

Automatic Clock/Watch Time Reading

2022/00/00

Environment

- ▶ OpenCV 4.5.5
 - ▶ C++
 - ▶ Visual Studio
-
- ▶ It is recommended to use Python as your programming language. Python is the most popular language people use now, it has many convenient functions, and there are enormous resources available on the Internet.

Process (First Method)

Pre-Process

- Load the image, resize the image→convert to gray scale→edge detection.

Find the Clock Face

- Find the possible clock faces using *HoughCircles* function, then recognize the most possible one as our clock face.

Find the Hands

- Find the most possible lines as the clock hands using *HoughLinesP* function, filter useless lines, and merge the lines that are possibly from the same hand.

Calculate Time

- Sort the hands by their length, the hands should be minute, second, and hour from long to short, and calculate the time using their angles.

Correct Errors

- Correct the possible calculation errors in the previous step by logic, and obtain more accurate time.

Result Output

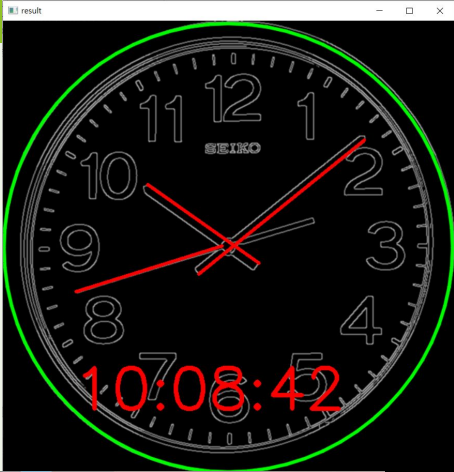
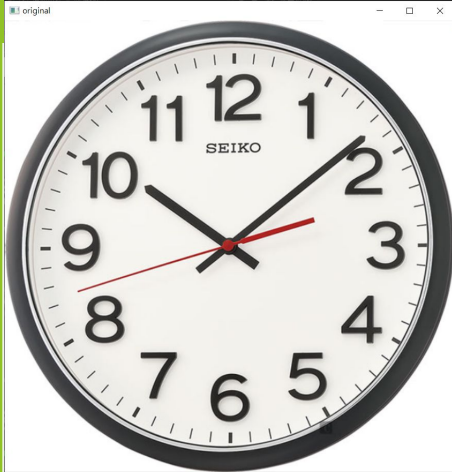
- Draw the results from previous steps on the image, show the image and output the data to console.

Results

Original Images

Processed Results

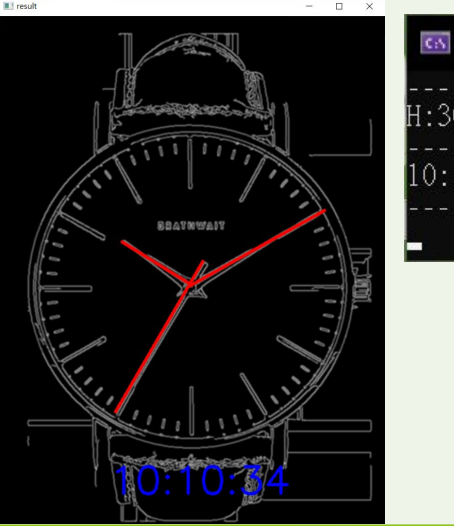

Outputs:
Angles
Calculated Time



C. - □ ×

H:305, M:51, S:252

10:08:42



C... - □ ×

H:300, M:60, S:209

10:10:34



C. - □ ×

H:42, M:151, S:355

01:25:59



C. - □ ×

H:59, M:0, S:0

02:00



C. - □ ×

H:191, M:40, S:16

03:06:02



C. - □ ×

H:302, M:60, S:0

10:10



C. - □ ×

H:301, M:358, S:47

10:00:07



C. - □ ×

H:132, M:132, S:132

04:22:22



C. - □ ×

H:156, M:64, S:76

05:10:12



C. - □ ×

H:294, M:294, S:294

09:49:49



C. - □ ×

H:301, M:50, S:0

10:08



C. - □ ×

H:303, M:58, S:261

10:09:43



C. - □ ×

H:92, M:43, S:0

03:07



C. - □ ×

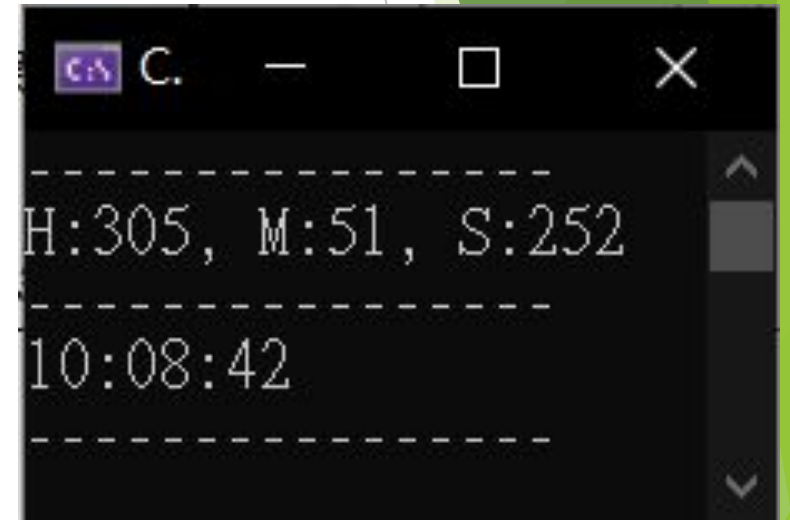
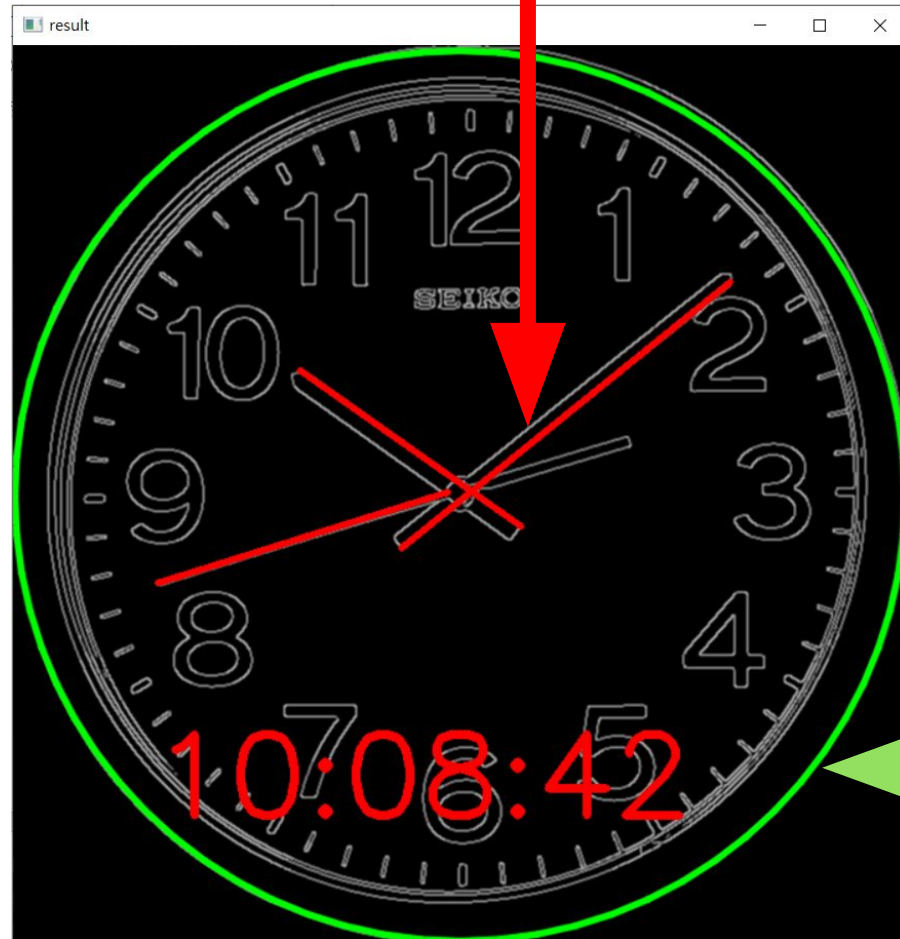
H:303, M:58, S:261

10:09:43

Data Output and Visualization



HoughLinesP



HoughCircles

Program Contents

- ▶ Variables Declaration

- ▶ Functions

Divide the program into several functions, make it easier to debug and modify.

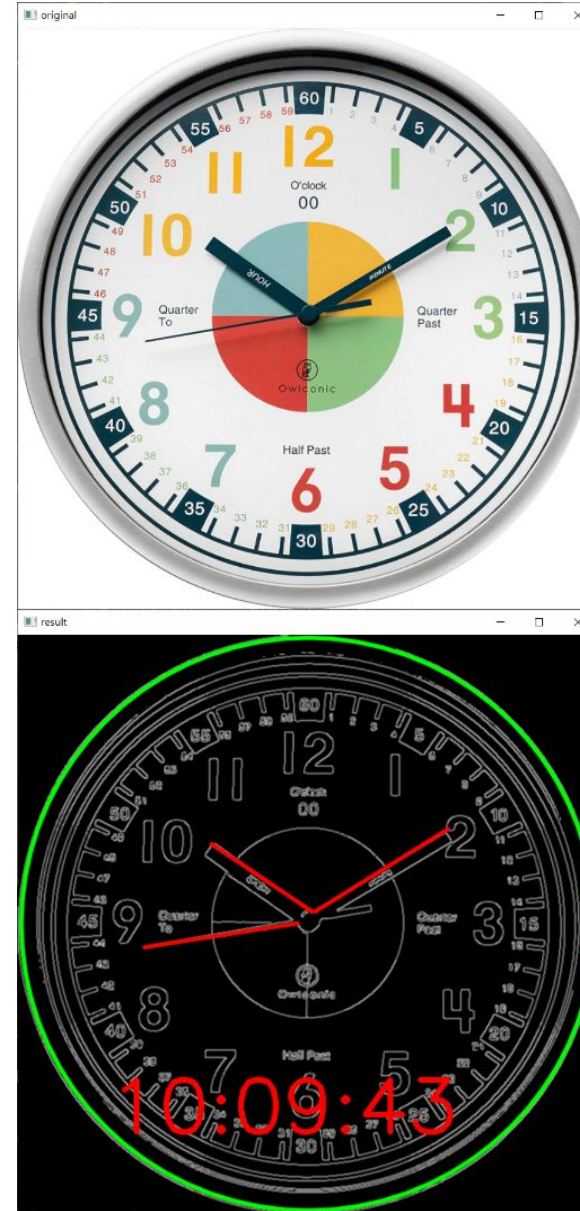
- ▶ Main program

Use the main program to call functions only.

```
1  #include <opencv2/opencv.hpp>
2  #include <iostream>
3
4  using namespace cv;
5  using namespace std;
6
7  float Length(Vec4i l);
8  float Angle(Vec4i l);
9  bool sortByFirst(const pair<float, Vec4i> a, const pair<float, Vec4i> b);
10
11 void imageProcess(string file);
12 void findClock();
13 void findPointers();
14 void calculateTime();
15 void checkError();
16 void showResult();
17
18 Mat originalImg, sizedIMG, grayIMG, blurIMG, cannyIMG, output;
19 Point center;
20 float radius;
21 vector<pair<float, Vec4i>> filteredPointers; //vector<pair<長度, 線段>>
22 float hour, minute, sec;
23 float hourAngle, minAngle, secAngle;
24
25 int main(int argc, char** argv) { ... }
26
27 //預處理圖片
28 void imageProcess(string file) { ... }
29
30 //找出鐘面
31 void findClock() { ... }
32
33 //找出指針
34 void findPointers() { ... }
35
36 //計算時間
37 void calculateTime() { ... }
38
39 //修正時間(僅針對整點附近，易造成較大誤差的部分處理)
40 void checkError() { ... }
41
42 //輸出及顯示時間
43 void showResult() { ... }
44
45 float Length(Vec4i l) { ... }
46
47 float Angle(Vec4i l) { ... }
48
49 bool sortByFirst(const pair<float, Vec4i> a, const pair<float, Vec4i> b) { ... }
```

Pre-Process Images

1. Load the image
2. Resize the image, make it shows properly on the screen.
3. **Gray scaling** the image. (For *HoughCircles* and *HoughLinesP* functions)
4. Blur the image. (To denoise)
5. **Edge detection** process.
6. Smooth the edges.
7. Convert the gray scaled image to **BGR image**, so we can draw on it.



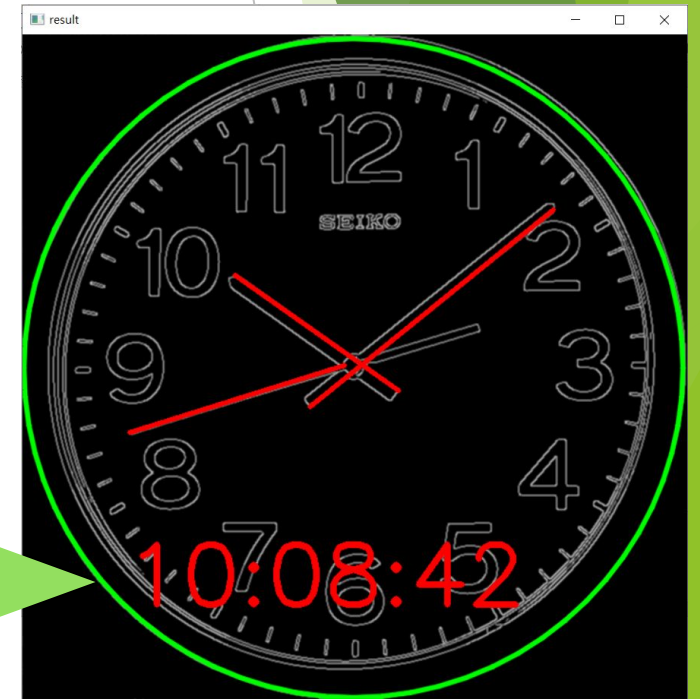
Recognize the Clock Face

1. Use *HoughCircles* function to find circles in the image, each circle could be the clock face.

(It's better to set loose parameters, giving higher chance to recognize the clock face in all conditions.)

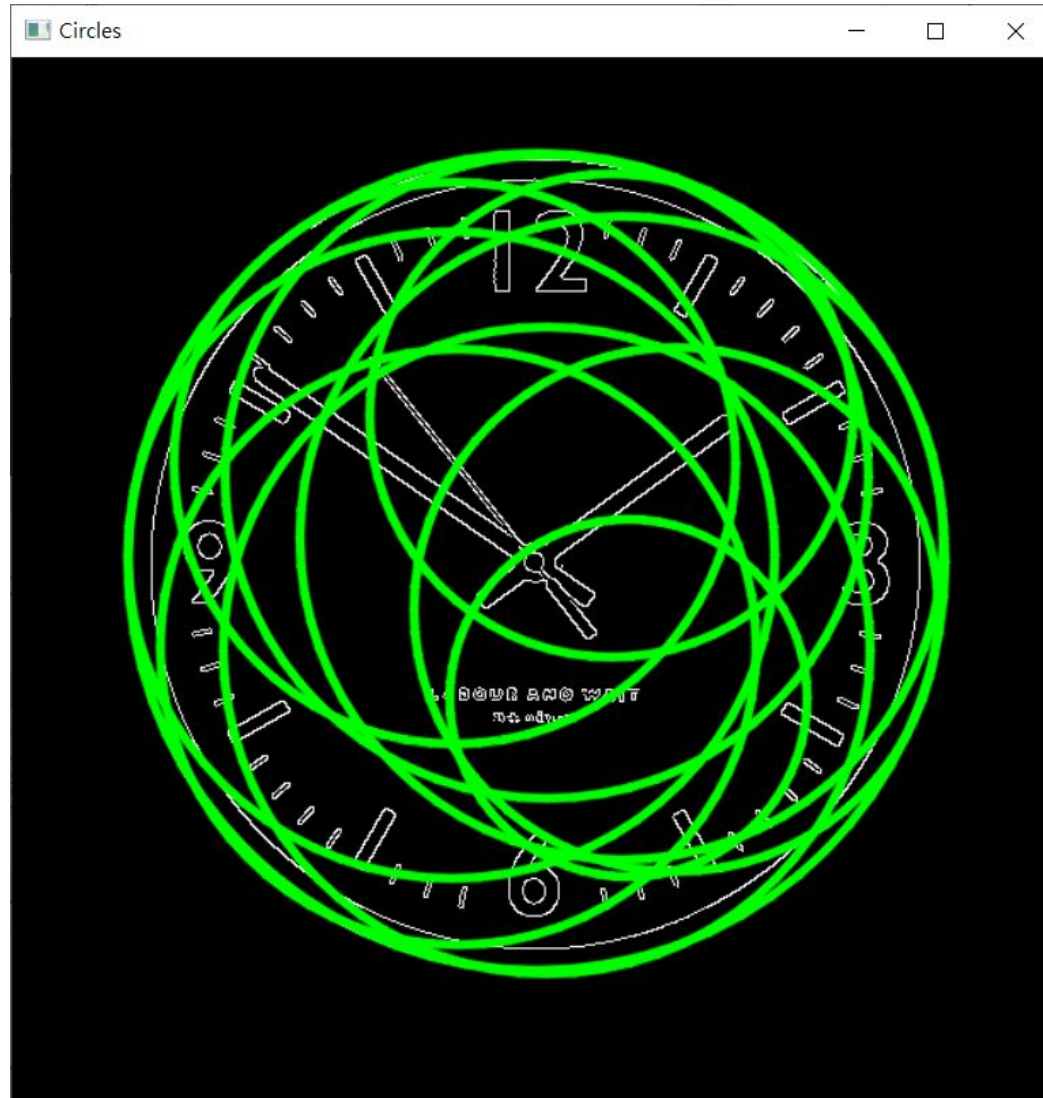
2. Find the circle with largest radius, and fits completely inside the image, as the clock face.
3. Store the clock's center and radius in variables, for later calculation.
4. Draw the clock face on the image. (*circle* function, can do this later)

HoughCircles



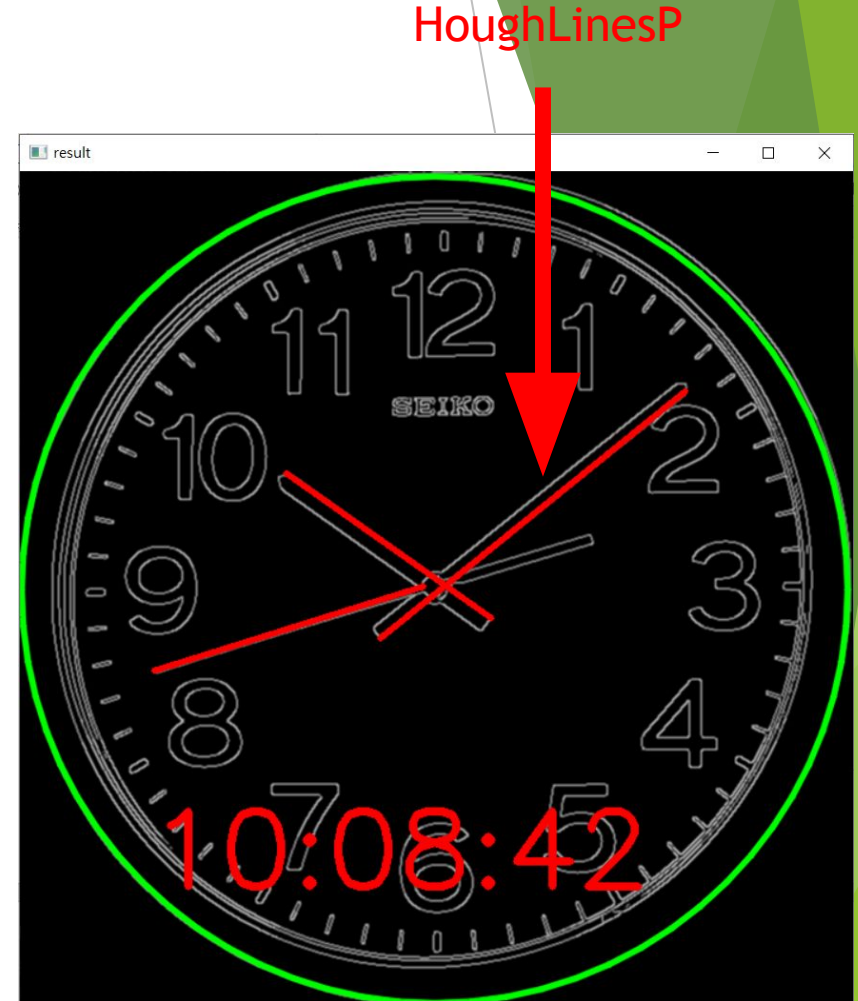
HoughCircles Function

- ▶ Example



Recognize the Clock Hands

1. Find the possible clock hands using *HoughLinesP* function.
(It's better to set loose parameters, giving higher chance to recognize the clock hands in all conditions)
2. Combine the lines with close angles into the same hand. Skip the lines that don't cross the center of the clock, which is not likely the hands we are looking for. Choose the longer line as the hand if angles of two lines are close.
3. Store all the possible hands and sort them by their length (The longest distance from two ends of line to the clock center).
4. Keep only the longest 3 hands in the end, other lines are not likely to be the hands.

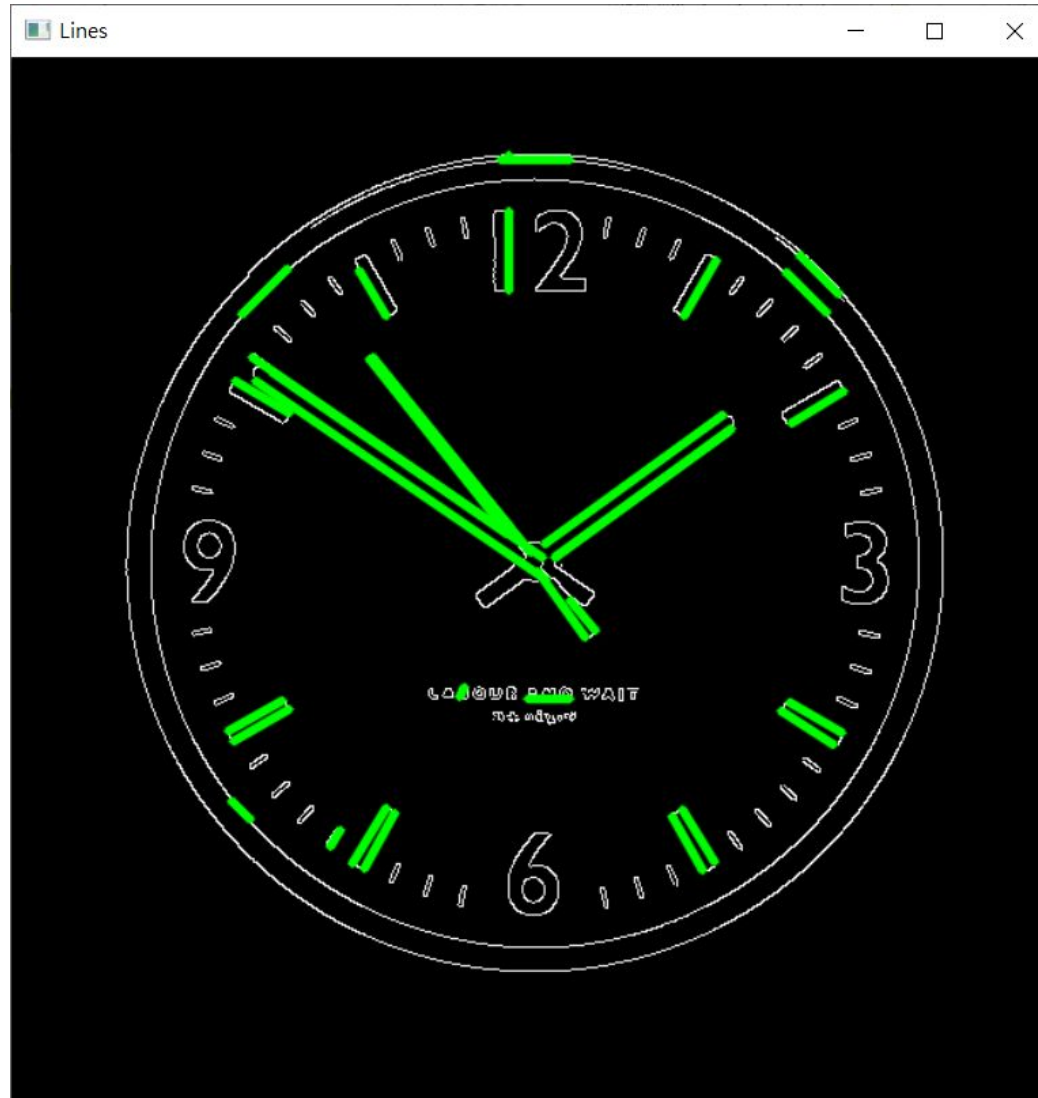


Recognize the Clock Hands (Not included in the code)

- ▶ The clock hands intercept in the center of the clock. Thus, we can determine the clock hands by this logic.

HoughLinesP Function

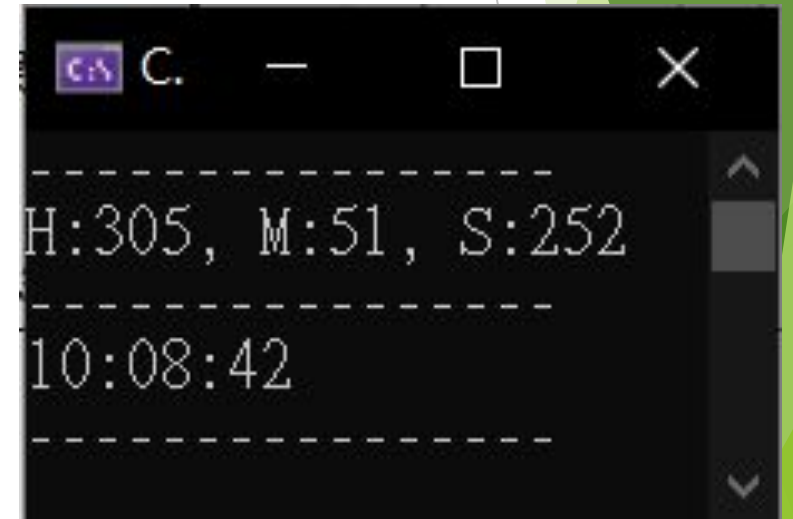
- ▶ Example



Calculate Time

1. If only one hand is found, this could mean the hands overlap, give the same angle to two other hands in this case.
2. Calculate time by the angles of the hands.

The calculation can be done by atan or atan2 function, other functions like asin, acos... also work.

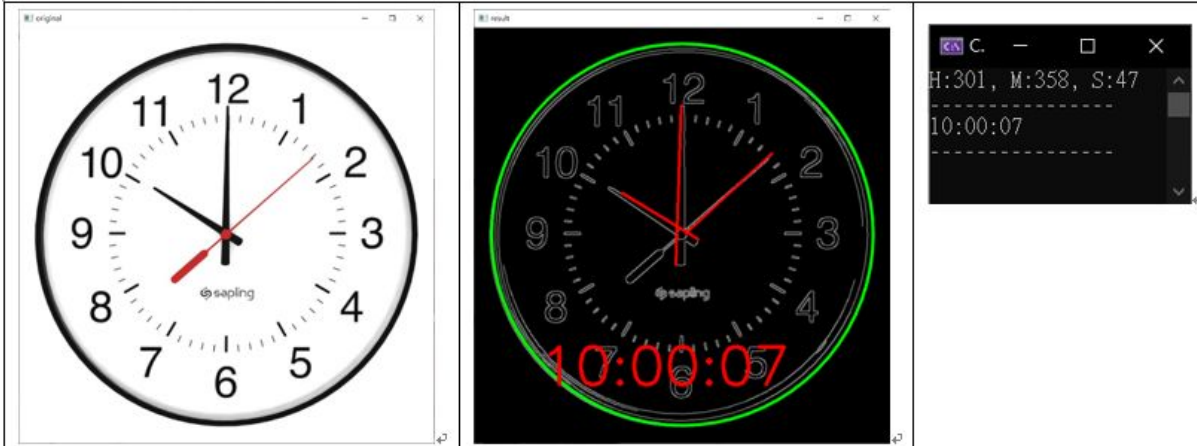


Correct the Errors

- ▶ In some cases, the hands' angles are too close to some limits, like 10:59 and 11:00. Correct the possible errors to obtain more accurate results.

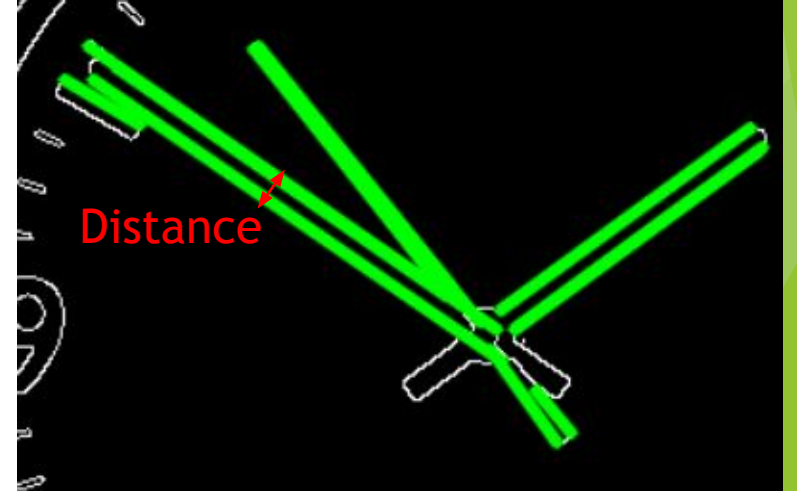
Process:

1. If minute hand is close to 12, judge the actual time by the second hand.
2. If hour hand is close to o'clock, judge the actual time by the minute hand. This can lead to an 1 hour difference if we don't correct it.



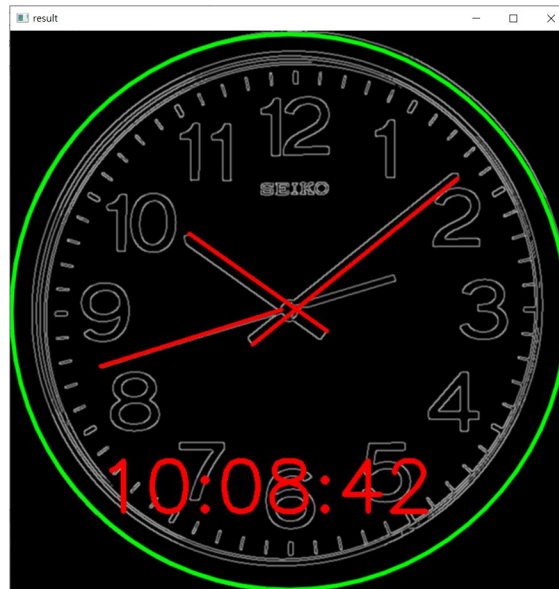
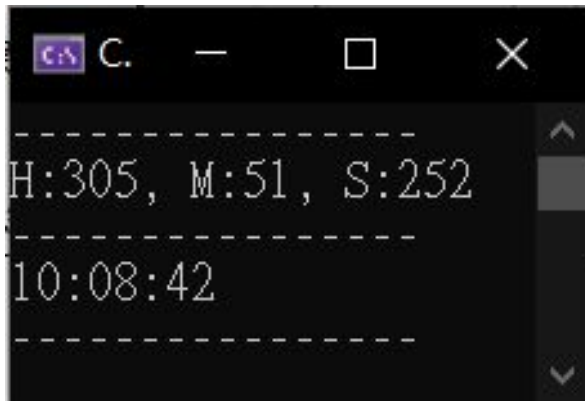
Correct the Errors

- ▶ Some clocks/watches have longer second hands, results in incorrect recognition between second and minute hands.
- ▶ This problem can be solved by several methods.
 1. We can usually find more lines on the minute hand since it's thicker, the provided code uses this way to correct the problem.
 2. The distance between two sides of minute hand is usually wider. (This method not included in the code)
 3. Use video instead of images to read the time, the movement of second hand is much faster than minute hand. (This method not included in the code)



Output Results

- ▶ Draw the hands (*line* function)
- ▶ Draw the data on the image (*putText* function)
- ▶ Print results in the console.
- ▶ Show the image.

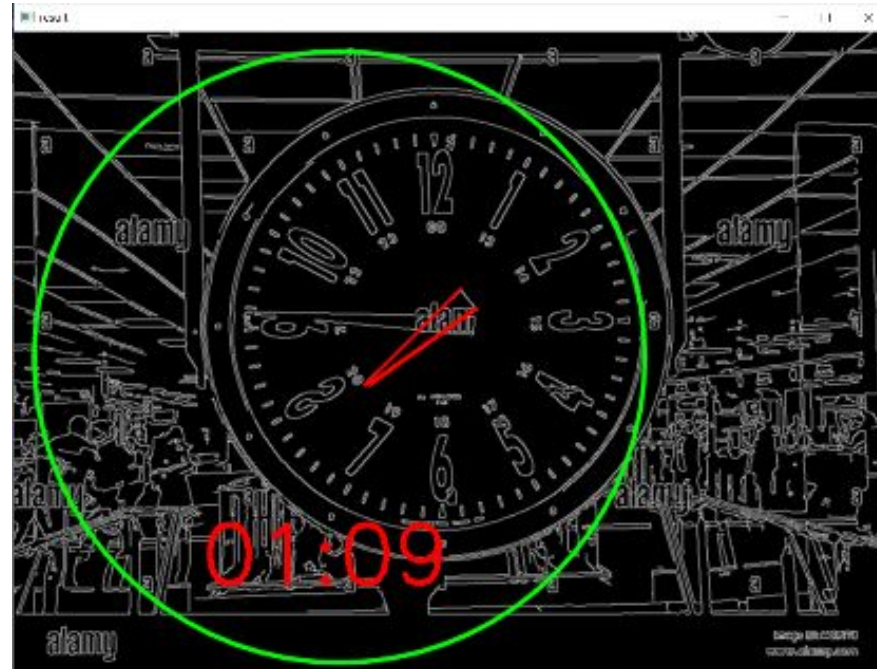
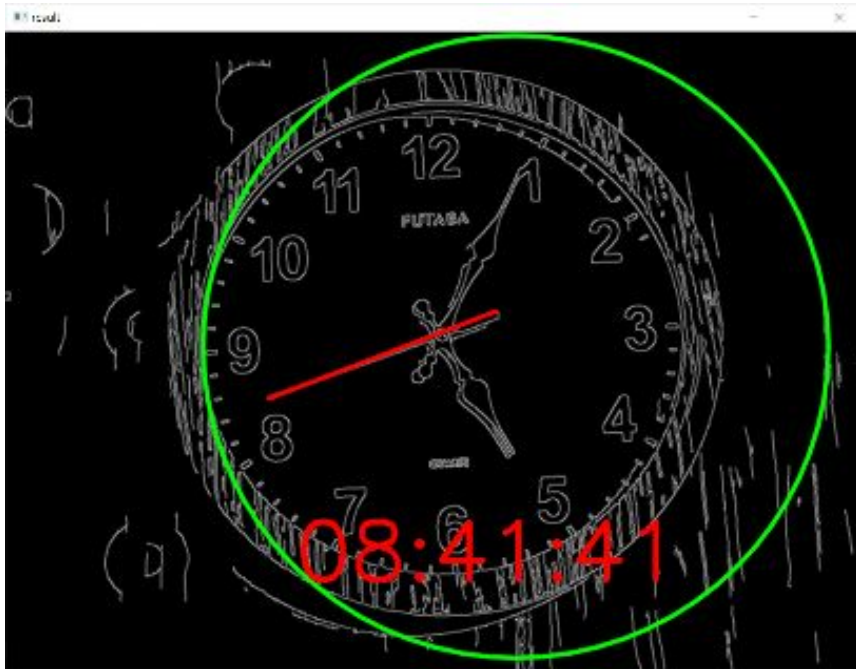


Functions Used

- ▶ Image recognition:
- ▶ HoughCircles:
https://docs.opencv.org/4.x/da/d53/tutorial_py_houghcircles.html
- ▶ HoughLinesP:
https://docs.opencv.org/3.4/d9/db0/tutorial_hough_lines.html
- ▶ Calculate angle:
- ▶ atan, asin, acos, etc.
- ▶ Draw results:
https://docs.opencv.org/4.x/dc/da5/tutorial_py_drawing_functions.html
- ▶ circle: draw circles
- ▶ line: draw lines
- ▶ putText: draw texts

Exceptions

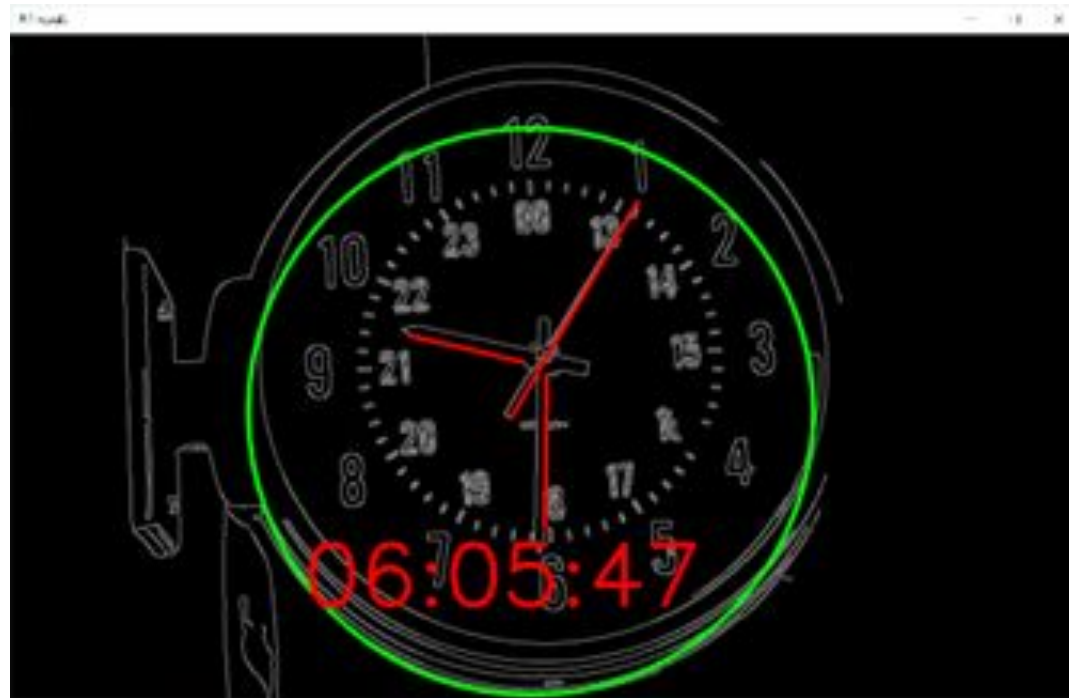
- Complex background



- Special shape of hands

Exceptions

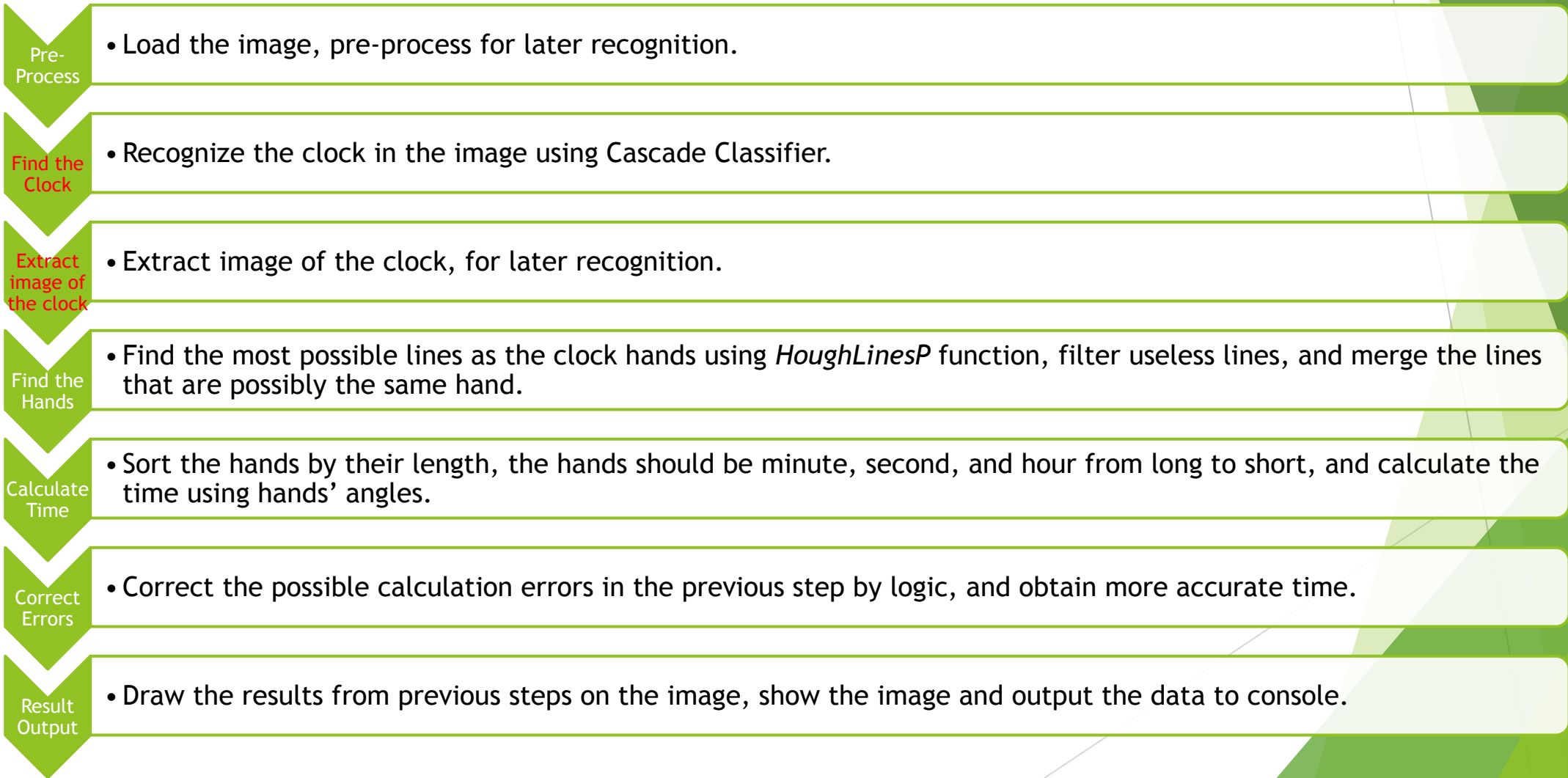
- Tilted camera angle



Conclusion - First Method

- ▶ This method is able to read time in certain conditions.
- ▶ It needs to be:
 - ▶ Clean background: Complex background could cause false recognition of clock/watch faces.
 - ▶ Good camera angle: Tilted camera angle could also cause false recognition of clock/watch faces.
 - ▶ Shape of hands is close to a straight line: Special shapes of clock hands could make *HoughLinesP* function unable to find the hands.

Process (Second Method) - Machine Learning

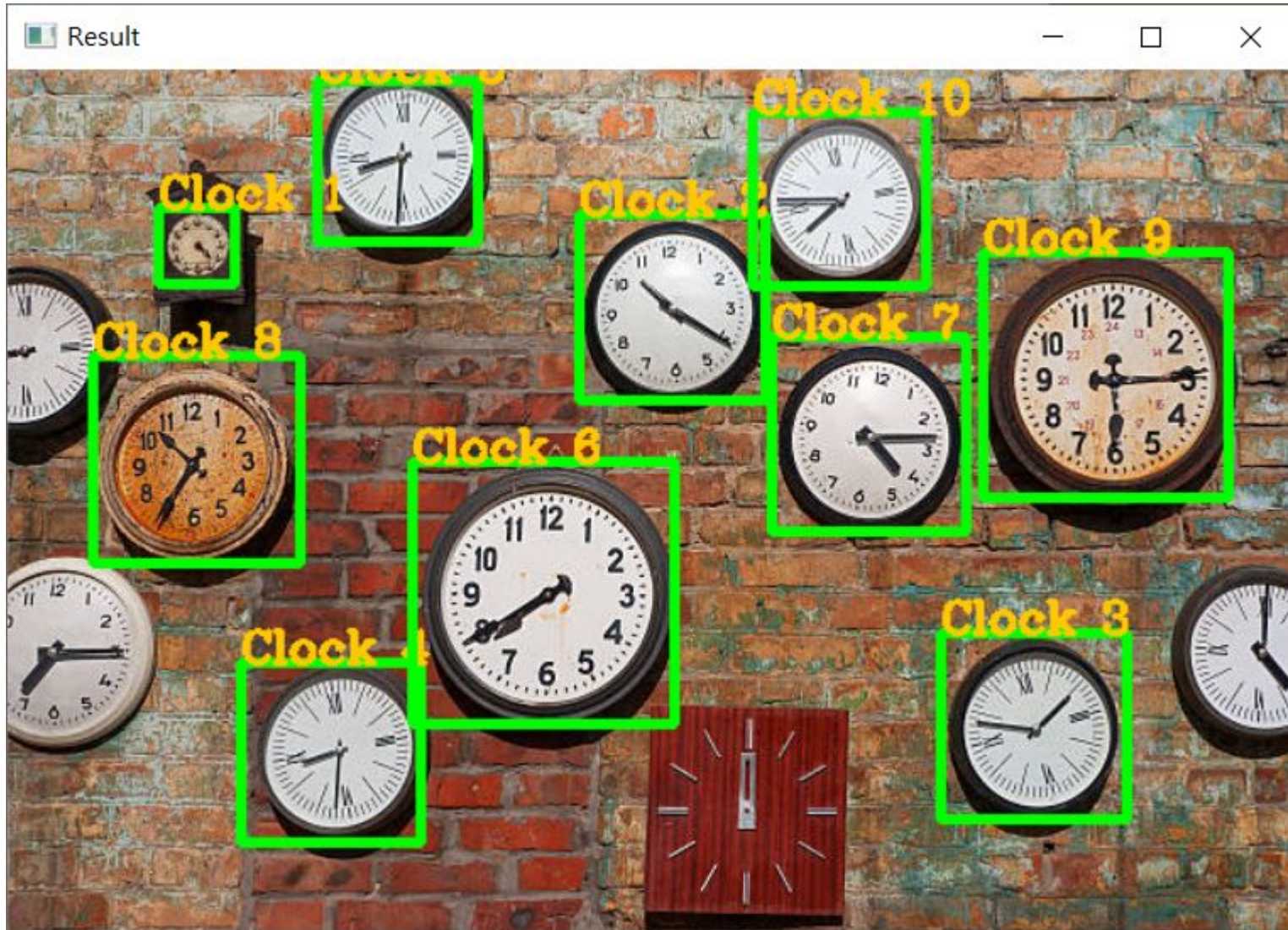


Cascade Classifier

- ▶ For training a boosted cascade of weak classifiers we need a set of positive samples (containing actual objects you want to detect) and a set of negative images (containing everything you do not want to detect). The set of negative samples must be prepared manually, whereas set of positive samples is created using the `opencv_createsamples` application.
- ▶ Example:
- ▶ https://docs.opencv.org/4.x/db/d28/tutorial_cascade_classifier.html
- ▶ Training:
- ▶ https://docs.opencv.org/4.x/dc/d88/tutorial_traincascade.html
- ▶ (There is also 3rd party GUI trainer)

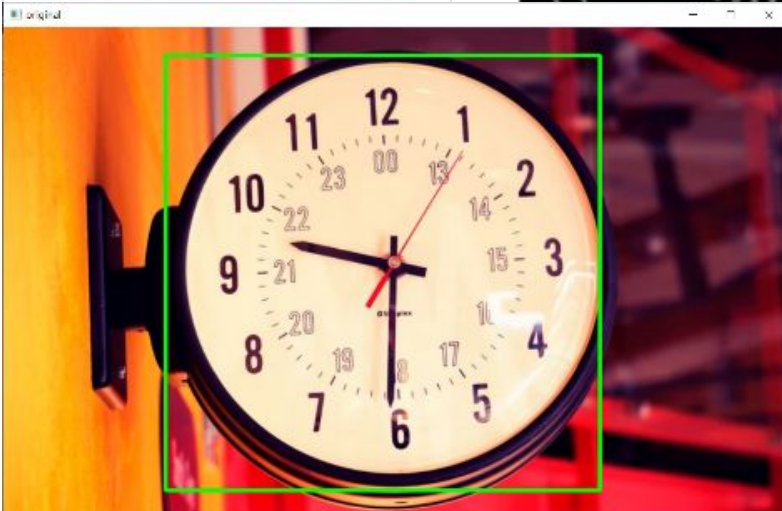
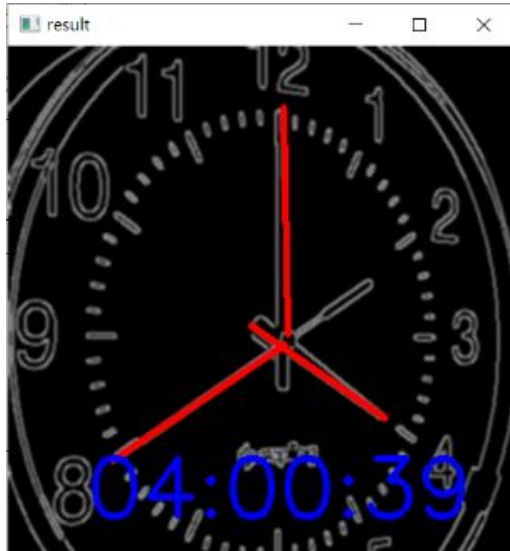
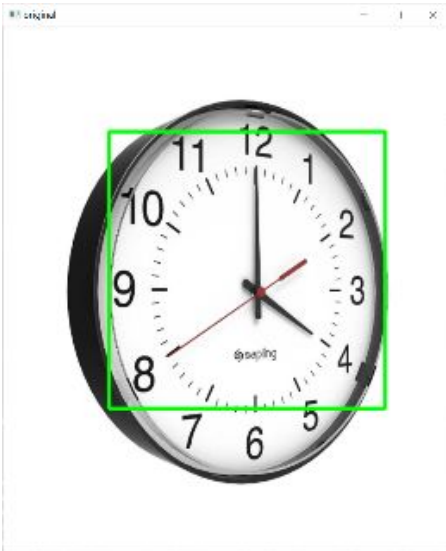
Cascade Classifier

- ▶ Example



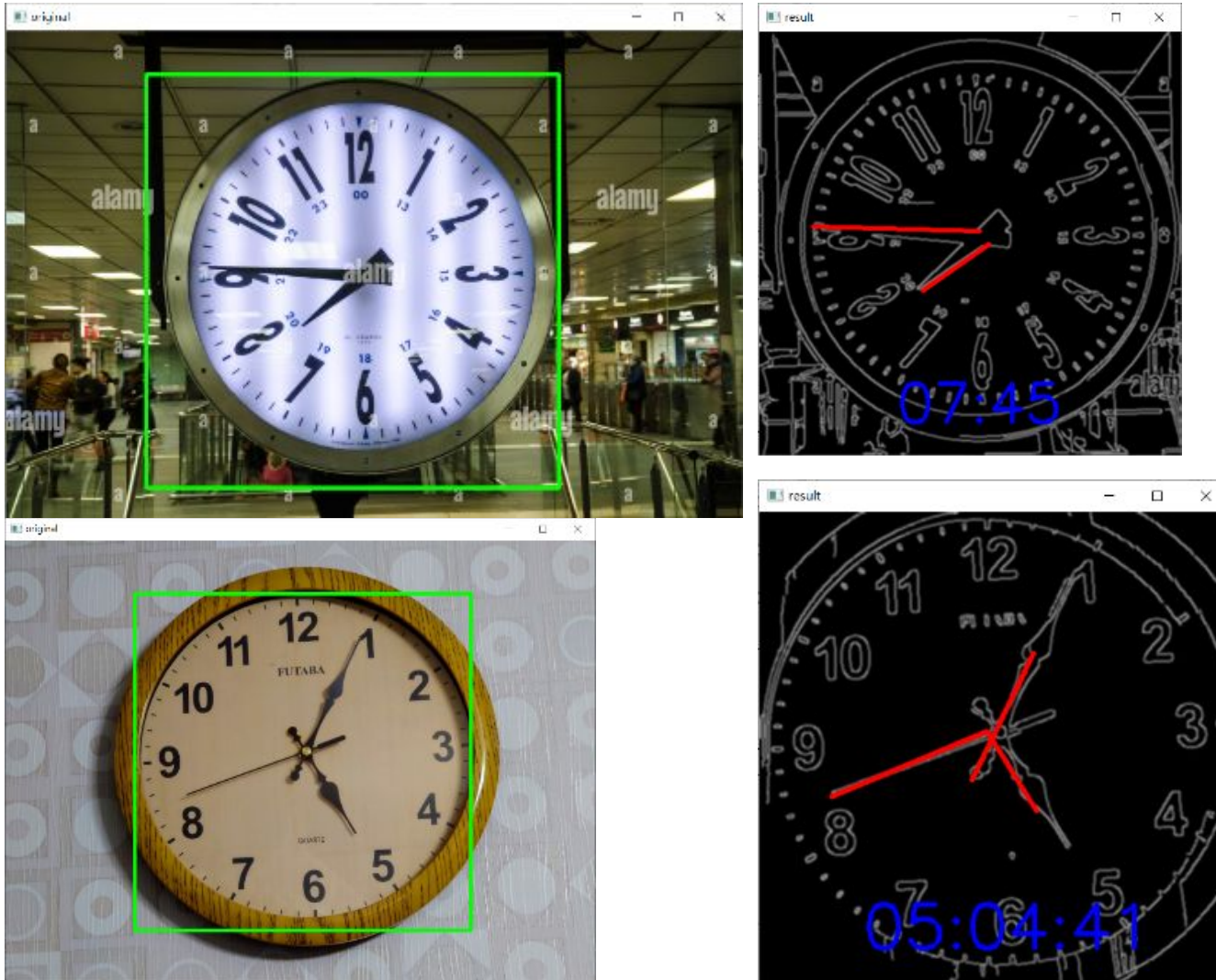
Process Results of Previous Exceptions

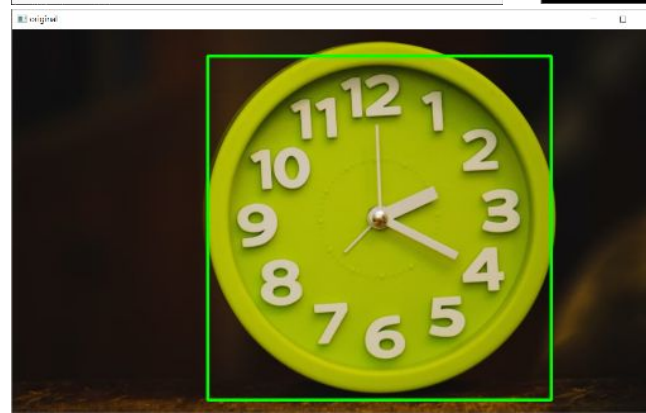
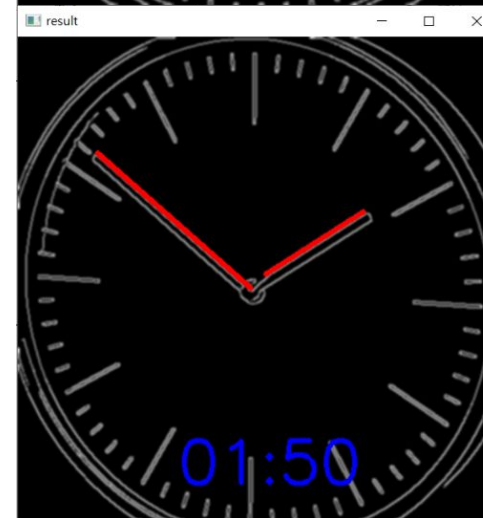
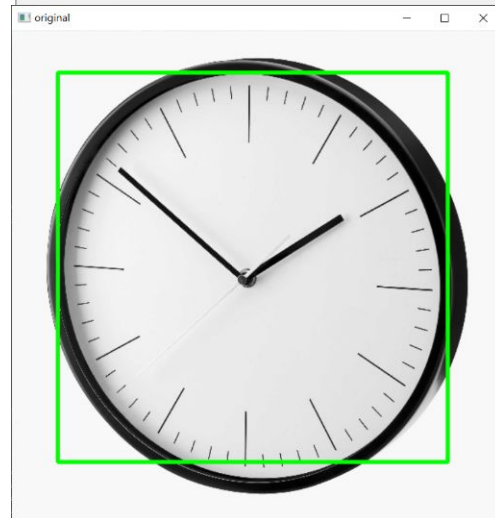
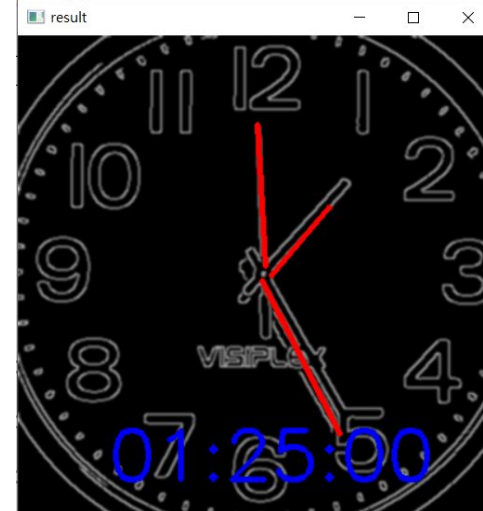
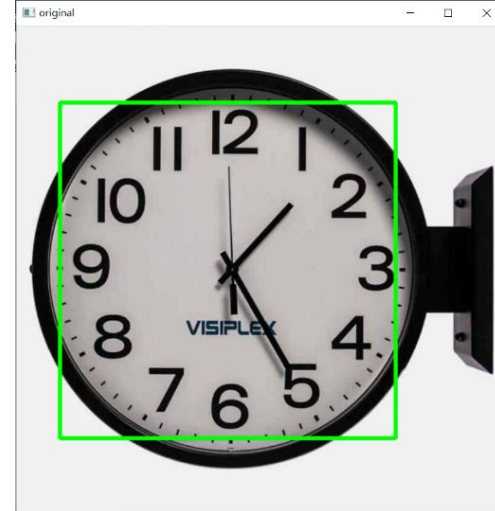
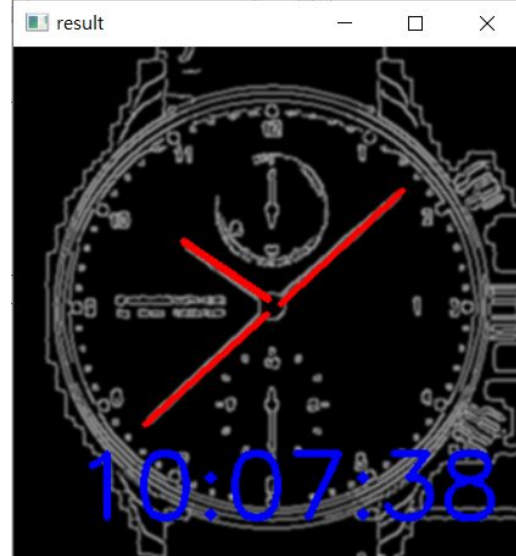
- Tilted camera angle



Process Results of Previous Exceptions

- ▶ Complex background: This method deals with complex backgrounds better.





Conclusion (Second Method)

- ▶ Able to read time of clocks/watches in most conditions.
- ▶ Conditions could possibly cause false results:
 - ▶ Clocks/watches that look too special.
 - ▶ Shape of hands are too special: Special shapes of clock hands could make *HoughLinesP* function unable to find the hands.
 - ▶ There are complex lines on the clock/watch face.
 - ▶ Can be solve by adjusting parameters of functions.

