شناسایی عقربه های ساعت:

در این پروژه می خواهیم ببینیم آیا می توانیم عقربه های یک ساعت را به عنوان یک خط در الگوریتم هاف شناسایی کنیم؟

پس از خواندن کتابخانه های مورد نیاز عکس مورد نظر را می خوانیم. و سپس آن را به یک عکس باینری تبدیل می کنیم.

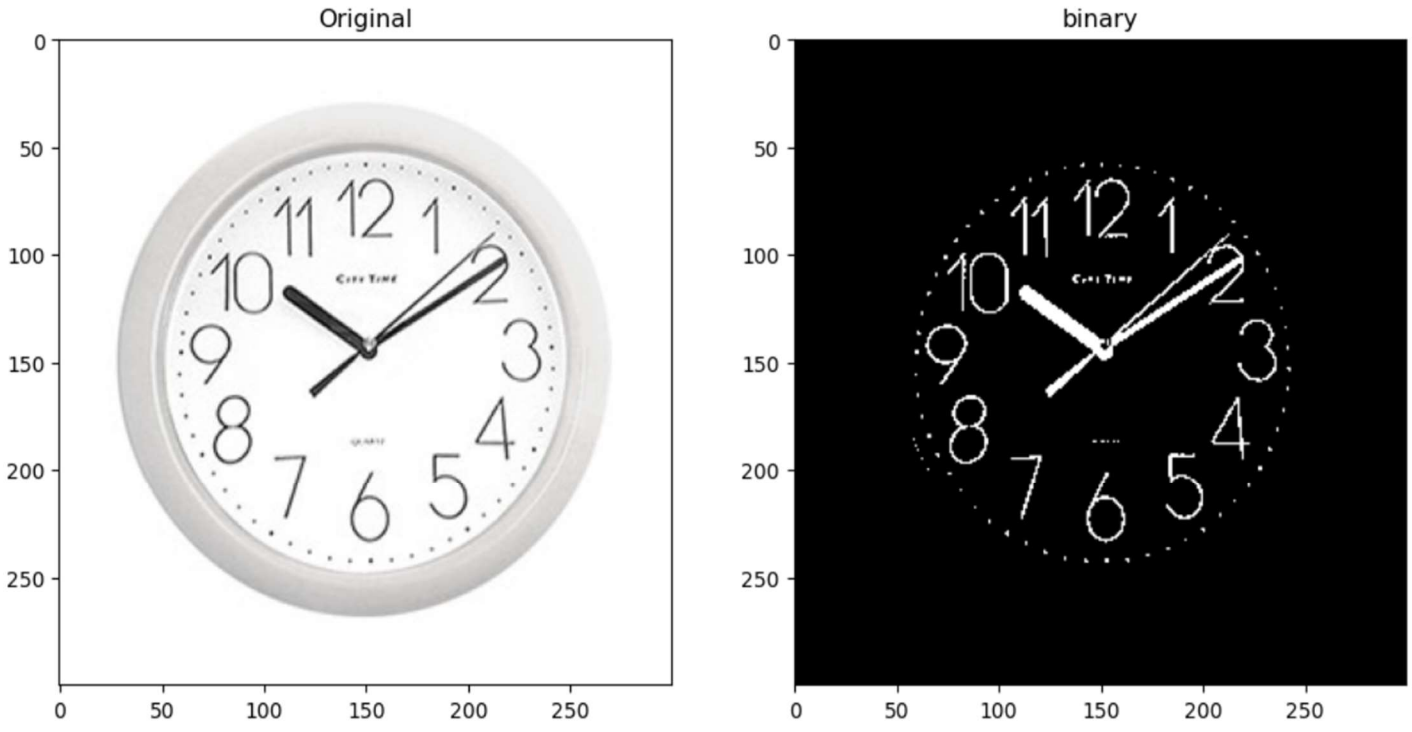
```
# Read image
image = cv2.imread("images/clock.jpg")


# Convert BGR back to grayscale:
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Threshold via Otsu:
_, binary = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)

plt.figure(figsize=[12,7])
plt.subplot(121);plt.imshow(image[...,::-1]);plt.title("Original");
plt.subplot(122);plt.imshow(binary, cmap='gray');plt.title("binary");
```

خروجی به این شکل خواهد بود:



در مرحله بعد به سراغ عملیات مورفولوژی می رویم:

```
# Get the structuring element:
structuringElement = cv2.getStructuringElement(cv2.MORPH_RECT, (3, 3))
opening = cv2.morphologyEx(binary, cv2.MORPH_OPEN, structuringElement)
plt.imshow(opening, cmap='gray')
```

عملیات مورفولوژی روی تصویر باینری انجام شد تا عقربه‌های ساعت (که به صورت خطوط باریک هستند) از نویزها جدا و شفاف‌تر شوند و بعد الگوریتم هاف بتواند آن‌ها را به عنوان "خط" شناسایی کند. وقتی تصویر ساعت را آستانه‌گذاری (Otsu) می‌کنید، همه بخش‌های روشن/تیره به صورت نواحی سیاه و سفید در می‌آیند. در این حالت:

- عقربه‌ها به شکل خطوط سفید باریک در زمینهٔ سیاه دیده می‌شوند.
- ولی اطرافشان ممکن است نویزهای کوچک (نقاط سفید اضافی) یا شکستگی در خطوط وجود داشته باشد.

چرا (Opening)؟

- **فرسایش :(erosion)** نقاط نویزی کوچک از بین می‌روند.
- **اتساع :(dilation)** عقربه‌ها دوباره به حالت خط باریک ولی پیوسته برمی‌گردند.

نتیجه:

- عقربه‌ها که ساختار کشیده و خطی دارند، سالم می‌مانند.
- نویزهای ریز که ساختار کوچک دارند، حذف می‌شوند.

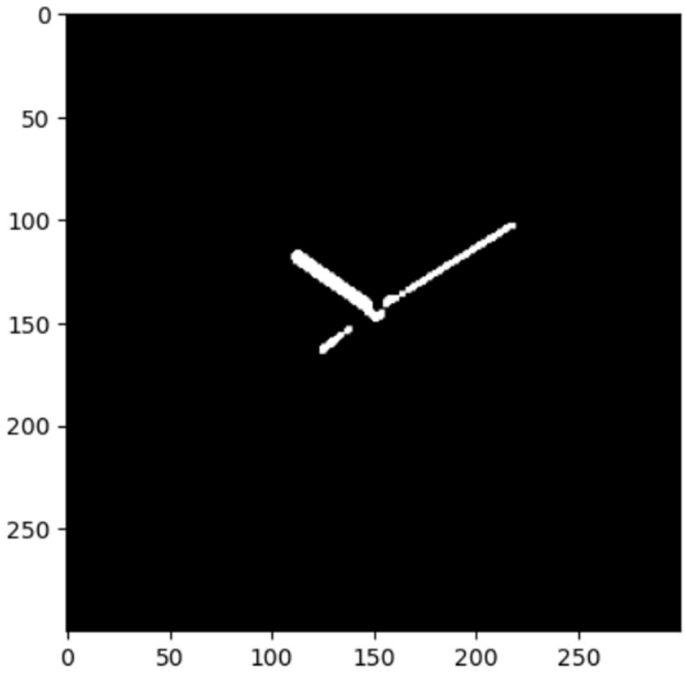
الگوریتم **HoughLinesP** فقط زمانی خوب کار می‌کند که ورودی‌اش شامل خطوط واضح و پیوسته باشد. اگر تصویر پر از نویز یا خطوط شکسته باشد:

- خطای تشخیص زیاد می‌شود.
- عقربه‌ها ممکن است ناقص یا چند خط جداگانه تشخیص داده شوند.

بنابراین، عملیات مورفولوژی در اینجا برای **آماده‌سازی تصویر** است تا عقربه‌ها به عنوان خطوط کامل دیده شوند و قابل شناسایی باشند.

خروجی عملیات مورفولوژی به شکل زیر است:

این خروجی ساعتی را نمایش میدهد که تنها عقربه های آن مشخص هستند.



پس از عملیات مور فولوژی نوبت به هاف و عملیات اصلی می رسد.

```
# Set HoughLinesP parameters:
lineThresh = 50
minLineLength = 20
maxLineGap = 100

# Run the line detection:
lines = cv2.HoughLinesP(opening, 1, np.pi/180 , lineThresh, None, minLineLength, maxLineGap)

# Prepare some lists to store every coordinate of the detected lines:
X1 = []
X2 = []
Y1 = []
Y2 = []
draw_image = image.copy()
# Store and draw the lines:
```

```
for [line] in lines:
    # First point:
    x1 = line[0]
    y1 = line[1]
    X1.append(x1)
    Y1.append(y1)

    # Second point:
    x2 = line[2]
    y2 = line[3]
    X2.append(x2)
    Y2.append(y2)

    # Draw the lines:
    cv2.line(draw_image, (x1,y1), (x2,y2), (0,0,255), 2)
    cv2.imshow("Lines", draw_image)
    cv2.waitKey(0)
cv2.destroyAllWindows()
plt.imshow(draw_image[...,::-1], cmap='gray')
```

بیایید این کد را با هم تشریح کنیم:

```
lineThresh = 50
minLineLength = 20
maxLineGap = 100
```

این سه پارامتر مشخص می‌کنند الگوریتم هاف چه خطوطی را پیدا کند:

- **`lineThresh`آستانه رأی‌دهی = 50**

در الگوریتم هاف هر خط باید به‌اندازه کافی "رأی" بگیرد تا معتبر شناخته شود.
یعنی حداقل ۵۰ پیکسل هم‌خط باید وجود داشته باشند.
اگر خیلی کوچک انتخاب شود ⟵ نویز هم به عنوان خط تشخیص داده می‌شود.
اگر خیلی بزرگ انتخاب شود ⟵ ممکن است بعضی عقربه‌ها از دست بروند.

- **`minLineLength`حداقل طول خط = 20 پیکسل**

فقط خطوطی شناسایی می‌شوند که حداقل ۲۰ پیکسل طول داشته باشند.
این کمک می‌کند تا خطوط خیلی کوتاه ناشی از نویز حذف شوند.
عقربه‌های ساعت معمولاً بلند هستند، پس این شرط آسیبی به آن‌ها نمی‌زند.

- **maxLineGap حداکثر فاصله = 100 پیکسل**

اگر یک خط شکسته باشد ولی فاصله‌ی بین قطعات آن کمتر از ۱۰۰ پیکسل باشد، هاف آن‌ها را یک خط واحد در نظر می‌گیرد.

این خیلی مهم است چون عقربه‌ها ممکن است در باینری یا مورفولوژی کمی قطع و وصل شده باشند.

```
lines = cv2.HoughLinesP(opening, 1, np.pi/180 , lineThresh, None, minLineLength, maxLineGap)

```

- **opening**: تصویر ورودی (بعد از مورفولوژی).
- **1**: دقت فاصله (رزولوشن ۱ پیکسل).
- **np.pi/180**: دقت زاویه (۱ درجه).
- **lineThresh**: حداقل تعداد رأی لازم.
- **None**: به پارامتر اختیاری نیاز نداریم.
- **minLineLength** و **maxLineGap**همانطور که بالا بیان شد.

```
X1 = []
X2 = []
Y1 = []
Y2 = []
draw_image = image.copy()
# Store and draw the lines:
for [line] in lines:
    # First point:
    x1 = line[0]
    y1 = line[1]
    X1.append(x1)
    Y1.append(y1)

    # Second point:
    x2 = line[2]
    y2 = line[3]
    X2.append(x2)
    Y2.append(y2)

```

خروجی linesیک آرایه از مختصات خطوط است بنابراین می توانیم مختصات خطوط را در لیست های متفاوتی ذخیره کنیم.

اینجا چهار لیست ساخته می‌شود تا مختصات نقاط ابتدایی و انتهایی هر خط جداگانه ذخیره شوند:

- `X1, Y1`: مختصات نقطه اول خط.
- `X2, Y2`: مختصات نقطه دوم خط.

این کار کمک می‌کند بعداً بتوانیم با داده‌ها محاسبات انجام دهیم (مثلاً طول خط، زاویه خط، پیدا کردن عقربه کوتاه/بلند و غیره).

خروجی `cv2.HoughLinesP` معمولاً آرایه‌ای با شکل **(N, 1, 4)** است) گاهی هم (N,4) بسته به نسخه:(OpenCV

- هر «خانه» نشان‌دهنده یک خط است.
- آن ۴ مقدار عبارت‌اند از **x1, y1, x2, y2** دو سر خط

چرا `for [line] in lines:`؟

- هر عنصر `lines` خودش یک آرایه ۱تایی است مثل `[[x1, y1, x2, y2]]`.
- نوشتن `[line]` یعنی «آن آرایه ۱تایی را باز کن و محتوایش را بده به متغیر line
- بعد از این بازکردن، `line` تبدیل می‌شود به آرایه **(4,)** یعنی دقیقاً `[x1, y1, x2, y2]`.
- به همین خاطر داخل حلقه می‌توانی بگویی `line[0], line[1], line[2], line[3]`.
- **استخراج مختصات :** چهار عدد را از `line` می‌گیرد. این‌ها نوعاً `int32` هستند و مستقیماً برای رسم مناسب‌اند.
- **ذخیره در لیست‌ها :** داشتن لیست‌های X1/Y1/X2/Y2 به شما اجازه می‌دهد بعداً طول، زاویه، فیلتر یا خوشه‌بندی خطوط را انجام دهید (مثلاً تشخیص عقربه کوتاه/بلند).
- **رسم :** هر بار همان تصویرِ `draw_image` (کپی تصویر اصلی) به‌روز می‌شود و خط جدید رویش افزوده می‌شود.