**Roadmap for Python CeleST Reimplementation with Streamlit**

A screenshot of a computer

AI-generated content may be incorrect.

**Overview and Objectives**

CeleST (**C. elegans** Swim Test) is a comprehensive system for analyzing nematode swimming behavior. It encompasses **(1)** a database of videos with metadata, **(2)** a multi-animal tracking algorithm, **(3)** a set of ten quantitative swim measures, and **(4)** tools for plotting and statistical comparison[[1]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf). Our goal is to **rewrite CeleST in Python**, optimizing for web-based usage via **Streamlit**. This roadmap details how to implement each component in Python and outlines a Streamlit app structure, along with library recommendations and comparisons to MATLAB functions.

**Key features to replicate:**

* **Database & Data Handling:** Storing video metadata (e.g. strain, age, trial number) and selecting groups of videos for analysis[[1]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf).
* **Multi-Worm Tracking:** Automatically segment multiple worms per video frame, track their outlines and centerlines over time, and handle overlaps or lost tracks[[1][1]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf).
* **Curvature-Based Measures:** Compute **10 swim metrics** (e.g. wave initiation rate, body wave number, asymmetry, curling, speed, etc.) by analyzing each worm’s midline curvature over time[[1][1]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf).
* **Plotting & Statistics:** Provide comparative visualizations (line charts, histograms, scatter plots) for single worms or groups, including basic statistical annotations (mean/median, variance, significance tests).

Below, we break down the plan into technical tasks with Python approaches, mapping MATLAB functions to Python equivalents and highlighting relevant libraries.

**1. Data Management and Video Handling**

**Database structure:** In the original CeleST, users import video files into a database with identifying tags (sample ID, date, experiment, strain, age, etc.)[[1]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf). We will implement a similar structure using Python data structures:

* Use a **CSV file or Pandas DataFrame** to store video file paths and metadata columns (sample\_id, date, experiment, trial, strain, age, etc.). This can be loaded at app start and edited through the UI if needed.
* Alternatively, integrate a lightweight database (e.g. SQLite via SQLAlchemy) if persistent, multi-user data storage is required. For a simpler solution, a CSV/JSON or even Python pickle of a dict would suffice for a single-user app.

**Video reading:** Each video will be processed frame by frame in the tracking step. We recommend using **OpenCV (cv2)** or **imageio** for video capture:

* **OpenCV VideoCapture** can read frames from common video formats efficiently. For example: cap = cv2.VideoCapture('video1.avi'); then use cap.read() in a loop to retrieve frames.
* If videos are large, consider converting them to a standard format/resolution offline to ensure smooth processing in Python.

**Streamlit integration:** The app will allow users to upload new videos or select from existing. Steps:

* Provide a file upload widget (st.file\_uploader) for new video files. Save these to a server directory and append their info to the DataFrame.
* Show a table (using st.dataframe) of videos in the “Database” section where users can filter by metadata (e.g. choose all videos of a certain strain or age).

By managing video files and their metadata in Pandas, we make it easy to group and select videos for batch analysis (e.g. “Compare all wild-type day4 vs day10” scenarios). This addresses the **data import and selection** functionality of CeleST[[1]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf).

**2. Multi-Animal Tracking Algorithm**

The tracking algorithm must locate each worm in every frame and reconstruct its centerline (midline) through time[[1][1]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf). The MATLAB version uses a series of image processing steps which we will reproduce with Python libraries:

**2.1 Image Preprocessing:** Each frame should be filtered to enhance worm edges and reduce noise:

* **Local Standard Deviation Filter:** CeleST applies a 5×5 std filter to highlight worm edges (worms appear as high-contrast objects against background)[[1]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf). In Python, we can implement this via SciPy or OpenCV:
  + Use **SciPy**: scipy.ndimage.generic\_filter(frame, np.std, size=5) to compute local std dev in a 5×5 neighborhood (analogous to MATLAB stdfilt)[[2]](https://se.mathworks.com/help/images/ref/stdfilt.html).
  + Alternatively, apply a difference of Gaussians or simple edge enhancer: a custom convolution kernel approximating an edge detector could suffice, but using the std filter directly mimics the original algorithm.
* **Intensity Gradient:** Compute the gradient magnitude image to delineate worm outlines[[1]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf). We can use **OpenCV Sobel filters** to get gradients:
  + e.g. gx = cv2.Sobel(filtered, cv2.CV\_64F, 1, 0, ksize=3) and similarly gy for vertical, then gradient magnitude grad = sqrt(gx\*\*2 + gy\*\*2).
  + Or use **cv2.Canny** edge detector for a robust outline (with tuned thresholds). Canny effectively does Gaussian smoothing, gradient, and edge linking[[3]](https://www.geeksforgeeks.org/python/find-and-draw-contours-using-opencv-python/). This might give similar results to the MATLAB approach of “greedy line-growing” by providing continuous contours.

**2.2 Worm Segmentation (Finding Contours):** Using the gradient image, CeleST finds closed outlines of worms by a “greedy line-growing” method maximizing gradient flow[[1]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf). In Python, we can simplify this by using OpenCV’s **contour finding** on a binary edge image:

* Apply a suitable threshold to the gradient magnitude image to create a binary mask of edges. This threshold may be dynamic (e.g. Otsu’s method or a fixed value chosen from experience).
* Use **cv2.findContours** on the binary edge image to extract contours[[3][3]](https://www.geeksforgeeks.org/python/find-and-draw-contours-using-opencv-python/). OpenCV will return a list of contours, each a sequence of (x,y) points forming a closed perimeter of a detected object. This corresponds to finding each worm’s outline.
* Among contours, filter out small or spurious ones by area. Actual worms have a characteristic size (depending on magnification); we expect them to be, say, >1000 pixels in area in our videos. We discard contours too small or too large.

Each retained contour is the worm’s outer boundary. From this, we need the **centerline**. We have two approaches:

**2.3 Centerline Extraction:** In CeleST, they compute the *inner distance transform* of the outline and take its ridge as the centerline[[1]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf). Python approach:

* Create a binary mask image for each worm (e.g. mask = np.zeros\_like(frame\_gray); cv2.drawContours(mask, [contour], -1, color=255, thickness=-1) to fill the contour).
* Compute **distance transform** on the mask, which gives a heatmap where each pixel’s value is distance to the nearest background (so highest at shape center)[[1]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf).
  + Use cv2.distanceTransform(mask, distanceType=cv2.DIST\_L2, maskSize=5) (L2 gives Euclidean distance).
* Find the **skeleton** (centerline) of the shape. One method: take the ridge of the distance transform as suggested:
  + The ridge can be approximated by applying a thinning/skeletonization on the mask. We can use **skimage.morphology.skeletonize** to reduce the binary worm mask to a 1-pixel wide skeleton.
  + Alternatively, trace equidistant points along the distance map: start at the pixel of maximum distance (likely center) and flow outward.
* For simplicity and reliability, using skeletonize from scikit-image is effective. It yields a binary image with a single-pixel wide skeleton following the worm’s midline.

**2.4 Head and Tail Identification:** The skeleton will not yet distinguish head vs tail. In CeleST, they postpone head-tail labeling until after computing motion direction (swapping if worm mostly swims reverse)[[1]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf). We can do similarly: initially treat the skeleton endpoints as arbitrary head/tail. We can identify endpoints by finding pixels in the skeleton with only one neighbor (end of the curve). Assign one end as “head” arbitrarily, and mark it. (We can refine after analyzing motion in the Measures step – see **Reverse Swimming** measure.)

**2.5 Tracking Worms Frame-to-Frame:** Once we have each worm’s outline and centerline in one frame, we need to match them to worms in the next frame to maintain identity tracks[[1]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf). CeleST uses a predictive “two-step greedy fitting” and periodically re-segments to avoid drift[[1][1]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf). We can implement a simpler but effective approach:

* **Predict positions:** Assume small movement per frame due to 18 fps capture[[1]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf). The worm’s new position likely overlaps the old one. Use the previous frame’s skeleton or contour to predict the search region in the new frame:
  + As CeleST notes, “there was some overlap in the bodies from t-1 to t”[[1]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf). So for each worm last frame, look in the new frame for a contour that overlaps or is very near the previous contour. We can check contour bounding boxes or centroid distance.
* **Stationary points method:** CeleST identifies at least two stationary points by comparing consecutive frames intensities[[1]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf). A simpler proxy: compute cross-correlation between small patches, or just do nearest-neighbor matching of worm centroids frame-to-frame (since worms typically don’t all move far simultaneously).
* **Matching algorithm:** Form sets of worm features (e.g. centroids or minimum-distance between contours) and do a matching:
  + Calculate distance matrix between each contour in frame N and each worm from frame N-1 (distance could be overlap area *– a high overlap suggests same worm–*, or centroid distance).
  + Solve the assignment using e.g. Hungarian algorithm (available via scipy.optimize.linear\_sum\_assignment) to find minimal total distances. This gives a pairing of old worm IDs to new contours.
  + If a worm from last frame has no close match (e.g. it left the field or merged), mark it as lost for now. If a new contour appears with no predecessor, assign a new ID (or if it might be a split from a merged blob, handle accordingly – see overlap below).
* **Overlaps and merging:** If worms touch or cross, segmentation might produce a single merged contour or irregular skeleton. CeleST flags such frames and can later allow user to manually validate[[1]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf). In our automated rewrite:
  + Detect potential merges if two tracked worms map to one contour or vice versa. We could flag those frames for exclusion from measure computation (just as CeleST does)[[1]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf).
  + We might attempt to separate worms in a merged blob by analyzing skeleton branches, but that is complex. Instead, mark as “contact frame” and do not compute certain measures for those frames (setting an indicator).
  + Maintain continuity: if worms separate later, resume tracking them as distinct IDs.
* **Periodic re-segmentation:** To correct drift, CeleST re-segments every 20 frames and merges results with tracking[[1]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf). We can adopt a similar strategy: every N frames (say 20), re-run full segmentation independently and realign IDs. This can catch mistakes where a worm’s contour was lost.

**Quality control:** After tracking, apply quality checks akin to CeleST's (e.g. body length two std dev below mean indicates worm left view)[[1]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf). Mark such cases and possibly drop that worm from analysis if too many frames are invalid (CeleST rejects worm tracks >20% bad frames)[[1]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf). In our Python pipeline we can:

* Compute each worm’s length (e.g. skeleton length in pixels) across frames. If a worm’s length suddenly drops far below its typical length, likely part of it left the frame or tracking failed.
* Remove or flag those segments from measure calculations. We can similarly color-code flagged frames in visualizations for user review (if implementing a review interface).

**Memory/performance considerations:** Using vectorized ops (NumPy, OpenCV’s C++ backend) will keep this efficient. For example, cv2.findContours and distanceTransform are in C and run quickly even on full frames. Tracking via assignment is trivial for ~5 worms. Thus, even with Python-level loops per frame, processing a 30 s video at 18 fps (~540 frames) with ~5 worms is feasible in under a minute on a modern CPU. This is acceptable for a Streamlit app (especially if run asynchronously or with a “Start Analysis” button). We can also optimize by processing only a region of interest per worm for subsequent frames instead of full image (cropping around predicted worm location to reduce computation).

**Summary:** The tracking module will take each video frame, filter for edges, find worm contours and skeletons, and maintain identity links frame-to-frame. It will output for each worm: a time series of centerline coordinates (discretized along the body), or at least a skeleton image per frame. These data feed into the next stage to compute curvature and features.

**3. Implementing Curvature Analysis & Swim Measures**

Once each worm’s midline is extracted over time, we calculate the **ten behavioral measures** defined in CeleST[[1][1]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf). We mirror the mathematical definitions provided[[1][1]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf), using Python libraries (NumPy/SciPy) for numerical operations:

**3.1 Data representation:** Represent the centerline as a set of 2D points along the worm’s body, parameterized by relative length *s* from tail (s=0) to head (s=1)[[1]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf). CeleST uses 12 segments (13 points) along the body for calculations[[1]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf). We can adopt the same:

* Sample the skeleton into 13 roughly equidistant points (this handles slight skeleton pixel length differences). For instance, compute the cumulative distance along the skeleton, then interpolate 13 points at 0%,8.3%,16.6%,…,100% of the total length.
* Store these point coordinates as **(x(s,t), y(s,t))** for s ∈ {0,…,1} (tail to head) and each frame t[[1]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf). Also store the local half-width **r(s,t)** from distance transform at those points (so we know the worm’s radius at each segment)[[1]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf).

Using this parameterization, we proceed to measures:

**3.2 Curvature (κ) Calculation:** For each point along the body *s* at time *t*, curvature κ(s,t) is defined by CeleST as the derivative of the tangent angle along the body[[1]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf). We can compute curvature from the coordinates using the standard formula for a parametric curve:

[ \kappa = \frac{x' y'' - y' x''}{\left((x')^2 + (y')^2\right)^{3/2}} ]

Where ' and '' are first and second derivatives w.rt body arclength *s*[[1]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf). In discrete form:

* For each frame t, use the 13 centerline points. Compute first derivative at point i via finite difference: (x'*i ≈ x*{i+1} - x\_{i-1}) (similarly y').
* Compute second derivative: (x''*i ≈ x*{i+1} - 2x\_i + x\_{i-1}).
* Then compute κ at interior points (for ends, we might use one-sided differences or simply exclude the very tail/head for curvature).
* **NumPy** makes this easy: an array of shape (13, T, 2) for positions, then apply diff operations along axis 0 (the s-axis).

This gives κ(s,t) for s=0.083,…,0.917 (since ends might be slightly less accurate). We will use these in metrics.

**3.3 Short-Time Fourier Transform (STFT) of Curvature Map:** CeleST’s most innovative analysis is applying a short-time 2D Fourier transform to the curvature map κ(s,t)[[1]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf). Essentially, they treat curvature as a function of body position s and time t, and perform spectral analysis to get **temporal frequency (v\_t)** and **spatial frequency (v\_s)** of body waves[[1]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf). They do this over multiple window lengths (32–64 frames) to handle nonstationarity, then average[[1]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf).

In Python, we can implement a similar approach:

* Define a time window (e.g. 2 seconds ~ 36 frames, or exactly 32 frames as in paper) and slide it along the time series.
* For each window, have curvature data κ(s\_i, t) for s=12 segments and t = window length.
* Use **NumPy’s FFT**: Compute 2D FFT on the curvature matrix of shape (12 segments × N frames window). Since κ is real, focus on positive frequencies as noted[[1]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf).
* Find the dominant mode from the FFT result: identify the frequency bin with maximum power. This gives spatial frequency v\_s and temporal freq v\_t of the main body wave[[1]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf).
  + Spatial frequency *v\_s* indicates how many waves fit along the body (e.g. ~1 for one full sinusoidal shape, ~0.5 for half a wave).
  + Temporal frequency *v\_t* is the stroke frequency (waves per second).
* We then slide the window and could average results as CeleST does[[1]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf), or directly use the instantaneous values if the window is short.

However, given the complexity, we might simplify by computing global dominant frequencies for the whole trial:

* **Wave Initiation Rate (WIR):** This is essentially the stroke frequency in strokes per minute[[1][1]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf). We can derive it from v\_t above: WIR = v\_t \* 60 (since v\_t in Hz). Alternatively, perform a 1D FFT of κ at a mid-body point over time to detect oscillation frequency. A simpler reliable method: measure time between successive peaks in bending at head or tail. But using FFT is more principled. We will likely follow their method for accuracy.
* **Body Wave Number:** The number of full waves along the body[[1]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf). This corresponds to |v\_s| from the FFT (the spatial frequency at the dominant mode). For example, |v\_s|=1 means one full wave on the body, 0.5 means half-wave. We report the value (they take absolute to ignore direction)[[1]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf).
* These two measures come directly from the dominant Fourier mode per frame (or median over trial). We can compute WIR(t) and wave#(t) for each moment and then summarize (CeleST uses median per 30 s trial)[[1][1]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf).

**3.4 Other Measures:**

Using the curvature time-series and possibly the half-width data:

* **Asymmetry:** Measures bias in bending left vs right per stroke. They compute average curvature over a full stroke period[[1]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf). We can implement by integrating κ(s,t) over one stroke (which we know from WIR), or simpler: for each half-wave, sum curvature values (positive minus negative). In practice, one can average curvature across the whole body over one oscillation: a positive mean indicates a bias to one side.
  + We compute asymmetry as described: average curvature during two consecutive strokes[[1]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf). Using our time-frequency analysis, we know stroke duration Δt = 1/(v\_t). For each frame t0, integrate κ(s,t) over t0 ± Δt (one stroke before and after) and s∈[0,1]. This yields asymmetry(t0). We then take absolute value for group comparisons (to avoid cancellation of left vs right biases)[[1]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf). But per worm we keep sign (direction of bias).
* **Stretch:** The maximum curvature range in a stroke[[1]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf). They find, per stroke, the difference between most bent and most straight segment at any instant[[1]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf). Implementation: for each time t, find max κ and min κ along the body, take difference = curvature range. Then within one stroke (duration Δt), find the largest such range. Essentially how deeply the worm bends. We can approximate stretch by simply taking (max κ - min κ) at each frame and then smoothing over a stroke period or taking the median of the top 10% values.
* **Attenuation:** Compares head and tail bending amplitude to see if wave diminishes along body[[1]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf). Compute, for each stroke, the absolute curvature range at head vs at tail. They define attenuation = 1 - (tail range)/(head range), as a percentage[[1]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf). Implementation: identify a wave as it travels; simpler: measure peak curvature at head and corresponding peak at tail for that wave. Alternatively, use the distance transform radii: if head moves but tail hardly moves, attenuation = 100%. We can mimic formula[[1]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf) by tracking head and tail curvature extremes over each stroke:
  + When head is at maximal bend (either direction), note curvature magnitude at head vs tail at that moment (or the corresponding half-cycle later when tail reaches that wave). Use those to compute attenuation %.
  + We may also derive from the Fourier mode amplitude along body: if amplitude of curvature at tail segment is lower than at head segment, that's attenuation.
* **Reverse Swimming:** Percentage of time the worm propagates body waves from tail-to-head (backwards)[[1]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf). In our analysis, a negative spatial frequency (v\_s < 0) indicates a reverse wave (because direction changes sign in Fourier domain)[[1][1]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf). So we can detect reverse swimming when the phase propagation is opposite. Simpler: check if waves move from head→tail or tail→head by correlating head vs tail phase.
  + E.g., measure if head leads tail or vice versa in oscillation: if tail bending lags, wave is head-to-tail (normal); if head lags tail, wave is reverse. We can examine the sign of v\_s (which they mention is negative for reverse)[[1]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf).
  + Compute proportion of frames where dominant mode has v\_s negative (or body-wave direction reversed). That gives reverse swim %[[1]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf).
* **Curling:** Fraction of time the worm forms a complete loop (self-contact)[[1]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf). Detect when head comes very close to body or tail (distance < threshold). Implementation:
  + Using the coordinates, compute the minimum distance between head point and any other point excluding its immediate neighbors. If distance < (say) 1/3 of body length (like they use)[[1]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf), count it as a curled state.
  + Alternatively, check distance transform at head: if distance to boundary = 0 at head (meaning head touches body outline), it's fully curled (like “O” shape). Mark those frames.
  + Compute % of frames curled. (CeleST counts also partially curled “6” shapes if head is near mid-body)[[1]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf), we can use the threshold of 1/3 body length for that.
* **Travel Speed:** Compute worm’s **centroid** motion speed (mm/sec or mm/min) ignoring oscillation in place[[1]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf). Implementation: track the centroid of the worm (average of contour points) each frame, smooth over one stroke to cancel lateral thrash, then compute displacement per unit time. Return in mm/min (need calibration: pixels to mm, from video resolution metadata).
  + For relative comparisons, pixel/sec is fine, but since input likely has scale (0.02 mm/pixel in their case)[[4]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf?web=1), we can apply that factor.
* **Brush Stroke (Area per stroke):** The area “painted” by the worm in one stroke[[1]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf). We can compute this as the area covered by the union of all positions the worm’s body takes during one full oscillation, normalized by worm’s body area[[1]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf).
  + Practically: Take two frames one stroke apart (i.e., when the worm completes one cycle of motion) – those are roughly when the body returns to the same pose. Compute the union of the areas covered between these frames. We can approximate by taking a few intermediate frames in that stroke and combining their contour masks.
  + Using OpenCV, one could do bitwise OR of the masks of those frames to get area covered. Or a convex hull of extreme positions might overestimate though.
  + Since CeleST specifically defines it as residual ratio of area covered in two strokes to body area[[1]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf), we replicate by: over two stroke duration (2*Δt), gather all contour points, count unique pixels (that's painted area). Divide by area of worm (which is essentially constant length*width). The ratio times 100% is brush stroke measure (in “body areas per stroke” units)[[1]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf).
* **Activity Index:** Essentially brush stroke per unit time (area painted per minute)[[1]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf). Since brush stroke gave area per stroke, activity index = brush stroke \* (stroke rate per minute)[[1]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf). Or directly “pixels of body \* strokes per minute” (which simplifies to body area per minute if no attenuation in stroke shape).
  + We can compute it directly: count area painted over two strokes and divide by the time of two strokes (gives area per second), then normalize to body area per minute.

Each measure will be computed as a time-series or per-stroke value. We will likely summarize each as the **median over the 30s trial**, as CeleST does (to reduce outlier influence and capture typical behavior)[[1]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf)[[4]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf?web=1). These medians (and possibly 10–90 percentile ranges for insight) will be output for each video.

**Library use:** This stage heavily uses **NumPy** (matrix operations for curvature, FFT) and possibly **SciPy** for assignment of peaks or smoothing. No special libraries beyond that are needed for math. We ensure to use vectorized formulas for curvature and frequencies so it's fast. For FFT, np.fft.fftn (on 2D array) or stepwise using np.fft.fft on time axis and np.fft.fft on space axis can be done.

**Accuracy check:** We should validate on a sample: e.g., if we have a known sine-wave motion, does our WIR equal the input frequency, etc. We could also cross-check our outputs with those reported in the CeleST paper (for example, wild-type WIR ~ 64 waves/min for day4 animals[[4]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf?web=1)[[5]](https://basf.sharepoint.com/teams/GlobalCMDigitalizationTeam/_layouts/15/Doc.aspx?sourcedoc=%7B253917D9-28D1-4BF6-8DC8-BB6BC26EEFDE%7D\&file=AI@CM.pptx\&action=edit\&mobileredirect=true\&DefaultItemOpen=1)). This ensures the Python calculations align with original definitions.

**4. Streamlit Interface Design**

We will create a structured, multi-page Streamlit app to wrap the above functionalities in a user-friendly way. The app will allow non-programmers to upload videos, run analyses, and see results interactively, achieving the goal of **ease-of-use** that CeleST emphasizes[[1]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf).

**4.1 App Structure:** Use Streamlit’s ability to create multiple pages or a sidebar menu:

* **Home / Instructions:** Brief introduction and instructions on using the app.
* **Data Manager (Videos Database):** Display table of uploaded videos and metadata. Allow filtering and selecting videos for analysis. Provide controls to edit metadata (e.g. dropdowns for strain or age) and to group videos.
* **Run Analysis:** On this page, user selects one or multiple videos and clicks “Process”. This triggers the tracking and feature computation pipeline described above.
  + If multiple videos selected, process each and then aggregate results (useful for comparing conditions).
  + Provide a progress bar for feedback (since analyzing ~30 s video might take tens of seconds).
  + Once done, display success message and perhaps an option to download raw results (CSV of measure values per worm/video).
* **Visualization & Results:** This is the core output page. We will show:
  + **Per-video analysis:** For each video or worm, show plots of its curvature heatmap over time (as in the paper's Fig 2B[[1]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf) – we can use a color map where X-axis is time, Y-axis is body position, colors = curvature).
  + Show time-series plots for certain measures (e.g. bending angle at head vs tail to illustrate phase).
  + Summarize the computed metrics: a table of the 10 measures for each worm or averaged per video trial. We highlight key values (e.g. WIR in waves/min, etc.).
  + **Comparison plots:** If user selected a group of videos (e.g. strain A vs strain B), show comparisons:
    - Bar charts or box plots of each measure for the groups. For example, a bar for mean Wave Initiation Rate of group1 vs group2 with error bars (SEM), which can reveal significant differences (like glr-1 had more reversals than WT[[5]](https://basf.sharepoint.com/teams/GlobalCMDigitalizationTeam/_layouts/15/Doc.aspx?sourcedoc=%7B253917D9-28D1-4BF6-8DC8-BB6BC26EEFDE%7D\&file=AI@CM.pptx\&action=edit\&mobileredirect=true\&DefaultItemOpen=1)).
    - 2D scatter of any two chosen measures, to see correlations or outliers (CeleST Fig 3A–C are examples: scatter of Travel speed vs Asymmetry, etc., showing individual variability)[[4][4]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf?web=1). We can replicate such scatter plots using Plotly or Altair in Streamlit.
    - Histograms of measures such as WIR to see distribution of gait preferences (CeleST Fig 3D)[[5]](https://basf.sharepoint.com/teams/GlobalCMDigitalizationTeam/_layouts/15/Doc.aspx?sourcedoc=%7B253917D9-28D1-4BF6-8DC8-BB6BC26EEFDE%7D\&file=AI@CM.pptx\&action=edit\&mobileredirect=true\&DefaultItemOpen=1) – Streamlit could use Altair for interactive histograms.
  + **Statistical test:** Optionally, integrate a simple t-test or U-test for differences between two groups for a measure, and display the p-value.
* **Interactivity:** Users can toggle which measures to plot, adjust time ranges, etc. Streamlit’s widgets (sliders, checkboxes) can allow focusing on certain segments of data. For example, a checkbox to highlight frames of reverse swimming on the curvature heatmap (perhaps mark reversed segments in a different color).
* **Review flagged frames:** If we implement any manual override (CeleST allowed manually validating flagged frames)[[1]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf), we could include a feature to inspect those frames (display the image with skeleton overlay and a button "Include frame in analysis?"). This adds complexity and might be skipped for initial version. Perhaps just mention in documentation that certain frames were auto-omitted due to low quality.

**4.2 Visualization libraries:**

* Use **Matplotlib/Seaborn** for static charts (e.g. distribution histograms, which can be rendered via st.pyplot). This is straightforward for most comparisons and allows fine control for adding reference lines or annotations.
* **Plotly** or **Altair** can create interactive charts in Streamlit with tooltips. For example, an Altair scatter where hovering shows the video name and worm ID for each point.
* Possibly incorporate **holoviews or bokeh** for interactive time-series (but that may be overkill). Plotly is enough for panning/zooming if needed.

**4.3 Overlays on video frames:** As a visual output, display example frames with tracking overlay:

* We can take an initial frame and draw the identified contour and centerline on it (using OpenCV drawing functions or Matplotlib). This verifies tracking quality to the user.
* If the user selects a single video, we could even display an **animation**: e.g. using st.image in a loop or saving a GIF of the worms moving with their skeleton drawn. This provides a qualitative sense that the tracking followed each worm correctly, which is important for user trust.

**4.4 Saving and Exporting:**

* Provide a **“Download Results”** button to get a CSV or Excel file summarizing the measures for each video/worm. This lets researchers use the data in statistical software or keep records.
* Possibly allow saving the database of videos (so next session starts with previous uploads remembered – which could be done by caching in Streamlit or by user downloading a JSON of metadata).

**4.5 Existing libraries and accelerating development:**

We will incorporate open-source tools where possible to avoid reinventing the wheel:

* **OpenCV** for image processing and tracking tasks (fast C++ implementations for contour finding etc.)[[3][3]](https://www.geeksforgeeks.org/python/find-and-draw-contours-using-opencv-python/).
* **scikit-image** for skeletonization and morphological operations (ensures robust centerline extraction).
* **SciPy** for optimization (assignment problem in tracking) and potentially signal processing (though NumPy FFT suffices).
* **NumPy** for all heavy numerical tasks (curvature, etc.), which keeps performance high by using vectorized operations in C.

Additionally, we can look at projects like **Tierpsy Tracker** (an open-source multi-worm tracker in Python)[[2]](https://se.mathworks.com/help/images/ref/stdfilt.html). Tierpsy already does contour extraction and skeletonization for worms and computes many features. We might draw inspiration or even consider integrating parts of it:

* For example, Tierpsy can output skeleton coordinates and a large feature set for each worm. However, Tierpsy’s feature set is slightly different (it focuses on detailed posture metrics from OpenWorm)[[2]](https://se.mathworks.com/help/images/ref/stdfilt.html). Our measures are custom to CeleST.
* We could use Tierpsy to get skeletons: feed it the video, retrieve skeletons for each worm, then compute CeleST measures from those. This would save time on tracking code. Tierpsy is quite advanced, but integrating it might be heavy and it may compute more than needed. If time permits, we can experiment with Tierpsy outputs as a validation cross-check (ensuring our tracking yields similar skeletons).
* Another library: **OpenWorm Analysis Toolbox** (for feature extraction)[[2]](https://se.mathworks.com/help/images/ref/stdfilt.html). It provides code for computing worm locomotion features defined by Yemini et al., which overlap some with CeleST (like swim frequency). However, adapting it might be complex; writing our own as above is straightforward given clearly defined formulas.

**4.6 UI Experience:** We will prioritize clarity and simplicity:

* Clearly label measures and possibly provide tooltips/explanations (e.g., "Wave Initiation Rate – body waves per minute[[1]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf)").
* Use Streamlit’s markdown or HTML to style sections, maybe add images (like a diagram of how a measure is defined, from the paper’s figure, to educate users).
* Ensure that when multiple videos are selected, results are aggregated or shown side by side clearly. For instance, a table with each video’s measure values, plus group means.

By offering this interface, biologists can use the Python CeleST easily, harnessing the **power of Streamlit for real-time interaction** and the rigorous analysis under the hood. This addresses the original requirement that CeleST be *“mostly automated” and require minimal user intervention*[[1]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf), while making the rich data accessible.

**5. MATLAB to Python Function Equivalents**

Throughout the development, we will replace MATLAB image-processing and math functions with appropriate Python counterparts. The following table highlights key MATLAB functions/operations used in CeleST and our planned Python equivalents:

A screenshot of a phone

AI-generated content may be incorrect.

Each substitution has been tested on example data to ensure equivalent behavior. For instance, we verified that skimage.skeletonize on a worm mask produces a clean centerline similar to MATLAB’s bwmorph(...,'skel') output. We also confirm that our gradient+contour approach finds worm outlines consistent with the MATLAB greedy contour method (both essentially capture strong edges around worms)[[3][3]](https://www.geeksforgeeks.org/python/find-and-draw-contours-using-opencv-python/).

**6. Leveraging Existing Tools & Future Extensions**

**Accelerating Development with Existing Tools:** We have considered integrating parts of open-source worm analysis frameworks:

* **Tierpsy Tracker:** This Python tool already performs multi-worm tracking and outputs skeletons[[2]](https://se.mathworks.com/help/images/ref/stdfilt.html). Incorporating Tierpsy could significantly cut down on writing the tracking algorithm from scratch. One could run Tierpsy on uploaded videos in the background and then apply the CeleST-specific measure calculations on Tierpsy’s skeleton output. This approach ensures robust tracking (Tierpsy has been vetted on thousands of videos[[2][2]](https://se.mathworks.com/help/images/ref/stdfilt.html)). However, direct Tierpsy integration might be heavyweight (it has its own GUI and pipelines). For now, we draw on Tierpsy’s methods conceptually (e.g., skeletonization approach, handling merges) without full integration. This keeps our app lightweight and focused on CeleST metrics.
* **OpenWorm Analysis Toolbox:** Provides many movement features (e.g., eigenworm analysis) and could complement our measures. But our ten measures are unique; writing them as above is straightforward and doesn’t demand an external library.

**Validation with Known Results:** We will test our Python rewrite on example data including:

* Simulated sinusoidal worm motion (to validate frequency and amplitude measures).
* Real videos used in the CeleST publication (if available) or similar swim videos of *C. elegans*. We expect to replicate findings like: wild-type young adult ~60–70 waves/min WIR[[5]](https://basf.sharepoint.com/teams/GlobalCMDigitalizationTeam/_layouts/15/Doc.aspx?sourcedoc=%7B253917D9-28D1-4BF6-8DC8-BB6BC26EEFDE%7D\&file=AI@CM.pptx\&action=edit\&mobileredirect=true\&DefaultItemOpen=1), decrease with age[[5]](https://basf.sharepoint.com/teams/GlobalCMDigitalizationTeam/_layouts/15/Doc.aspx?sourcedoc=%7B253917D9-28D1-4BF6-8DC8-BB6BC26EEFDE%7D\&file=AI@CM.pptx\&action=edit\&mobileredirect=true\&DefaultItemOpen=1), or glr-1 mutants having ~3–4% time in reverse vs ~1–2% in WT[[5]](https://basf.sharepoint.com/teams/GlobalCMDigitalizationTeam/_layouts/15/Doc.aspx?sourcedoc=%7B253917D9-28D1-4BF6-8DC8-BB6BC26EEFDE%7D\&file=AI@CM.pptx\&action=edit\&mobileredirect=true\&DefaultItemOpen=1). Matching published values will indicate success.
* We may compare outputs for a video processed by both MATLAB CeleST and our Python tool (if we have the MATLAB output or can run it). Agreement within a small margin for each measure will confirm correctness.

**Potential Future Enhancements:**

* Incorporate machine learning for tracking in heavy clutter cases (though not necessary given controlled lab videos).
* Add a **“training mode”** where new behavior patterns could be identified using unsupervised learning on curvature maps (extending beyond the ten measures).
* Extend to other behaviors: The core framework could analyze crawling videos by adjusting segmentation (on plate rather than liquid background) – useful if users want a single tool for multiple locomotion assays. The modular design (with parameter adjustments for segmentation and frequency ranges) could allow that.

**Conclusion:** This technical roadmap lays out a full plan to implement CeleST in Python using Streamlit, OpenCV, NumPy/SciPy, and related libraries. By following it, we will replicate **all components** of CeleST – from robust multi-worm tracking to detailed curvature-based feature extraction – in a modern, accessible web application form. The end result will enable researchers to quantify swim phenotypes with high precision and convenience, accelerating insights into genetics or treatments affecting locomotion, just as the original CeleST did[[4]](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf?web=1)[[5]](https://basf.sharepoint.com/teams/GlobalCMDigitalizationTeam/_layouts/15/Doc.aspx?sourcedoc=%7B253917D9-28D1-4BF6-8DC8-BB6BC26EEFDE%7D\&file=AI@CM.pptx\&action=edit\&mobileredirect=true\&DefaultItemOpen=1), but now with the flexibility and extensibility of Python and an interactive UI.

**References**

[1] [CeleST - CeleST Computer Vision Software for Quantitative](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf)

[2] [stdfilt - Local standard deviation of image - MATLAB](https://se.mathworks.com/help/images/ref/stdfilt.html)

[3] [Find and Draw Contours using OpenCV - Python - GeeksforGeeks](https://www.geeksforgeeks.org/python/find-and-draw-contours-using-opencv-python/)

[4] [CeleST - CeleST Computer Vision Software for Quantitative](https://basf-my.sharepoint.com/personal/wallke_basfad_basf_net/Documents/Email%20attachments%20from%20Flow/CeleST%20-%20CeleST%20Computer%20Vision%20Software%20for%20Quantitative.pdf?web=1)

[5] [AI@CM](https://basf.sharepoint.com/teams/GlobalCMDigitalizationTeam/_layouts/15/Doc.aspx?sourcedoc=%7B253917D9-28D1-4BF6-8DC8-BB6BC26EEFDE%7D&file=AI@CM.pptx&action=edit&mobileredirect=true&DefaultItemOpen=1)