Automated Measurement of Contact Angles for Sessile Droplets using MATLAB Image analysis Library:

Larkin Liu,
Thermodynamics and Kinetics Laboratory, Department of Mechanical and Industrial Engineering. University of Toronto, Canada M5S 3G8

Introduction:

In the paper, "*Sessile-Water-Droplet Contact Angle Dependence on Adsorption at the Solid-Liquid Interface*", by H. Ghasemi & C.A. Ward, the experiment was conducted to investigate experimentally the possible role of line tension in determining the contact angle of sessile water droplets on a polished Cu substrate. Revisiting this phenomena, it would be valuable to confirm this relationship with varying pressures and substances. Previously, the measurements obtained in this experiment involved the use of depth measurements using a manually adjusted dial. This report outlines a new procedure we can use to measure the contact angle of the sessile droplet, and plot the sessile droplet's boundary.

Using source code embedded in the MATLAB image processing library, we can re-measure the contact angles under more precise methods to obtain a more definite and precise result. This report will outline a new methodology which can be used to evaluate the contact angle between the liquid/solid interface of a small sessile droplet less than ~20 mm in diameter. This software is also capable of obtaining a visual trace of the sessile droplet, in which we can fit a mathematical function to model the varying heights in the sessile droplet.

The potential benefit of this software allows the user to obtain a more precise and systematic way to obtain measurements from images of small objects. This process also involves the use of image processing algorithms which employs the use of automated boundary detection, allowing the computer to automatically complete the measurements and data-plot without user-input related to the mapping of the image. In short, the computer is able to automatically identify and measure the contact angle of the sessile droplet, and plot the tracing of the three-phase line by itself.

Future applications of this software can be used for the automation of the vacuum chamber in which the sessile droplet is contained in. Using this software, the computer will be able to automatically identify the equilibrium status of the sessile droplet. If interfaced with the mechanical apparatus of the experiment, the computer can adjust the parameters of the vacuum chamber in accordance to the equilibrium state.

Procedure and Algorithm:

As mentioned earlier, the method used to evaluate our data involves the use of the MATLAB image processing library. The source code can be found in this folder of the MATLAB installation:

```
cd 'C:\Program Files\MATLAB\R2010a\toolbox\images\images'
```

The default directory should be changed to this directory, or all of the files contained within this directory should be directly copied over to the directory we are working with.

Further technical documentation of the image processing toolbox can be found on this website: http://www.mathworks.com/help/toolbox/images/ref/f3-23960.html

New functions and scripts were added to the MATLAB image processing library to accomplish this task:

sessile_angle.m –serves as the main function of the software, calls other functions to determine the contact, and identifies all files within a folder. Also crops images if necessary to reduce file size.

find_angle.m – The function used to find the contact angle and trace the boundary of the three phase line.

poly_line.m –Used to determine the equation of a polynomial fit through a set of points, and computes another set of point based on the polynomial.

derive_angle.m – Derives the angle between the intersection points of two given mathematical functions.

*Refer to source code for technical documentation

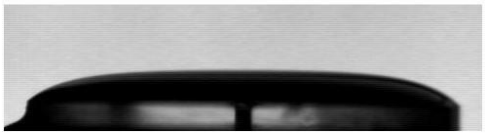
Before beginning to trace the three-phase line, it is important to convert the grayscale version of the image to an absolute black and white version of the image such as the one below:

The image represents an absolute black and white conversion from a grayscale image of a sessile droplet. Notice that the bottom portion of the image in predominantly black (all black in this case), and that the three-phase line of the sessile droplet is visible without interruptions. The base of the sessile droplet is recognizable by the sharp bend in the smooth curve representing the line of contact between the liquid and solid.

The grayscale image can be converted into an absolute black and white image using the “im2bw” function, and adjusting the threshold value. See technical documentation on the MATLAB website.

```
threshold = graythresh(I) + thresh_inc;
```

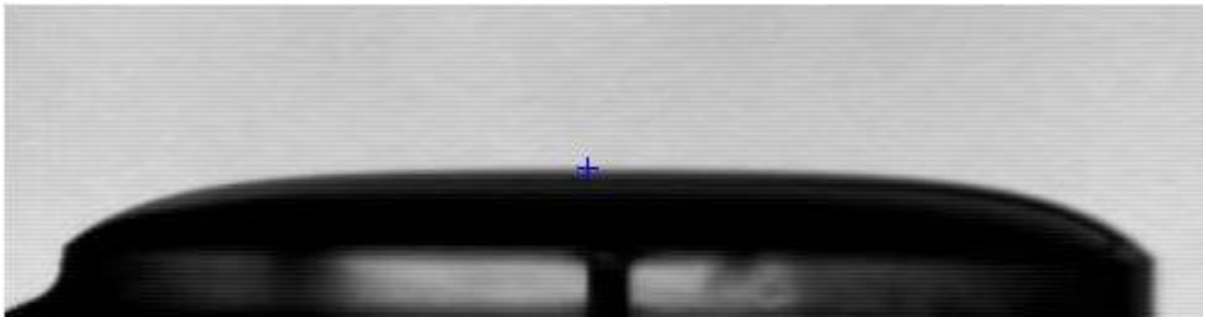
```
BW = im2bw(I, threshold);
```



Referencing 20-65 of the MATLAB file “find_angle.m”:

```
% Trace a boundary for the parabolic function, refer to MATLAB online help
% resource library for the function "bwtraceboundary" for detailed
description.

      direc_str1 = 'clockwise';
      boundary_apex = bwtraceboundary(BW, [ini_Y, ini_X], 'W', 8,
      para_length, direc_str1);

    % Determine the value of the apex of the boundary trace, find the
    % maximum Y value(or row vector)

      [Yval, Yind] = min(boundary_apex(:,1));
       Xval = boundary_apex(Yind + para_length)

       apex = [Yval, Xval];

   %Because there may be multiple values of the same maximum Y, we identify
   %all these values and center the apex.

        dot_count = 0;
        for i = 1:para_length
            if boundary_apex(i, 1) == Yval
                dot_count = dot_count + 1;
            end
        end

        for i = 1:(dot_count/2+5) %incrememnting apex 5 to the right
            Yind = Yind + 1;
        end

        Xval = boundary_apex(Yind + para_length);
        Yval = boundary_apex(Yind);

        apex = [Yval, Xval]
```

This step is to identify the apex of the of the droplet, and determine the point where the tracing is to begin. This can be accomplished by beginning a trace from the bottom left corner of the image for a considerable length approximately the width of the droplet, and then finding the maximum row vector value of that trace.



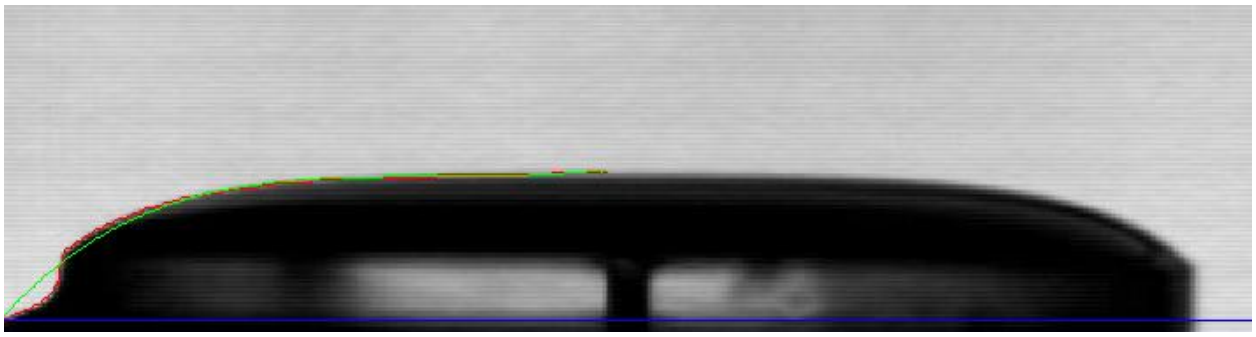
From lines 58-111:

```
%Trace boundaries down from the apex:
  cd C:\Users\larkz\Documents\sessile_detection;
  cut_off = 5;
  boundary_left = bwtraceboundary(BW, apex, 'N', 8, para_length/2-
  cut_off, 'clockwise');
  boundary_right =  bwtraceboundary(BW, apex, 'N', 8, para_length/2-
  cut_off, 'counterclockwise');

%Produce a polynomial fit onto the curve, using the function
"poly_line"
[f_lef, f_rig, f_line, f_lef_val, f_rig_val, f_line_val, X_lef, X_rig,
X_line] = poly_line(boundary_left, boundary_right,
max(boundary_left(:,1)));
```

Once the apex is identified, we can begin our tracing from the apex. In this code, the default length, “para_length”, is automatically set to the width of the image plus an additional margin to ensure there is enough room for correction and accommodate for higher resolution imaging.

We begin tracing from the apex outwards towards the left from the apex, we trace the three-phase line in the southeast direction. See the technical documentation for MATLAB’s “bwtraceboundary” function for details.

Because there is no way to identify the final point on the three-phase line boundary trace, we can expect that on the first trace, assuming there we have given sufficient room for trace overflow, that there will be a trace over flow.



As we can see, there is a trace overflow as marked by the red portion outline. This means that we set the trace length for too long, and must decrease it. Marked in red is the sessile boundary trace. In green a 4th order polynomial is fitted across the boundary using the function, "poly_line.m". Marked in blue is the base line, with reference to the final point of the boundary trace.

By over tracing the boundary, we are forced to fit the polynomial across points which do not necessarily fall on the sessile droplet. There by creating a significant portion of error.

In this version, we use the "mean squared error" to determine our threshold error value. Where the mean error is governed by the equation:

$$\mathrm{MPE} = \frac{1}{n}\sum_{t=1}^{n}\frac{f_t - a_t}{a_t}$$

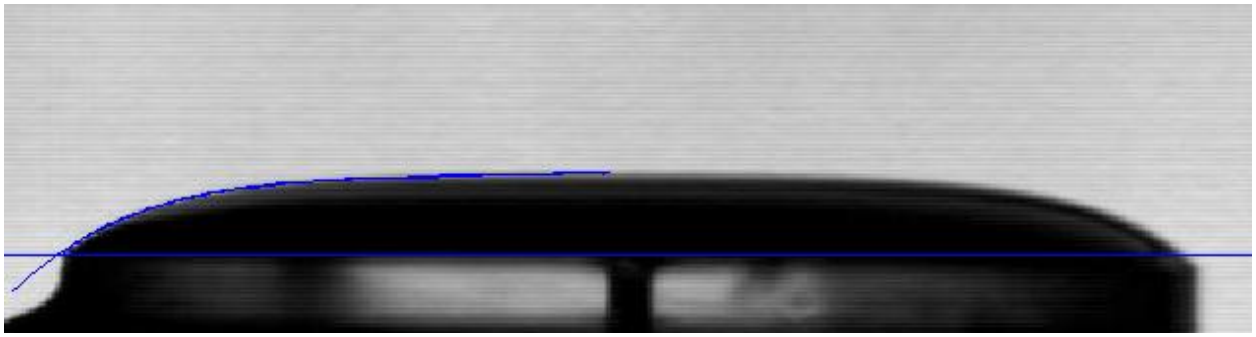
Where,
n = the number of boundary points
ft = the boundary vaule
at = the value found on the quartic fit

```
% If the error is greater than 0.185, recompute the boundary
% increase the cut off (shorten the boundary) and recompute the
% boundary trace.
while(err_lef > 0.185)
      boundary_left = bwtraceboundary(BW, apex, 'N', 8,
      para_length/2-cut_off, 'clockwise');

      [f_lef, f_rig, f_line, f_lef_val, f_rig_val, f_line_val,
      X_lef, X_rig, X_line] = poly_line(boundary_left,
      boundary_right);

      err_lef = 0;

      trace_points = boundary_left(:,1);
      siz_lef = size(trace_points);

%Adds on each error value for every difference value
    for i = 1:min( [numel(f_lef_val), numel(trace_points) ] )
```

```
            err_lef = err_lef + ( abs(f_lef_val(i) -
            trace_points(i)))/trace_points(i);
        end

    %Divide the total error by number of points in boundary.
        err_lef = err_lef/siz_lef(1);
        cut_off = cut_off + 1;
    end
```

The program automatically reduces the mean error by reducing the trace length until the error is contained to a certain degree. In this case 0.155 is our mean error value, in terms of pixel distance, and is the error value will be used as a threshold value. If a mean error value greater than 0.155 is detected, then the trace is repeated until it falls below that value, incrementing the cut off value on each iteration.

A horizontal baseline is then plotted across the image, referencing from the endpoint of the three-phase line trace, the row vectors of the straight line remain constant creating a perfectly horizontal line fit, which will be used to measure the contact angle.



*The same method is used to evaluate the sessile droplet on the right side.

Using the function "derive_angle.m", we can find the point of intersections between the curve and the line, and limit those intersections to the two within the range of this image. After we take the derivative of the quartic function, using the MATLAB built-in "polyder.m", we evaluate the tangent at the point of intersections, and return the respective contact angle.

The angle between two straight lines can be calculated using the equation (implemented by the function derive_angle.m):

$$\tan\theta = \frac{m_1 - m_2}{1 + m_1 m_2}$$

Where, m1 & m2 = the respective slopes of each line.

The function, find_angle.m returns the left contact angle, the right contact angle, the height of the sessile droplet, and the boundary of points combining the trace of the left and right trace of the three phase line (in mm).

Example:

```
Results:
    filename: 'shot4000001.jpg'
     ang_lef: 34.5464
     ang_rig: 35.5003
      height: 1.3755
```

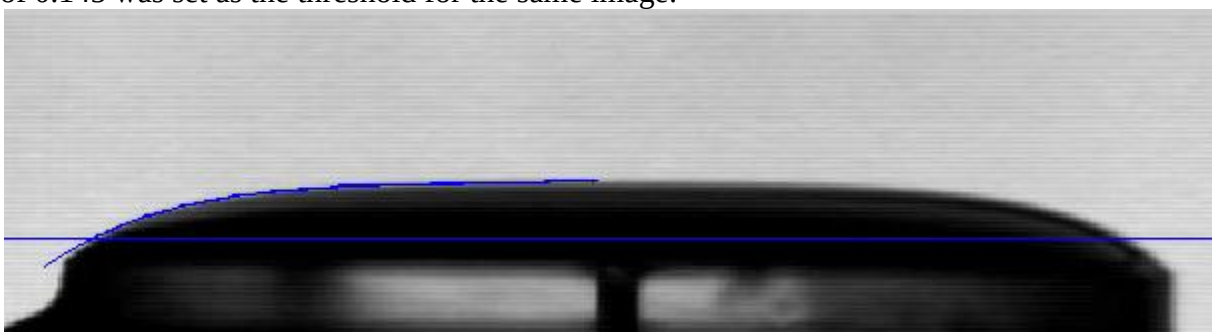
Future Considerations:

There are a few things to take into consideration if one plans to take this software to the next level.

1) *Error finding*
   The method of evaluating error in this procedure was the mean percentage error value. This error may not always produce the desired result as it is the simplest and most efficient way in deriving the error. I would suggest a more comprehensive error checking method such as the "mean squared method" etc.

2) *Providing a Graphic User Interface:*
   A GUI would make this software more user-friendly for those who are not accustomed to the MATLAB scripting interface.

3) *Improving the automation algorithm:*
   Currently, the algorithm for detecting the boundary between the baseline and the sessile droplet depends on the error values predicted. This may sometimes result in the computed baseline being set higher than where the actual base line is. For example, an error value of 0.145 was set as the threshold for the same image:



   At this moment the threshold error value can only be experimentally determined by the user this could affect the software as a whole. A better automated form of edge detection which is independent of the threshold error value could eliminate this problem.

4) *Integration of automated contact angle detection with Simulink:*
   The MATLAB built in tool Simulink contains features which can be used to interface with mechanical parts outside of the computer, such as the vacuum chamber itself. If the

current software is reliable and consistent enough to determine conditions of equilibrium, then it could be used as a control bridge between the automation of the vacuum chamber and the computer.