LAB3: Tension Detection of Rolling Metal Sheet

Date: 2025-may-5

Author: Yechan Kim 22100153

Introduction

I. Objective

Goal: Detecting the level of tension in the rolling metal sheet.

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.

You 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.

II. Preparation

Software Installation

- Visual Studio Code
- Python 3.9.21
- openCV 4.11.0

Dataset

Dataset link:

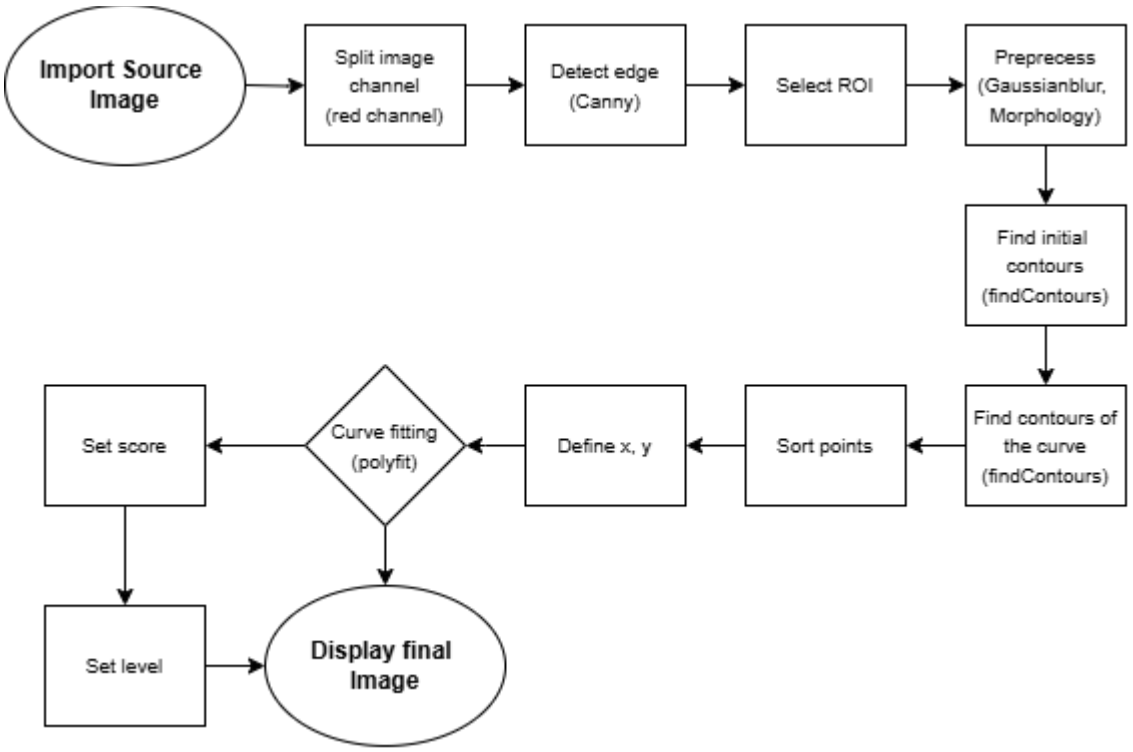
Challenging Dataset

Video

Algorithm

I. Overview

Here is a flowchart describing the algorithm.

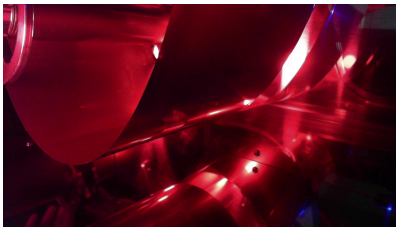
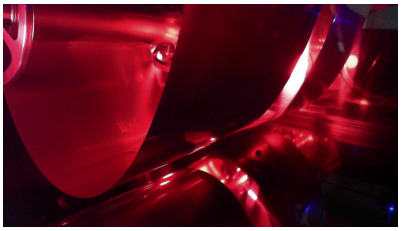
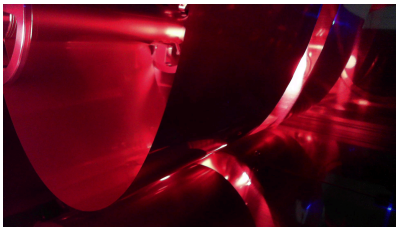


This flowchart represents the image processing steps for a single frame of the source video. The process is repeated for each frame and complied into the final video.

II. Procedure

0. Source image

Here are source images:

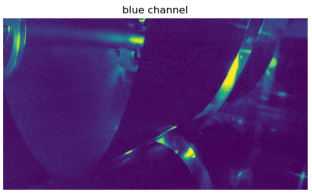
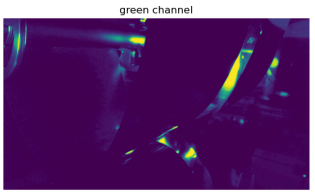
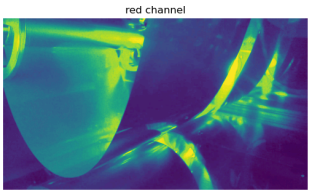
Level 1	Level 2	Level 3
		

For better readability in the report, the process was explained using the 'LV3' image.

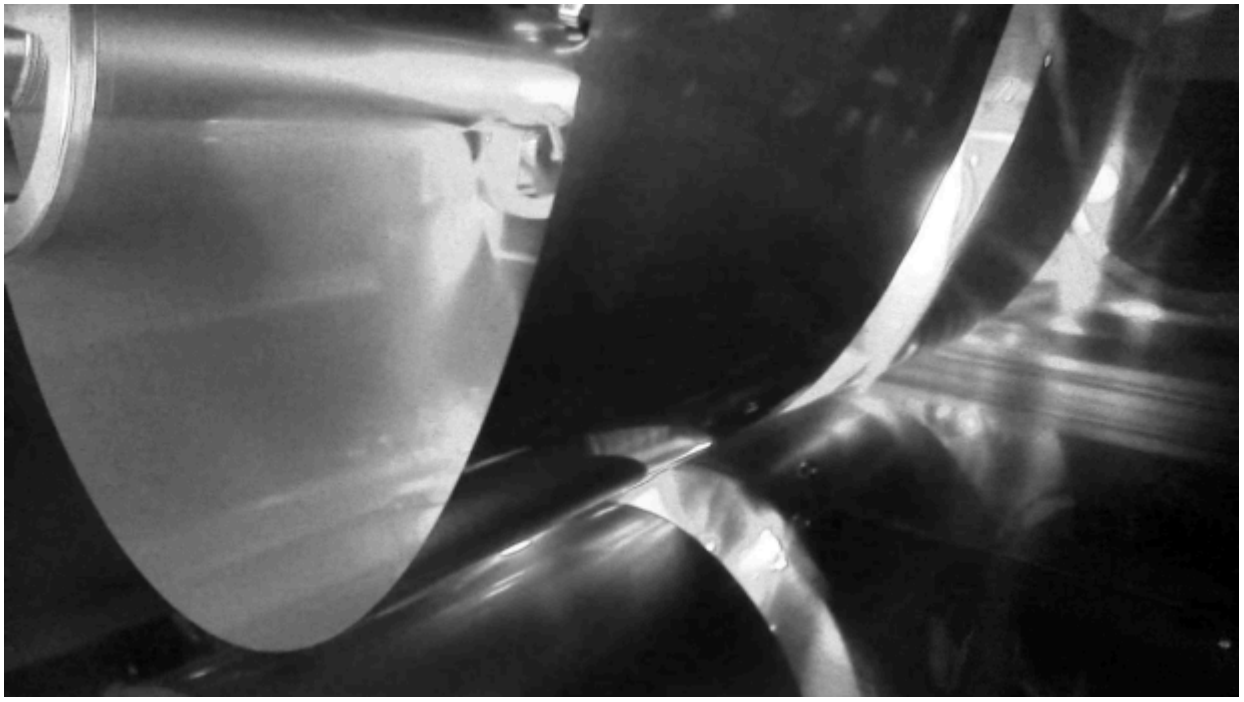
1. Edge detecting

1-1. Channel split

The metal sheet is categorized into levels 1, 2, and 3 based on its height. Since the sheet generally exhibits a reddish color, first split the source image into B, G, and R channels, and use R channel prior to performing Canny edge detection.

Blue Channel	Green Channel	Red Channel
		

When the red channel is displayed in grayscale, it appears as follows:



1-2. Canny edge detection

For the Canny edge detection, the optimal values for the two thresholds were approximately 67 and 133 when applied to a single image. However, for consecutive frames, these fixed values might not be suitable. Therefore, the thresholds were generalized using the median pixel intensity of each frame, and this adaptive approach was applied to the Canny edge detection.

The result is below:

```
source_blue, source_green, source_red = cv.split(source)

    # red 채널 기준 median 계산
    v = np.median(source_red)

    # lower, upper threshold
    lower = int(max(0,  0.90*v))
    upper = int(min(255, 2.16*v))

    source_edges = cv.Canny(source_red, lower, upper)
```

2. ROI & Preprocessing(Filtering, Morphology)

2-1. ROI

To create the ROI, four points surrounding the curve were specified to form a trapezoid. When selecting points, care was taken **to prevent distortion of the curve as the level varied.**

Points is as fallow:

-Top-left: (3, 468)

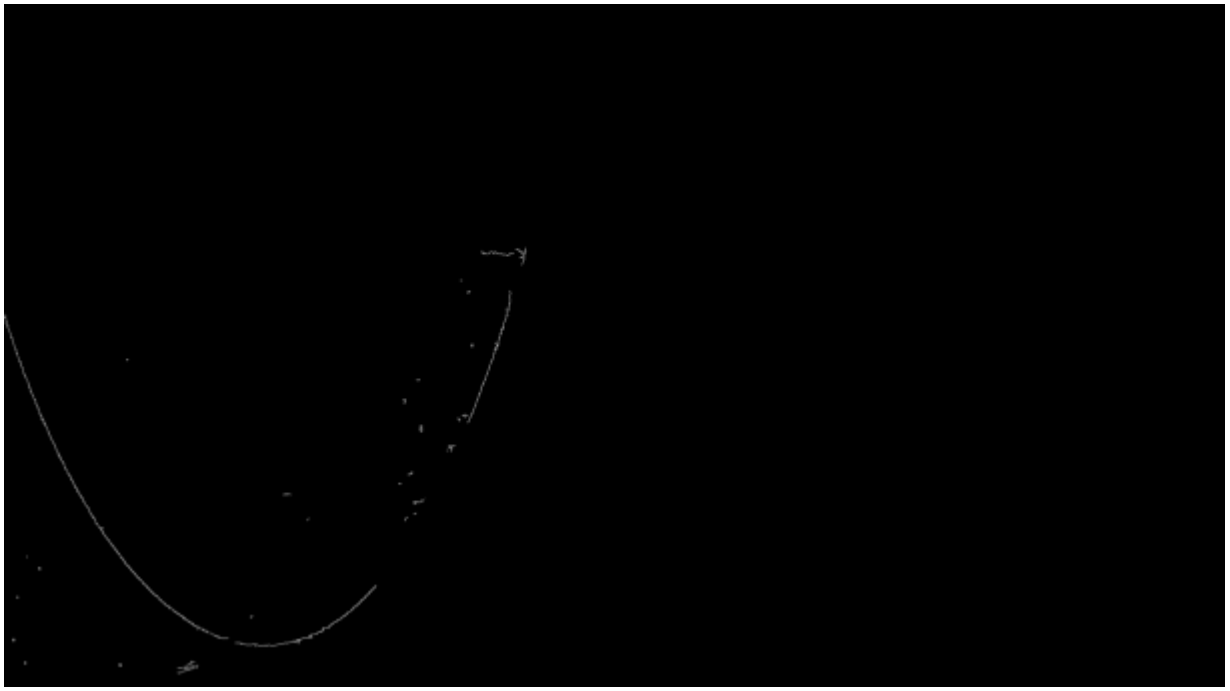
-Top-right: (790, 380)

-Bottom-left: (495, 1080)

-Bottom-right: (3, 1080)

A mask of the same size as the edge-detected image was created, with only the inside of the trapezoid filled in white. The mask was then combined with the edge image using the bitwise_and operator to generate the final ROI.

The result is below:



2-2. Gaussian filter and Morphology

Additionally, Gaussian filter and morphology was applied. For Morphology, To make the curve clearer, Closing was applied.

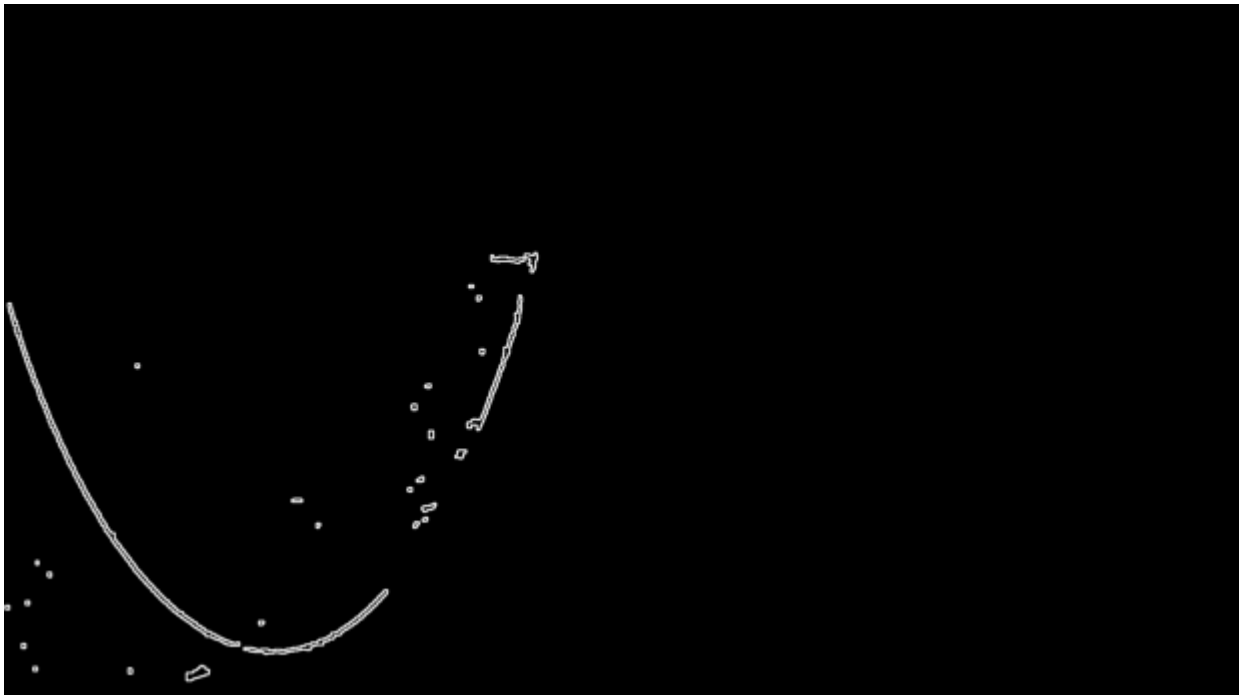
```
#roi에 대한 필터링, 모폴로지
roi = cv.GaussianBlur(roi, (5, 5), 5)
roi = cv.morphologyEx(roi, cv.MORPH_CLOSE, kernel)
```

3. Finding contours

3-1. Initial contours

To identify the curve's contour, the ROI image obtained from the previous step was first processed to detect initial contours.

The result is below:



```
#컨투어 이미지를 넣기 위한 빈 행렬
binary = source_edges.copy()
no_filtered_contours_image = np.zeros_like(binary)

#초기 roi이미지에 대해서 컨투어 찾기
contours_roi, _ = cv.findContours(roi, mode=cv.RETR_EXTERNAL, method=cv.CHAIN_APPROX_SIMPLE)
cv.drawContours(no_filtered_contours_image, contours_roi, -1, 255, 2)
```

3-2. Contour filtering

To isolate only the contour of the curve, the following criteria were applied:

- Relatively large elements
- Relatively long elements

cv.contourArea()

To calculate the area of the each contour, we used `cv.contourArea()`.

```
for cnt in contours_roi:
    area = cv.contourArea(cnt)
```

cv.boundingRect(cnt)

To calculate the the height and length of the each contour, we used `cv.boundingRect()`, which is a function that calculates the circumscribed circle of a given contour, and its output is as follows:

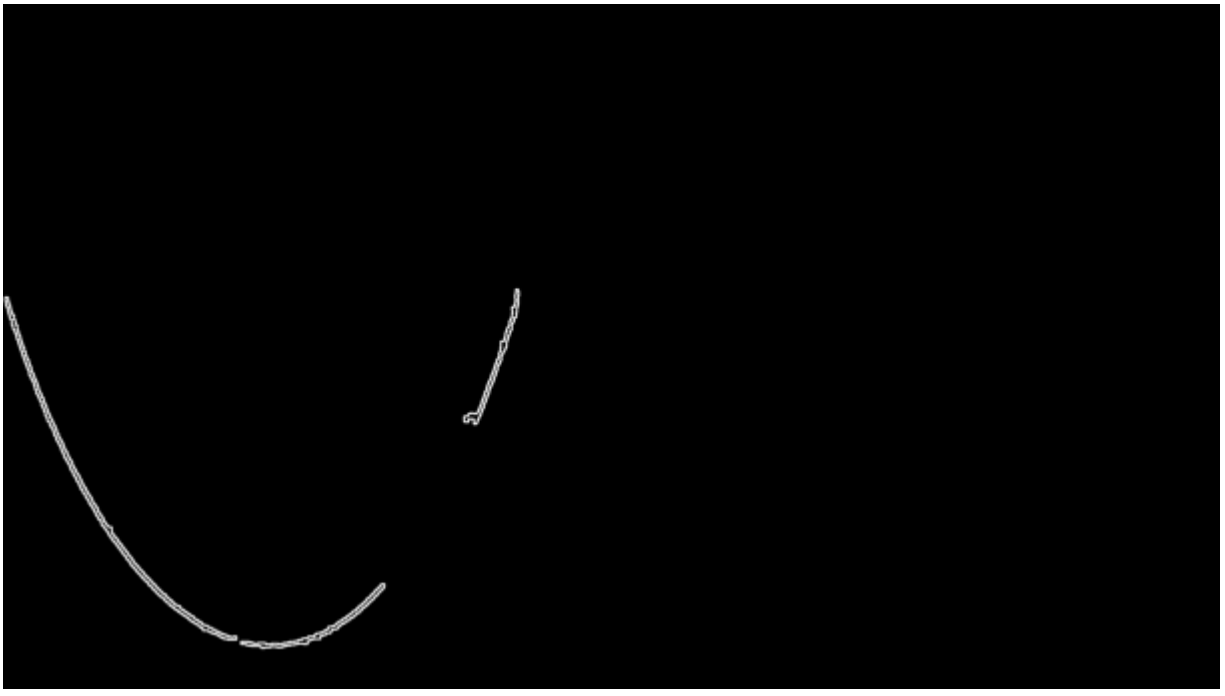
- The coordinate of the rectangular
- The width and height of the rectangular

```
 for cnt in contours_roi:
    x, y, w, h = cv.boundingRect(cnt)
```

The optimal criteria were as follows:

Criteria 1	Criteria 2
`contour area`>=1300	`height/width>0.7` and `600<=contourArea<=1300`

The results obtained by filtering the initial contours based on the given criteria are as follows:



```
for cnt in contours_roi:

    #컨투어별 면적 계산
    area = cv.contourArea(cnt)
```

```
    #외접원의 좌표, 너비, 높이
    x, y, w, h = cv.boundingRect(cnt)

    #높이에 대한 너비의 비율
    ratio=h/w

    #분모가 0인 상황 방지
    if w == 0:
        continue

    #곡선의 기준을 만족하는 컨투어를 새로 저장
    if (area >= 1300)  or (600<=area<1300 and ratio>0.7):
        contours_filtered.append(cnt)
```

4. Curve fitting

4-1. Preprocessing

Before fitting the curve, preprocessing the coordinates in filtered contours is needed.

Preprocessing steps are as below:

1. Stacking all filtered contour points into a single array and squeezed to remove any singleton dimensions.

```
# 컨투어 점들 모두 모으기
all_points = np.vstack(contours_filtered).squeeze()
```

2. Sorting the points in ascending order based on their x-coordinate values and Extract x and y coordinates.

```
# x 기준 정렬
all_points = all_points[np.argsort(all_points[:, 0])]
x = all_points[:, 0]
y = all_points[:, 1]
```

3. Using the least squares method, fit a second-degree polynomial to the data points .

```
#2차함수형태로 곡선 근사
coeffs = np.polyfit(x, y, deg=2)
poly = np.poly1d(coeffs)
```

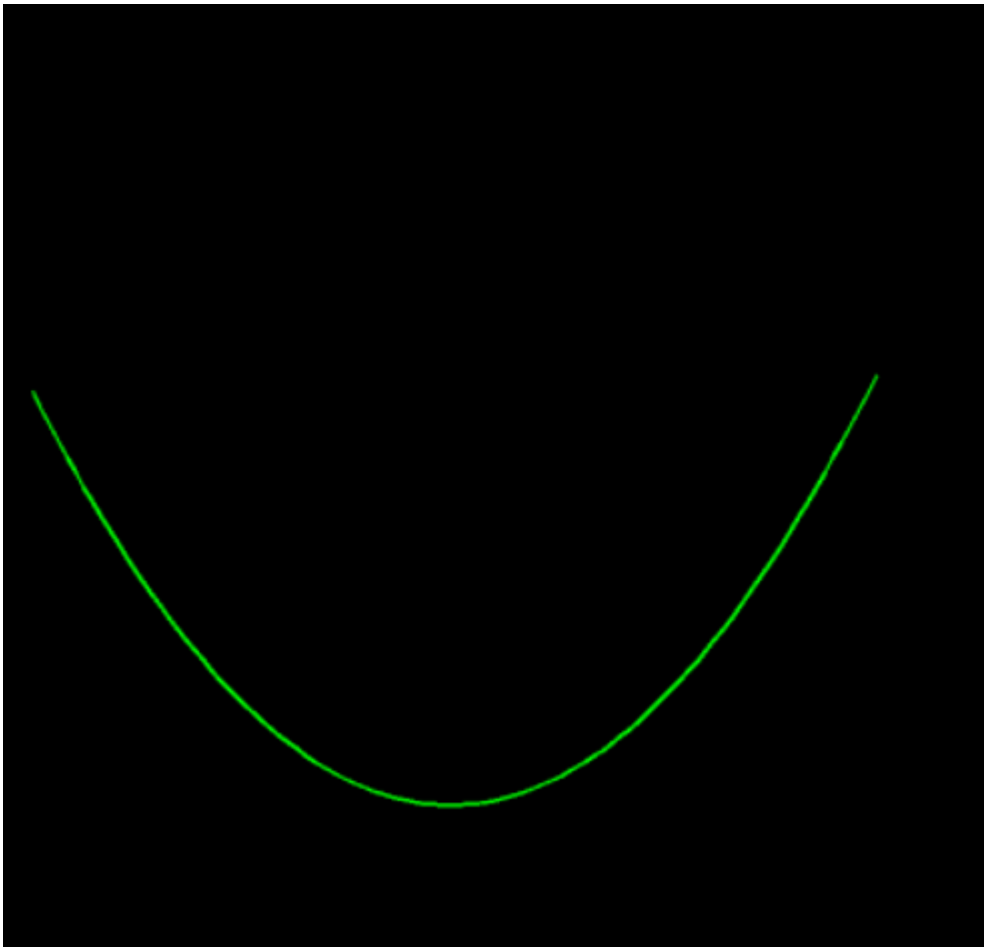
4. Defining x and y values for the fitted curve.

```
#fitting된 함수에 대입할 x,y값 정의
x_fit = np.linspace(x.min(), x.max(), 100)
y_fit = poly(x_fit)
```

4-2. Curve fitting

First, draw the fitted curve on a blank image.

Bellow is the result:

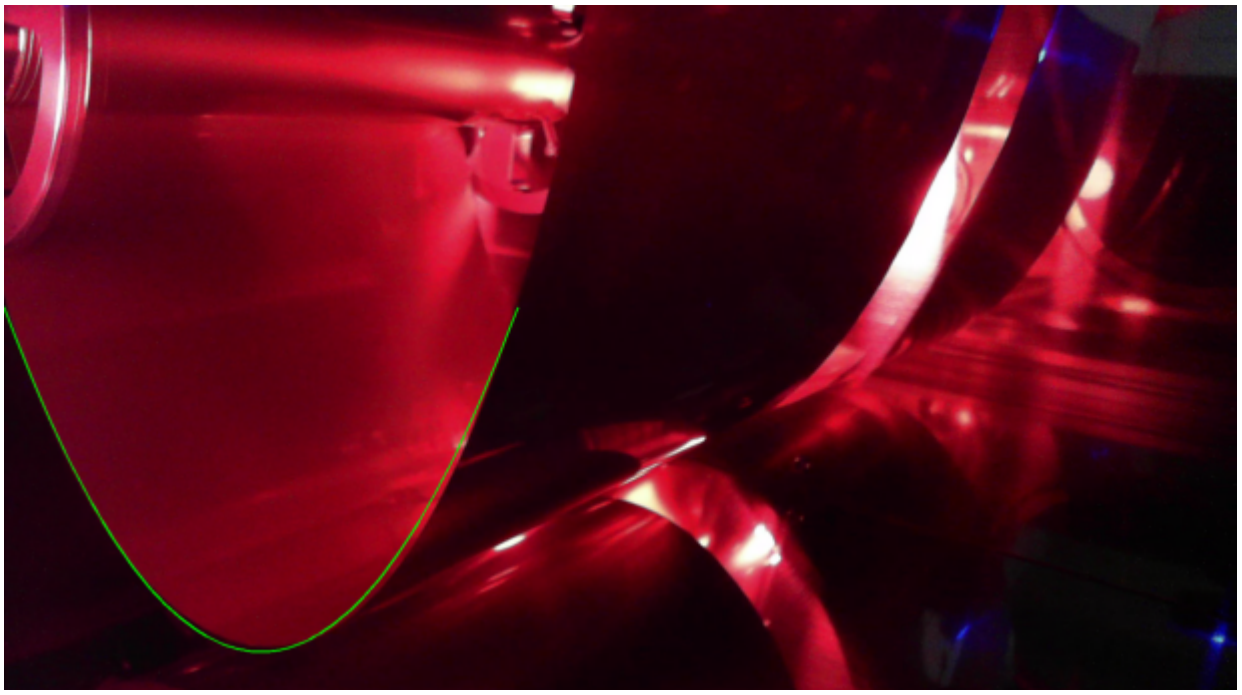


```
#피팅된 곡선을 그릴 행렬 정의
final_line=np.zeros_like(binary)
final_line=cv.cvtColor(final_line,cv.COLOR_GRAY2BGR)

#빈 이미지에 피팅된 곡선을 그려넣는다.
for i in range(len(x_fit) - 1):
    pt1 = (int(x_fit[i]), int(y_fit[i]))
    pt2 = (int(x_fit[i + 1]), int(y_fit[i + 1]))
    if 0 <= pt1[1] < source.shape[0] and 0 <= pt2[1] < source.shape[0]:
        cv.line(final_line, pt1, pt2, (0, 255, 0), 2)
```

After that, combine the curve with the original image using `cv.addWeighted()`

Bellow is the result:



```
final_output = cv.addWeighted(source, 1.0, final_line, 1.0, 0)
```

5. Score &Level

5-1. Setting the tension score

In this project, Score is y-position [px] of the curvature vertex from the bottom of the image

To do this, extract the The coordinates of the non-zero pixels from the final line image.

```
#최종 라인을 복사해와 이미지 내에서 0이 아닌 부분을 모두 가져온다.
final_line=cv.cvtColor(final_line,cv.COLOR_BGR2GRAY)
points_final_line=cv.findNonZero(final_line)
```

And then, the score is defined as the distance between the bottom of the image and the highest y-coordinate.

```
#y좌표 내에서 가장 큰 것을 score로 삼는다.
bottom_image, width=final_output.shape[:2]
ys = points_final_line[:, 0, 1]
score = bottom_image-np.max(ys)
```

5-2. Setting the tension level

The criteria of the score is as follows:

- 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

```
#스코어에 대한 레벨 설정
if score>250:
    level=1
elif score<=250 and score>=120:
    level=2
else:
    level=3
```

5-3. Drawing the level line and text

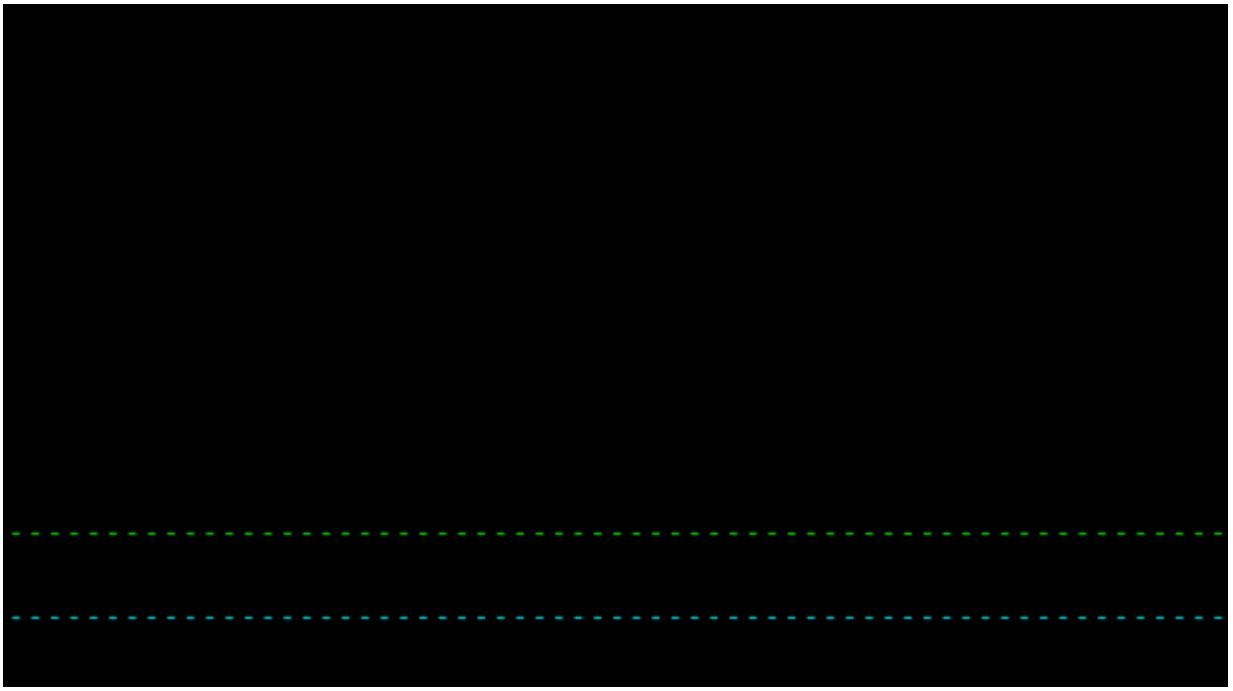
To indicate the levels on the image, y-coordinates corresponding to each level were established, and dotted lines were rendered accordingly.

```
#레벨별 y좌표
threshold_1 = int(bottom_image-250)
threshold_2 = int(bottom_image-120)
```

- Level 1: the score point is above `threshold_1`
- Level 2: the score point is below `threshold_1` and above `threshold_2`
- Level 3: the score point is below `threshold_2`

And then, draw thresholds using dotted line.

Bellow is the result:



```
#레벨에 따라 점선 그리기
for y_val, color in zip([threshold_1, threshold_2], [(0, 255, 0), (255, 255, 0)]):
    for x_pos in range(0, width, 30):
        cv.line(final_output, (x_pos, y_val), (x_pos + 10, y_val), color, 2)
```

Lastly, put score and level on the image as a text form using `putText()`

Bellow is the result:

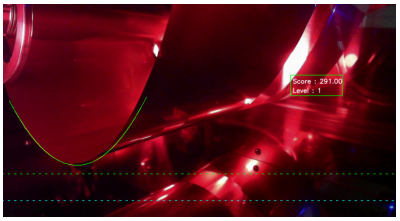
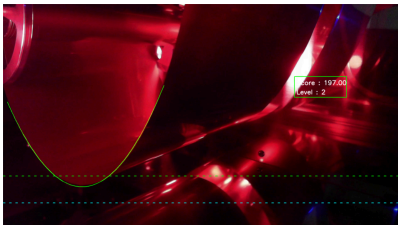
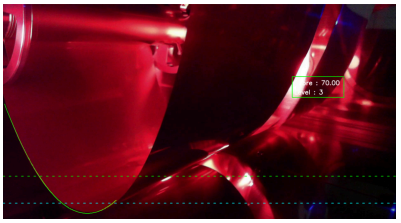

```
# 스코어
cv.putText(final_output, f'Score : {score:.2f}',(box_x + 10, box_y + 40),
cv.FONT_HERSHEY_SIMPLEX ,1 ,(255, 255, 255), 2, cv.LINE_AA)

# 레벨
cv.putText(final_output, f'Level : {level}', (box_x + 10, box_y + 85),
cv.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 2, cv.LINE_AA)
```

Result and Discussion

I. Final Result

Bellow is the final result:

Level 1	Level 2	Level 3
		

Bellow is the final result video YouTube link:

Final result

II. Discussion

While processing, the levels were successfully matched according to the degree of tension in the metal sheet, fulfilling the primary objective of the project . However, when applying level detection to video data rather than single frames, various changing conditions introduced slight errors in curve detection. In Level 1 cases, additional edges were frequently detected near the curve, which hindered accurate curve extraction. To address this, a tighter ROI (Region of Interest) was applied, but this led to curve degradation at Level 3. Consequently, a trade-off was observed between Levels 1 and 3. To solve this problem, it was necessary to identify optimized parameters for edge detection, ROI setting, and contour filtering that could consistently extract the curve across all levels.

As a result, there were no issues in detecting the levels for Levels 1 and 3, but the curves were not perfectly extracted. It is expected that if more optimized criteria had been applied during contour filtering—beyond just area, length, and width—the curve detection would have been more successful. Through this process, it became clear how important it is to establish proper criteria not only in image processing but also when addressing any engineering problem.

Conclusion

The objective of this project was to detect curves corresponding to a moving metal sheet and measure the degree of tension, classifying it into three levels. If the lowest point of the curve's y-coordinate was more than 250 pixels away from the bottom of the image, it was classified as Level 1. If it was between 120 and 250 pixels, it was classified as Level 2, and if it was less than 120 pixels, it was classified as Level 3.

To achieve this, Canny edge detection was first applied to the original image to extract the overall edges. To isolate the curve and remove the surrounding background, a Region of Interest (ROI) was defined, followed by image processing techniques such as morphological operations and Gaussian blurring. Contours were then extracted from the edges, and contour filtering was performed to retain only the contours corresponding to the curve. For the filtered curve contour, curve fitting using the least squares method was applied to detect the final curve. The level was determined based on the proximity of the curve to the bottom of the image.

As a result, the project successfully achieved its goal of classifying tension levels. However, challenges arose in consistently fitting the curve while the metal sheet was in motion. Considerable effort was made to establish appropriate criteria to distinguish the contours of the metal sheet from the background. This process underscored the importance of selecting suitable criteria when addressing engineering problems.

Appendix

Bellow is the source code:

```
import numpy as np
import cv2 as cv

cap = cv.VideoCapture('Image/LAB3_Video.mp4')
if not cap.isOpened():
    print("Error: 비디오 파일을 열 수 없습니다.")
    exit()

#모폴로지를 위한 커널을 설정
kernel = np.ones((3, 5), np.uint8)

while True:

    #비디오 불러오기
    ret, source = cap.read()

    if not ret:
        break


    #================================edge=======================================
    source_blue, source_green, source_red = cv.split(source)

    # red 채널 기준 median 계산
    v = np.median(source_red)


    # lower, upper threshold 계산
    lower = int(max(0,  0.90*v))
    upper = int(min(255, 1.80*v))

    source_edges = cv.Canny(source_red, lower, upper)
   #================================roi=========================================
```

```
#roi좌표
pts = np.array([[
    (3, 468),   #좌측상단
    (790, 380), #우측상단
    (495, 1080),#좌측하단
    (3, 1080)   #우측하단
    ]], dtype=np.int32)

#선택한 좌표 내부만 흰색인 마스크 완성
binary = source_edges.copy()
mask= np.zeros_like(binary)
cv.fillPoly(mask,pts,255)

#검출된 엣지와 선정한 부분만 흰색인 부분을 곱하여 roi설정
roi = cv.bitwise_and(mask,source_edges)

#roi에 대한 필터링, 모폴로지
roi = cv.GaussianBlur(roi, (5, 5), 5)
roi = cv.morphologyEx(roi, cv.MORPH_CLOSE, kernel)


#==============================contour=======================================

#컨투어 이미지를 넣기 위한 빈 행렬을 준비
no_filtered_contours_image = np.zeros_like(binary)
filtered_contours_image = np.zeros_like(binary)

#초기 roi이미지에 대해서 컨투어 찾기
contours_roi, _ = cv.findContours(roi, mode=cv.RETR_LIST, method=cv.CHAIN_APPROX_SIMPLE)
cv.drawContours(no_filtered_contours_image, contours_roi, -1, 255, 2)


# 초기 roi이미지에 대해서 컨투어 찾기
contours_roi, _ = cv.findContours(roi, mode=cv.RETR_LIST, method=cv.CHAIN_APPROX_SIMPLE)
cv.drawContours(no_filtered_contours_image, contours_roi, -1, 255, 2)


#필터링된 컨투어를 저장할 공간
contours_filtered = []

for cnt in contours_roi:

    #컨투어별 면적 계산
    area = cv.contourArea(cnt)

    #외접원의 좌표, 너비, 높이
    x, y, w, h = cv.boundingRect(cnt)

    #높이에 대한 너비의 비율
```

```
        ratio=h/w

        #분모가 0인 상황 방지
        if w == 0:
            continue

        #곡선의 기준을 만족하는 컨투어를 새로 저장
        if (area >= 1300)  or (600<=area<1300 and ratio>0.7):
            contours_filtered.append(cnt)

    #필터링된 컨투어가 없을 경우 예외처리
    if not contours_filtered:
        continue
    #==============================fitting=========================


    #필터링된 컨투어 점들을 모두 모으기
    all_points = np.vstack(contours_filtered).squeeze()

    #x축 기준 정렬, x, y 정의
    all_points = all_points[np.argsort(all_points[:, 0])]
    x = all_points[:, 0]
    y = all_points[:, 1]

    #2차함수형태로 곡선 근사
    coeffs = np.polyfit(x, y, deg=2)
    poly = np.poly1d(coeffs)

    #fitting된 함수에 대입할 x,y값 정의
    x_fit = np.linspace(x.min(), x.max(), 100)
    y_fit = poly(x_fit)



    #피팅된 곡선을 그릴 행렬 정의
    final_line=np.zeros_like(binary)
    final_line=cv.cvtColor(final_line,cv.COLOR_GRAY2BGR)

    #빈 이미지에 피팅된 곡선 그리기
    for i in range(len(x_fit) - 1):
        pt1 = (int(x_fit[i]), int(y_fit[i]))
        pt2 = (int(x_fit[i + 1]), int(y_fit[i + 1]))
        if 0 <= pt1[1] < source.shape[0] and 0 <= pt2[1] < source.shape[0]:
            cv.line(final_line, pt1, pt2, (0, 255, 0), 2)

    #소스 이미지와 피팅된 곡선 합치기
    final_output = cv.addWeighted(source, 1.0, final_line, 1.0, 0)

    #예외처리
    if not contours_filtered:
        cv.putText(final_output,
                'No contours found',
                (50, 50),
```

```
                cv.FONT_HERSHEY_SIMPLEX,
                1,
                (0, 0, 255),
                2,
                cv.LINE_AA)
        cv.imshow('final', final_output)
        if cv.waitKey(30) & 0xFF == ord('q'):
            break
        continue


    #============================score=======================================

    #최종 라인 이미지 내에서 0이 아닌 부분 가져오기
    final_line=cv.cvtColor(final_line,cv.COLOR_BGR2GRAY)
    points_final_line=cv.findNonZero(final_line)


    #가장 큰 y좌표가 이미지 하단에서 떨어진 정도를 score로 삼음
    bottom_image, width=final_output.shape[:2]
    ys = points_final_line[:, 0, 1]
    score = bottom_image-np.max(ys)


    #스코어에 대한 레벨 설정
    if score>250:
        level=1
    elif score<=250 and score>=120:
        level=2
    else:
        level=3

    #레벨별 y좌표
    threshold_1 = int(bottom_image-250)
    threshold_2 = int(bottom_image-120)

    #레벨에 따라 점선
    for y_val, color in zip([threshold_1, threshold_2], [(0, 255, 0), (255, 255, 0)]):
        for x_pos in range(0, width, 30):
            cv.line(final_output, (x_pos, y_val), (x_pos + 10, y_val), color, 2)

    # ============================print============================

    #스코어와 레벨 텍스트를 담기 위한 사각형 설정
    box_x, box_y, box_w, box_h = 1400, 350, 250, 100
    cv.rectangle(final_output, (box_x, box_y), (box_x + box_w, box_y + box_h), (0, 255, 0),
2)

    # 스코어 텍스트 입력
    cv.putText(final_output,
               f'Score : {score:.2f}',
               (box_x + 10, box_y + 40),
               cv.FONT_HERSHEY_SIMPLEX,
```

```
                1,
                (255, 255, 255),
                2,
                cv.LINE_AA)

    # 레벨 텍스트 입력
    cv.putText(final_output,
                f'Level : {level}',
                (box_x + 10, box_y + 85),
                cv.FONT_HERSHEY_SIMPLEX,
                1,
                (255, 255, 255),
                2,
                cv.LINE_AA)

    # 노트북에서 1080*1920프레임의 영상이 다 담기지 않아 축소
    display_scale = 0.5

    display_frame = cv.resize(final_output, (int(width * display_scale), int(bottom_image *
display_scale)))
    cv.imshow('final output', display_frame)
    if cv.waitKey(30) & 0xFF == ord('q'):
        break

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