LAB: Tension Detection of Rolling Metal Sheet

Date: 2025-May-05

Author: Joungbin Choi 22200757

Github: repository link

Introduction

1. Objective

The environment in which steel plates are manufactured is extremely high in temperature, making it impossible for sensors or machines to be placed directly in the production zone. As a result, it becomes challenging to measure the stress applied to the steel plates. To overcome this limitation, cameras are utilized. Installed in isolated areas away from the high-temperature zone, these cameras observe the curvature of the steel plates to assess their condition.

In this lab, various image processing techniques and numerical analysis methods are employed to evaluate the condition of the steel plates, expressing the results in terms of a "level" and "score."

2. Preparation

Software Installation

- OpenCV 3.83, Visual Studio Code
- CUDA 12.6, cudatoolkit 12.6,, Python 3.8.5

Dataset

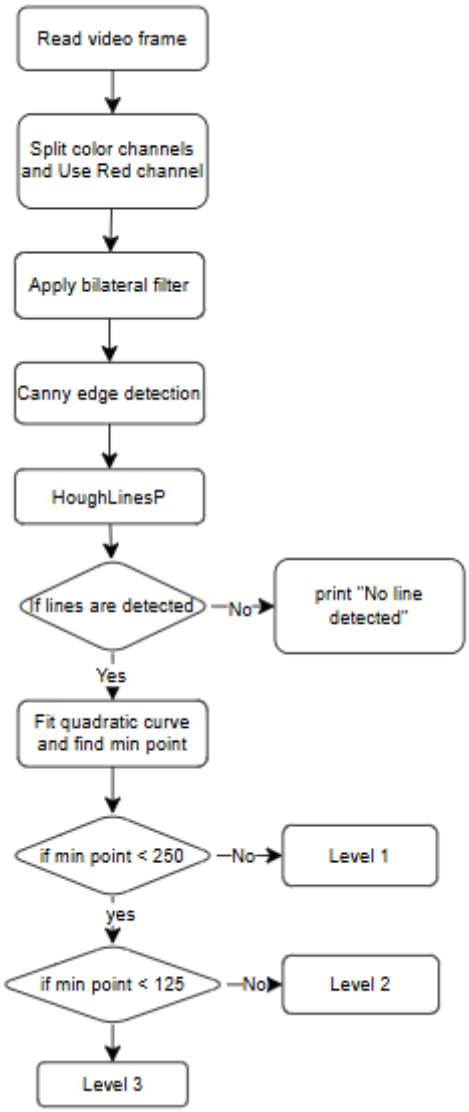
There are 3 images of simple testing, other 3 of challenging testing, and the final video.

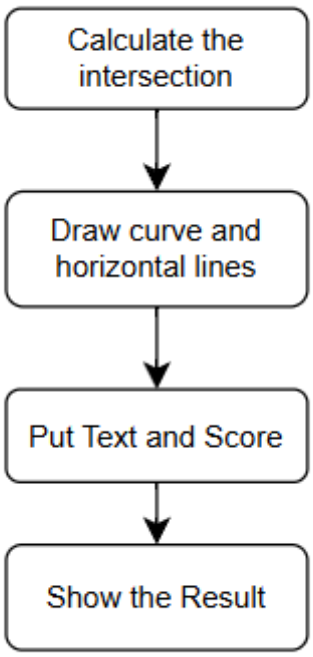
Dataset link:

1. Download the simple test image
2. Download the challenging test image
3. Download the test video

Algorithm

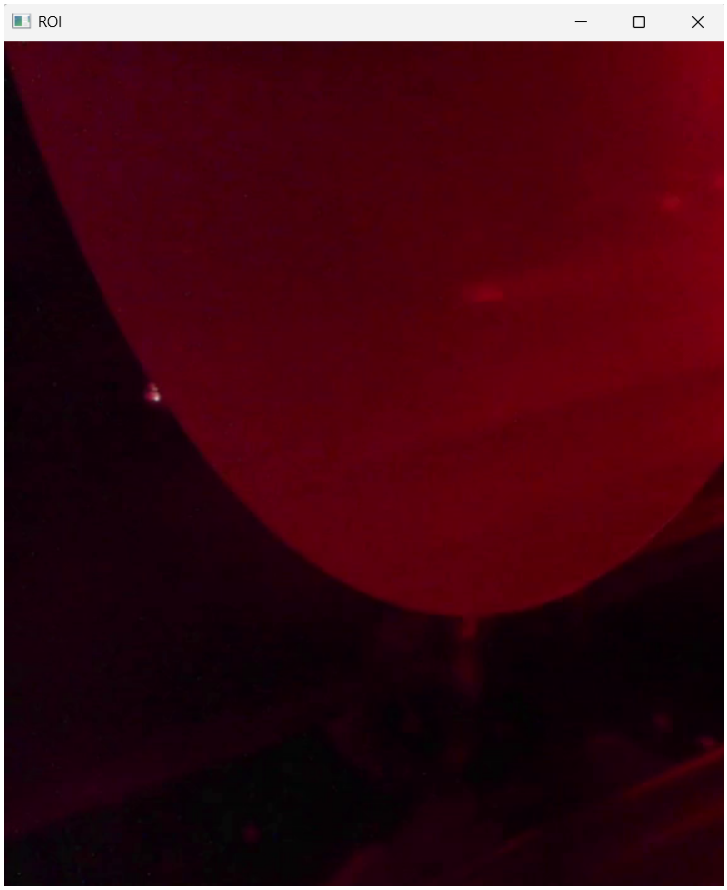
1. Overview


Read video frame
Split color channels and Use Red channel
Apply bilateral filter
Canny edge detection
HoughLinesP
If lines are detected
No
print "No line detected"
Yes
Fit quadratic curve and find min point
if min point < 250
No
Level 1
yes
if min point < 125
No
Level 2
Level 3


Calculate the intersection
Draw curve and horizontal lines
Put Text and Score
Show the Result

2. Procedure

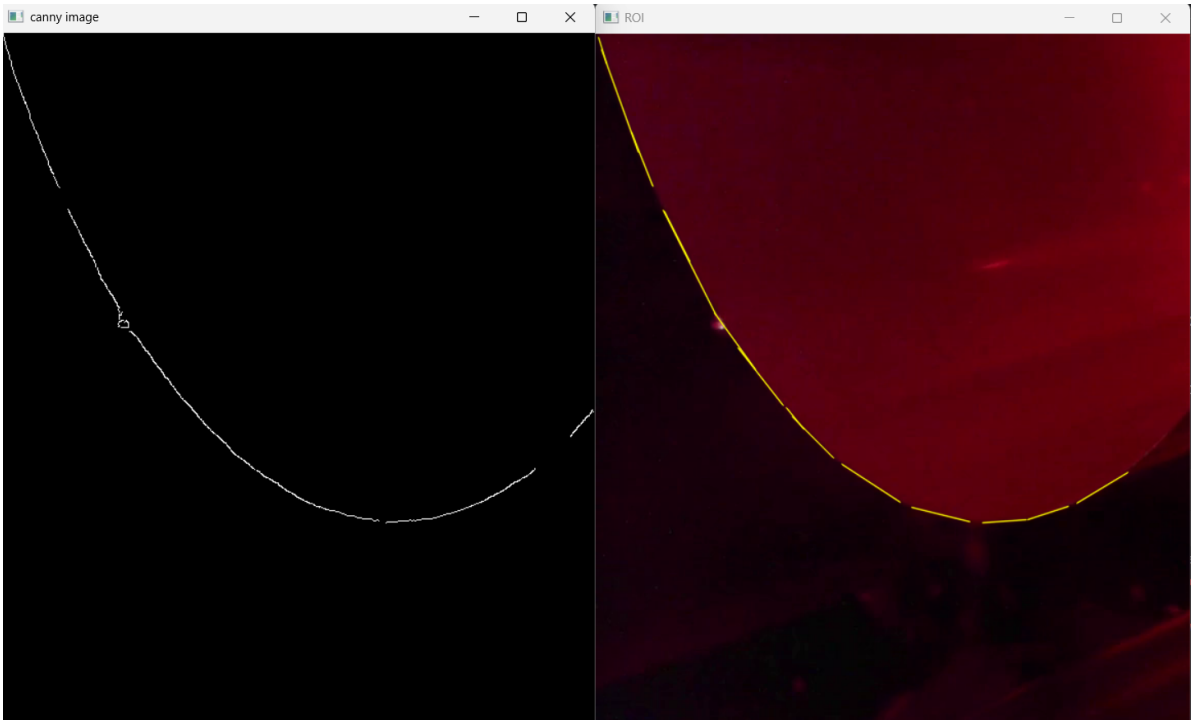
ROI Selection and Preprocessing



Extract a specific ROI to focus processing and reduce noise. For this lab, the size of window is limited from (0, 400) to (580, 1080).

Since the manufacturing environment is dark and red, the red channel is used and applied bilateral filtering to reduce background noise. Additionally, it is filtered with `bilateralFilter()` to decrease the noise and maintain the edges. The diameter of each pixel is 9, the filter sigma in the color space and coordinate space are both 95.

Edge Detection and Line Detection



The filtered red channel image underwent Canny edge detection (`cv.Canny()`) to highlight the edges. Probabilistic Hough Line Transform (`cv.HoughLinesP()`) was applied to detect straight line segments from the edge map.

Curve Fitting

After detecting line segments using the Hough Transform, the coordinates of their endpoints were extracted and stored in two lists: X and Y. When values are stored, ROI value has to be added because the final line will be appeared on the original image. With those lists, a second-degree polynomial curve (quadratic function) is modeled and is fitted using the least squares method.

To determine the coefficients of a second-degree polynomial curve, the following matrix formulation was used.

$$A = \begin{bmatrix} x_1^2 & x_1 & 1 \\ x_2^2 & x_2 & 1 \\ \vdots & \vdots & \vdots \\ x_n^2 & x_n & 1 \end{bmatrix}, Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}, c = \begin{bmatrix} a \\ b \\ c \end{bmatrix}$$

A is a matrix of input x-value and Y is corresponding y-values. The matrix c is the coefficient vector to determined with the least squares solution which is written below.

$$(A^T A)c = A^T Y$$

To avoid numerical approximation error, `linalg.inv` function from `numpy library` is used.

Curve-fitting Code

Score and Level Determination

Calculate the minimum y-value of the curve to determine the depth with simple math.

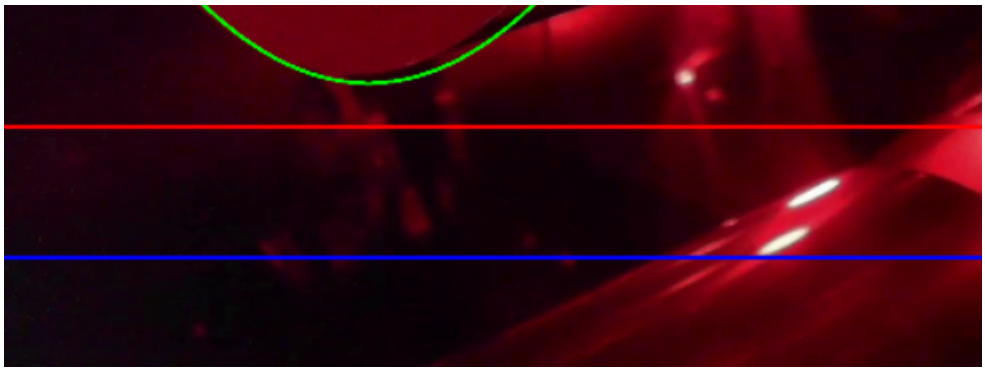
$$x_{min} = -b/(2 \times a)$$
$$y_{min} = a \times x_{min}^2 + b \times x_{min} + c + roi\ of\ y$$

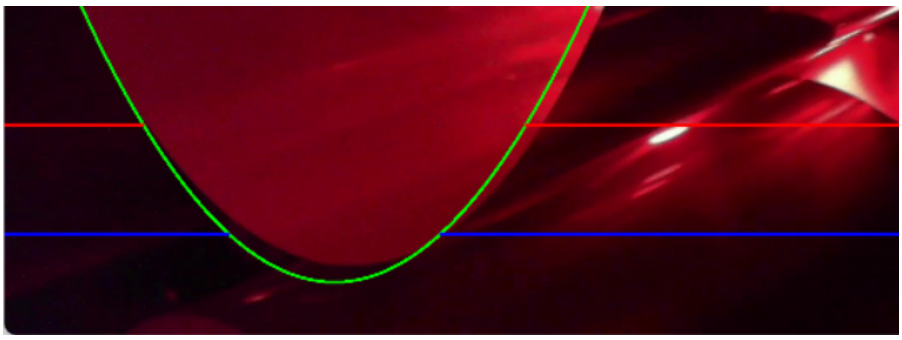
Compute a "score" based on the distance from the ROI bottom.

```
score = roi_h - y_min
```

Based on the score, determine the level:

- **Level 1**: Score ≥ 250
- **Level 2**: 120 ≤ Score < 250
- **Level 3**: Score < 120





Each level has different method to draw horizontal lines which shows the levels. If y is below than certain values(250px, 120px), their x value it scored as a vector. Draw the line from the 0 to first point of vector and draw another line from last point of vector to end of width. In conclusion, there is line until the intersection and there is no line on concaving area. Again, there is a line after concave.

Put Text

Level and score are displayed on window. They are in the rectangular box which is weighted with `addWeighted()` function to make it readable.

```
    box_x, box_y = 880, 350
    box_w, box_h = 240, 80
    overlay = img.copy()
    alpha = 0.5
    cv.rectangle(overlay, (box_x, box_y), (box_x + box_w, box_y + box_h), (0, 0,
0), -1)
    cv.addWeighted(overlay, alpha, img, 1 - alpha, 0, img)

    cv.rectangle(img, (box_x, box_y), (box_x + box_w, box_y + box_h), (0, 255,
0), 2)

    cv.putText(img, f'Level: {level}', (box_x + 10, box_y + 35),
               cv.FONT_HERSHEY_SIMPLEX, 1.2, (255, 255, 255), 2, cv.LINE_AA)
    cv.putText(img, f'Score: {int(score)}', (box_x + 10, box_y + 70),
               cv.FONT_HERSHEY_SIMPLEX, 1.2, (255, 255, 255), 2, cv.LINE_AA)
```

Result and Discussion

1. Final Result

First, each frame was processed by extracting the region of interest (ROI) from the left part of the image. The red channel was separated using channel splitting, as it provided the most relevant contrast for detecting features. The image was then denoised using a bilateral filter, which preserved edges while reducing noise.

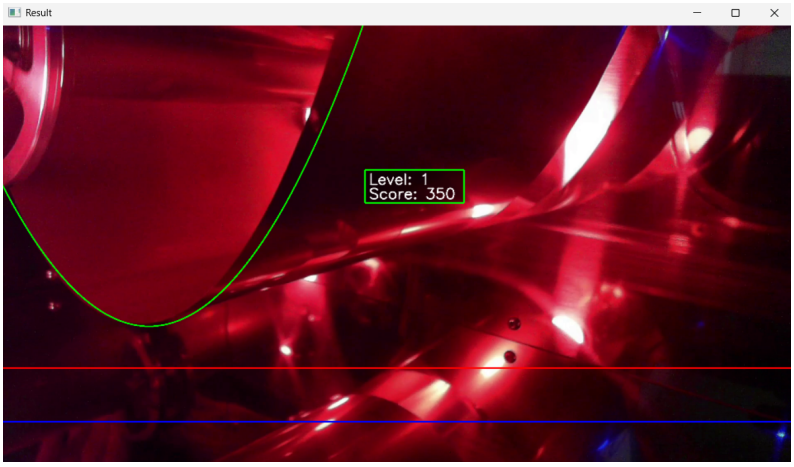
Next, Canny edge detection was applied to highlight contours within the red channel. Using these edges, line segments were extracted through the Hough Line Transform (`HoughLinesP`). The endpoints of the detected lines were collected and used for second-degree polynomial (quadratic) curve fitting using the least squares method.

From the fitted curve, the lowest point (vertex) was calculated and used to determine a score, representing how deep the curve extended into the ROI. Based on this score, the system assigned a level from 1 to 3:

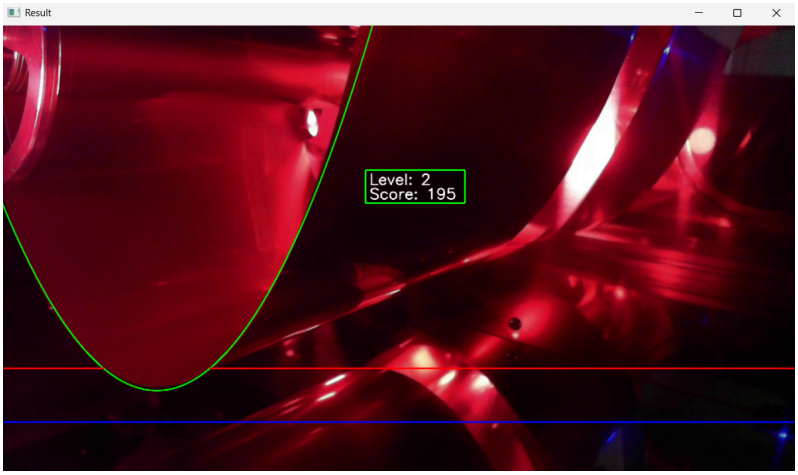
- **Level 1**: Shallow curve (Less tension)
- **Level 2**: Medium curve (medium tension)
- **Level 3**: Deep curve (strong tension)

On average, the system produced stable curve fits across most frames, and levels were accurately assigned based on curve depth.

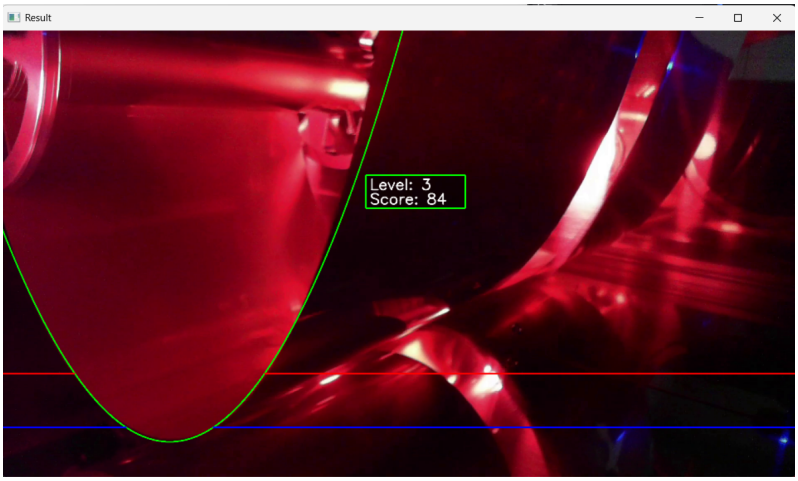
The Level 1:



The Level 2:



The Level 3:



2. Discussion

However, there were some limitations observed during the process. Although bilateral filtering helped reduce noise, it could not eliminate all irrelevant edges. As a result, some noisy or unintended line segments were still detected during edge detection, which occasionally led to outlier points in the curve fitting stage. These outliers could slightly distort the shape of the fitted curve.

Nevertheless, in the provided test video, such distortions were minor and did not significantly affect the level or score classification. The system remained stable and reliable for the given data.

Conclusion

In this lab, we addressed the challenge of monitoring steel plates in high-temperature manufacturing environments where direct sensor placement is not feasible. By employing a camera-based system, we successfully detected the curvature of the steel plates and evaluated their tension state without physical contact. Through the use of image processing techniques such as edge detection and polynomial curve fitting, combined with a scoring and classification system, we were able to assess the condition of the steel plates in real-time. This approach demonstrates the practicality and effectiveness of computer vision as a non-invasive solution for industrial monitoring in extreme conditions.

Appendix

```
import numpy as np
import cv2 as cv

cap = cv.VideoCapture('LAB3_Video.mp4')
while True:
    ret, frame = cap.read()
    if not ret:
        break

    # Open the Image in Gray Scale and Filtering
    img = frame.copy()
    height, width = img.shape[:2]

    # Cropped the image in ROI
    roi_x = 0; roi_y = 400; roi_w = 580; roi_h = height
    roi = img[roi_y:roi_h, roi_x:roi_w]
    img2 = roi.copy()

    # Split the channel and convert the red to gray scale
    img_b, img_g, img_r = cv.split(img2)
    img_denoised = cv.bilateralFilter(img_r, d=9, sigmaColor=75, sigmaSpace=75)

    # Canny Edge Detection
    img_canny = cv.Canny(img_denoised, 50, 80)
    cv.imshow('denoised image', img_denoised)
    cv.imshow('canny image', img_canny)

    lines = cv.HoughLinesP(img_canny, 1, np.pi/180, 30, minLineLength=20,
maxLineGap=10)
    X = []
    Y = []
    if lines is not None:
        for i in range(lines.shape[0]):
            x1 = lines[i][0][0] + roi_x
```

```
            y1 = lines[i][0][1]
            x2 = lines[i][0][2] + roi_x
            y2 = lines[i][0][3]
            cv.line(img2, (x1, y1), (x2, y2), (0, 255, 255), 1, cv.LINE_AA)
            X.extend([x1, x2])
            Y.extend([y1, y2])

        # Curve Fitting
        A = np.vstack([np.power(X, 2), X, np.ones(len(X))]).T
        Y_np = np.array(Y).reshape(-1, 1)
        ATA = A.T @ A
        ATY = A.T @ Y_np
        coeffs = np.linalg.inv(ATA) @ ATY

        x = np.arange(0, width)
        y = coeffs[0] * x ** 2 + coeffs[1] * x + coeffs[2] + roi_y

        mask_level1 = y > (height - 250)
        mask_level2 = y > (height - 120)
        under_x_250px = x[mask_level1]
        under_x_120px = x[mask_level2]

        points = np.array(list(zip(x.astype(int), y.astype(int))),
dtype=np.int32)
        cv.polylines(img, [points.reshape((-1, 1, 2))], False, (0, 255, 0), 2,
cv.LINE_AA)
        cv.imshow('HoughLinesP', img2)

        # Minimum Value
        a, b, c = coeffs
        x_min = -b / (2 * a)
        y_min = a * x_min ** 2 + b * x_min + c + roi_y
        bottom_point = (x_min, y_min)

        # Calculate Score and Level
        score = roi_h - y_min
        if score < 120:
            level = 3
            if under_x_250px.size > 0:
                cv.line(img, (0, roi_h - 250), (under_x_250px[0], roi_h - 250),
(0, 0, 255), 2, cv.LINE_AA)
                cv.line(img, (under_x_250px[-1], height - 250), (width, roi_h -
250), (0, 0, 255), 2, cv.LINE_AA)
            if under_x_120px.size > 0:
                cv.line(img, (0, roi_h - 120), (under_x_120px[0], roi_h - 120),
(255, 0, 0), 2, cv.LINE_AA)
                cv.line(img, (under_x_120px[-1], roi_h - 120), (width, roi_h -
120), (255, 0, 0), 2, cv.LINE_AA)
        elif 120 <= score < 250:
            level = 2
            if under_x_250px.size > 0:
                cv.line(img, (0, roi_h - 250), (under_x_250px[0], roi_h - 250),
(0, 0, 255), 2, cv.LINE_AA)
                cv.line(img, (under_x_250px[-1], height - 250), (width, roi_h -
250), (0, 0, 255), 2, cv.LINE_AA)
            cv.line(img, (0, roi_h - 120), (width, roi_h - 120), (255, 0, 0), 2,
cv.LINE_AA)
        elif score >= 250:
```

```
            level = 1
            cv.line(img, (0, roi_h - 250), (width, roi_h - 250), (0, 0, 255), 2,
cv.LINE_AA)
            cv.line(img, (0, roi_h - 120), (width, roi_h - 120), (255, 0, 0), 2,
cv.LINE_AA)

    else:
        print('No lines detected')

    box_x, box_y = 880, 350
    box_w, box_h = 240, 80
    overlay = img.copy()
    alpha = 0.5
    cv.rectangle(overlay, (box_x, box_y), (box_x + box_w, box_y + box_h), (0, 0,
0), -1)
    cv.addWeighted(overlay, alpha, img, 1 - alpha, 0, img)

    cv.rectangle(img, (box_x, box_y), (box_x + box_w, box_y + box_h), (0, 255,
0), 2)

    cv.putText(img, f'Level: {level}', (box_x + 10, box_y + 35),
               cv.FONT_HERSHEY_SIMPLEX, 1.2, (255, 255, 255), 2, cv.LINE_AA)
    cv.putText(img, f'Score: {int(score)}', (box_x + 10, box_y + 70),
               cv.FONT_HERSHEY_SIMPLEX, 1.2, (255, 255, 255), 2, cv.LINE_AA)

    cv.namedWindow('Result', cv.WINDOW_NORMAL)
    cv.resizeWindow('Result', width=960, height=540)
    cv.imshow('Result', img)

    if cv.waitKey(1) == 27:
        break

cap.release()
cv.destroyAllWindows()
```