ChatGPT

Development Plan for a Python-Based Droplet Shape Analysis Tool

Introduction

The goal is to develop a Python-based software for analyzing droplet shapes from side-profile images in order to determine key physical properties. The application will handle **pendant drops** (hanging drops) to measure surface tension and **sessile drops** (sitting on a surface) to measure contact angle, along with general droplet geometry (volume, height, diameter). The software will provide a **graphical user interface (GUI)** so that users can easily load images (single or in batch), visualize droplet outlines, overlay predicted theoretical shapes, and obtain quantitative results. It will be designed to work with arbitrary droplet images (e.g. taken with a camera against a backlight) without requiring a specialized imaging system or pre-calibration – though manual scale calibration will be supported for physical units.

Key Features and Requirements:

- **Python-Based GUI:** Implement the application primarily in Python, using suitable libraries for GUI (e.g. PyQt5 or Tkinter) and image processing. External tools or languages will only be considered if performance demands require it (e.g. a C/C++ extension for heavy computations), but the core should remain in Python for flexibility.
- **Image Processing Capabilities:** Read common image formats and perform preprocessing to isolate the droplet. Include robust image segmentation to detect the droplet's edge (outline) against the background. This should work for a variety of images (not tied to one camera or background), possibly by using adaptive thresholding or edge detection.
- **Single & Batch Processing:** Allow users to analyze one image at a time with interactive feedback, and also provide a batch mode to process multiple images automatically (with results saved or summarized). This is useful for experiments where many droplet images need analysis.
- **No Fixed Scale Assumption:** The software will not assume a known pixel-to-length scale; it will compute shape and properties in pixel units by default. However, it will allow manual calibration – e.g. the user can input a known distance or droplet volume to scale the measurements into physical units (like mm or µL).
- **Multiple Theoretical Models:** Provide several mathematical models to fit or predict the droplet shape: (1) the physical Young–Laplace equation (and its numerical solution via the Bashforth-Adams approach for axisymmetric drops), (2) simpler geometric models like circular or elliptical fits (spherical cap approximation), and (3) an axisymmetric drop shape analysis (ADSA) method for high-precision fitting [1] [2]. These models will be used to either directly compute properties or to overlay an ideal shape for comparison.
- **Overlay of Predicted Shape:** After analyzing an image, the software should overlay the computed droplet profile on the image for visual confirmation. For example, a fitted curve (from one of the models) will be drawn on top of the droplet's silhouette to show how well the model matches the experiment.
- **Computed Outputs:** The application will output key droplet features: surface tension (for pendant drops), contact angle (for sessile drops), droplet volume, height, and diameter. These may be displayed in the GUI and also saved to a file. Additional details like droplet curvature or surface area could be included if needed, but the focus is on the listed properties.

Technology Stack and Libraries

To meet the above requirements, we recommend the following Python libraries and tools:

- **GUI Framework: PyQt5** (or PySide) is suggested for building the GUI. PyQt5 provides a rich set of widgets and is well-suited for displaying images and custom graphics. It supports events (for interactive image analysis) and has OpenGL accelerated views for efficient drawing. An alternative is **Tkinter** (built-in to Python), but PyQt5 offers more modern UI elements and better image rendering performance. PyQt5 also allows embedding Matplotlib plots or using QGraphicsScene for overlays. (Kivy or wxPython could be other options, but PyQt5 is more established for scientific apps.) A comparison is summarized in **Table 1** below.

- **Image I/O and Processing: OpenCV (cv2)** and **scikit-image** will be used for image handling and processing. OpenCV is highly optimized for tasks like reading/writing images, color conversion, filtering, thresholding, and finding contours. Scikit-image (skimage) provides a high-level interface to many image algorithms and works nicely with NumPy arrays (e.g. for filters, morphology, segmentation, and even an active contour (snake) implementation). Both libraries can be used in tandem: for instance, OpenCV for basic operations and skimage for more advanced segmentation if needed.

- **Numerical Computation: NumPy** and **SciPy** will be used for numerical analysis. SciPy provides tools for curve fitting (`scipy.optimize.curve_fit`), optimization algorithms, integration routines, and even ODE solvers (`scipy.integrate.solve_ivp`) which will be crucial for solving the Young–Laplace differential equation. The SciPy library also has functions for image processing (in `scipy.ndimage`) and measure (like center of mass or edge filters) that can complement OpenCV/skimage.

- **Curve Fitting & Analysis:** Besides SciPy's optimization, we may use specialized routines for geometric fitting: e.g. fitting a circle or ellipse to points (OpenCV has `cv2.fitEllipse` for contour points which yields an ellipse parameters). For circle fitting, we could also implement a least-squares solution. For overlaying shapes, **Matplotlib** can be used if we choose to display the image in a Matplotlib Axes (for example, embedding a Matplotlib canvas in the PyQt GUI), which would make it easy to plot curves on the image. Alternatively, PyQt's own drawing functions (QPainter) could draw lines on a QPixmap.

- **Data Handling:** For batch processing outputs, we can use **pandas** to organize results (especially if saving to CSV or Excel). Each image's results can be one row with columns for each property.

- **Optional Machine Learning:** If advanced segmentation is needed, we might integrate **TensorFlow/Keras or PyTorch** for a trained image segmentation model (e.g. a U-Net to segment droplet vs background). This would be optional and only if the classical methods struggle on the user's dataset. Similarly, a trained neural network could be used to directly predict surface tension from the image (as recent research suggests [3]), but this is beyond the initial scope.

Table 1: Comparison of GUI Framework Options

Framework	Advantages	Drawbacks
PyQt5/ PySide	Rich widget set; good image display and drawing capabilities; can embed plots; cross-platform.	Requires installing Qt bindings; slightly steeper learning curve.
Tkinter	Built into Python; simple API; no extra installs.	Limited widgets; styling is dated; manual image handling can be slower for large images.
Kivy	Modern, touch-friendly, good for simple UIs.	Less conventional for desktop apps; not as mature for complex widgets; uses its own rendering pipeline.

Given the need for an interactive image display with overlays and potentially complex dialogs (for batch processing settings, calibration input, etc.), **PyQt5** is the recommended choice.

GUI Design and User Interaction

The GUI will be designed for clarity and ease of use, with logically organized sections and controls:

- **Main Window Layout:** Likely a two-panel design. The left side will contain the image display area where the droplet image is shown. The right side (or bottom) will have controls and outputs. Controls include: file open (single image), folder open (batch mode), calibration input, model selection (dropdown or checkboxes for which analysis method to use), and a “Process” button. Outputs include text fields or labels showing the calculated surface tension, contact angle, volume, etc., for the current image. There may also be an output console or log area for messages (e.g. errors or process info).

- **Image Display:** The droplet image will be shown (fit to window or 1:1 scale toggle). The detected droplet edge (contour) can be drawn on this image (e.g. as a highlighted edge in a distinct color). The predicted model shape (from Young–Laplace solution, circle fit, etc.) will also be drawn – likely as a smooth curve overlay. For clarity, each type of overlay can use a different color or style (e.g. blue for detected edge, red dashed line for theoretical fit). The user should be able to zoom/pan if high-resolution images are used, which PyQt’s graphics view or Matplotlib can handle.

- **Controls for Segmentation:** Because lighting and droplet contrast can vary, the GUI might offer basic controls for the segmentation step (e.g. a slider for threshold value, or a dropdown for choosing an algorithm: “Automatic (Otsu)”, “Adaptive”, “Canny Edge” etc.). In many cases, Otsu’s automatic threshold will suffice to binarize the image into drop vs background [4] [5]. But the user could override if needed. For example, if the automatic method fails due to unusual background, the user can manually set a threshold or use a region-of-interest tool to isolate the drop.

- **Batch Mode Operation:** In batch mode, the GUI will allow selecting a directory or multiple files. Upon clicking “Process All”, it will iterate through images, apply the analysis, and collect results. The results can be displayed in a table within the GUI (each row per image) and optionally exported to CSV. In batch mode, the image display might either update sequentially to show progress, or remain blank to speed up processing (with just a progress bar). The user can review individual images afterward by clicking on a result row to display that image and its overlays.

- **Calibration Input:** A small dialog or section will allow inputting a scale. Options include: entering a known pixel-to-length ratio (if the user knows, e.g., that 50 pixels = 1 mm), or using a reference object in the image (e.g. the needle tip diameter or a scale marker) – the user could drag a line on the image and input what length it represents. This calibration factor will be used to convert pixel measurements to real units (mm, µL, etc.). If no calibration is provided, results for surface tension will be given in dimensionless form or assuming a default DPI, and volumes in pixel^3 (with a note that they are in arbitrary units). Typically, surface tension calculation actually *requires* physical units because the Young–Laplace equation includes gravity and density (which are in SI units) [2] . So for surface tension, the user must input the droplet liquid density (and surrounding density, e.g. air) and either a scale or the capillary dimension, otherwise the calculation can only give a relative value.

- **Model Selection and Settings:** The GUI will present options for which analysis models to apply. For example, a checklist: "Circle Fit (quick contact angle)", "Ellipse Fit", "Young-Laplace Fit (precise)", etc. The user can select one or multiple. Advanced settings for the models can be tucked into an "Advanced" panel (for instance, the Young–Laplace fit might allow specifying if the drop is pendant or sessile, the density difference, and an initial guess for surface tension to speed up convergence). Most defaults will be sensible (e.g. assume pendant drop if surface tension box is checked, etc.).

- **Output Display:** Once analysis is run on an image, the numerical results will be shown with appropriate units (if calibrated). For a pendant drop image, it might show "Surface Tension: 72.8 mN/m", "Volume: 12.5 µL", "Height: 8.4 mm", "Max Diameter: 5.2 mm". For a sessile drop image, it might show "Contact Angle: 85.3°", "Volume: 5.0 µL", "Height: 2.1 mm", "Base Diameter: 4.5 mm", etc. Each result will be derived from the chosen model – if multiple models are selected, potentially multiple results could be shown (e.g. contact angle by circle fit vs by ellipse fit vs by Young–Laplace, for comparison). The GUI should make it clear which value comes from which method.

- **Figure: Example Interface Outputs:**
  Example of a pendant drop analysis: the software detects the drop contour (blue) and fits a theoretical shape (red) by solving the Young–Laplace equation. The fitted curve closely matches the drop silhouette, and the results (surface tension, volume, etc.) are shown in a table on the right. [6]

Image Preprocessing and Droplet Edge Detection

Accurate edge detection is the first crucial step. The application will use a combination of techniques to robustly segment the droplet from the background:

- **Grayscale Conversion:** Input images (which could be color) will be converted to grayscale for analysis, since shape detection relies on intensity differences. If the droplet is backlit (common in tensiometer setups), it will appear dark on a bright background or vice versa. The software may offer an "invert image" option in case the droplet is bright on dark background, so that the droplet is consistently treated as the darker region for thresholding (or the algorithm can simply handle both by looking at histogram bimodality).

- **Noise Reduction:** A slight Gaussian blur (using OpenCV or skimage) can be applied to reduce noise that might interfere with thresholding or edge finding [5] . This is especially useful if the image has sparkles or pixel-level noise. However, the blur will be kept minimal to avoid distorting the drop shape edges.

- **Thresholding:** For initial segmentation, **Otsu's method** is a good choice for automatic threshold determination. Otsu's algorithm finds an optimal threshold that separates the image histogram into two classes (foreground and background) by maximizing inter-class variance [4] [7]. The droplet being a distinct object should create a bimodal intensity distribution (one peak for background, one for droplet). Using OpenCV's `cv.threshold` with `cv.THRESH_OTSU` flag yields the Otsu threshold automatically [8] [9]. The result is a binary image where ideally the droplet is white (1) and background black (0) or vice versa. If lighting is uneven, **adaptive thresholding** (mean or Gaussian) could be tried, which computes local thresholds [10] [11]. Adaptive threshold might help if, for example, the top of the image is brighter than the bottom.

- **Morphological Cleanup:** After thresholding, small speckles or holes can be present. We will use morphological operations: **erosion/dilation** or closing to fill small holes in the droplet region, and **opening** to remove small isolated noise blobs. For instance, if the droplet has a specular highlight that appears as a white spot inside a dark drop, the threshold might classify it as background hole; a morphological closing will fill that in. Similarly, dust in the background could be erroneously picked up; a size filter can remove regions below a certain area. Using `cv2.findContours`, we can retrieve all connected contours in the binary image and then choose the one with largest area (assuming the droplet is the largest dark object in view). We will make sure to ignore contours touching the image border (if background is not properly cropped, it could produce a large contour).

- **Edge Detection (Alternative):** If thresholding is tricky (for instance, if the droplet and background have similar brightness but the edge is clearly visible), we can use **edge detection** like the Canny algorithm. Canny edge detection finds strong gradients in the image. By tuning its thresholds, it can highlight the droplet outline. Then, we could use a dynamic programming or contour following algorithm to trace a continuous edge. However, pure edge detection might produce a double-edged outline (inner and outer edge). In practice, thresholding to a filled blob and then extracting the contour of that blob is simpler and more robust for a well-contrasted drop image.

- **Active Contours (Snakes):** To improve the precision of the detected edge, especially in suboptimal images, an **active contour model** can be applied. The idea is to initialize a spline (curve) around the droplet and let it shrink onto the actual edge by minimizing an energy functional [12] [13]. We can leverage `skimage.segmentation.active_contour` for this, using the binary image or the gradient image as input. For example, after Otsu threshold, we have a rough binary mask; we can find its contour and use that as an initial guess for the snake, which will then adjust to better lock onto true edges (especially helpful if the droplet edge is smooth and we want sub-pixel accuracy). The active contour's parameters (alpha, beta, etc., which control tension and smoothness [14]) can be tuned for droplet shapes (which are generally smooth and convex). Another approach is the **"DropSnake"** method mentioned in literature [15], which uses B-spline snakes to accurately determine contact points and angles [16] [17]. Implementing a full custom snake might be complex, but using the scikit-image snake provides a similar benefit in refining the outline.

- **Reflection Handling (Sessile drop):** Sessile drop images often show the droplet and its mirror image below the surface (if a reflective surface or if backlit on a horizontal surface). This reflection can confuse edge detection. A strategy to handle this is to detect the **baseline** (surface line) first, then ignore contours below it. The baseline can be found by looking for a straight line segment beneath the droplet: often the substrate is visible as a horizontal line or simply deduced as the lowest y-coordinate of the droplet contour where it flattens out. The EPFL LBADSA approach cleverly fits both drop and its reflection simultaneously to find the contact point [18], but in our implementation, a simpler method is: find the lowest point of the detected droplet contour and assume that is the bottom of the drop; a line through that point horizontally that matches a line of symmetry in the image intensity could be the baseline. We could also ask the user to click the baseline if the automation fails. Once baseline is known, we treat anything below it as reflection and exclude it.

In summary, the segmentation module will combine global thresholding (with optional adaptive mode) and contour extraction as the primary method, augmented by edge detection or active contours for fine-tuning. By the time this step is done, we should have a set of coordinates (x,y) representing the droplet boundary (usually one side from apex to bottom for pendant drop, or both sides from one contact point to the apex to the other contact point for sessile drop, assuming symmetry about a vertical axis).

Calibration and Scaling

Since the software is to work with arbitrary images without known scale, all measurements default to pixel units. For relative comparisons or dimensionless analysis, this is fine. However, to report physically meaningful values (surface tension in N/m, volume in μL, etc.), we need a scale. We provide the user with the following calibration options:

- **Manual Scale Entry:** The user can enter a value like “X pixels = 1 mm”. If the imaging system pixel size is known (e.g. from a calibration target or manufacturer specs), they can input it directly. This single factor will convert all linear dimensions (thus areas $\propto$ factor^2, volumes $\propto$ factor^3).

- **On-Image Calibration:** The user can draw a line segment on the image corresponding to a known length (for example, if a millimeter ruler or a known-diameter needle tip appears in the image). The software will read the pixel length of that segment and ask the user for the real length. From this, the pixel-to-mm ratio is computed. This approach is user-friendly if a reference object is present.

- **Droplet Volume Calibration:** In some cases, the droplet volume might be known (e.g. you dispensed 10 μL). The software could adjust the scale such that the volume measured from the image (in pixel^3) matches the known volume in real units. This is a bit indirect and assumes the droplet was fully in view and measured correctly in pixels, but it can serve if no other reference is available.

- **Using Theoretical Capillary Length:** For pendant drops, if the fluid density is known, one might calibrate by using the physics of the shape. The Young–Laplace equation introduces a natural length scale called the *capillary length* `(γ/(Δρ g))^0.5`. If we guess or have an idea of surface tension, we could estimate the capillary length in physical units (e.g. ~2.7 mm for water/air). If we see the drop shape, we might fit it in dimensionless form to determine how many pixels correspond to one capillary length. However, this is an advanced approach and likely not needed if manual calibration is available.

For our implementation, we’ll primarily include the first two (direct scale input and line measurement). The calibration factor (pixels per mm) will be applied whenever computing physical properties: e.g. the droplet height in pixels multiplied by (mm/pixel) gives height in mm; surface tension computed dimensionlessly will be converted by appropriate combination of length scaling and known density (more on that in the next section). The GUI will clearly indicate if results are “(in pixels or arbitrary units)” when no calibration is set, to avoid confusion.

Mathematical Models for Droplet Shape and Analysis

A core aspect of the application is the inclusion of multiple mathematical models to analyze and predict droplet shapes. Each model has its assumptions and applicable scenarios. We will implement a selection of models, allowing the user to choose which to use for their analysis:

1. Young–Laplace Equation (Axisymmetric Drop Shape)

The Young–Laplace equation is the fundamental equation describing the shape of a fluid interface under surface tension and pressure differences. For an axisymmetric droplet (pendant or sessile), it can be formulated as a second-order ODE relating the curvature of the drop profile to the pressure difference caused by gravity and surface tension [19]. In practice, solving this equation for a droplet yields a family of shapes parameterized by a capillary length or Bond number. Key points:

- **Shape Physics:** Gravity tends to elongate the drop (making it “pendant”), while surface tension tends to make it spherical. Equilibrium is reached when pressure differences balance the curvature *γ* (surface tension) according to Young–Laplace: ΔP = γ * (curvature). For a hanging drop, ΔP varies with height due to hydrostatic pressure of the liquid column [20]. The result is a characteristic drop shape that can be captured by the Young–Laplace equation. The competition between gravity and surface tension is quantified by the capillary length *ℓ<sub>c</sub>*, where *ℓ<sub>c</sub> = sqrt(γ/(Δρ g))* [20]. Large drops (relative to ℓ<sub>c</sub>) are significantly distorted by gravity, while small drops approach spherical shape.

- **Bashforth–Adams Equation:** In practice, the Young–Laplace shape ODE is often solved in a dimensionless form known as the Bashforth–Adams equation [21]. Historically, Bashforth and Adams solved this equation numerically and produced tables for drop profiles. Today we can integrate it with a computer. We will set up the ODE as follows (in dimensionless terms):

$$\frac{d^2y}{dx^2} = \frac{1 + (dy/dx)^2}{x} - 1,$$

(This is a simplified form assuming certain scaling; the actual form we implement will handle units and density explicitly.) We will use SciPy’s ODE solver (e.g. 4th order Runge–Kutta) to integrate from the apex of the drop. The apex (top point for a pendant drop) has a curvature related to the radius of curvature R<sub>0</sub>. We treat R<sub>0</sub> as a parameter to be determined. The ODE integration will produce a shape profile (x(r), y(r)) of the drop contour [22].

- **Numerical Solution & Optimization:** Solving the shape ODE alone is not enough – we need to match it to the actual drop. This is done by optimizing parameters such as the integration constants or directly the surface tension value. For a pendant drop of known density difference Δρ, if we guess a surface tension γ, we can compute ℓ<sub>c</sub> and generate a dimensionless shape. By scaling that shape to pixel units, we can compare to the detected contour. We then adjust γ to minimize the mismatch between the theoretical contour and the experimental contour [22]. Essentially, we are fitting the Young–Laplace model to the data; the fitting parameter is γ (and possibly one more parameter for overall scale if not calibrated). The ImageJ plugin *Pendent_Drop* follows this approach by minimizing the mean square distance between the theoretical curve and the extracted drop border [22], using an optimization method

(Powell's method in their case). We will implement a similar fitting routine: for example, use `scipy.optimize.minimize` to find the γ that best aligns the curves. This yields the surface tension directly as the optimal γ [22] [2]. If the scale is unknown, γ can only be determined up to a factor (because pixel size factor couples with γ), so calibration or known scale is necessary for absolute surface tension.

- **Sessile Drop with Young–Laplace:** For sessile drops, the Young–Laplace equation can also describe the shape, but an additional boundary condition comes into play: the contact angle at the solid surface. To fit a sessile drop, one can treat the contact angle as an unknown parameter in addition to γ (or instead of γ if γ is known). ADSA (Axisymmetric Drop Shape Analysis) for sessile drops often involves fitting both γ and the contact angle by considering both the droplet profile and its reflection [18]. In our implementation, we may not initially do a full ADSA for contact angle (since contact angle can be measured more directly), but it is possible to include. If we do, we'd solve Young–Laplace ODE for a sessile drop given a contact angle and see which contact angle yields the best match to the observed profile.

- **Outputs from Y–L Model:** Once the best-fit Young–Laplace shape is found, we can calculate droplet volume by integrating the shape (rotate the profile around the vertical axis). We can also get the radius of curvature at the apex (R<sub>0</sub>) and other geometric parameters. The surface tension is obtained from the fitting parameter. In fact, there is a known formula relating surface tension to shape for a pendant drop:

$$\gamma = \Delta\rho \, g \, R_0/\beta,$$

where *R<sub>0</sub>* is the apex radius of curvature and β is a dimensionless shape factor determined by the drop profile [23]. Our approach essentially finds the correct β (through the ODE solution) and hence γ. Modern computational methods allow solving the Young–Laplace equation iteratively for this purpose [24].

The Young–Laplace model (with numeric solution) will be the **most accurate** method for surface tension of pendant drops [2], and it also provides a physically consistent way to predict the entire shape (used for overlay).

2. Simple Geometric Models (Circle/Ellipse Fits)

Not every situation demands the full physics-based solution. Often, for contact angle measurements or quick estimates, simpler models are used:

- **Circular (Spherical Cap) Fit:** If a droplet is small such that gravity's effect is negligible, the droplet's profile is essentially a circular arc (since the drop takes the shape of a spherical cap). In this case, one can fit a circle to the droplet's outline. The circle fit is done by selecting points along the droplet edge and finding the circle that best goes through them [25]. Many commercial contact angle tools use this **circle-fitting method** because of its simplicity and good accuracy for moderate contact angles [26]. The contact angle can then be calculated as the angle between the circle's tangent at the contact point and the horizontal baseline [27]. Practically, after detecting the droplet contour, we identify the contact points (where the droplet meets the surface) and a few points along the upper part of the drop, then solve for the circle passing through them (or use least-squares to fit if more points than needed). The circle's equation gives its center and radius. If the circle intersects the baseline at the contact point, the contact angle θ is obtained from the geometry: for a spherical cap, **θ = 2 * arctan(h / r)** [28] (where *h* is droplet height and *r* is half the base width). This is known as the height-width or half-angle method, suitable for smaller symmetric drops [28]. Using that formula directly is a quick approximation: $\theta = 2 \arctan(h/r)$ [28].

- **Ellipse Fit:** For larger sessile drops (or those with 90°+ contact angles), the shape can appear "flattened" by gravity. An ellipse can sometimes better fit the profile than a circle [29]. The ellipse-fitting method is similar: we take points on the drop outline and perform an ellipse fit (which yields parameters like major/minor axes). The contact angle is then found by the slope of the ellipse at the baseline intersection [30]. The advantage of the ellipse model is that it can accommodate shapes where the top is more flattened and the sides more tapered, covering a wider range of contact angles (up to ~130° according to Apex Instruments docs) [31]. The downside is that ellipse fitting is more complex (requires solving a polynomial) and might over-fit in some cases. We can leverage OpenCV's `fitEllipse` for implementation, which returns the best-fit rotated ellipse for a set of contour points.

- **Tangent (Polynomial) Fit:** Another method (sometimes called **tangent or polynomial method**) is to fit a smooth curve (like a polynomial) through the droplet profile and then simply take the slope at the contact point [32]. For example, one could fit a second- or third-degree polynomial to the upper half of the drop profile (for sessile drop, with x-axis along baseline and y-axis vertical) [32]. The derivative at the contact point (x at the edge) gives tan(θ). While this method makes no assumption of a circular shape, it's largely empirical and works best when the drop is not too large (so the profile is still monotonic and single-valued above the baseline). The Krüss DSA software mentions a polynomial method for intermediate angles [1]. We can include a simple version: e.g. fit a quadratic curve to the side profile and compute θ. However, given that we have circle and ellipse which cover similar ground, the polynomial fit might not be necessary except as a cross-check.

Use Cases: The simple geometric fits are primarily for **contact angle measurement on sessile drops**. They require identifying the baseline and contact points accurately. Once that is done, these fits yield the contact angle quickly without needing fluid properties. They are not used for surface tension directly (because they don't consider gravity or density) – except in the trivial case of a small droplet, one could estimate surface tension if one somehow knew the curvature radius at the top and used an approximation $\gamma \approx \Delta\rho\ g$ R<sub>0</sub>^{2} (but this is not standard). So, geometric fits are supplemental for contact angle and shape overlay. They also give volume estimates: if a drop is approximated as a spherical cap, volume can be calculated by the spherical cap formula [33] (see Section "Droplet Volume Calculation").

Geometric models are very fast computationally (fitting a circle or ellipse is quick) and can be used in batch mode when speed is needed. They also serve as initial guesses for the more complex Young–Laplace fit (for example, the circle fit's radius can seed the starting curvature for the ODE integration).

3. Axisymmetric Drop Shape Analysis (ADSA)

ADSA refers to a comprehensive approach of fitting the entire drop shape to the solution of the Laplace equation (Young–Laplace) for axisymmetric drops [34]. In essence, it's what we described for the pendant drop, extended and applied generally (including sessile cases). ADSA typically uses an optimization algorithm to adjust parameters until the theoretical shape best matches the image [22].

- For pendant drops, ADSA adjusts surface tension (and possibly a scaling factor or the drop volume in the model) to match the profile. This yields surface tension with high precision (often within 0.1% if the image is good) [35] . Commercial tensiometers always use this method for surface tension [2] .

- For sessile drops, ADSA can adjust the contact angle as a parameter. One implementation is the **Low-Bond Axisymmetric Drop Shape Analysis (LBADSA)** which is a variant optimized for small Bond numbers (small or moderately sized drops). LBADSA uses a first-order perturbation solution of the shape, which is much faster than full numerical integration, but still fits the whole profile [36] . It can retrieve contact angle by considering the drop plus its mirror image for the reflection, thus finding the true contact angle at the contact line [18] . This is an advanced feature we can note as a potential extension: to improve contact angle accuracy, fit both the drop and reflection contour in one step, which eliminates the need to pinpoint the contact point manually – the algorithm finds the best contact angle that makes the theoretical shape and its mirror align with both the actual drop and reflection in the image [18] .

- **Computational Considerations:** Full ADSA optimization (fitting multiple parameters) is computationally heavier. Each iteration involves solving the ODE (or an approximation) and comparing to the contour. However, for single images this is still quite feasible (the 2016 ImageJ plugin reported achieving ~1% precision with a simple webcam, indicating it runs in a matter of seconds or faster) [37] [38] . We will implement the core of this (especially for pendant drop surface tension). For sessile drop contact angle via ADSA, we might initially rely on geometric methods, but the architecture will be such that an ADSA module could be added.

- **Validation:** We should mention that part of the development will include validating the ADSA results against known values or even against a commercial instrument if possible. The ImageJ Pendent_Drop authors validated their results with known surface tensions and got good agreement [35] . We can create some synthetic images using known parameters (e.g. generate a theoretical drop shape, render it, add noise) to test if our fitting recovers the input parameters.

In summary, ADSA is the "all-encompassing" physical model approach. In the software, enabling "Young-Laplace fit" is essentially doing ADSA for the chosen scenario (pendant or sessile). Simpler options like "circle/ellipse fit" are quicker but limited models, as summarized in **Table 2**:

Table 2: Drop Shape Models and Their Characteristics

Model	Applicable Scenario	Inputs Required	Outputs Directly	Typical Accuracy
Young–Laplace Fit (full ADSA)	Pendant (surface tension) and Sessile (contact angle, if extended)	Density difference; initial guess for γ (for pendant) or contact angle (for sessile); image scale for absolute units.	Surface tension (pendant); contact angle (sessile) if implemented; volume; shape curve.	High (best method, <1% error for γ [35])
Bashforth–Adams numeric (internal to Y–L)	Pendant or sessile (computes shape for given γ)	γ, density, etc. (used within ADSA fit)	— (Generates shape for overlay or comparison)	High (but only as good as input γ)

Model	Applicable Scenario	Inputs Required	Outputs Directly	Typical Accuracy
Circle Fit (spherical cap)	Sessile drop (contact angles < ~30° very accurate, up to 90° acceptable) [26]	Image of drop (no fluid properties needed). Baseline must be known.	Contact angle, base width, height (also radius of fitted circle).	Good for small symmetric drops (errors increase for large drops) [39].
Ellipse Fit	Sessile drop (contact angles 10°–130°) [31]	Image of drop; baseline.	Contact angle, can also get height/ width.	Good for wide range, but more complicated fit (sensitive to noise).
Height-Width (Half-angle)	Sessile small drop quick estimate	Drop height *h*, base half-width *r*.	Contact angle (via $\theta \approx 2 \arctan(h/r)$) [28]; volume via spherical cap formula.	Approximate (assumes perfect spherical cap; best when gravity negligible).
Polynomial (Tangent) Fit	Sessile drop (any angle)	Drop profile coordinates.	Contact angle (from slope at base).	Medium (depends on fit region and polynomial order).

The software will allow the user to choose one or multiple methods. For example, one might use circle fitting to get a quick contact angle, and also apply the Young–Laplace fit for a more precise result if needed. The GUI will clarify which result came from which method.

Computing Physical Properties from the Image

With the droplet contour detected and a model chosen, the next step is to compute the quantitative properties: surface tension, contact angle, volume, height, and diameter. Here we detail how each is obtained:

Surface Tension (Pendant Drop Method)

For a pendant drop, the surface tension is derived from the shape of the hanging drop. We will implement this via the Young–Laplace fitting described earlier:

- **Young–Laplace Fit Method:** Using the detected contour of the pendant drop, we iterate on the surface tension value in the model until the theoretical shape matches the contour. The final γ that gives the best fit is reported as the surface tension. This method inherently uses the Young–Laplace equation (solved via Bashforth–Adams numeric integration) and is the same principle used by optical tensiometers [2]. The user must provide the **density difference Δρ** (liquid density minus ambient density, typically e.g. 1000 kg/m^3 for water in air) and the **gravitational acceleration g** (we can assume 9.81 m/s^2 unless the experiment is in a different gravity field). The reason is that the shape scaling relates to the capillary length ℓ<sub>c</sub> = sqrt(γ/(Δρ g)), so without Δρ and g, we cannot get γ in absolute units [2]. If the image is calibrated (pixels to mm), we can apply those units; if not, the software should warn that surface tension cannot be determined meaningfully without a scale.

- **Pendant Drop Shape Factor Method (Optional):** There is a classical approach to estimate γ from a few drop dimensions without a full numerical fit. It involves measuring the diameter of the drop at its widest point and the ratio of some vertical distances. Historically, one could measure the drop shape and use precomputed tables (from Bashforth and Adams) to get β (shape factor) then use γ = Δρ g R<sub>0</sub>/β [23]. In practice, our software can automate this: detect the apex curvature radius R<sub>0</sub> (by fitting a circle to the top tip of the drop) and measure, say, the diameter at some distance from the apex. However, since we are already extracting the full contour, it's just as easy to do the full fit. We might include a "quick estimate" mode: using the **aspect ratio of the drop** (height/width) to estimate the capillary length [40]. A rough formula is mentioned in literature that the aspect ratio correlates with γ [40], but it's not very precise. So this will not be the primary method.

- **Output:** Surface tension will be given in units of mN/m (millinewton per meter, a common unit for surface tension, equivalent to dyn/cm). For example, pure water might come out around 72 mN/m at room temperature [41]. We will likely display it with one decimal place precision by default. If the user's image lacks calibration or density info, we either cannot compute γ or we provide it in relative units (e.g. "$\gamma * (px^2)$", which is not very useful, so better to insist on needed info).

The success of this measurement will depend on a clear contour and a pendant shape that is well-formed (hanging pendant or an inverted drop (bubble) in another fluid – the algorithm is similar). If the drop is too small (nearly spherical), the surface tension calculation becomes very sensitive to noise (since the shape is almost symmetric, gravity effect minimal). There is a lower limit where pendant drop method is reliable (usually Bond number should be above ~0.1 or so). We will document that the drop should have a visible "tear" shape for best results [42].

Contact Angle (Sessile Drop Method)

For sessile drops on a solid surface, the **contact angle** is the angle between the droplet liquid and the solid at the line of contact. Our approach to measuring contact angle from the image will involve geometry of the droplet outline:

- **Finding the Baseline and Contact Points:** First, we identify where the droplet touches the surface (the "three-phase contact points"). In the binary contour, these are typically the two bottom-most points of the droplet that also lie on a horizontal line. We can find the convex hull of the droplet and check where it intersects a horizontal line at the bottom – that should give the left and right contact points. Alternatively, we project the contour onto the horizontal axis and find the span; the endpoints of that span on the contour are contact points. Once we have those, we define the baseline as the line between them (which should be horizontal if the camera was level). If the baseline is not exactly horizontal due to camera tilt, we might detect it as slightly tilted – we could correct by assuming it should be horizontal (or allow a small tilt, but usually better to correct the image orientation in preprocessing if needed).

- **Circle Fit Method:** Using the contact points and the upper contour, fit a circle. Specifically, take the coordinates of the two contact points and one additional point on the upper part of the droplet (perhaps the apex) – three points define a circle uniquely. We can use those or use more points for a least-squares fit. Once we have the circle's center (which will lie below the baseline for a sessile drop) and radius, we compute the contact angle. For a circle of radius *R* that intersects the plane at distance *R - h* below its top, the contact angle θ relates by simple geometry: cos(θ) = (R - h) / R, where *h* is the droplet height (which is also the distance from the top of the drop to the baseline). And sin(θ) = (base_radius) / R. But more directly, the tangent line at the contact point on the circle makes an angle whose complement corresponds to how far around the circle the contact point is. We can calculate θ by finding the slope of the radius line to the contact point: if the circle center is at (x<sub>c</sub>, y<sub>c</sub>) and contact point at (x<sub>cp</sub>, y<sub>cp</sub>), then the line from center to contact has slope (y<sub>cp</sub> - y<sub>c</sub>) / (x<sub>cp</sub> - x<sub>c</sub>). The tangent at the circle is perpendicular to the radius, so the tangent's slope is the negative reciprocal. The contact angle θ is the angle between this tangent and the horizontal baseline. In practice, that simplifies to θ = arctan(|x<sub>cp</sub> - x<sub>c</sub>| / |y<sub>c</sub> - y<sub>cp</sub>|). Since y<sub>c</sub> - y<sub>cp</sub> is essentially the drop height, and x<sub>cp</sub> - x<sub>c</sub> is half the base width, this reduces to θ = arctan(h / r) and then double it (because the angle inside the drop is symmetrical) [28]. Indeed that matches θ = 2 * arctan(h/r) earlier [28]. The Apex documentation noted that circle fitting is especially suitable for θ < 30° for best accuracy [43], but generally used up to ~90°. We'll implement it for all cases and see if an ellipse is needed for bigger angles.

- **Ellipse Fit Method:** If selected, we perform an ellipse fit on all the contour points of the droplet (above the baseline). OpenCV's `fitEllipse` returns the ellipse's orientation and axes. From that, we find where the ellipse intersects the baseline (there will be two intersection points, ideally matching the contact points). The slope of the ellipse at that intersection can be found by implicit differentiation of the ellipse equation. But simpler: we know the ellipse parameters: center (x<sub>c</sub>, y<sub>c</sub>), major axis *a*, minor axis *b*, and tilt angle φ. We can plug in the contact point coordinates into the ellipse equation and differentiate to get the tangent. This is somewhat involved; an easier route: since we have the equation, we can directly compute the angle via the normal vector at that point. However, to keep it straightforward, a numerical approach can be used: take a point slightly above the contact point along the contour and compute the slope between that and the contact point. That slope (Δy/Δx) then θ = 180° - arctan(slope) (if slope is measured along the droplet outline). Actually, since we can identify left and right contact angles, we should measure each side's tangent separately and possibly average if needed. Many software report left and right contact angles (which should be the same on a perfectly symmetric drop, but can differ on an imperfect drop as an indicator of asymmetry or surface tilt) [44]. Our tool can report both or an average.

- **Polynomial/Tangent Method:** If chosen, we will take, say, the lowest 5-10 points on each side of the droplet contour (from the contact point upward a small distance) and fit a line or quadratic through them. The slope at the contact point is approximated by this local fit. This is essentially doing a tangent by local derivative. It might be less robust to noise, but it's a method that doesn't assume a global shape form. This could be useful if the drop is very large (flattened) or irregular, where circle/ellipse might not fit perfectly. The polynomial method was recommended for higher contact angles and larger drops by Krüss (see "conic section method" vs "polynomial" for >100°) [1]. We can incorporate it to cover that range: for example, fit a second-degree polynomial $y = A x^2 + B x + C$ to the outline points and get derivative at contact. If the drop is axisymmetric and we align the x-axis through the center, we might fit even a symmetric polynomial to both sides, but it's simpler to do one side at a time.

After computing θ (one for left and right if both visible), we output the contact angle in degrees. If there is a noticeable difference between left and right (e.g. one side 84°, other 87°), we may output both values. In a perfect scenario, they should be the same; differences could hint at tilt or droplet asymmetry.

Droplet Volume Calculation

The volume of the droplet can be derived once we have the droplet's shape or key dimensions. We will implement volume computation in two ways, depending on the approach:

- **From Fitted Shape (Young–Laplace):** If we have the full shape function (r(z) where r is radius as a function of vertical coordinate z), the volume is obtained by revolving this profile around the vertical axis. In continuous form, volume *V* is:

$$V = 2\pi \int_{z_{\text{bottom}}}^{z_{\text{top}}} r(z)^2 \, dz,$$

  where *r(z)* is the horizontal radius at height *z*. Our numeric shape is discrete, so we can do a discrete sum (trapezoidal integration) of π $r(z)^2$. The ODE solver might give us directly the dimensional shape coordinates, or we reconstruct them from the fitted parameters. This will give volume in whatever units the coordinates are (so if mm, then mm^3, convert to μL where 1 μL = 10^3 mm^3).

- **Analytical Spherical Cap Formula:** If using the spherical cap approximation (like for small sessile drops), we can compute volume using the formula for a spherical cap:

$$V = \frac{\pi h(3a^2 + h^2)}{6},$$

  where *h* is the drop height and *a* is the base radius [33]. Note that base diameter = 2a, and height is measured from base to apex. This formula assumes the droplet is a section of a sphere (which is the same assumption as the circle fit method). We will apply this if the user selects the simple volume calculation or if no better model is available. It tends to slightly overestimate volume if the drop is flattened by gravity, but for small angles it's very accurate. We will cite this formula in documentation and perhaps automatically use it when contact angle < some threshold and no other info.

- **Numerical Integration from Contour:** Even without the Young–Laplace model, we can numerically integrate the volume from the detected outline. Since we have the pixel coordinates of the contour, we can do the following: find the center line (for symmetry, we assume the drop is symmetric about a vertical centerline which goes through the apex for pendant or the midpoint between contact points for sessile). Then, take the right-side contour (from apex down to base for sessile, or apex to bottom for pendant) and use it as r(z). We reflect it to the other side for full drop. The volume is then approximated by summing circular disk volumes across the height. For example, take small vertical slices (each pixel row is a slice): for each y (row), determine the half-width of the drop (distance from centerline to edge) = r. Then slice volume ≈ area of circle of radius r * slice thickness. In discrete terms,

$$V \approx \sum_{y=y_{\text{bottom}}}^{y_{\text{top}}} \pi [r(y)]^2 \, \Delta y.$$

If the image is calibrated, Δy in mm. If not, it's pixel volume. We have to be careful about pixel geometry (if pixel aspect ratio is 1:1, which we assume). This method will be implemented for generality because it works for any shape given just the contour. It might be slightly noisy (the contour might not be perfectly smooth), but a smoothing of r(y) or fitting a spline can help. The ImageJ plugin also computed volume after fitting [45], and we can compare our integration with what a fitted model gives to ensure consistency.

- **Output Units:** Volume will be reported in microliters (μL) if calibrated (since typical drop volumes are a few μL to tens of μL in many experiments). If using pixel units, we might just give "(in pixel^3)" unless calibrated. Alternatively, we could allow the user to input the liquid density and compute the drop volume by weight if they measured drop weight, but that's out of scope – we'll stick to geometric volume.

Droplet Height and Diameter

These are straightforward geometric quantities once the droplet contour or key points are known:

- **Droplet Height:** For a pendant drop, height is measured from the top (apex) of the drop to the bottom tip. In the image, once segmented, we find the minimum y (top) and maximum y (bottom) of the droplet's pixels. The difference is the pixel height. We can refine the top if the drop is attached to a needle – usually, we'd exclude the portion above the neck where it attaches, focusing on the "free pendant part" [6]. Our software might allow the user to select a top cutoff (like drawing a box around just the drop, excluding the needle tip). We will document that if the drop is hanging, the height is measured from the bottom of the needle or wherever the drop detaches. For a sessile drop, height is measured from the baseline (surface) to the apex. That is simply the y-distance between the baseline (contact point y) and the highest point of the droplet contour. Because we identify contact points and baseline, we get this easily. Height is usually reported in mm (or pixels if uncalibrated).

- **Droplet Diameter (Width):** For a sessile drop, the relevant "diameter" is the base width (the distance between the two contact points along the baseline). That we get once we know contact points. For a pendant drop (which has no solid base), a characteristic width is the **maximum horizontal diameter** of the drop. We can find this by scanning the contour: find the maximum distance between the left and right edges of the drop at any vertical level. Alternatively, the contour points can be used to get the bounding box width, but for a pendant drop the widest part might be some way down from the top (the "equator" of the pendant shape). We can find that by finding the farthest pair of points on opposite sides of the droplet. Since we usually segment as one blob, the simplest is to take the binary mask of the drop and compute its horizontal extent for each row, then find the max. This will yield the max diameter in pixels. In a well-formed pendant drop, the max width might be near the middle or upper part depending on shape. We will output that as "max diameter". If needed, for pendant drops, we could also output the diameter at the neck (if analyzing near detachment), but that's more for research; not required here.

- **Reporting Height/Width:** These geometric values can serve as sanity checks and are also used in some empirical correlations. We will report them likely in mm (to one decimal if appropriate, since droplet heights might be a few mm). They are also used in intermediate calculations (like height and width for the half-angle method of contact angle, or computing aspect ratio for a quick γ estimate).

Both height and width can be determined very reliably from the binary droplet mask, so they should be simple to compute even in batch mode without any advanced fitting.

Summary of Outputs and Equations

To summarize how each property is computed, we provide the following reference:

- **Surface Tension (γ):** Computed for pendant drops by fitting Young–Laplace equation to droplet contour (numerical solution of ODE with adjustable γ). Requires density difference and scale. Equation used internally: γ = Δρ g R<sub>0</sub> / β [23], with β found by matching shape.
- **Contact Angle (θ):** Computed for sessile drops. By default, via circle fitting or half-angle formula $\theta = 2 \arctan(h/r)$ [28] (h = height, r = half base width). Or via direct curve fitting (circle/ellipse) to find the tangent at contact [27]. No fluid properties needed.
- **Volume (V):** If using spherical cap assumption: $V = (\pi h (3r^2 + h^2)) / 6$ [33] (for sessile drop, r = base radius, h = height). Generally, computed by numerical integration of the contour (rotational volume). Requires scale for real units.
- **Height (H):** = apex_y - baseline_y (for sessile, baseline_y known; for pendant, baseline is top attachment). Computed directly from pixel measurements of contour.
- **Diameter (D):** For sessile = distance between contact points (base width). For pendant = max horizontal span of drop. Computed from contour coordinates.

These formulas and methods ensure that each required output is backed by either a physical model or a clear geometric calculation.

Overlaying Predicted Shape on the Image

Visualizing the results is important for user confidence and for diagnosing the quality of the fit. The software will overlay predicted shapes onto the original image as follows:

- After computing a model (say Young–Laplace or a circle fit), we have either an explicit function or a set of points for the droplet's profile. For instance, the Young–Laplace ODE solver might return a list of (x,z) coordinates in physical units (with z vertical and x horizontal radius). We will convert those into image pixel coordinates. This requires knowing the mapping: likely we treat the apex or the base as an anchor. For a pendant drop, we can align the theoretical shape's apex to the detected apex of the real drop. Then scale the theoretical shape horizontally and vertically to match the real size. Actually, in the fitting process, we implicitly determine the scale (that's part of finding γ). Once aligned and scaled, we have a series of points (x_pix, y_pix) that represent the model outline. We will draw this as a smooth curve on the image. Typically, drawing can be done by converting points to an OpenCV PolyLine or using Matplotlib's plot.

- For sessile drop shape overlays: if we solved an axisymmetric shape (e.g. LBADSA approximation), we would similarly align the baseline and apex. However, in most sessile cases, we might just overlay the circle or ellipse that was fitted. A circle can be drawn easily if we know its center and radius (OpenCV `cv2.circle` or using QPainter). An ellipse similarly with `cv2.ellipse` given its parameters. We'll ensure to only overlay the portion of the circle/ellipse that corresponds to the droplet (not the part "inside" the solid or below the baseline – though for clarity maybe a full outline in a different color is fine). Alternatively, just draw the arc between the two contact points.

- **Color and Legend:** We will choose distinct colors for different models: e.g. red curve for Young–Laplace fit (since that's often considered the theoretical shape) [41], green for a circle/ellipse fit, blue for the raw detected edge. The image overlay should be annotated in a small legend (or in text) if multiple overlays are shown simultaneously. However, to avoid clutter, we might allow only one model overlay at a time or have checkboxes to toggle them.

- **Scaling Issues:** We must be careful that the overlay aligns with the image. If the image was rotated or cropped, we need to ensure we use the same coordinate origin. Typically, image coordinates (pixels) start at top-left. For consistency, we might convert everything to an (x,y) Cartesian system where y=0 at baseline (for sessile) or at apex (for pendant) internally, do calculations, then map back. For pendant: the apex might be easier as origin (since ODE is solved from apex downward), then we find where bottom is, etc. The fitting process likely already determined the alignment implicitly. In any case, after fit, we will have to do a translation of the shape: e.g. add the pixel coordinates of the real apex to the model shape apex to overlay correctly.

- **Graphics Implementation:** In PyQt5, one approach is using a `QGraphicsView` with the image as a pixmap item, and then adding QGraphicsPathItem for the contour lines. This allows easy overlay and zooming. Another approach: use Matplotlib by showing the image and plotting lines (this is simpler for coding but less interactive for a GUI context, though matplotlib can be embedded). We prefer QGraphicsView for a fluid interactive GUI. We will generate the path for the theoretical contour by connecting the model points (which are already in correct order). If resolution of model points is coarse, we can interpolate a smooth curve (the ODE solver can output fine increments, or we can spline-fit the points).

- **Example Overlay:** As shown in the earlier figure, the blue outline was the detected edge and red the theoretical [41]. Our software will replicate such visualization. For sessile drops, an overlay might be the fitted circle arc in green matching the edge near the base.

- **User Interaction:** The user might want to manually adjust if the overlay seems off (which could indicate a poor fit or wrong input like density). We might allow the user to tweak parameters and see the overlay update in real time (e.g. a slider for surface tension that moves the red curve slightly). This could be a helpful educational feature. However, initially, we can just display the final result.

- **Saving Results:** The GUI should allow saving the image with overlays (for reports or lab records). We can implement a "Save annotated image" which will combine the pixmap and overlay drawings into a single image file.

Batch Processing and Automation

In batch mode, the software will process multiple images without user intervention on each. The main differences in batch mode:

- **No interactive adjustment per image:** The segmentation and analysis will use either the same parameters for all images or an automatic mode. For example, if one image needed a slightly different threshold, the batch might not catch that unless we implement an adaptive method for each (like always using Otsu per image, which we plan to do). We will assume images in a batch are similar in setup so that one configuration works for all.

- **Performance:** We will optimize critical loops by using NumPy vectorization where possible (image threshold, projection, etc. are all vectorized in OpenCV/NumPy). The Young–Laplace fitting might be the slowest part if done for each image; we can estimate – solving ODE and optimizing might take maybe 0.1–1 second per image depending on resolution and required precision. Processing, say, 100 images should still be reasonable (on the order of a couple minutes). If needed, we could offer a simpler mode for batch (like only do circle fitting for contact angle to go faster, or only do surface tension calc at the very end of a video sequence for one or two key frames).

- **Output of Batch:** All results will be compiled into a table (like a CSV). Each row: filename, surface tension (if applicable), contact angle (if applicable), volume, height, width, etc. Possibly also the date/time, or any error flag if a particular image failed (say couldn't detect drop). The GUI might show a preview of this table and allow copy-pasting it or saving. If a user is doing time-series (like watching a drop evaporate or oscillate), this could feed into further analysis (we might support simple plotting of results vs time if images are timestamped).

- **Logging:** For batch processing, a log of actions for each image might be kept (especially if errors or warnings occur, like "Image3: Droplet not found" if something goes wrong). This log can be shown or saved.

Optional: Machine Learning Enhancements

While the initial design relies on traditional image analysis, machine learning (ML) could improve robustness, especially for segmentation:

- **Droplet Segmentation via ML:** A convolutional neural network (CNN) could be trained to perform semantic segmentation, labeling each pixel as droplet or background. A U-Net architecture, for example, could likely outperform simple thresholding when background conditions vary (different lighting, reflections, complex backgrounds). The downsides are the need for a training dataset and the added dependency on ML frameworks. This might be offered as a plugin or future update where the user can supply some manually annotated images to train a model for their setup. For a general tool, we might provide a pre-trained model if we gather enough droplet images (for example, one could simulate droplet shapes on random backgrounds to train a model to find them). For now, we note it as a possible improvement if thresholding proves to be a bottleneck.

- **Improving Fitting with ML:** Another emerging idea is using ML to directly predict surface tension from the shape without iterative solving [3]. Researchers have trained deep networks on simulated drop shapes to output γ directly [3]. This could make the analysis instantaneous at runtime. We could consider incorporating such a model (perhaps as a checkbox "Use ML estimator for surface tension"). It would still require density and scale to output in correct units, but it could bypass the need for solving ODE each time. However, careful validation would be needed to trust this approach.

- **Contact Angle ML:** Similarly, one could train a model to output contact angle from the image of a drop on a surface. But since geometric methods are already fast, this is less of a need.

- **Quality Control with ML:** Another use might be to classify if an image is good or not (sharp vs blurry, properly backlit vs not) to warn the user if a particular image might yield poor results.

Given the complexity, ML features will be optional and likely off by default. They might appeal to advanced users or in research settings where they want to push throughput (e.g. analyzing hundreds of drop images quickly).

Development Phases and Testing

To implement this plan, we foresee these phases:

1. **Prototype with Core Functionality:** Start with reading an image, doing threshold segmentation, contour detection, and basic measurements (height, width, contact angle via simple geometry). Implement the GUI skeleton with image display and single-image analysis. Ensure the basic contact angle measurement works on known images (e.g. a test image where we know the angle ~90°).

2. **Add Physics Models:** Implement the ODE solver for the Young–Laplace equation (using SciPy). Test it on known cases: e.g., synthetic shape generation. Then implement the optimization loop to fit surface tension for a pendant drop. Validate with a reference – for example, create a fake drop shape assuming γ = 72 mN/m, see if the algorithm recovers ~72. This phase also includes implementing the circle and ellipse fitting functions and verifying on simple shapes (like a perfect circle image).

3. **GUI Refinement:** Integrate the model computations into the GUI, allowing the user to choose models and see overlays. Add calibration input functionality and ensure units convert correctly (check that if we simulate a known mm scale, the output numbers match).

4. **Batch Mode & Robustness:** Implement batch processing loop. Test on a set of images (we can take a short video of a pendant drop and extract frames, then process all to see if we get consistent γ). Optimize any slow parts (profile the code; if the ODE fitting is slow, consider reducing the search space by using the measured drop size to guess an initial γ or use the shape factor formula as initial guess).

5. **Testing on Real Samples:** Collect or use known benchmark images (perhaps literature or open-source datasets, if available). For contact angle, one can use images where the angle was measured manually and compare. For surface tension, maybe use a known liquid (water, known 72 mN/m) and see if result is close. Adjust algorithms as needed (tuning threshold strategies, snake parameters, etc.).

6. **Error Handling and UI Polish:** Make sure the program gracefully handles cases like: no droplet found (pop up "Droplet not detected, please adjust threshold or crop image"), extremely large drops (maybe out of frame), etc. Also ensure the UI elements are intuitive (labels, tooltips, documentation).

7. **Optional ML integration:** If time permits or as a future update, experiment with using a pre-trained model for segmentation. This could run via OpenCV dnn module or TensorFlow. It should be toggleable. Ensure that even without it, the classical approach works well in general scenarios.

Throughout development, we'll maintain and cite references for the implemented equations and methods, both in code comments and in any documentation, to acknowledge sources like the ImageJ Pendent_Drop plugin (which gave guidance on the algorithm) [22], the EPFL DropSnake project for contact angle methods [16], and standards from companies (Krüss, Biolin) that outline best practices [25] [1]. This grounding in known methods will help users trust the results.

Conclusion

In this development plan, we have outlined a comprehensive approach to building a Python-based GUI application for droplet shape analysis. By leveraging powerful libraries (OpenCV, scikit-image, SciPy, PyQt5) and implementing both straightforward geometric algorithms and advanced physical models, the tool will be capable of analyzing pendant and sessile drop images to yield surface tension, contact angle, and other droplet characteristics. We have addressed the requirements of single and batch processing, calibration flexibility, multiple analysis models, and result visualization. The combination of these features will make the software useful for researchers and engineers in surface science, offering an open and extensible alternative to commercial drop analysis software. With careful implementation and validation [35], the final application will provide accurate measurements (e.g. surface tension within ~1% and contact angle within ~0.5° under good conditions) and a user-friendly experience for droplet image analysis.

---

[1] [2] Drop shape analysis | KRÜSS Scientific
https://www.kruss-scientific.com/en-US/know-how/glossary/drop-shape-analysis

[3] Pendant Drop Tensiometry: A Machine Learning Approach - arXiv
https://arxiv.org/abs/2006.10111

[4] [5] [7] [8] [9] [10] [11] OpenCV: Image Thresholding
https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html

[6] [18] [20] [22] [35] [37] [38] [40] [41] [45] Pendent_Drop: An ImageJ Plugin to Measure the Surface Tension from an Image of a Pendent Drop | Journal of Open Research Software
https://openresearchsoftware.metajnl.com/articles/10.5334/jors.97

[12] [13] [14] Active Contour Model — skimage 0.25.2 documentation
https://scikit-image.org/docs/0.25.x/auto_examples/edges/plot_active_contours.html

[15] [16] [17] [36] BIG>Drop Analysis
https://bigwww.epfl.ch/demo/dropanalysis/

[19] [23] [24] [42] Pendant drop method for surface tension measurements
https://www.biolinscientific.com/blog/pendant-drop-method-for-surface-tension-measurements

[21] [PDF] Modeling of a sessile droplet with the curvature dependence of ...
https://journals.tubitak.gov.tr/cgi/viewcontent.cgi?article=1134&context=physics

[25] [26] [27] [28] [29] [30] [31] [39] [43] Contact Angle Measurement Technology — Apex Instruments
http://www.apexicindia.com/technologies/contact-angle-measurement-technology

[32] Contact Angle Measurement, Theory & Relation to Surface Energy
https://www.ossila.com/pages/contact-angle-theory-measurement

[33] Spherical cap - Wikipedia
https://en.wikipedia.org/wiki/Spherical_cap

[34] Axisymmetric Drop Shape Analysis (ADSA) for the determination of ...
https://www.researchgate.net/publication/239441646_Axisymmetric_Drop_Shape_Analysis_ADSA_for_the_determination_of_surface_tension_and_contact_angle

44 Droplet image and contact angle measurement result suing circle fit... | Download Scientific Diagram
https://www.researchgate.net/figure/Droplet-image-and-contact-angle-measurement-result-suing-circle-fit-model-Left-angle_fig1_314161064