Droplet Contact Angle Tool

Written by Umar Hussain

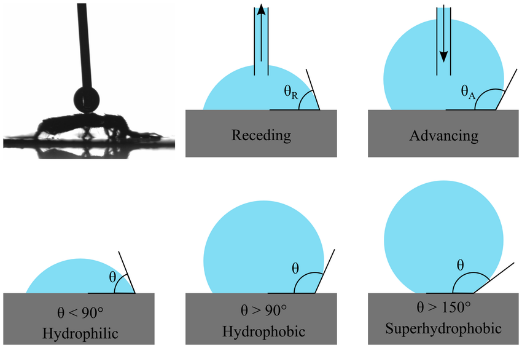
This report aims to highlight the features and flow of the Droplet Contact Angle Python Script.

**Motivation: What is the Importance of Contact Angles?**

“Dueto the importance of monitoring outdoor non-ceramic insulators aging condition, many utilities around the globe have developed several methods to evaluate insulators’ surface conditions. Most of these techniques depend on the observation of leakage current (LC) and partial discharge activities (PD) like infrared (IR) cameras, image intensifiers and current transformers (CT). Other techniques, such as contact angle measurements and hydrophobicity evaluation techniques are used to monitor the quality and the level of ageing in the polymer material.” [1]

The Contact Angle of a water droplet can be found at then interior of the three-point-junction of surrounding (room) air, the surface the droplet is on, and the water (or liquid) of the droplet.

“Deterioration of the polymeric material is always accompanied with loss of its hydrophobicity. Hydrophobicity of the material is its ability to repel and resist water flow on its surface. When the surface of the insulator is hydrophobic, water droplets will form independent small droplets with contact angles >90°. When the surface of the insulator experiences ageing, this contact angle starts to decrease until the surface is completely hydrophilic and the water droplets transfer to water films.” [1]

[2]

**What does it do?**

This Python script can input an image file (.jpg and .png were tested) containing a closeup side view of a droplet. Using the assumption that the side profile of the droplet can be fitted to **a rotated ellipse**, we try to determine the inner **contact angles,** where the horizontal surface is our reference to find the intersection/contact points.

**Implementation - Summary:**

1. Foreground extraction using GrabCut.
2. Image converted to grayscale and blurred using a 4x4 kernel.
3. Adaptive Thresholding applied.
4. Canny Edge Detection applied, and detected pixel contours are extracted.
5. Contours are fit to ellipses and the largest ellipse is chosen.
6. Angles Calculated between ellipse tangents and horizontal.

**Implementation – Detailed:**

**Step 1 - GrabCut:**

Motivation: Previous implementations that did not utilise image segmentation prior to edge detection result in very unclean edges, which were not suitable for fitting the data to an ellipse or polynomial. Using GrabCut isolates the outer shape of the droplet, allowing us to study its side profile directly.

A picture containing graphical user interface

Description automatically generated

Diagram

Description automatically generated

* User inputs the bounding box. Everything outside this rectangle will be taken as sure background (That is the reason it is mentioned before that your rectangle should enclose the droplet). Similarly, any user input specifying foreground and background are considered as hard-labelling which means they won't change in the process.
* Computer does an initial labelling depending on the data we gave. It labels the foreground and background pixels (or it hard-labels)
* Now a Gaussian Mixture Model (GMM) is used to model the foreground and background.
* Depending on the data we gave, GMM learns and create new pixel distribution. That is, the unknown pixels are labelled either probable foreground or probable background depending on its relationship with the other hard-labelled pixels in terms of color statistics (It is just like clustering).
* A graph is built from this pixel distribution. Nodes in the graphs are pixels. Additional two nodes are added, **Source node** and **Sink node**. Every foreground pixel is connected to Source node and every background pixel is connected to Sink node.
* The weights of edges connecting pixels to source node/end node are defined by the probability of a pixel being foreground/background. The weights between the pixels are defined by the edge information or pixel similarity. If there is a large difference in pixel color, the edge between them will get a low weight.
* Then a mincut algorithm is used to segment the graph. It cuts the graph into two separating source node and sink node with minimum cost function. The cost function is the sum of all weights of the edges that are cut. After the cut, all the pixels connected to Source node become foreground and those connected to Sink node become background.
* The process is continued until the classification converges.

Link to Research Paper: <https://doi.org/10.1145/1186562.1015720>

**Step 2 – Grayscale and Blur:**

GrabCut depends on color distributions, which is why we used the input image as is (in RGB) Grayscale conversion is applied only after segmentation because Canny edge detection used in Step 3 requires a single channel image. To further reduce edge noise and lower the effect of incorrectly segmented parts of the droplet near its borders, the grayscale image is then smoothed with a 4-pixel by 4-pixel kernel passed over the image. As a future improvement the size of this kernel should be proportional to image size. A 3x3 normalized box filter (kernel) would look like the following:

A picture containing text, antenna

Description automatically generated

OpenCV functions:

cv.cvtColor(image,cv.COLOR\_BGR2GRAY)

cv.blur(image, (4, 4), 0)

**Step 3 – Adaptive Thresholding:**

So far, we have an image that is a gray smoothed picture of an isolated droplet, but it is still not ready for detecting edge contours, we must convert it to a binary image.

Here, the matter is straight forward. If a pixel value is greater than a threshold value, it is assigned one value (may be white), else it is assigned another value (may be black).

However, using a global threshold value may not be effective in different lighting conditions in different areas. Here, the algorithm determines the threshold for a pixel based on a small region around it. So, we get different thresholds for different regions of the same image which gives better results for images with varying illumination (e.g., shadows).

cv.adaptiveThreshold(image, 255, cv.ADAPTIVE\_THRESH\_MEAN\_C, cv.THRESH\_BINARY, 11, -2)

**Step 4 – Canny Edge Detection:**

Canny Edge Detection is a popular edge detection algorithm (due to the resistance to noise) developed by John F. Canny in 1986 as a multi-stage algorithm. The returned contours (edges) from this stage are what we will be fitting an ellipse to. ‘Edges’ can be thought of as continuous strings of pixels, which tend to occur at the junction of contrasting gradients within the image.

Stages:

1. Finding Intensity Gradient of the Image
2. Non-maximum Suppression
3. Hysteresis Thresholding

Detailed explanations of each step can be read here:

<https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_canny/py_canny.html>

**Step 5 and 6 – Choosing the Ellipse of Interest and Calculating Contact Angles:**

(A formal paper by Mark C. Hendricks, Ph.D. with derivations of the used formulas can be found here:

<http://quickcalcbasic.com/ellipse%20line%20intersection.pdf> )

In the previous step we had fit ellipses to all detected contours where many of them are not on the water droplet profile. To find the largest ellipse we look for max((major\*minor-axis)). Since we have isolated the droplet as a foreground image and all contours detected are within, we can assume that the correct ellipse will most likely be the largest one. This is not the most accurate or efficient implementation in my opinion and in future may be done in a smarter way.

The bottom of the bounding box passes through the contact points, so we will use this horizontal line as reference to pinpoint where the fitted ellipse intersects it. We then find the tangents to the ellipse at the aforementioned contact points. The angles are then calculated for both the left and right sides of the droplet/ellipse as detailed in the paper above.

**User Guide:**

To run script from terminal, run script and pass the path to the image file.

Two windows are displayed: ‘input’ and ‘output’

* Input: This is the window the user should interact with/click on
* Output: This window allows you to view the foreground object as you mark it so that it can be improved if needed.

**Imports:**

numpy, cv2, sys, math

**Drawing Bounding Box:**

Once the script has been run and the windows are displayed, you must right click to begin drawing the rectangle, while holding the right click, drag the mouse pointer to a satisfying position until the box surrounds the drop leaving a mouse-pointer sized margin of space between the box and the edge of the droplet (left, right, and top)

* You must make sure the bottom of the bounding box is along the bottom of the droplet as this will be used as the horizon level, you may restart drawing if needed.

**Segment and Markup:**

Press the **‘n’** key on your keyboard to segment the droplet within the bounding box, if satisfied with the droplet you see in the ‘output’ window you may proceed to step 5 otherwise continue.

If the ‘output’ window shows an unsatisfying segmentation of droplet as a foreground object you need to add or remove foreground or background respectively by marking it off:

* Add ‘Sure Foreground’ marks by pressing the **‘1’** key and drawing on the places you want to mark using the left mouse button (and dragging) on the ‘input’ window – this will draw white dots.
* Similarly, you may remove ‘Sure Background’ areas by pressing **‘0’** and drawing on them, this will draw black dots.
* Press the ‘n’ key one more time so that the image can be re-segmented with the markings you just made. If you are satisfied with the output, continue – otherwise repeat step 5.

**Save and View Results:**

* Press the ‘s’ key to save your output images and view the results (they will also be printed on the terminal output).

**References**

**[1]:** I. Jarrar, K. Assaleh and A. H. El-hag, "Using a pattern recognition-based technique to assess the hydrophobicity class of silicone rubber materials," in IEEE Transactions on Dielectrics and Electrical Insulation, vol. 21, no. 6, pp. 2611-2618, December 2014, doi: 10.1109/TDEI.2014.004523.

**[2]:**  Gundersen H, Leinaas HP, Thaulow C (2014) Surface Structure and Wetting Characteristics of Collembola Cuticles. PLoS ONE 9(2): e86783. https://doi.org/10.1371/journal.pone.0086783