Image Processing for Quantification of Nanowire Alignment on Surfaces

April 14, 2018

Description

This program quantifies the orientation order of SEM nanowire images by using a specific order parameter. The program takes an SEM nanowire image in the .png format and applies multiple filters on it. The final filtered image is then processed to determine nanowire orientation. It returns:

- A graph of the orientation order parameters.
- histogram of the unweighted orientations of each nanowire.
- A histogram of the weighted orientations of each nanowire.

You may also choose to display the SEM image after each filter.

High-Level Overview

The user uploads a .png format SEM image to the program. The program then does 4 sequential tasks to modify the image:

Image filtering

- The image is converted into grayscale.
- A denoising filter is applied.

Thresholding

- Each pixel in the image is compared to a specified value. If the pixel is above the threshold, it is assigned to the foreground, else it is assigned to the background.

Object Detection

- Topological skeletons of the nanowires are created.

Shape Fitting

  - End-points and branch-points (regions of nanowire overlap) are determined from the topological skeleton.

Afterwards, using the end-points and branch-points, the program determines orientation order parameters for the nanowires.

Orientation Order Parameter

The **orientation order parameter S** is defined as:

$$S = \langle 2\cos^2\theta_i - 1\rangle = \frac{1}{N}\sum_{i=1}^{N}(2\cos^2\theta_i - 1)$$

Where:

- N is the total number of nanowires.
- θ_i is the angle between an average alignment vector and the ith nanostructure alignment vector.

The orientation order parameter can have values ranging from 0 to 1. The closer the parameter is to 1, the closer the nanowires are to perfect alignment.

Image Modification Functions

load_image

load_image(filename, display=False)

> **filename**: a string that indicates the path to the image to be processed.
>
> **display**: a boolean that determines if the grayscale image is to be shown.

The function takes the image to be processed and converts it into grayscale. Returns the grayscale image.

digitize_image

digitize_image(grey_image, block_size, threshold='otsu', denoising_strength=30, display=False)

> **grey_image**: the grayscale version of the image to be processed.
>
> **block_size**: an integer which determines the size of the pixel neighborhood used to compute a threshold value.
>
> **threshold**: a string, either specified otsu or adaptive, that determines the type of thresholding to be done.

otsu: automatically calculate a threshold value based off the histogram for the bimodal image.

adaptive: determines specific threshold values for small regions of the image.

denoising_strength: an integer which regulates the filter strength. The higher the value, the stronger the denoising.

display: a boolean that determines if the image applied through the filter and threshold is shown.

The function applies an OpenCV denoising filter to the image and then applies either otsu or adaptive thresholding. Returns a binary version (pixel values represented as 1s and 0s) of the original image.

_hit_and_miss

_hit_and_miss(image, hit, miss)

image: the grayscale version of the image to be processed.

hit: a matrix which defines the criteria for a pixel to be considered a foreground pixel.

miss: a matrix which defines the criteria for a pixel to not be considered a foreground pixel.

The function applies a hit-or-miss transform to the image. The transform sets all pixels which conform to the hit kernel to foreground and all pixels which do not conform to the hit kernel / conform to the miss kernel to background. Returns the modified image.

_thin

_thin(image, hit, miss)

image: the grayscale version of the image to be processed.

hit: a matrix which defines the criteria for a pixel to be considered a foreground pixel.

miss: a matrix which defines the criteria for a pixel to not be considered a foreground pixel.

This function applies the hit-or-miss transform in such a way as to skeletonize the image. Returns the modified image.

_erosion

_erosion(image, kernel, invert=False)

> **image**: the binary image to be processed.
>
> **kernel**: a matrix which defines the criteria for a pixel to be considered a foreground pixel.
>
> **invert**: a boolean which swaps foreground and background pixels after erosion if true.

This function implements morphological erosion unto the image. Erosion operates by superimposing the input kernel onto every pixel in the input image. When the kernel is superimposed onto an input pixel, if all the pixel values underneath the kernel match up with the respective values in the kernel, the input pixel is left as foreground, else it is set to background. Returns the eroded image.

skeletonize_image

skeletonize_image(bw_image, prunings=0, display=False)

> **bw_image**: the image to be processed.
>
> **prunings**: an integer which determines how much to prune the image. Pruning the image removes artifacts.
>
> **display**: a boolean that determines if the skeletonized image is shown.

This function creates a skeleton image of the input bw_image. A skeleton image removes most foreground pixels while maintaining the connectivity of the original image. This function creates a skeleton image using morphological thinning and carefully selected structuring elements. Returns the skeleton image of the original bw_image.

Image Analysis Functions

find_branch_points

find_branch_points(bw_image, skel_image, display=False)

> **bw_image**: the binary image to be processed.
>
> **skel_image**: the skeleton image obtained from applying the skeletonize_image function the bw_image.
>
> **display**: a boolean that determines if the image with indicated branch points is shown.

This function determines the branch points of the skeletonized image, with branch points being defined as the intersection between two nanowires. Branch points are determined using specially selected structuring elements. Returns a list of branch point coordinates and the number of branch points.

find_end_points

find_end_points(bw_image, skel_image, display=False)

> **bw_image**: the binary image to be processed.
>
> **skel_image**: the skeleton image obtained from applying the skeletonize_image function on the bw_image.
>
> **display**: a boolean that determines if the image with indicated branch points is shown.

This function determines the end points of the skeletonized image, with end points being defined as the ends of a nanowire. End points are determined using specially selected structuring elements. Returns a list of end point coordinates and the number of end points.

find_segments

find_segments(bw_image, skel_image, cutoff, prunings=0, display=False

> **bw_image**: the binary image to be processed.
>
> **skel_image**: the skeleton image obtained from applying the skeletonize_image function on the bw_image.
>
> **cutoff**: an integer representing the maximum length of a nanowire in pixels.
>
> **prunings**: an integer which determines how much to prune the image. Pruning the image removes artifacts.
>
> **prunings**: a boolean that determines if the modified image is shown.

This function applies the find_end_points and find_branch_points functions to the skel_image. From there, the function determines end point pairs that exist within the cutoff. Returns a list of valid end point pairs and rejected end point pairs.

Orientation Distribution Function (ODF) and Orientation Order Parameter Functions

compute_order

compute_order(nw_segments, order=6, display=False)

> **nw_segments**: a list of coordinates of end point pairs.
>
> **order**: the maximum order to be calculated up to.
>
> **display**: currently not implemented.

This function, in conjunction with the fit_odf_samples function determines the orientation of the nanowires and translates this information into a set of orientation order parameters up to order orders. Returns the weighted and unweighted order parameters, the length of the nanowires, and the weighted and unweighted orientations of the nanowires.

fit_odf_samples

fit_odf_sample(orientation, length, order, display=False)

> **orientation**: the orientation of each nanowire, given by compute_order.
>
> **length**: the length of each nanowire, given by compute_order.
>
> **order**: the maximum order to be calculated up to, given by compute_order.
>
> **display**: currently not implemented.

This function calculates weighted and unweighted orientation order parameters. Returns the weighted and unweighted order parameters, and the weighted and unweighted orientations of the nanowires.

compute_odf

compute_odf(S, n=10000, display=False)

> **S**: a list of S parameters calculated by compute_order
>
> **n**: text
>
> **display**: text

This function creates the orientational order distribution function from the orientation order parameters generated by compute_order. Returns the polar coordinate (theta) and the associated value obtained from the odf.

GUI Functions

plot_to_image

plot_to_image(image)

> **image**: an array of values meant to represent an image that can be plotted by matplotlib.

This function converts an array of values, image, into a matplotlib plot and then converts it into a b64 encoded image. Returns a b64 encoded image.

graph_to_image

graph_to_image(orientation, nbins, theta, p, p0)

> **orientation**: an array of the orientations of the nanowires.
>
> **nbins**: the number of bins to add to the histogram.
>
> **theta**: an array of theta values.
>
> **p**: the result of the ODF given a particular theta value.
>
> **p0**: the initial value of the ODF.

This function, given data on the nanowires, plots a p(theta) vs. theta graph and converts it into a b64 encoded image. Returns a b64 encoded image.