LAB: Tension Detection of Rolling Metal Sheet

Date: 2024-May-06

Author: Taegeon Han 21900793

Github: https://github.com/hhangun/open_picture/issues

I. Introduction

This is a simplified industrial problem for designing a machine vision system that can detect the level of tension in the rolling metal sheet.

The tension in the rolling process can be derived by measuring the curvature level of the metal sheet with the camera.

The surface of the chamber and the metal sheet are both specular reflective that can create virtual objects in the captured images. We need to design a series of machine vision algorithms to clearly detect the edge of the metal sheet and derive the curvature and tension level.

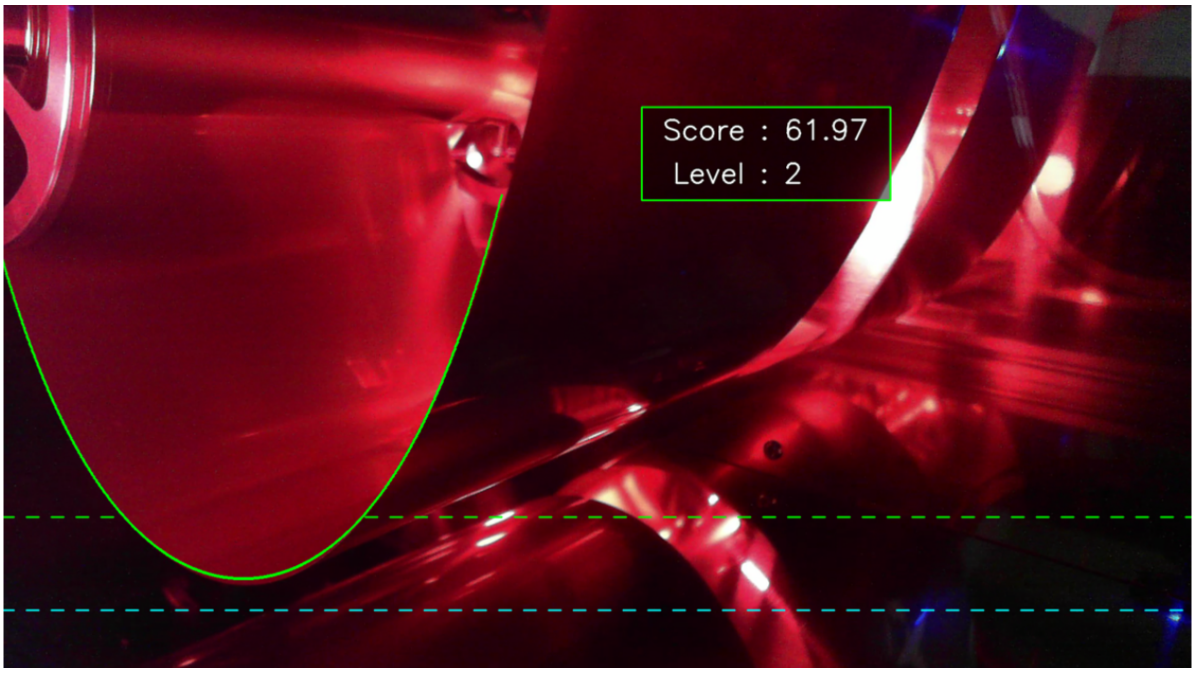


Figure 1. Completed example picture

1. Problem Conditions

- Use Python OpenCV (*.py)
- Don't use Chat-GPT or any other online materials/search
- Measure the metal sheet tension level from Level 1 to Level 3.
  - Use the minimum y-axis position of the metal sheet curvature

    - Level 1: >250px from the bottom of the image
    - Level 2: 120~250 px from the bottom of the image
    - Level 3: < 120 px from the bottom of the image
- Display the output on the raw image
    - Tension level: Level 1~3
    - Score: y-position [px] of the curvature vertex from the bottom of the image
    - Curvature edge
- Algorithm will be evaluated on similar test images
- We can choose either simple images or challenging images

2. Preparation

Software Installation

- OpenCV 4.9.0 with Visual Studio 2022

Dataset

This is a picture of the object.

Challenging dataset: Download the test images

II. Problem

Detect the edge to measure the metal sheet tension level from Level 1 to Level 3 and score from 0 to 100.

1. Procedure

1. Understand fully about the design problem
    - Download the dataset images
2. From the color raw image, cover to a gray-scaled image
    - We can use `cv.split()` to see individual channel
3. Apply Pre-processing such as filtering
4. Find ROI (region of interest) of the metal sheet from the image
    - For ROI, it does not have to be a rectangle
5. Within the ROI, find the edge of the metal sheet
6. Detect and Display the curvature of the metal edge
7. Measure the curvature's vertex (minimum point of Y-axis [px]) as the tension SCORE
    - Measure the height from the bottom of the image.
8. Detect the tension level from Level 1 to Level 3
9. Display the Final Output
    - Tension level: Level 1~3
    - Score: y-position [px] of the curvature vertex from the bottom of the image
    - Curvature edge overlay

2. Algorithm

1. Overview

This is a flowchart of this lab. It measures the metal sheet tension level and score.

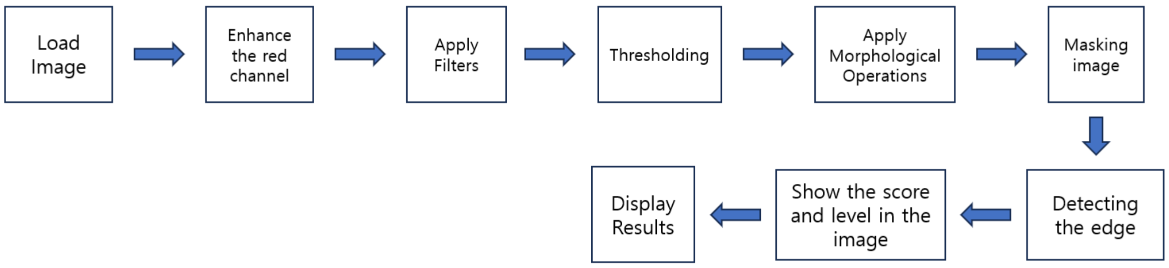


Figure 2. Outline to results

2. Assumption

- This code is based on challenge image. Therefore, if a photo and video having a pixel size different from that of challenge image are used, the condition for pixel should be changed within the code like kernel size, masking size and so on.

3. Process

Load Image & Enhance the red channel

The image was downloaded from the DLIP GitBook. Since the picture has a red color overall, cv.split() was used to improve the contrast of red. Histogram equalization was applied to the red channel to enhance its contrast, making the reds more vivid and pronounced in the resulting image. This enhancement allows red objects or features within the image to be more easily identified. We can also see it more clearly in the picture below. The figure below is Challenge Lv.3, but Challenge Lv.1 and Challenge Lv.2 similarly improved the red channels.

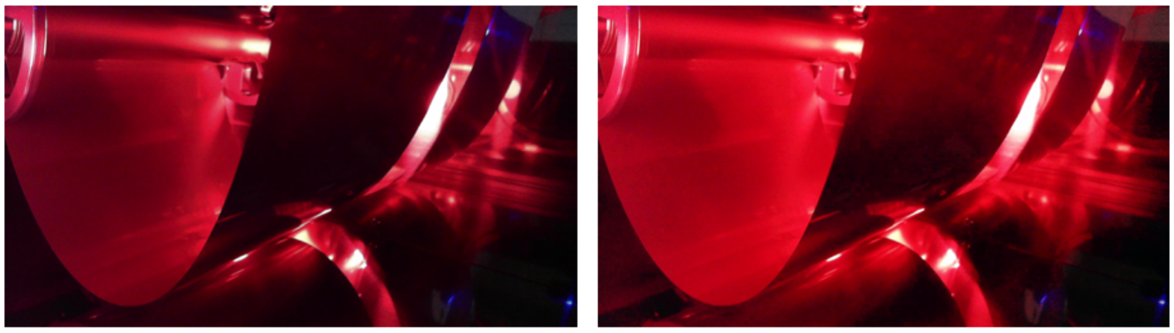


Figure 3. Original Image(Left) and Enhanced red channel image(Right)

Apply Filters and Thresholding

Filtering and thresholding were performed to classify and detect objects in the image. This step is important to improve the quality of the image and to detect the object well. Before Threshold, I wanted to reduce noise and sharpen the edges. Extreme noise was removed with median blur, and the overall image was smoothed with Gaussian blur. After that, the threshold was applied to 49, and it was used to represent the contours of all challenge images effectively.

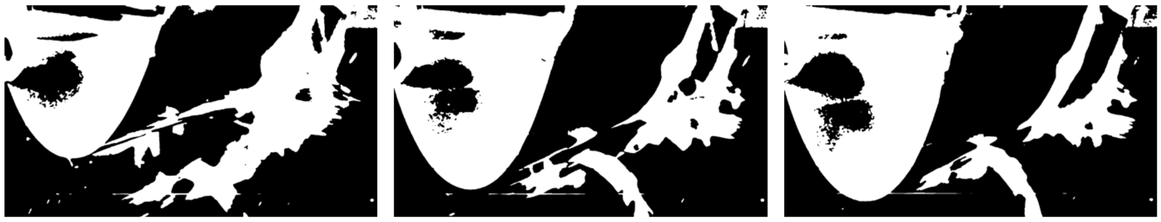


Figure 4. Filters and thresholding applied challenge images

Apply Morphology

As we can feel the difference from Figure 4, we used the Morphology technique to remove small white parts and separate others from what we want to detect the outer edge. In addition, we set the type as Threshold_binary to separate the background and the object well.

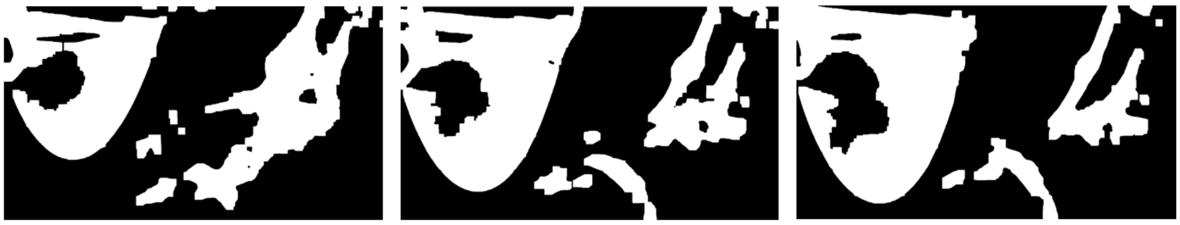


Figure 5. Threshold challenge Image

Masking Image

If we check the outline directly in Figure 5, we can recognize all the unwanted parts. Therefore, we mask only to see the part we want. However, there is a limit that masking is only a square. Therefore, parts that we didn't think of or didn't want may be included. Still, it is much better than Figure 5.

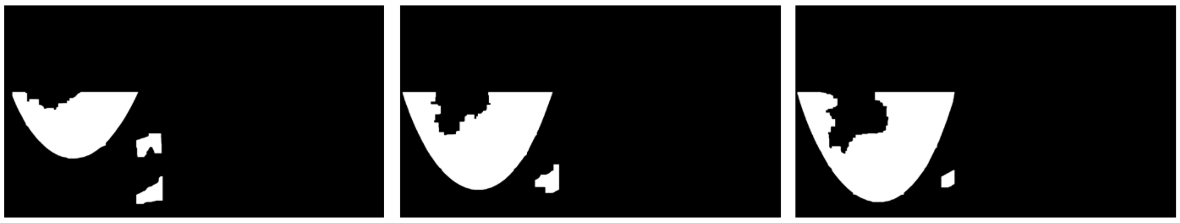
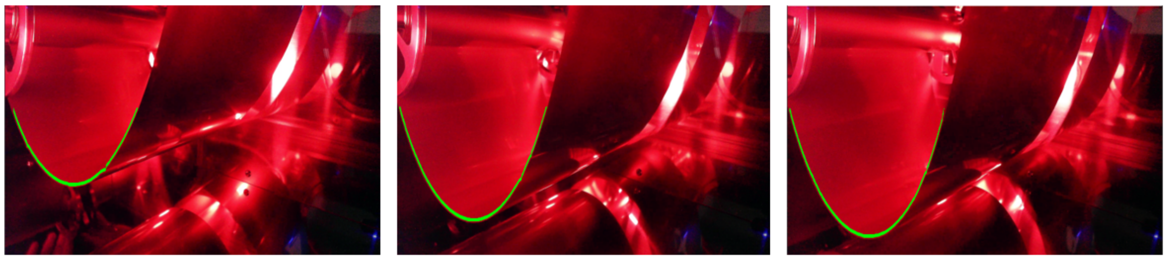


Figure 6. Masking challenge image

Detecting the edge

Check all pixels in the masked image. In the process, find a white pixel, that is, a value of 255, and draw a green circle on the corresponding pixel according to a specific condition. Certain conditions are methods to process the pixel only when it is within a given value in the y and x directions from the previous pixel. Therefore, it may look as if a line has been drawn by drawing a continuous circle.



Challenge Lv.1 Challenge Lv.2 Challenge Lv.3

Figure 7. Detected the edge image

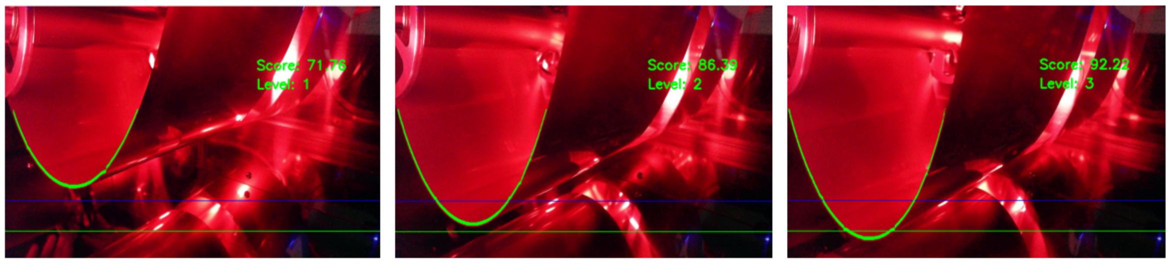
Show the score and level in the image

In detecting the edge part, it increases in the y-axis as it increases in the x-axis. In the process, we draw a circle, and we find the circle located at the bottom. If the y-value of the current pixel is larger than the previous pixel in which the circle was drawn, the lowest value stored before is newly updated with the y-value. If it goes the other way, it will not be updated and the existing value will be maintained. Remember the position of the circle drawn at the bottom like this and use it to evaluate the level and score. The bottom part of the image is 100 points and the top part is 0 points. Therefore, the score is measured by calculating the ratio according to the position of the lowest circle. In addition, the level is divided according to the value given in the Problem Condition above, and when the lowest circle is located in that value, the corresponding level is output as it is.

III. Result and Discussion

1. Final Result

As can be seen in Figure 8, the challenge image was divided according to the level and confirmed. In addition, the edge of the metal sheet was marked as described in the above process. For level measurement, the level reference line according to the pixel was drawn by referring to Problem Condition. And each level and score was evaluated at the bottom of the metal sheet, and the values were shown on the image.



Challenge Lv.1 Challenge Lv.2 Challenge Lv.3

Figure 8. Results of each Challenge Pictures

2. Discussion

The table below shows the scores and levels for each pictures.

Picture	Score	Level
Challenge Lv.1	71.76	1
Challenge Lv.2	86.39	2
Challenge Lv.3	92.22	3

Since the objective of this project is to Measure the metal sheet tension level from Level 1 to Level 3 and score, the proposed algorithm has achieved the project goal successfully.

Through the process, the edge of the metal sheet is detected clearly and the derive the curvature and tension level. Looking at the table above, it can be seen that the tension level for the corresponding score came out normally. As can be seen in Figure 8, it can be visually confirmed that the level is divided according to the blue line and the green line.

IV. Conclusion & Troubleshooting

1. Conclusion

In this project, we aimed to measure the metal sheet tension level. For this, we need to find the line of the metal sheet. Using the masking image, we narrowed the range to detect the edge, and as the x and y values increased, we drew a circle to make the edge visually visible if the condition related to the difference between the previous pixel and the current pixel within the set distance was satisfied. As a result, it can be seen that the line was properly drawn in the three level images as shown in Figure 8, and the score and level were accurately output. However, the algorithm used in this LAB has a limitation in that it has a large amount of computation because it proceeds with a lot of repetitions. Therefore, if you think of a way to reduce the amount of computation, you will get a faster result and the reaction speed will be faster.

2. Troubleshooting

In this project, it was difficult to clearly detect the edge of the metal sheet. There were sharp, large protruding parts on the edge, so it was not recognized properly in detecting the edge. So, I couldn't draw the desired line. However, I was able to draw the line as I wanted by finding the filter and proper threshold, and adjusting the morphology as well. In particular, I spent a lot of time finding the kernel size of each value. It was quite difficult to control these fine values. In addition to these values, pixel values within a specific distance were used to draw lines along the curve, and it was difficult to select a specific distance. This is also because it is heavily influenced by the pre-process.

V. Appendix

- Code for detecting the edge of the metal sheet

```
"""
* DLIP_LAB3.py
* author: Taegeon Han
* Date: 2024-05-06
* DLIP_LAB3_Tension Detection of Rolling Metal Sheet
* Measure the metal sheet tension level
"""

import cv2 as cv
import numpy as np


# Challenge picture
image_width = 1920
image_height = 1080
src = cv.imread('Challenge_LV1.png')
src = cv.resize(src, (image_width,image_height))

# Create a new image highlighting the details of the red channel
B, G, R = cv.split(src)
R_enhanced = cv.equalizeHist(R)
result_image = cv.merge((B, G, R_enhanced))

# Draw the line to divide level
cv.line(result_image, (0,result_image.shape[0]-120),
(result_image.shape[1],result_image.shape[0]-120), (0,255,0), 2, cv.LINE_AA)
cv.line(result_image, (0,result_image.shape[0]-250),
(result_image.shape[1],result_image.shape[0]-250), (255,0,0), 2, cv.LINE_AA)
dst = cv.cvtColor(result_image, cv.COLOR_BGR2GRAY)

while  cv.waitKey(1) != ord('q'):

    # Apply blur
    median = dst
    for i in range(0,1):
        median = cv.medianBlur(median,9)
    for i in range(0,1):
        median = cv.GaussianBlur(median, (7, 7), 0)

    # Apply Thresholding
    thVal = 49
    ret,thresholding = cv.threshold(median,thVal,255,cv.THRESH_BINARY)

    # Apply Morphology
    size = 21
    size2 = 15
    kernel = np.ones((size, size),np.uint8)
    kernel2 = np.ones((size2, size2),np.uint8)

    erosion = cv.erode(thresholding,kernel,iterations = 1)
    opening = cv.morphologyEx(erosion, cv.MORPH_OPEN, kernel)
    opening = cv.dilate(opening, kernel2, iterations = 1)
    # Masking Image
    mask = np.zeros(dst.shape, dtype="uint8")
    cv.rectangle(mask, (0,440), (800,1080), 255, -1)
    mask2 = np.zeros(dst.shape, dtype="uint8")+255
    cv.rectangle(mask2, (618,630), (1920,1080), 255, -1)
    mask_image = cv.bitwise_and(opening, opening, mask=mask)
    #mask_image = cv.bitwise_and(mask_image2, mask_image2, mask=mask2)


    x_limit, y_limit = mask_image.shape[1], mask_image.shape[0]  # Length and
width of mask_image
    last_x, last_y = -1, -1     # Store the x, y coordinates of previously
discovered pixels
    y_dist = 12     # Allowable pixel distance in y-direction

    low_circle = 1  # Bottom Circle

    # Check all the pixels in the image
    for x in range(x_limit):
        for y in range(y_limit):
            if mask_image[y, x] == 255:  # Check white pixel in mask_image
                # Distance conditions from previous pixels
                if last_y == -1 or abs(last_y - y) <= y_dist and abs(last_x - x)
<= 3:
                    cv.circle(result_image, (x, y), 1, (0, 255, 0), -2)  # Draw
circle
                    last_x = x
                    last_y = y

                    # Find the bottom circle
                    if low_circle < last_y:
                        low_circle = last_y
                    else:
                        low_circle = low_circle


    # resize the picture
    resize_width = 600
    resize_height = 400
    result_image = cv.resize(result_image, (600, 400))

    # Calculate by the changed size
    low_circle = low_circle*(resize_height/image_height)

    score = (low_circle / resize_height) * 100
    location_score = (int(resize_width-resize_width/3), int(resize_height/4))
    location_level = (int(resize_width-resize_width/3), int(resize_height/4 +
30))

    # Show the score, level in the picture
    if low_circle < result_image.shape[0]-250*(resize_height/image_height): #
Level 1
        cv.putText(result_image, f"Score: {score:.2f}", location_score,
cv.FONT_HERSHEY_SIMPLEX, 0.7, (0,255,0), 2)
        cv.putText(result_image, f"Level: 1", location_level,
cv.FONT_HERSHEY_SIMPLEX, 0.7, (0,255,0), 2)
        mask_image = cv.bitwise_and(mask_image, mask_image, mask=mask2)
    elif low_circle >= result_image.shape[0]-120*(resize_height/image_height): #
Level 3
        cv.putText(result_image, f"Score: {score:.2f}", location_score,
cv.FONT_HERSHEY_SIMPLEX, 0.7, (0,255,0), 2)
        cv.putText(result_image, f"Level: 3", location_level,
cv.FONT_HERSHEY_SIMPLEX, 0.7, (0,255,0), 2)
    else:   # Level 2
        cv.putText(result_image, f"Score: {score:.2f}", location_score,
cv.FONT_HERSHEY_SIMPLEX, 0.7, (0,255,0), 2)
        cv.putText(result_image, f"Level: 2", location_level,
cv.FONT_HERSHEY_SIMPLEX, 0.7, (0,255,0), 2)



    cv.imshow("Contours", result_image)

    if cv.waitKey(1) & 0xFF == ord('q'):  # Press 'q' to quit
        break

cv.waitKey(0)
cv.destroyAllWindows()
```