Thickness measurement of a coating on a cross-sectional sample image using `Python`

Luis Rufino

I. INTRODUCTION

The primary objective of implementing an automated measurement script was to enhance the efficiency of experimental testing by addressing a key bottleneck associated with the manual imaging and measurement of coating thickness on the outer edges of a cross-sectional sample image. Leveraging the Python Library `OpenCV` [2] alongside its diverse array of algorithms, including `Median Blur`, `Canny edge detection`, and `Hough Line` [6], the developed script effectively identifies the coating thickness of the catheter. Consequently, a substantial reduction in time expenditure is achieved, bolstering the overall throughput of the testing process.

II. METHODOLOGY

After acquiring an image with distinct contrast and separation between the sample and coating. To use edge detection algorithms provided by `OpenCV` and many other image-processing Python libraries, you will need to convert the image into a grayscale. Since edge detection is sensitive to noise, the next step is to "smooth" the image, effectively removing the noise. This can be done through various smoothing techniques such as a standard blur, Gaussian Blur, Median Blur, and a Bilateral Filter.

Smoothing, also known as blurring an image is a simple technique to reduce noise in an image and preparation for any edge detection algorithm. A filter is applied across the whole image to perform any smoothing process on an image. Filters are applied to an image by performing a convolution. A convolution is a weighted sum of neighboring pixels, where the filter kernel determines the weights. The filter kernel is a matrix that defines the characteristics of the filter [3].

For example, a simple filter kernel could be an NxN matrix of ones, which would mean that the output pixel is the average of the surrounding pixels. This would result in a smoothed image with reduced noise.

Let $g(i, j)$ be an output of the pixel's value, of the i'th row and the j'th column, let $f(i, j)$ be the image data. $g(i, j)$ is then determined by a convolution with a filter kernel matrix $h(k, l)$ on the image

$$g(i, j) = \sum_{k,l} f(i + k, j + l)h(k, l) \tag{1}$$

The following is an example of a 3x3 Gaussian blur kernel

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \tag{2}$$

Using a median blur or any filter can be summarized using `OpenCV` in Python.

```
#img: Image source image
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
img_g_b = cv2.medianBlur(img, 9)
edges2 = cv2.Canny(img_g_b, 50, 50)
```

```
## image with an arbitrary kernel
Kernel = np.array([[0, -1, 0],
                   [-1, 5, -1],
                   [0, -1, 0]])
dst = cv2.filter2D(img, -1, kernel)
img_concat = cv2.hconcat([img, dst])
cv2.imwrite("concat.jpg", img_concat)
```

Now that a kernel filter is applied, in this case, I've applied a convolution of the median blur filter, and edge detection algorithms can be applied. Canny edge detection is used for this process. The Canny algorithm is known for its ability to detect edges accurately while suppressing noise and minimizing false positives. It provides good localization, minimal response to noise, and thin, continuous edges, making it suitable for various applications such as object detection, image segmentation, and feature extraction. Here it's used to detect the coating which can be easily pieced out from the sample.

The Canny [4] algorithm follows a series of steps to detect an edge after noise reduction.

1. Gradient calculations. The algorithm begins with performing gradient calculations on the image through the application of distinct filters in the horizontal x and vertical y directions. These filters, represented by matrix kernels, are employed to determine the rate of intensity change within the image, thereby facilitating the identification of edges.

$$G_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \text{ and } G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{3}$$

$$\text{Edge Gradient } G = \sqrt{G_x^2 + G_y^2} \tag{4}$$

2. Non-Maximum Suppression. In this stage, the algorithm identifies the local maxima within the gradient magnitude of the edges (referred to as Edge Gradient, denoted by G). By comparing each pixel's Edge Gradient with those of its neighboring pixels along the direction of the gradient, non-maximum pixels are suppressed. This process effectively diminishes the presence of extraneous edges, retaining only the most salient ones.

3. Double Thresholding. Subsequently, the Edge Gradient is subjected to a two-tiered thresholding mechanism, offering flexibility in threshold selection. The Edge Gradient is categorized into three distinct groups: strong edges, weak edges, and non-edges. Strong edges are delineated when the Edge Gradient surpasses the higher threshold, while non-edges are identified when the Edge Gradient falls below the lower threshold. The intermediate range between the two thresholds designates weak edges.

4. Edge tracking by hysteresis. The final phase aims to establish a coherent contour by connecting each edge to its adjacent strong edges. Commencing with the identification of a strong edge pixel, the Canny edge detection algorithm traverses the edge by assessing neighboring pixels. When a neighboring pixel is categorized as a weak edge, it is assimilated into the existing edge detection, as part of the same contour. This iterative process continues until no further weak edges can be connected. Through this mechanism, the algorithm ensures the formation of continuous contours in the detected edges.

The result of applying the Canny edge detection algorithm is the production of a binary image, wherein the detected edges are represented as white pixels, contrasting against a black background.

The primary task having been completed, the subsequent objective entails determining the inter-line distance established by the `Canny edge detection` algorithm. To achieve this, the utilization of the `Hough lines` algorithm becomes instrumental. Renowned for its resilience to noise, the `Hough lines` algorithm possesses the capability to detect lines even in scenarios where they are partially obscured or fragmented. Its versatile applications encompass lane detection in autonomous vehicles, shape recognition, and the extraction of lines in digital images.

The algorithm commences by portraying an image as an accumulator space, referred to as the Hough space. This parameter space serves as a representation wherein each point corresponds to a feasible line within the original image. The angle parameter, denoted as θ, along with the distance parameter, designated as ρ, are the key factors taken into consideration.

In order to construct the Hough space, the algorithm iteratively traverses all the edge points present in the input image. An edge point, characterized by a pixel that signifies a substantial discrepancy in intensity or color in relation to its neighboring pixels, is subjected to analysis. For each edge point, the algorithm calculates a set of prospective lines within the Hough space that may have generated said edge point.

When given an edge point (x, y), the algorithm repetitively loops over a predefined range of θ values (e.g., from 0 to 180 degrees) and subsequently computes the corresponding ρ value utilizing the following equation: $\rho = x\cos\theta + y\sin\theta$. Each pairing of (θ, ρ) corresponds to a line within the Hough space, thereby leading the algorithm to increment the corresponding accumulator cell.

Upon processing all edge points, an examination of the Hough space ensues to identify the most prominent lines. This examination involves scrutinizing the accumulator cells and identifying peaks that surpass a pre-established threshold. Such discerned peaks correspond to the lines that garner the greatest support from the input image. Subsequently, the algorithm is capable of reverting the lines to their original image space by employing the associated ρ and θ values derived from the identified peaks.

The Hough algorithm produces outputs in the standard Euclidean space, where each line is represented by two points denoted as P_1 and $P_2 \in \mathbb{R}$. By performing a 90-degree rotation around the midpoint, we can identify the intersection point with another edge line. The distance between this intersection point and the original line corresponds to the sample's thickness. Utilizing a rotation matrix R in Equation: 6, simplifies and enhances the efficiency of rotating all lines. Consequently, we can obtain a measurement that is perpendicular to the initial edge line generated by the Hough algorithm.

$$\text{Rotation matrix } R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \tag{5}$$

$$L_1' = \begin{bmatrix} x_1' \\ y_1' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} \tag{6}$$

The process of finding the intersection points between two lines is a well-established mathematical operation. In our context, we are provided with two points for each line segment. To ensure that the resulting intersection points lie within the bounds of the line segments, we define lines L_1 and L_2 using Bézier parameters [5].

$$L_1 = \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} + t \begin{bmatrix} x_2 - x_1 \\ y_2 - y_1 \end{bmatrix}, L_2 = \begin{bmatrix} x_3 \\ y_3 \end{bmatrix} + u \begin{bmatrix} x_4 - x_3 \\ y_4 - y_3 \end{bmatrix} \tag{7}$$

$$t = \frac{\begin{vmatrix} x_1 - x_3 & x_3 - x_4 \\ y_1 - y_3 & y_3 - y_4 \end{vmatrix}}{\begin{vmatrix} x_1 - x_2 & x_3 - x_4 \\ y_1 - y_2 & y_3 - y_4 \end{vmatrix}} = \frac{(x_1 - x_3)(y_1 - y_2) - (y_1 - y_3)(x_3 - x_4)}{(x_1 - x_2)(y_3 - y_4) - (y_1 - y_2)(x_3 - x_4)} \tag{8}$$

and

$$u = \frac{\begin{vmatrix} x_1 - x_3 & x_1 - x_2 \\ y_1 - y_3 & y_1 - y_2 \end{vmatrix}}{\begin{vmatrix} x_1 - x_2 & x_3 - x_4 \\ y_1 - y_2 & y_3 - y_4 \end{vmatrix}} = \frac{(x_1 - x_3)(y_1 - y_2) - (y_1 - y_3)(x_1 - x_2)}{(x_1 - x_2)(y_3 - y_4) - (y_1 - y_2)(x_3 - x_4)} \tag{9}$$

and

$$(P_x, P_y) = (x_1 + t(x_2 - x_1), y_1 + t(y_2 - y_1)) \text{ or } (P_x, P_y) = (x_3 + u(x_4 - x_3), y_3 + u(y_4 - y_3)) \tag{10}$$

The determination of the intersection point entails the utilization of real variables, specifically denoted as t and u. The coordinates of the intersection points, representing the meeting of two lines, are represented by the list (P_X, P_y). It is imperative to emphasize that an intersection occurs when the values of t and u satisfy the condition $0 \leq t \leq 1$ and $0 \leq u \leq 1$. However, if the denominator of either equation equates to zero, it indicates the absence of an intersection. Consequently, the magnitude of these intersection points serves as a quantitative measure of the thickness of the drug coating applied to the catheter.

III. RESULTS

Let us shift our attention toward the images in a meticulous manner. To initiate our analysis, let us closely examine a cross-sectional image specimen featuring an outer surface coating. It is imperative to acknowledge that the preparation of samples plays a pivotal role prior to capturing any image. At this scale, the presence of metal shavings resulting from cutting the specimen can be discerned under the microscope. To mitigate this issue, I have discovered that the application of adhesive tape on the surface effectively eliminates, or at the very least significantly reduces, the presence of metal shavings, thus yielding a clearer image.

Given that the primary objective of this procedure is to minimize image noise, it is prudent to forego demonstrating the variance between the sample image and its gray-scale counterpart. Instead, I shall focus on highlighting discrepancies in the behavior of the `Canny` algorithm when applied to an unfiltered image, thereby omitting redundant information.

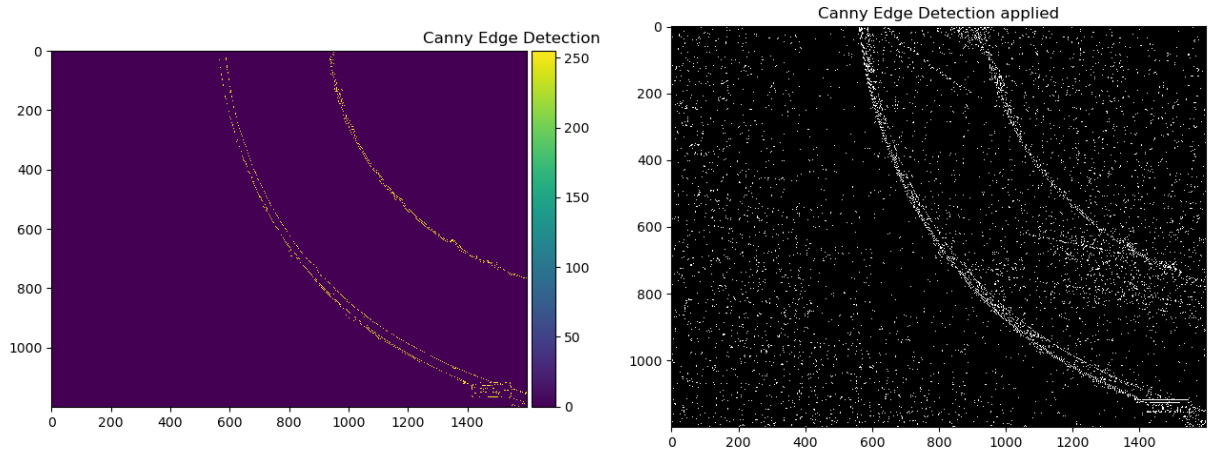

FIG. 1. The image displayed on the left demonstrates the sample image upon the application of the Median Blur filter, while the image on the right depicts the identical image in its unfiltered state, where noise reduction has not been employed.

The visual examination reveals the necessity of applying a noise reduction filter to achieve optimal results. Initially, discerning noticeable disparities between the two images proves challenging. It is worth noting the significance of Equation 4 in the `Canny` algorithm, which heavily relies on the prescribed minimum and maximum thresholds. Applying a filter kernel such as a blur or in this case, median blur meshes neighboring pixels, effectively diminishing noise and accentuating prominent color transitions, particularly those associated with the edges in the image.

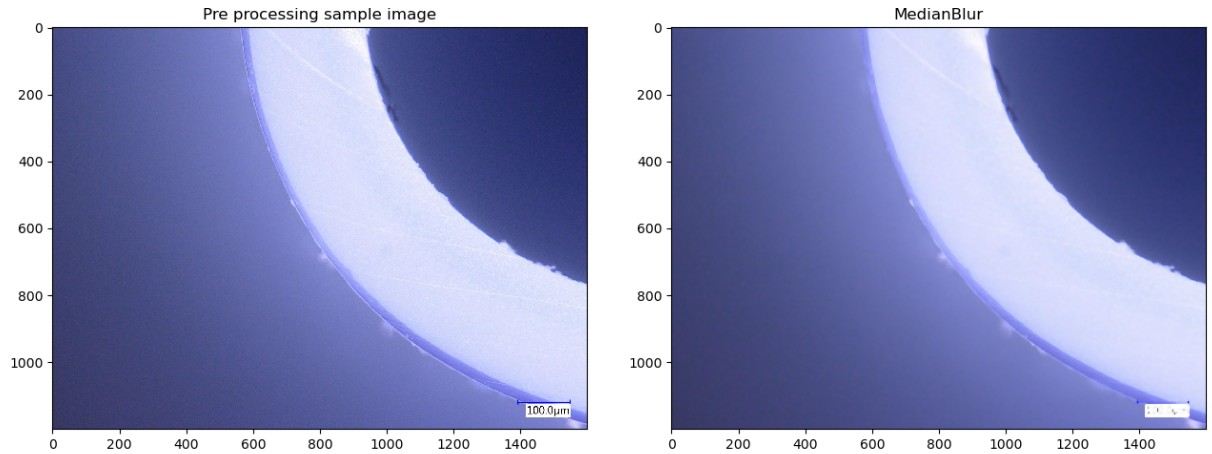

FIG. 2. Applying a median blur filter.

Now that `Canny` has successfully detected the edges in the image we can process with the algorithm, and measure the thickness of the coating.

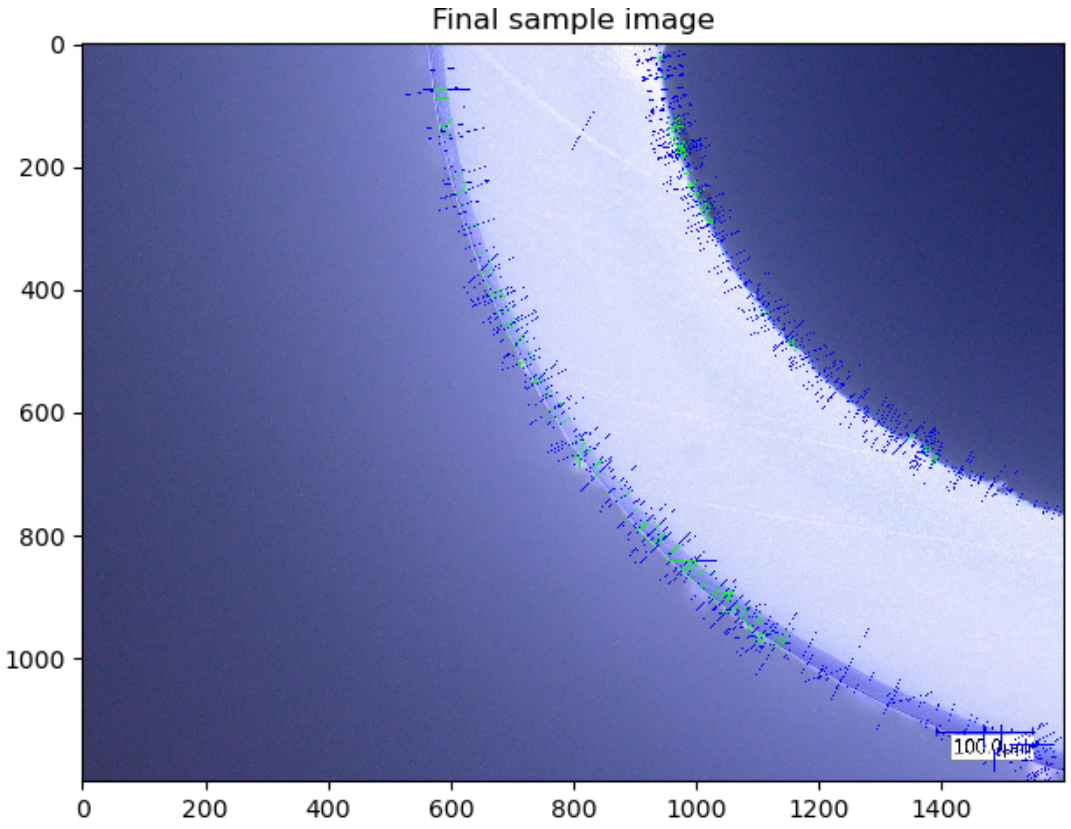

FIG. 3. Final image. The green lines are the successful lines that correspond to the length of the coating, and as a result the thickness of the coating

Summarizing, the green lines are the lines normal to the edge line, which should result in an accurate measurement of the coating thickness in pixels which can be easily converted to real units using the scale bar. It does a good job for the most part, but let's take a closer look.

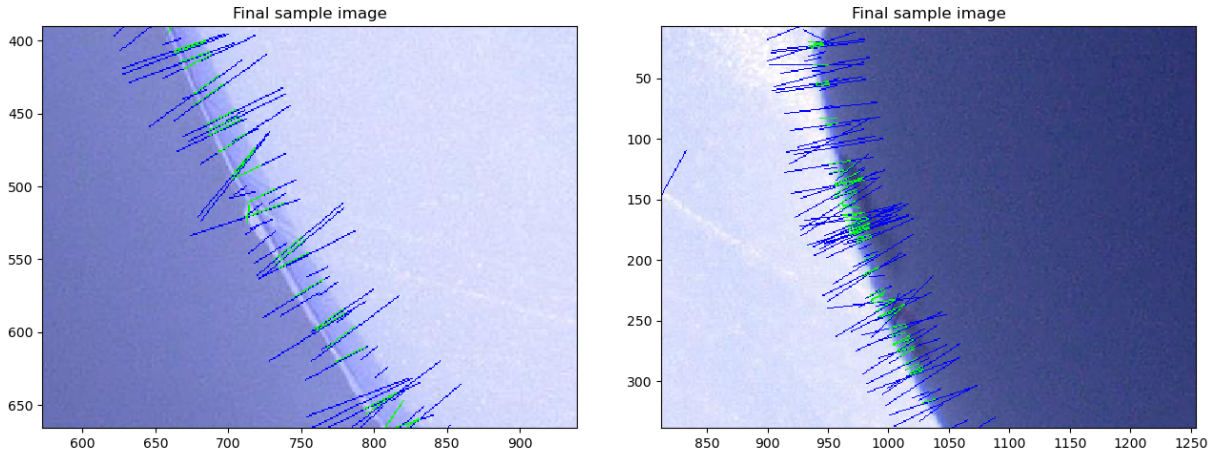

FIG. 4. Final image. The green lines are the successful lines that correspond to the length of the coating, and as a result the thickness of the coating

As you can see it does an excellent job at detecting the coating, but it also detects something inside the sample. We don't want this to skew our averages. At this moment I haven't found an excellent way to remove these outliers. Promising methods include Multivariate Outlier Detection methods such as Mahalonobis Distance [1] or Euclidean distance but I have yet to implement this into the script.

---

[1] Mahalanobis distance https://en.wikipedia.org/w/index.php?title=Mahalanobis_distance&oldid=1154436029
[2] OpenCV Documentation https://docs.opencv.org/4.x/
[3] Kernel Image processing https://en.wikipedia.org/wiki/Kernel_(image_processing)
[4] Canny edge detection https://docs.opencv.org/4.x/da/d22/tutorial_py_canny.html
[5] Line-Line intersection https://en.wikipedia.org/wiki/Line%E2%80%93line_intersection
[6] Hough Lines https://docs.opencv.org/3.4/d9/db0/tutorial_hough_lines.html