FIBRELYTICS

Composite Fiber Analysis Tool

Date: August 2025

Author: Shouq Alrumaih

1. Overview

Fibrelytics is a computer vision-powered application for analyzing cross-sectional images of fiber-reinforced composite tapes. It automates the detection and quantification of individual microscopic fibers embedded in resin. The system extracts precise measurements such as fiber centers, average radii, segmented contour areas, and computes the overall fiber volume fraction.

2. Architecture Overview

The project consists of two core components: a front-end GUI built using tkinter and enhanced with sv_ttk, and a back-end image analysis pipeline implemented using OpenCV, SciPy, and scikit-image. The front-end allows users to upload images, tweak processing parameters, draw masking polygons, and view visual feedback of each stage. The back-end handles the heavy lifting of image enhancement, segmentation, and measurements, organized into modular functions. This separation allows the front-end to stay lightweight while keeping all scientific logic in a reusable backend.

3. Dependencies

To ensure functionality, the project requires the following Python packages:

- opencv-python: Used for filtering, thresholding, contouring, masking, and more.
- numpy: Core numerical operations including arrays and mathematical calculations.
- scikit-image: Offers higher-level image analysis like watershed segmentation and peak detection.
- scipy: Used for labeling connected components in binary images.
- pandas: Used to format and export results to CSV.
- Pillow (PIL): Allows display of OpenCV images in tkinter.
- sv_ttk: A custom modern theme for a more aesthetic and user-friendly GUI.

4. Processing Pipeline

Note: The original image Figure 0 used for demonstration throughout this documentation was sourced from the article referenced as [1].

The core of the application is the function process_image_pipeline, which orchestrates several image processing steps to extract useful data from the input image.

Here is a detailed breakdown of each stage:

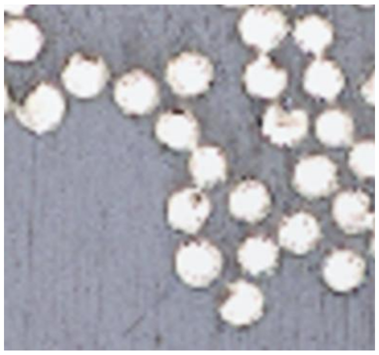


Figure 0: The original composite tape image used for analysis.

4.1 Mean Shift Filtering

Mean Shift Filtering is used to simplify the image by grouping nearby pixels with similar color intensities Figure1. This smoothing preserves edges while eliminating unnecessary color noise. The algorithm works by iteratively replacing each pixel's value with the average of neighboring pixels within a spatial (sp) and color (sr) radius. The result is an image with clustered regions, making it easier to detect fiber boundaries in later steps. This step is implemented using OpenCV's cv2.pyrMeanShiftFiltering().

4.2 Gaussian Blurring

After mean shift, a Gaussian Blur is applied to the image to reduce high-frequency noise that may cause false detections Figure1. A Gaussian filter is a weighted moving average where nearby pixels have more influence than distant ones. This softens the image and smooths out any jagged edges, preparing it for grayscale conversion. The kernel size is defined by the user (blur_size), and a larger kernel gives more blur. This step uses cv2.GaussianBlur().

4.3 Grayscale Conversion

The blurred image is then converted from color (BGR) to grayscale Figure1 using cv2.cvtColor(). This reduces the three-channel image into a single-channel intensity image, which is essential for subsequent contrast enhancement and thresholding. Since the analysis focuses on shape and contrast rather than color, working in grayscale simplifies computation and makes it compatible with most segmentation algorithms.

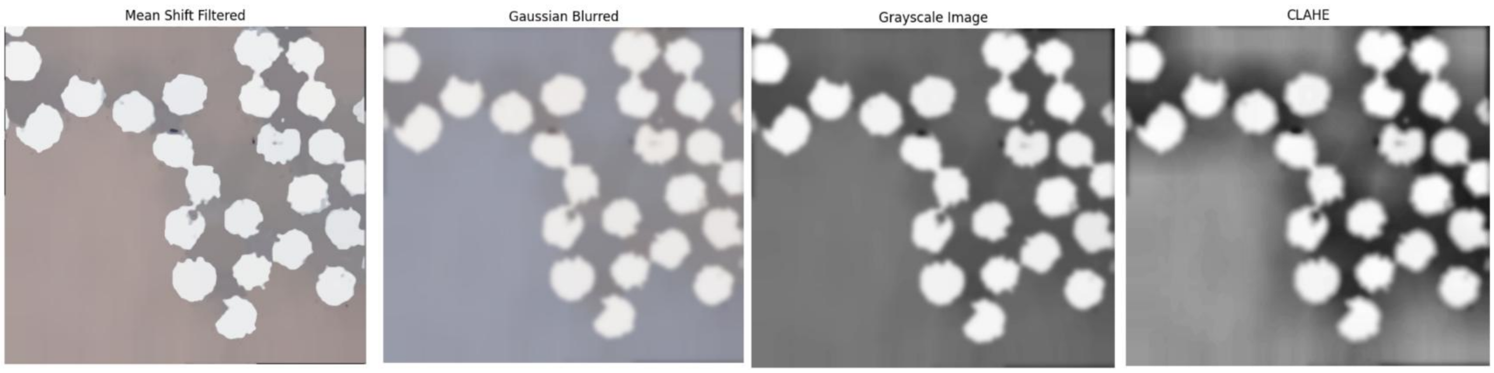


Figure 1: Shows the results of mean shift filtering, Gaussian blurring, grayscale conversion, and CLAHE contrast enhancement, respectively.

4.4 Contrast Enhancement (CLAHE)

Grayscale images may have uneven lighting or poor contrast, especially in microscopy images. CLAHE (Contrast Limited Adaptive Histogram Equalization) improves local contrast by dividing the image into tiles and enhancing each tile individually. This makes fibers appear clearer against the resin background Figure1, especially in regions with poor illumination. The user can control clip_limit (intensity cutoff) and tile_grid_size (tile shape). The implementation uses OpenCV's cv2.createCLAHE().

4.5 Thresholding (Otsu Method)

Next, the CLAHE-enhanced grayscale image is thresholded into a binary image (black-and-white) using the Otsu method Figure2. This is an automatic, data-driven thresholding technique that separates bright objects (fibers) from the darker background (resin). The binary result serves as the base for further segmentation. This is achieved via cv2.threshold() with THRESH_BINARY + THRESH_OTSU.

4.6 Euclidean Distance Map

Once the binary image is created, the Euclidean Distance Transform is applied. This computes, for each white pixel (fiber), the shortest distance to the nearest black pixel (background). The result is a grayscale image where the center of fibers appears brightest Figure2. These peaks represent the most probable fiber centers. This map is generated using OpenCV's cv2.distanceTransform().

4.7 Local Maxima Detection

After we create the distance map, we now need to pinpoint bright centers. To locate the exact centers of each fiber, we search for the local maxima in the distance map, these are the highest intensity points, typically located at the center of each fiber blob. The function peak_local_max() from scikit-image is used for this task. It identifies these peaks while respecting a min_distance parameter, which ensures that nearby fibers are not detected as the same object.

4.8 Connected Components

The binary map of local maxima is passed through connected component labeling to assign a unique integer label to each fiber center. This ensures that small clusters of adjacent peak pixels are treated as one seed region. scipy.ndimage.label() performs this task by grouping each connected region of pixels i.e., each cluster of neighboring maxima into a single labeled component. These labels serve as distinct markers for the upcoming watershed segmentation step.

4.9 Watershed Segmentation

The watershed algorithm uses the labeled markers and the original binary mask to segment touching fibers. It treats the binary mask like a topographic surface and "floods" from the peaks, building boundaries between regions Figure2. This effectively separates fibers that were clumped together in earlier steps. Implemented using skimage.segmentation.watershed().

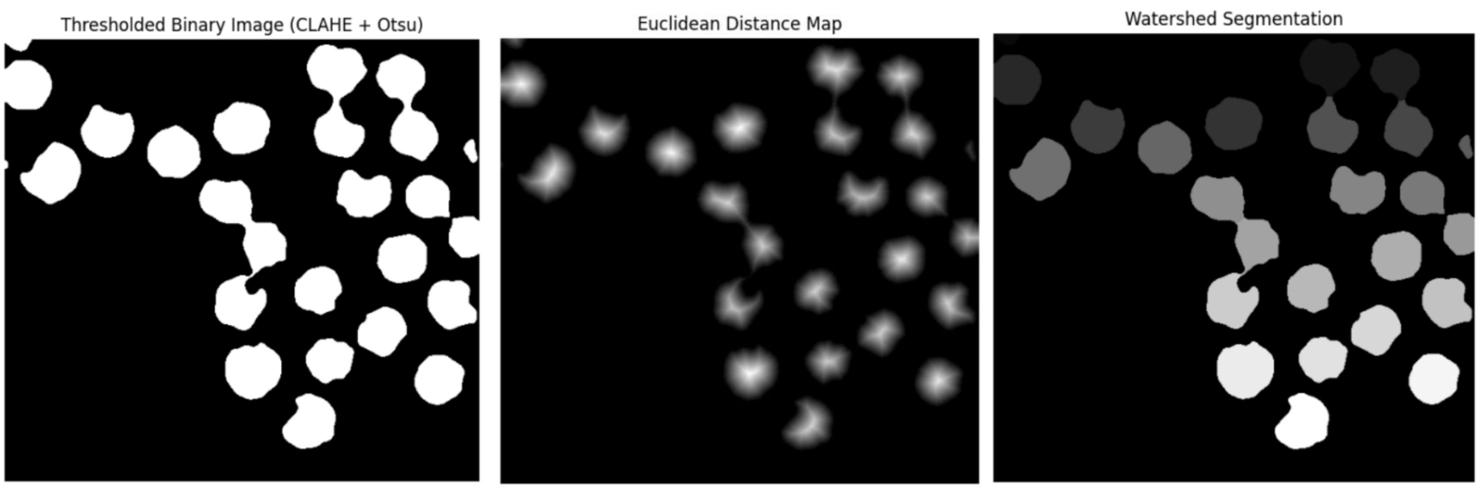


Figure 2: Shows the results of thresholding, Euclidean distance mapping, and watershed, respectively.

4.10 Contour & Center Extraction

Using the watershed result, contours are extracted for each fiber region using cv2.findContours(), and centroids (fiber centers) are calculated using regionprops() Figure 3 from skimage.measure.

4.11 Proximity Filtering

Sometimes, false positives appear too close to one another, suggesting duplicate detections. A custom filter checks all detected centers and removes those that are too close, based on a threshold distance.

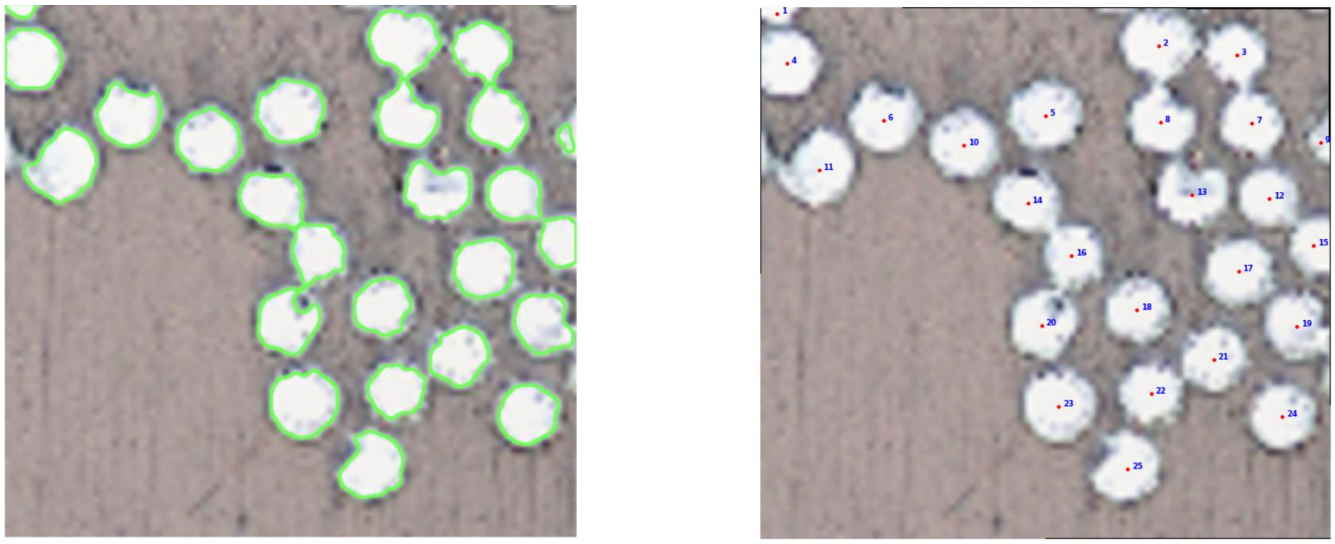


Figure 3: Shows the results of contouring and center extraction respectively.

4.12 Average Radius Estimation

Each accepted center is evaluated using a ray-casting method: rays are cast in 8 directions from the center, and the distance to the contour is measured along each ray. These distances are averaged to obtain a reliable estimate of the fiber's radius. This method adapts well to irregular shapes Figure 4.

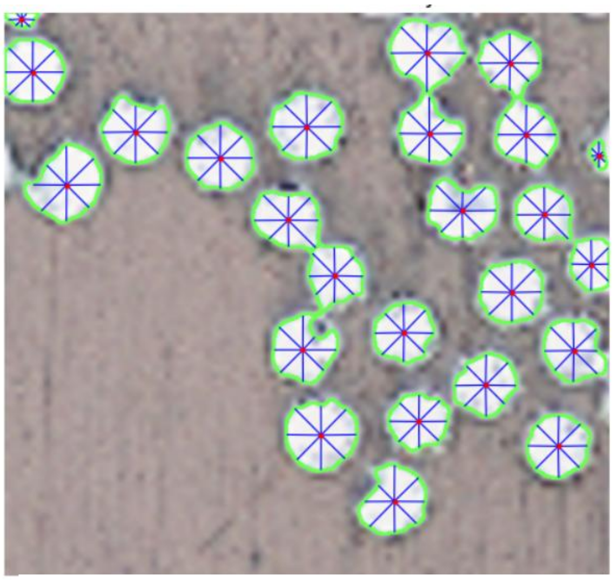


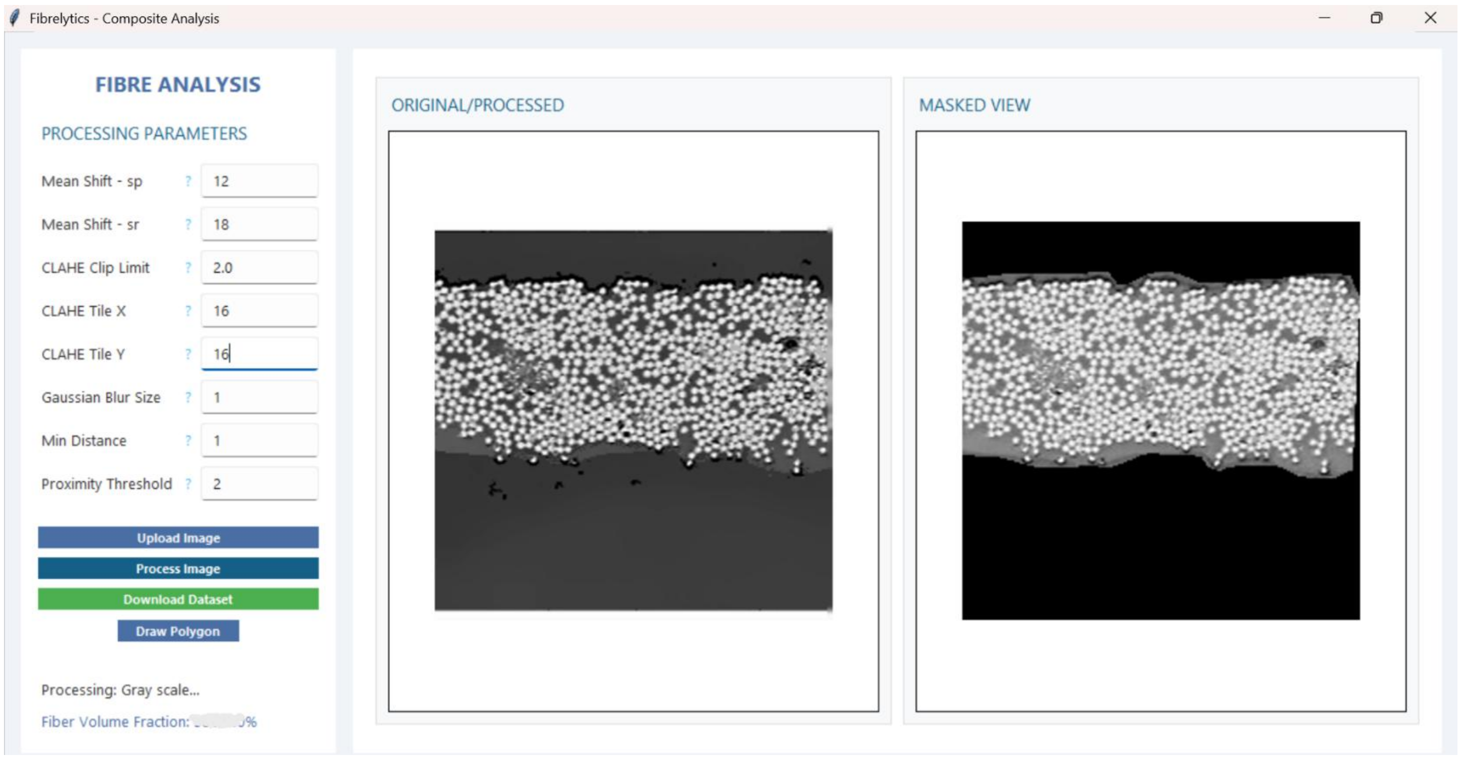
Figure 4: Shows ray-casting method used for estimating the avg radius of each fiber

4.13 Fiber Volume Fraction (Vf)

The Fiber Volume Fraction is computed as the ratio between the sum of all fiber areas (calculated from contours) and the total image area. If the user has drawn a polygon mask Figure 5, only the masked region is considered for the denominator. This final value is crucial for material characterization and is returned as a percentage.

5. Optional Masking with Polygon

To ensure that the analysis focuses strictly on the tape region, users can optionally draw a polygon to exclude irrelevant parts of the image Figure 5. This is particularly useful when the image contains noisy borders or elements outside the tape area that could distort the results. The polygon is converted into a binary mask using cv2.fillPoly() and applied to the image using cv2.bitwise_and(). This mask used in the fiber volume fraction calculation to ensure that only the selected tape region is considered.



Mean Shift - sp ? 12
Mean Shift - sr ? 18
CLAHE Clip Limit ? 2.0
CLAHE Tile X ? 16
CLAHE Tile Y ? 16
Gaussian Blur Size ? 1
Min Distance ? 1
Proximity Threshold ? 2

Processing: Gray scale...
Fiber Volume Fraction: [redacted]%

Figure 5: The Fibrelytics graphical user interface (GUI)

6. GUI Features and Styling

The front-end is built with tkinter and styled with sv_ttk Figure 5. Users can:

- Upload images
- Adjust parameters with tooltips
- Visualize intermediate results step-by-step
- Draw polygon masks
- Monitor progress using a progress bar
- Download a CSV of all fiber data

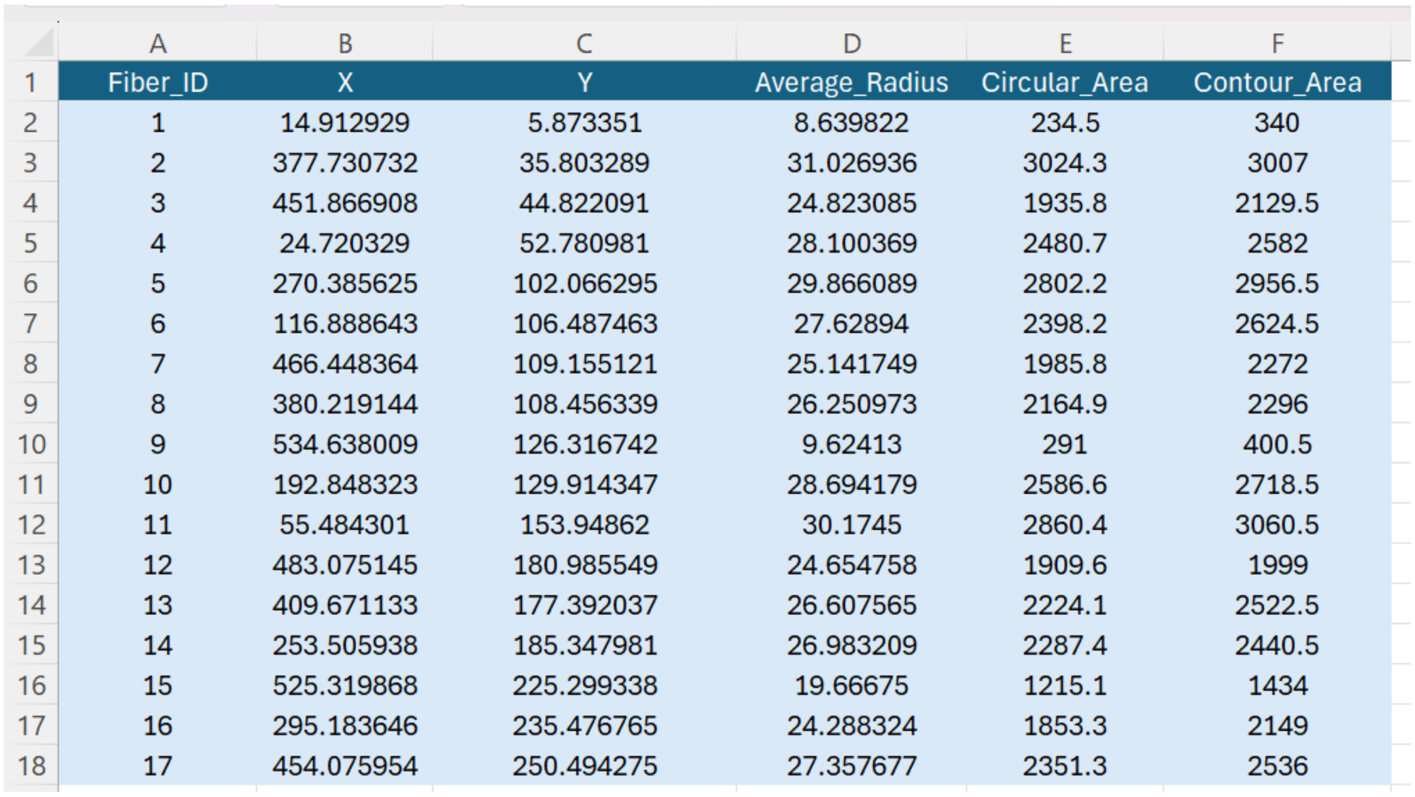
The GUI includes two main panels: a control panel (parameters and actions) and a display panel (original and masked image previews).

7. Data Export

The function save_fiber_data() saves all measurements to a CSV file. Each row represents a fiber, and includes:

- Fiber_ID: Unique index
- X, Y: Coordinates of the fiber center
- Average_Radius: Mean of ray lengths
- Circular_Area: Area computed from radius (πr^2)
- Contour_Area: Area from the actual segmented contour

This dataset enables further statistical analysis or comparison across samples Figure 6.



	A	B	C	D	E	F
1	Fiber_ID	X	Y	Average_Radius	Circular_Area	Contour_Area
2	1	14.912929	5.873351	8.639822	234.5	340
3	2	377.730732	35.803289	31.026936	3024.3	3007
4	3	451.866908	44.822091	24.823085	1935.8	2129.5
5	4	24.720329	52.780981	28.100369	2480.7	2582
6	5	270.385625	102.066295	29.866089	2802.2	2956.5
7	6	116.888643	106.487463	27.62894	2398.2	2624.5
8	7	466.448364	109.155121	25.141749	1985.8	2272
9	8	380.219144	108.456339	26.250973	2164.9	2296
10	9	534.638009	126.316742	9.62413	291	400.5
11	10	192.848323	129.914347	28.694179	2586.6	2718.5
12	11	55.484301	153.94862	30.1745	2860.4	3060.5
13	12	483.075145	180.985549	24.654758	1909.6	1999
14	13	409.671133	177.392037	26.607565	2224.1	2522.5
15	14	253.505938	185.347981	26.983209	2287.4	2440.5
16	15	525.319868	225.299338	19.66675	1215.1	1434
17	16	295.183646	235.476765	24.288324	1853.3	2149
18	17	454.075954	250.494275	27.357677	2351.3	2536

Figure 6: A sample of the generated dataset showing key measurements for each detected fiber, including center coordinates (X, Y), average radius, circular area, and contour area. The coordinates help visualize the spatial distribution of fibers across the tape. Additionally, the comparison between circular and contour areas reveals that while many fibers are nearly circular, others differ significantly suggesting the presence of irregular or non-circular fibers.

8. Conclusion

Fibrelytics was built to make fiber analysis easier, faster, and more visual. Instead of manually estimating fiber boundaries or measuring radii and area by hand, users can now rely on an interactive tool that clearly illustrates what's happening at each stage. Every step is transparent, intuitive, and guided by visual feedback. While the current version was developed in just two weeks meaning there's still room for improvement and extended testing, it already transforms what was once a tedious process into a streamlined and reproducible one.

You can find the code here: Click Here.

9. Reference

[1] N. Katuin, D. M. J. Peeters, and C. A. Dransfeld, "Method for the Microstructural Characterisation of Unidirectional Composite Tapes," *Journal of Composites Science*, vol. 5, no. 10, p. 275, Oct. 2021. [Online]. Available: https://doi.org/10.3390/jcs5100275