dài thứ tự giảm dần lần lượt là kim giây, phút và giờ

Bước 10: Lấy thời gian dựa trên những kim đó và xuất nó ra

2. Xây dựng các hàm sử dụng trong bài

Để có thể xây dựng thuật toán theo các bước sau ta sẽ cần những hàm riêng biệt đóng vai trò nền tảng trong một cấu trúc tổng thể của thuật toán. Sau đây là các hàm sau

2.1 Hàm tính khoảng cách Euclide

```
def euclidean_distance(line1, line2):
    midpoint1 = [(line1[0][0] + line1[0][2]) / 2, (line1[0][1] + line1[0][3]) / 2]
    midpoint2 = [(line2[0][0] + line2[0][2]) / 2, (line2[0][1] + line2[0][3]) / 2]
    return np.linalg.norm(np.array(midpoint1) - np.array(midpoint2))
```

Hình 3.1: Hàm euclidean_distance để tính khoảng cách Euclide

Trong hàm này, tham số truyền vào sẽ là hai đường thẳng cần tính khoảng cách có cấu trúc như sau $[\boldsymbol{x_1}, \boldsymbol{y_1}. \boldsymbol{x_2}. \boldsymbol{y_2}]$ với $(\boldsymbol{x_1}, \boldsymbol{y_1})$ và $(\boldsymbol{x_2}, \boldsymbol{y_2})$ là tọa độ 2 điểm đầu và cuối của đường thẳng. Trong đó, hàm sẽ lấy trung điểm của hai đường đó và dùng nó để tính khoảng cách giữa hai điểm đó và trả về kết quả

2.2 Hàm tính góc giữa hai đường thẳng

```
def angle_between_lines(line1, line2):
    angle1 = np.arctan2(line1[0][3] - line1[0][1], line1[0][2] - line1[0][0])
    angle2 = np.arctan2(line2[0][3] - line2[0][1], line2[0][2] - line2[0][0])
    return np.abs(angle1 - angle2) * (180 / np.pi)
```

Hình 3.2: Hàm angle_between_lines để tính góc giữa 2 đường thẳng

Trong hàm này, tham số truyền vào là hai đường thẳng có cấu trúc như sau $[x_1, y_1. x_2. y_2]$ với (x_1, y_1) và (x_2, y_2) là tọa độ 2 điểm đầu và cuối của đường thẳng. Đầu tiên ta sẽ sử dụng hàm arctan2 của thư viện numpy để tính giá trí arctan đường thẳng đó, trong ngữ cảnh này nó sẽ tính góc giữa đường thẳng và trục ngang. Cuối cùng trả về giá trị tuyệt đối góc giữa 2 đường thẳng đó với đơn vị là độ.

2.3 Hàm gộp hai đường thẳng Hough Line

```python
def merge_close_hough_lines(lines, distance_threshold, angle_threshold):
    merged_lines = []

    while len(lines) > 0:
        current_line = lines[0]
        group = [current_line]

        i = 0
        while i < len(lines):
            distance = euclidean_distance(current_line, lines[i])
            angle_diff = angle_between_lines(current_line, lines[i])
            # print("Distance",distance)
            # print("Angle",angle_diff)
            if (distance < distance_threshold) and (angle_diff < angle_threshold):
                group.append(lines[i])
                lines = np.delete(lines, i, axis=0)
            else:
                i += 1

        # Merge lines in the group
        if len(group) > 1:
            merged_line = np.mean(np.array(group), axis=0).astype(int)
            merged_lines.append(merged_line.tolist())
        else:
            merged_lines.append(group[0][0].tolist())

    return merged_lines
```

Hình 3.3: Hàm merge_close_hough_lines để gộp đường thẳng

Trong hàm này, ta sẽ giải thích như sau:

- Đầu tiên, hàm nhận vào một list chứa các đường thẳng, giá trị ngưỡng của khoảng cách và góc
- Tiếp theo tạo một list rỗng để chứa các đường đã gộp
- Tạo một vòng lặp để lấy dữ liệu từ những đường thẳng cho đến khi nó rỗng. Trong vòng lặp đó, các đường thẳng sẽ được lấy ra và tạo ra một list tên group để chứa nó

- Tạo một vòng lặp khác để lặp qua các đường thẳng còn lại để kiểm tra tính tương đồng giữa hai đường đó. Tại đây ta sẽ xét theo hai tiêu chí: distance_threshold để gộp những đường thẳng cách nhau một khoảng nhỏ hơn ngưỡng đó và angle_threshold để gộp những đường thẳng tạo thành một góc nhỏ hơn. Và nếu hai đường thẳng đó thỏa các 2 điều kiên trên. Đường thẳng đang được lấy ra để so sánh sẽ được loại khỏi list
- Sau khi đã đi hết toàn bộ đường thẳng còn lại, ta sẽ gộp những đường thẳng đó lại.
- Và lặp lại cho đến khi hết danh sách đường thẳng truyền vào, sau đó trả về danh sách các chuỗi đã gộp

2.4 Hàm tìm giao điểm giữa 2 đường

```
def get_intersection_points_from_lines(lines):
    intersection_points = []

    for i in range(len(lines)):
        for j in range(i + 1, len(lines)):
            line1 = lines[i]
            line2 = lines[j]

            x1, y1, x2, y2 = line1[0]
            x3, y3, x4, y4 = line2[0]

            # Check if lines are not parallel (slope condition)
            if (x2 - x1) * (y4 - y3) != (x4 - x3) * (y2 - y1):
                # Solve for intersection point using linear algebra
                A = np.array([
                    [y2 - y1, x1 - x2],
                    [y4 - y3, x3 - x4]
                ])

                B = np.array([x1 * (y2 - y1) - y1 * (x2 - x1), x3 * (y4 - y3) - y3 * (x4 - x3)])

                intersection_point = np.linalg.solve(A, B)
                intersection_points.append((int(intersection_point[1]), int(intersection_point[0])))

    return intersection_points
```

Hình 3.4: Hàm get_intersection_point_from_lines để tìm giao điểm của 2 đường

Trong hàm này, tham số truyền vào là hai đường thẳng có cấu trúc như sau $[x_1, y_1. x_2. y_2]$ với (x_1, y_1) và (x_2, y_2) là tọa độ 2 điểm đầu và cuối của đường thẳng. Đầu tiền hàm sẽ kiểm tra hai đường thẳng có song song không, nếu có thì trả vè None còn không thì thực hiện bước tiếp theo đó là trả về kết quả giao điểm của hai đường thẳng đó

2.5 Hàm lấy thời gian từ kim đồng hồ

```
def get_time_from_clock_hands(hour_hand, minute_hand, second_hand ):
    # Calculate the angles of the clock hands with respect to the 12 o'clock position
    hour_angle = np.arctan2(hour_hand[0][3] - hour_hand[0][1], hour_hand[0][2] - hour_hand[0][0]) * (180 / np.pi)
    minute_angle = np.arctan2(minute_hand[0][3] - minute_hand[0][1], minute_hand[0][2] - minute_hand[0][0]) * (180 / np.pi)
    second_angle = np.arctan2(second_hand[0][3] - second_hand[0][1], second_hand[0][2] - second_hand[0][0]) * (180 / np.pi)

    print(hour_angle)
    print(minute_angle)
    print(second_angle)
    # Normalize angles to be positive
    hour_angle = (hour_angle + 450) % 360
    minute_angle = (minute_angle + 450) % 360
    second_angle = (second_angle + 450) % 360

    # Convert angles to time values
    hour = int((hour_angle / 30) % 12)  # Each hour is 30 degrees, and modulus 12 for 12-hour clock
    minute = int(minute_angle / 6)  # Each minute is 6 degrees
    second = int(second_angle / 6)  # Each second is 6 degrees

    return hour, minute, second
```

Hình 3.5: Hàm lấy thời gian

Trong hàm này ta sẽ truyền lần lượt các kim đồng hồ đã được nhận dạng trước đó để có thể lấy thời gian. Tại đây các đường thẳng với định dạng $[x_1, y_1. x_2. y_2]$ với (x_1, y_1) và (x_2, y_2) là tọa độ 2 điểm đầu và cuối của đường thẳng sẽ được tính góc arctan của nó, sau đó các góc đó được cộng thêm 450 độ thay vì 360 vì arctan sẽ lấy theo trục ngang nên các kim sẽ bị lệch. Sau đó chia theo tỉ lệ của kim đồng hồ ta ra được kết quả.

3. Ứng dụng các hàm giải quyết bài toán

Khi đã có được các hàm cần thiết ta sẽ thực hiện thuật toán theo các bước một cách chi tiết như sau:

Bước 1: Đọc hình ảnh đầu vài và chuyển nó sang ảnh xám\

```
# Read image
img = cv2.imread('clock_image3.jpg')
hh, ww = img.shape[:2]

# convert to gray
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
```

Hình 3.6: Đọc hình ảnh và chuyển sang ảnh xám

Bước 2: Tính ngưỡng để ảnh chuyển về ảnh nhị phân và đảo ngược màu của ảnh

```
# threshold
thresh = cv2.threshold(gray, 200, 255, cv2.THRESH_BINARY)[1]

# invert so shapes are white on black background
thresh = 255 - thresh
```

Hình 3.7: Xử lí ngưỡng của ảnh

Bước 3: Tìm từng đường nét của ảnh và sắp xếp nó theo khu vực của mình

```
# get contours and save area
cntrs_info = []
contours = cv2.findContours(thresh, cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)
contours = contours[0] if len(contours) == 2 else contours[1]
index=0
for cntr in contours:
    area = cv2.contourArea(cntr)
    cntrs_info.append((index,area))
    index = index + 1

# sort contours by area
def takeSecond(elem):
    return elem[1]
cntrs_info.sort(key=takeSecond, reverse=True)
```

Hình 3.8: Lấy đường nét của ảnh và sắp xếp nó theo khu vực

Bước 4: Tạo một mảng arms với kích thước bằng ảnh gốc nhưng toàn bộ giá trị là 0. Sau đó lấy 3 đường nét có khu vực lớn nhất sau đó rồi vẽ ra trên mảng arms với toàn bộ giá trị là 1. Sau đó ta sẽ thu gọn những đường nét đó và chuyển các giá trị của ảnh đó sang các giá trị pixel là từ 0 đến 255

```
# get third largest contour
arms = np.zeros_like(thresh)
index_third = cntrs_info[2][0]
cv2.drawContours(arms,[contours[index_third]],0,(1),-1)

arms_thin = skeletonize(arms)
arms_thin = (255*arms_thin).clip(0,255).astype(np.uint8)
```

Hình 3.9: Lấy xương sống của những cây kim trong đồng hồ

Bước 5:

```
if merged_lines is not None:
    # Sort merged lines by length
    merged_lines.sort(key=lambda x: np.linalg.norm(np.array([x[0][0], x[0][1]]) - np.array([x[0][2], x[0][3]])),
                      reverse=True)
    # Assign labels to lines
    labels = ["second", "minute", "hour"]
    labeled_lines = dict(zip(labels, merged_lines))
    # Draw the clock hands and label them by length
    colors = [(0, 255, 0), (255, 0, 0), (0, 0, 255)]
    for label, l in labeled_lines.items():
        color = colors[labels.index(label)]
        cv2.line(result, (l[0][0], l[0][1]), (l[0][2], l[0][3]), color, 3, cv2.LINE_AA)
        cv2.putText(result, label, (l[0][2], l[0][3]), cv2.FONT_HERSHEY_SIMPLEX, 0.5, color, 2)

    hour, minute, second = get_time_from_clock_hands(
        labeled_lines["hour"],
        labeled_lines["minute"],
        labeled_lines["second"]
    )

    cv2.putText(result, f"{hour:02d}:{minute:02d}:{second:02d}", (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)


    cv2.imshow('result', result)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
else:
    print("Not enough lines found to determine clock hands.")
```

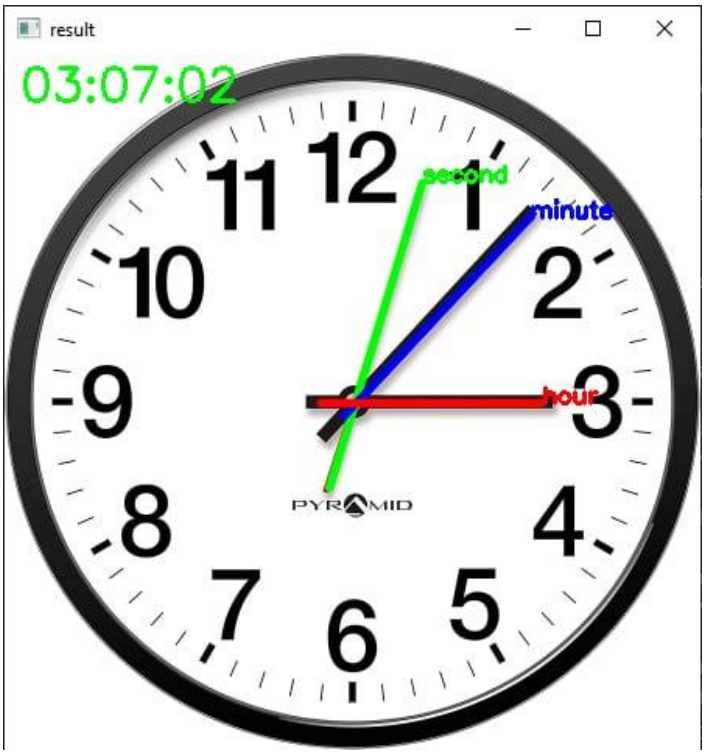
Hình 3.10: Bước thực hiện cuối cùng của bài toán

Ở bước cuối cùng, thuật toán sẽ được thực hiện theo trình tự sau:

- Lần 1: Ta sẽ thực hiện tạo một điều kiện để kiểm tra xem có đủ kim để truy xuất thời gian hay không
- Lần 2: Sắp xếp những đường thẳng theo độ dài của nó
- Lần 3: Dán nhãn kim giờ, phút và giây cho những đường thẳng đó, vì những kim đó được xếp theo thứ tự từ dài tới ngắn nên ta sẽ gán kim giây tới kim phút rồi mới kim giờ ngoài ra còn chọn màu cho mỗi loại
- Lần 4: In những đường thẳng trên những cây kim đó và ghi tên cho chúng
- Lần 5: Lấy giờ từ những cây kim trên và sau đó in lên ảnh gốc

4. Kết quả đầu ra

Sau đây là kết quả đầu ra của một đồng hồ đã được truy xuất hình ảnh



Hình 3.11: Kết quả đầu ra

CHƯƠNG 4 – KẾT LUẬN

1. Kết quả đạt được

Thông qua bài này, em đã đạt được những thành tựu sau:

- Lên kế hoạch xây dựng mô hình thuật toán
- Thiết kế thuật toán trích xuất ảnh từ đồng hồ bằng Python và thư viện của ngôn ngữ đó

Từ việc sử dụng các thư viện trong OpenCV và Scikit-Image, em đã thực hiện được việc nhận diện kim từ đồng hồ và từ đó tính được thời gian.

2. Những nhược điểm của thuật toán

Sau đây là những nhược điểm mà thuật toán gặp khó khăn:

Do có rất nhiều ảnh có định dạng khác nhau nên việc chọn ra một ngưỡng xử lí chung cho toàn bộ hình gần như rất khó.

Đối với những hình đồng hồ có những loại kim đặc biệt thì rất khó như không thể nhận diện được.

Một số khó khăn trong việc xử lí góc của những cây kim vì nó có thể tính sai góc của hàm arctan2 trong thư viện Numpy

3. Định hướng

Dựa trên những tìm hiểu của em về vấn đề này, để có thể cải tiến thuật toán ta có thể dùng những cách như sau:

- Áp dụng các mô hình học máy
- Sử dụng thuật toán khác tối ưu hơn trong việc nhận diện đường thẳng

Vì vậy trong tương lai, em sẽ cố gắng tìm hiểu kỹ hơn về những vấn đề trên để có thể hoàn toàn giải quyết vấn đề này cũng như có thêm kiến thức về xử lí ảnh

TÀI LIỆU THAM KHẢO

Tiếng Việt

Tiếng Anh

1. Tài liệu về Skeletonize của thư viện Scikit Image. Truy cập ngày 17/12/2023
2. Tài liệu về Hough Line của thư viện OpenCV. Truy cập ngày 18/12/2023