LAB: Tension Detection of Rolling Metal Sheet

Date: 2024-05-07

Author: Soonho Lim 21900610

Github: repository link

Introduction

1. Objective

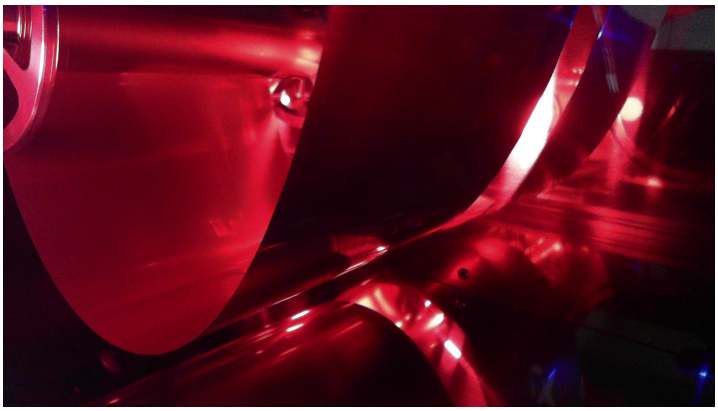

Figure 1

This lab concentrates on detecting tension in rolling metal sheets through image processing techniques. The primary objective is to develop a methodology for isolating metal sheets within images and employing curve fitting techniques to evaluate their tension. This task is crucial for ensuring quality control in industrial manufacturing processes.

Goal: Measure the tension of the Metal sheet in three levels. The level is divided by the calculating the distance between metal sheet's global minimum point and the bottom of the image frame.

- level 1: bigger than 250[pix]
- level 2 : bigger than 120[pix], smaller than 250[pix]
- level 3: smaller than 120[pix]

2. Preparation

Software Installation

- OpenCV 4.7, Visual Studio Code
- Python 3.9

Dataset

- challenging images containing 3 levels, data type "png"
- video of metal sheet, data type "mp4"

Algorithm

1. Overview

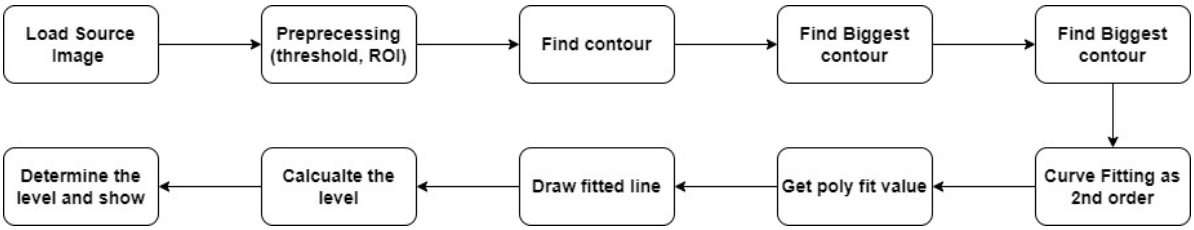

Figure 2

Initially, it loads the image and extracts the region of interest (ROI) containing the metal sheet. It then applies a binary threshold to segment the metal sheet and identifies contours within it. The largest contour, representing the metal sheet, is determined, and curve fitting using a second-order polynomial is performed to model its shape. This fitted curve is drawn onto the original image. The script calculates the maximum y-pixel value of the fitted curve to determine the height, which is crucial for assessing tension. Based on fitted curve, it determines the tension level and draws boundaries accordingly. Additionally, it displays the tension score and level on the image.

2. Procedure

ROI selection

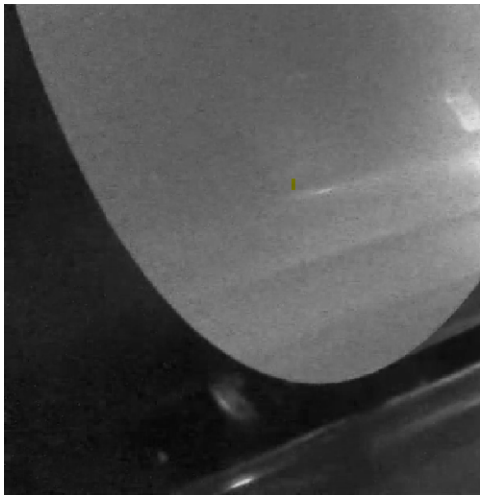

Figure 3

ROI is selected by predefining the points of the source image. Without defining the ROI, it is hard to get the curvature of the metal sheet because other unwanted contours or edges are detected along with it.

Curve Fitting

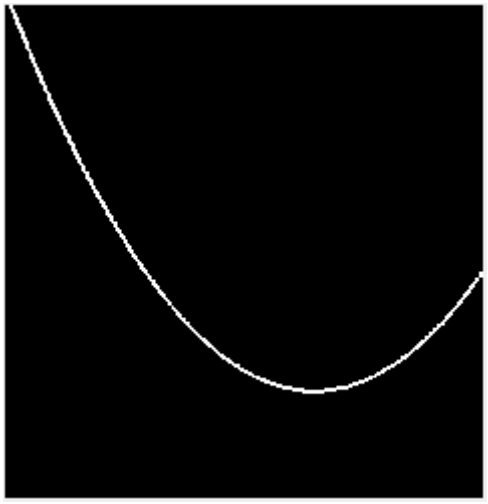

Figure 4

curve fitting is needed to get the global minimum point of the curvature. With the function "drawContours()", it is hard to get the smooth continuous values of the curve. Since the metal sheet's curvature is a 2nd order polynomial curve, it is fitted as it is.

Drawing Boundary lines

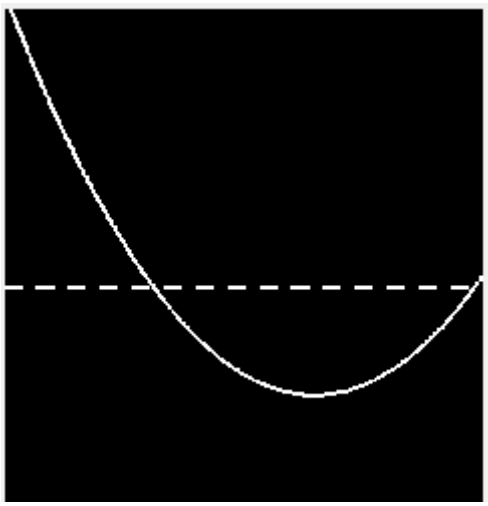

Figure 5

Because there is dash line function in openCV library, defining function for the dash lines is needed. Figure 5 shows the dash line drawn with the fitted curve.

Result and Discussion

1. Final Result

- Challenging Image level 1

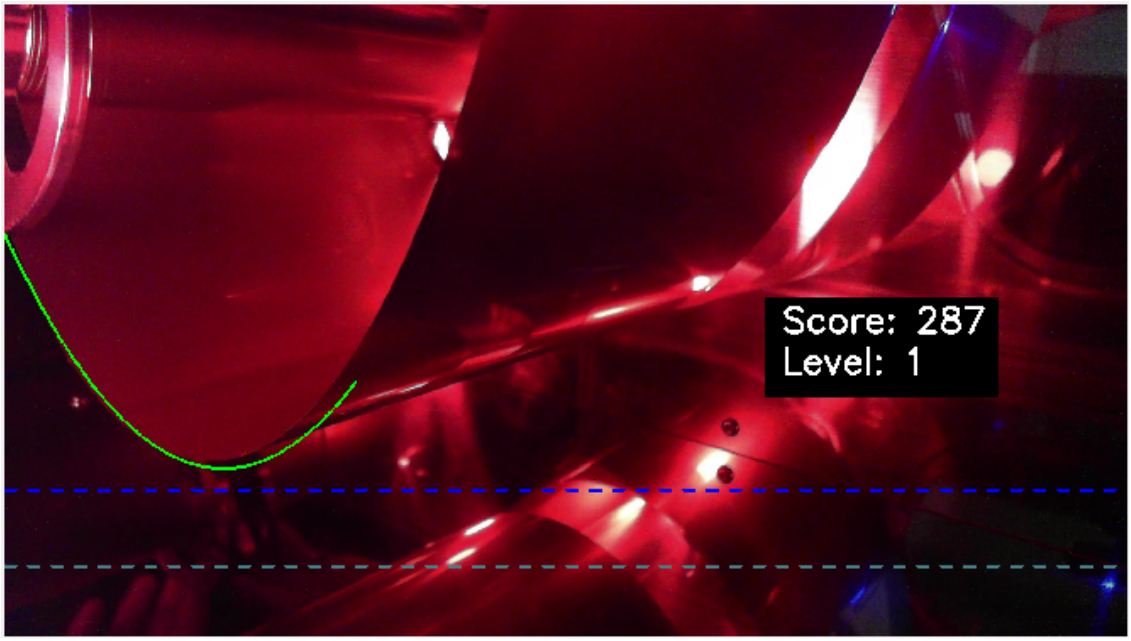

Figure 6

- Challenging Image level 2

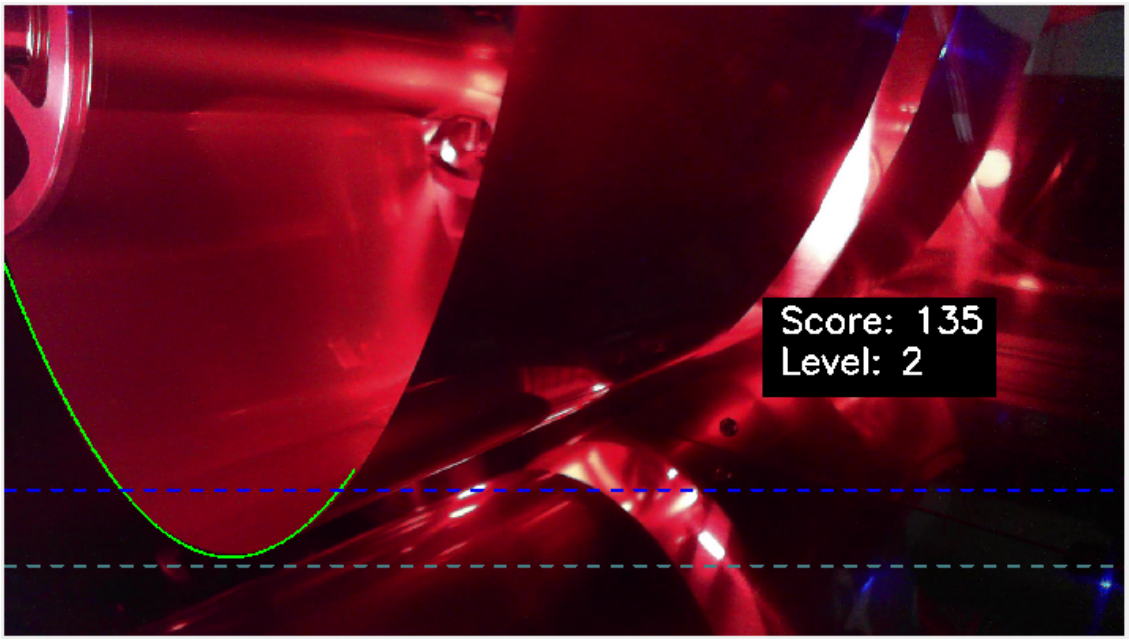

Figure 7

- Challenging Image level 3

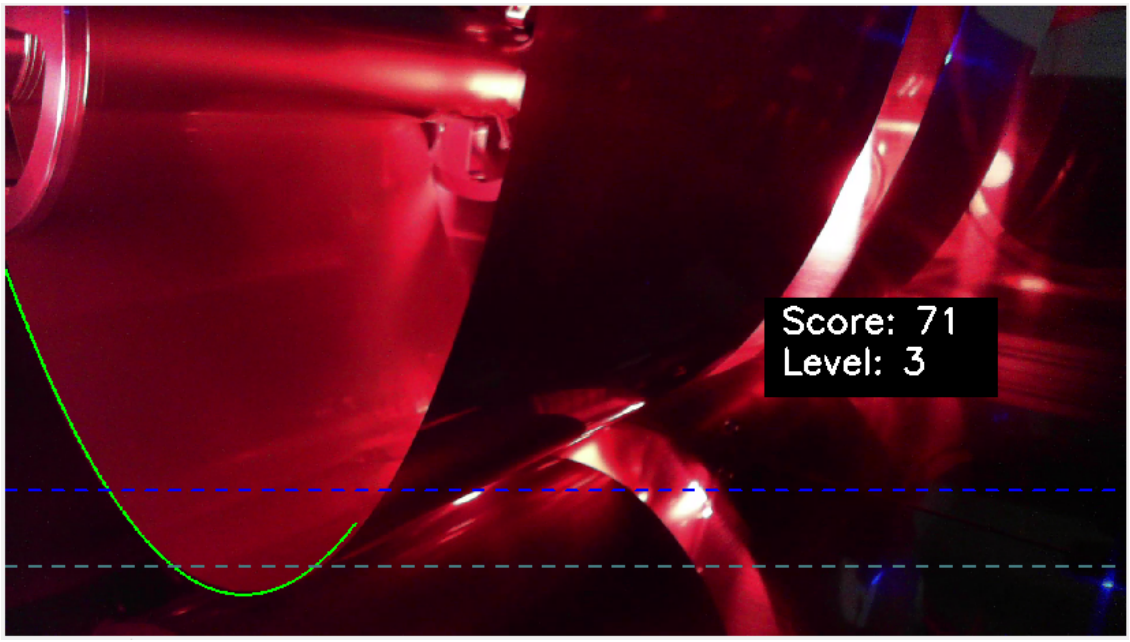

Figure 7

2. Discussion

The main reason for outlier of the curve fitting curvature was because of threshold value and the contour that was found. Especially, in level 1, the error resulted was able to verify with naked eye. However even though there was some error the result seemed appropriate. This makes that the error might be seem little big but it is not actually changing the result.

Image	Estimated	Accuracy
level 1	level 1 (score: 287)	100%
M6 Bolt	level 2 (score: 135)	100%
M6 Hex Nut	level 3 (score: 71)	100%

Table 1

Conclusion

This lab on tension detection of rolling metal sheets provided valuable insights into image processing techniques and curve fitting methods essential for industrial quality control processes. By developing a robust methodology for isolating metal sheets and analyzing their tension levels, we gained practical experience in utilizing OpenCV and Python for real-world applications.

Appendix

- **For images**

This code below is for three images given (challenging image)

```
# Load image
img = cv.imread("./Challenging_Dataset/LV3.png")
result = img
img_split = cv.split(img)
```

Load the source image and also split the image and get the channel of the red. Because the image is highly composed with red.

```
# Select ROI
x_start = 0; y_start = 460
w = 600; h = 1070
roi = img_split[2][y_start:y_start+h, x_start:x_start+w]
```

The ROI is selected by experimental method.

```
# Change to binary data
_,thresh = cv.threshold(roi, th, 255,  cv.THRESH_BINARY)

# Find contours
contours, _ = cv.findContours(thresh, cv.RETR_EXTERNAL, cv.CHAIN_APPROX_SIMPLE)

# Find the biggest contour
max = cv.contourArea(contours[0])
maxIdx = 0
for idx in range(0, len(contours)):
    if(max < cv.contourArea(contours[idx])):
        max = cv.contourArea(contours[idx])
        maxIdx = idx
```

The code above make the ROI image to binary data and finds the biggest contour that contains which is the metal sheet curvature.

```
# Save points as numpy array
x = np.zeros(len(contours[maxIdx]))
y = np.zeros(len(contours[maxIdx]))
for point in range(len(contours[maxIdx])):
    x[point] = contours[maxIdx][point][0][0]
    y[point] = contours[maxIdx][point][0][1]

# Curve fitting as 2nd order
fit = np.polyfit(x,y,2)
x_new = np.linspace(x_start, x_start+600, 500)
y_new = np.polyval(fit, x_new)
```

Then it save the contour values as numpy and use function "polyfit" and "polyval" to get the curve that best fits the metal sheet.

```
# Draw poly lines
points = np.vstack((x_new, y_new)).T.reshape(-1, 1, 2).astype(np.int32)
new_points = points + np.array([x_start, y_start]) #offset for the polyline
cause of ROI
cv.polylines(result, [new_points], isClosed=False, color=(0, 255, 0),
thickness=4)

# Retrieve maximum y pixel value
max_y = 0
for idx in range(len(new_points)):
    if(max_y < new_points[idx][0][1]):
        max_y = new_points[idx][0][1]
```

Draw the fitted curve on the source image and calculate the maximum value of y of the curve to get the tension measurement.

```
# Draw boundaries
y_1 = 1080 - 250
y_2 = 1080 - 120
x_end = img.shape[1]

def dash_lines(img, start, end, y, dashlength, color, thick):
     length = end - start
     dash_num = round(length/dashlength)

     x1 = start
     x2 = start + dashlength

     for i in range(dash_num):
         cv.line(img, (x1, y), (x2, y), color, thickness = thick)
         x1 = x2 + dashlength
         x2 = x1 + dashlength

dash_lines(result, x_start, x_end, y_1, 20, (255, 0, 0), 3)
dash_lines(result, x_start, x_end, y_2, 20, (130, 125, 70), 3)
```

This code draws dash line to show division of the boundaries on the source image.

```
# Show score and text on the image
pt1 = [1300, 500]
pt2 = [1700, 670]
cv.rectangle(result, pt1, pt2, (0,0,0), thickness = -1)
text = "Score: " + str(height)
cv.putText(result, text, [pt1[0]+30, pt1[1]+60], cv.FONT_HERSHEY_SIMPLEX, 2,
(255,255,255), thickness = 5)
text2 = "Level: " + str(level)
cv.putText(result, text2, [pt1[0]+30, pt1[1]+130], cv.FONT_HERSHEY_SIMPLEX, 2,
(255,255,255), thickness = 5)

# Show result image
cv.namedWindow(win,0)
cv.imshow(win, result)
```

Showing result on the source image like score and tension level.

- ## Video

  The code for the video is same as the image ,but there is a while loop and inside the loop it continues to capture the frame.

```
import numpy as np
import matplotlib.pyplot as plt
import cv2 as cv

win = "result" #Window name
th = 60

# Load image
cap = cv.VideoCapture("LAB3_Video.mp4")

if not cap.isOpened():
    print("Error: Could not open Video")
    exit()

while cap.isOpened():

    ret, frame = cap.read()

    if not ret:
        break

    img = frame

    img_gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
    img_split = cv.split(img)

    # Select ROI
    x_start = 0; y_start = 470
    w = 600; h = 1070
    roi = img_split[2][y_start:y_start+h, x_start:x_start+w]

    # Change to binary data
    _,thresh = cv.threshold(roi, th, 255,  cv.THRESH_BINARY)

    # Find contours
    contours, _ = cv.findContours(thresh, cv.RETR_EXTERNAL,
cv.CHAIN_APPROX_SIMPLE)

    # Find the biggest contour
    max = cv.contourArea(contours[0])
    maxIdx = 0
    for idx in range(0, len(contours)):
        if(max < cv.contourArea(contours[idx])):
            max = cv.contourArea(contours[idx])
            maxIdx = idx

    # Save the contour points as numpy array
    x = np.zeros(len(contours[maxIdx]))
    y = np.zeros(len(contours[maxIdx]))
    for point in range(len(contours[maxIdx])):
        x[point] = contours[maxIdx][point][0][0]
        y[point] = contours[maxIdx][point][0][1]
```

```
    # Curve fitting as 2nd order
    fit = np.polyfit(x,y,2)
    x_new = np.linspace(x_start, x_start+600, 500)
    y_new = np.polyval(fit, x_new)

    # Draw poly lines
    points = np.vstack((x_new, y_new)).T.reshape(-1, 1, 2).astype(np.int32)
    new_points = points + np.array([x_start, y_start]) #offset for the polyline
cause of ROI
    cv.polylines(img, [new_points], isClosed=False, color=(0, 255, 0),
thickness=4)

    # Retrieve maximum y pixel value
    max_y = 0
    for idx in range(len(new_points)):
        if(max_y < new_points[idx][0][1]):
            max_y = new_points[idx][0][1]

    # Determination of the level
    height = img.shape[0] - max_y
    level = 0
    if height > 250:
        level = 1
        print('level 1')
    elif height < 250 and height > 120:
        level = 2
        print('level 2')
    else:
        level = 3
        print('level 3')

    # Draw boundaries
    y_1 = 1080 - 250
    y_2 = 1080 - 120
    x_end = img.shape[1]

    def dash_lines(img, start, end, y, dashlength, color, thick):
        length = end - start
        dash_num = round(length/dashlength)

        x1 = start
        x2 = start + dashlength

        for i in range(dash_num):
            cv.line(img, (x1, y), (x2, y), color, thickness = thick)
            x1 = x2 + dashlength
            x2 = x1 + dashlength

    dash_lines(img, x_start, x_end, y_1, 20, (255, 0, 0), 3)
    dash_lines(img, x_start, x_end, y_2, 20, (130, 125, 70), 3)

    # Show score and text on the image
    pt1 = [1300, 500]
    pt2 = [1700, 670]
    cv.rectangle(img, pt1, pt2, (0,0,0), thickness = -1)
    text = "Score: " + str(height)
```

```
    cv.putText(img, text, [pt1[0]+30, pt1[1]+60], cv.FONT_HERSHEY_SIMPLEX, 2,
(255,255,255), thickness = 5)
    text2 = "Level: " + str(level)
    cv.putText(img, text2, [pt1[0]+30, pt1[1]+130], cv.FONT_HERSHEY_SIMPLEX, 2,
(255,255,255), thickness = 5)

    cv.namedWindow(win, cv.WINDOW_NORMAL)
    cv.imshow(win, img)

    cv.waitKey(10)
```