`BundleTracker` Software

Joseph M. Marcinik[1]

[1]*Department of Physics & Astronomy, University of California, Los Angeles, California, 90095, USA*

(Dated: June 6, 2024)

We have developed intuitive software to track small-scale displacements. The software includes both graphical and script-based interfaces. It simultaneously tracks regions of interest (ROIs) from a grayscale video. It processes ROIs independently in three primary stages: preprocessing, tracking, and postprocessing. First, it preprocesses pixel intensities in the video, preparing them for tracking. Next, it tracks ROIs selected by the user, calculating their displacements over time. Finally, it postprocesses these traces, optionally rotating and/or detrending them. While processing, it propagates error estimates, and it exports sufficient metadata to reproduce the analysis. To demonstrate, we used the software to track a collection of five hair bundles within minutes.

I. INTRODUCTION

Optical tracking is used in many fields. It outputs the displacement of an object over time, relative to the first frame of a video. Many different objects have been tracked optically, including particles [1–3], glaciers [4–16], organs [17], tendons [18], hair cells [19–23], and mice whiskers [24–30]. This makes optical tracking a prominent technique to measure displacements.

However, optical tracking requires performing heavy computations. Optical datasets consist of videos, so they contain abundant information. All of this information must be processed, which demands sufficiently fast algorithms.

Furthermore, optical tracking requires inputting numerous parameters. The user must cater their analysis to their specific biological system (e.g. hair cells in our demo). The user requires system-specific processing parameters, such as those used to reduce noise. Only after inputting these parameters can the user track their system.

To alleviate these issues, we created software to track small-scale displacements effectively. This software tracks any object that remains roughly within a predetermined fixed region of interest. It estimates displacements efficiently, particularly when processing batches of predetermined ROIs.

We implemented both a graphical user interface (GUI) and its concomitant application programming interface (API). By having a GUI, our software is transparent and accessible. By having an API, our software can be fully automated by importing previous sessions. With a GUI and API, our software is both accessible and automatable.

To start, the user imports a video. The software supports grayscale videos, not RGB videos. The software supports 8-bit and 16-bit MJ2 (Motion JPEG 2000) as well as 8-bit AVI (Audio Video Interleave).

After importing a video, the user draws ROIs on an image. They can draw four different shapes (i.e. rectangle, ellipse, polygon, and freehand). After drawing, the user can adjust its size and position.

After drawing ROIs, the user commands the software to process them. The software preprocesses, tracks, and postprocesses each ROI independently. Before tracking, it preprocesses pixel intensities. To track, it calculates the displacement for each frame. To postprocess, it optionally rotates and/or detrends these displacements. While processing, it propagates error throughout all steps, unless otherwise noted. Importantly, it saves sufficient metadata to reproduce and import tracking sessions in the future.

II. DESIGN AND IMPLEMENTATION

We implemented three feature-rich GUIs for the following tasks: 1) tracking subpixel (or few-pixel) displacements, 2) detecting blobs automatically, and 3) thresholding ROIs automatically. The primary GUI tracks batches of oscillating objects such as collections of hair bundles. The latter two GUIs aid the user by automatically detecting object locations and determining preprocessing parameters, respectively. Altogether, these three GUIs provide an intuitive and precise means to track objects quickly.

The primary GUI consists of three major parts [??? reference screenshot]. These include 1) the global editor, 2) the region editor, and 3) the action (menu and tool) bars and toolbar. By partitioning the primary GUI, the user more easily intuits its functions.

In the global editor, the user adjusts desired ROIs. To draw ROIs, the user has three options. First, they can manually draw them by clicking and dragging their mouse. Next, they can automatically insert them by using our blob-detection GUI [??? reference blob-detection GUI]. This GUI allows the user to quickly add numerous ROIs into the global editor. Finally, they can manually duplicate them in the context menu of an existing ROI. The user quickly and intuitively adds new ROIs into the global editor.

Also in the global editor, the user sets default parameters. These parameters include methods to process ROIs, such as which tracking algorithm to use. They also include parameter values to process ROIs with, such as thresholds for pixel intensities. When adding a new ROI,

the software copies default parameter values into the new ROI.

In the region editor, the user tweaks parameters for each ROI. The user independently sets parameter for each ROI. These include all parameters shown in the global editor. With this freedom, the user caters parameter values precisely for each ROI. Further, they can compare processing algorithms and preprocessing inputs by duplicating a single ROI. With the region editor, the user tracks every ROI independently.

With the action bars, the user performs various useful actions described later in this section. We list only a few notable actions. In the menu bar, the user imports videos and/or ROIs. They open the blob-detection and image-thresholding GUIs. In the toolbar, they choose from four shape tools to draw ROIs, one as general as freehand. Most importantly, they command the software to start tracking (or they can use the optional keyboard shortcut shown in table II). After tracking, they can import their results into a waterfall plot. By incorporating these action bars, the software includes many convenient operations.

To speed up processing time, we implemented automated (though optional) blob-detection and image-thresholding GUIs. By automating blob detection, the user quickly and precisely generates multiple ROIs in the global editor. By automating image thresholding, the user quickly and accurately determines preprocessing parameters for all ROIs. Used in conjunction, these two automated algorithms can save hours during a single tracking session.

To encourage future expansion, we wrote modular and object-oriented code. This makes our code readily extendable. Developers can add new processing algorithms or modify existing ones. In our scripts, they can tweak parameters to their specific needs, with more control than the GUI elicits.

A. Mathematical Notation

We introduce useful notation to communicate our algorithms. We define the 3D matrix $\mathbf{I}$ to represent pixel intensities for the grayscale video. In this matrix, the first two indices correspond to position, and the third index corresponds to time. We define $\mathbf{X} = \begin{pmatrix} \mathbf{x} \\ \mathbf{y} \end{pmatrix}$ to represent trace positions. To represent its position at frame k, we subscript $\mathbf{X}$ as $\mathbf{X}_k \in \mathbb{R}^2$. We let $\mathbf{R}_\theta$ denote the rotation matrix, which rotates counter-clockwise by θ.

We introduce notation for common operations and functions. To denote expected value and standard deviation of matrix $\mathbf{M}$ taken over axes x_i, we use $\mathbb{E}_{x_i}[\mathbf{M}]$ and $\sigma_{x_i}[\mathbf{M}]$, respectively. When omitting the axes, these operations are taken over the entire matrix.

B. Preprocessing

We implemented two preprocessing steps, both optional. The software filters then inverts the intensity. After each preprocessing steps, the intensity remains normalized.

The software filters the range of pixel intensities. The user filters by selecting lower and upper bounds I_{min} and I_{max}, respectively. To filter the image, the software maps

$$I \to \begin{cases} 0, & I \le I_{min} \\ 1, & I \ge I_{max} \\ \dfrac{I - I_{min}}{I_{max} - I_{min}}, & \text{otherwise} \end{cases} \tag{1}$$

for all $I \in \mathbf{I}$. Notice that $\mathbf{I} \to \mathbf{I}$ if $I_{min} = 0$ and $I_{max} = 1$. This makes filtering effectively optional.

The software inverts the pixel intensities, optionally. The user chooses whether to invert the image. If the user decides to invert, then the software inverts the image. To invert the image, the software maps $I \to 1 - I$ for all $I \in \mathbf{I}$. Otherwise, the software does nothing. This makes inverting optional.

The software displays pixel intensity for the first preprocessed frame. We created a three-part colormap to show pixel intensities: one color for undersaturated pixels, one color for oversaturated pixels, and a standard colormap for the remaining pixels. With these contrasting colors, the user can clearly see under- and over-saturated pixels. The software applies this colormap to the already-preprocessed frame, changing the display in real time as the user changes preprocessing parameters.

C. Tracking

After preprocessing ROIs, the software tracks their position. At the moment, we implemented three tracking algorithms, described in this section. The software outputs a 2D vector of positions $\mathbf{X}$.

The software finds displacement by calculating a *Centroid*. For each frame, it sets centroid weights proportional to pixel intensity. It estimates error based on the deviation in pixel intensity. It propagates this error when calculating the centroid. The centroid center and concomitant error represent the position for the ROI.

The softwares generates an error map $\delta I : [0, 1] \to \left[0, 1/\sqrt{2}\right]$. Here $\delta I(I)$ represents error for intensity $I \in \mathbf{I}$. Employing least-squares regression, the software calculates the two best-fit parameters in the power law $y = Ax^m$, where $y \in \sigma_t[\mathbf{I}]$ and $x \in \mathbb{E}_t[\mathbf{I}]$. Finally, it estimates $\delta I(I) \approx AI^m$ and maps $I \to I \pm \delta I(I)$.

The software calculates a centroid for each frame. For the centroid $\mathbf{X}_k$, the software applies weights $w_{ijk} \propto I_{ijk}$ normalized over each frame k. The software uses this $\mathbf{X}_k$ as the position at frame k.

The software finds displacement by calculating a *2D Gaussian* fit. It fits intensities in frame k with respect

to pixel positions. Employing the Levenberg-Marquardt algorithm, it calculates seven best-fit parameters in a rotated 2D Gaussian. These parameters include 1-2) center (x_0, y_0), 3) angle θ, and 4-5) axes (r_x, r_y) of the ellipse, along with 6) offset C and 7) amplitude A of the Gaussian. Employing nonlinear least-squares estimates, the software calculates a 95% confidence interval, using half this range as the error for each parameter. The software uses $\mathbf{X}_k = (x_0, y_0)$ as the position at frame k. We set initial conditions shown in table I.

The software finds displacement by calculating a *Maximum Cross Correlation*. The software assumes null displacement for frame 1. Then for each frame $k \geq 2$, it finds the displacement $\mathbf{X}_k$ at which intensities in frame k experience their maximum cross-correlation relative to those in frame 1. The software uses $\mathbf{X}_k$ as the position at frame k. Before calculating cross-correlation, the software upsamples each frame $100\times$ [31]. By upsampling, the software produces error as low as $1/100^{\text{th}}$ of a pixel.

D. Postprocessing

We implemented five postprocessing steps, four optional. The software orients, shifts, scales, rotates, and detrends the traces. With these steps, the software produces publication-ready traces.

The software orients traces. To orient a trace, the software reflects and rotates it. To reflect a trace, the software maps $\mathbf{X}$ such that $\mathbf{y}$ points $\pi/2$ counter-clockwise from $\mathbf{x}$. Now $\mathbf{x}$ points in the +x-direction, while $\mathbf{y}$ points in the +y-direction as seen in the image. The user rotates by selecting some angle θ_0 from the eight cardinal directions (counter-clockwise from the +x-direction). To rotate a trace, the software maps $\mathbf{X} \to \mathbf{R}_{\theta_0}^\top \mathbf{X}$. Notice that $\mathbf{X} \to \mathbf{X}$ if $\theta_0 = 0$. This makes orienting effectively optional.

The software shifts traces. To shift a trace, it maps $\mathbf{X} \to \mathbf{X} - \mathbb{E}_t[\mathbf{X}]$. Shifting is not optional.

The software scales traces. The user scales by selecting a scale factor A. To scale a trace, the software maps $\mathbf{X} \to A\mathbf{X}$. Notice that $\mathbf{X} \to \mathbf{X}$ if $A = 1 \pm 0$. This makes scaling effectively optional.

The software rotates traces. The user chooses whether to rotate a trace by selecting one of five algorithms in section II D 1. If the user decides to rotate, then the software maps $\mathbf{X} \to \mathbf{R}_{\theta^*}^\top \mathbf{X}$ using θ^* calculated from the chosen algorithm. The software further maps $\mathbf{X} \to \mathbf{R}_{\frac{\pi}{2}}^\top \mathbf{X}$ if $\sigma_t[\mathbf{x}] < \sigma_t[\mathbf{y}]$, placing the larger-amplitude component in the x-direction. If the user decides not to rotate, the software does nothing. This makes rotating optional.

The software detrends traces. The user chooses whether to detrend by selecting one of three algorithms in section II D 2. If they decide to detrend, then the software detrends each direction independently using the chosen algorithm. Otherwise, the software does nothing. This makes detrending optional.

1. Rotating

For most rotation algorithms, the software calculates errors in the same way. It uses 4-fold cross-validation subsets of $\mathbf{X}$ to calculate of set of angles $\boldsymbol{\theta}^*$. As the error for θ^*, the software uses $\delta\theta^* = \sigma[\boldsymbol{\theta}^*]$. Unless otherwise noted, it uses this method to calculate angular error.

The software rotates by *Minimum Covariance*. It finds

$$\theta^* = \underset{\theta \in \left[-\frac{\pi}{4}, \frac{\pi}{4}\right]}{\arg\min} \left| \left(\mathbf{R}_\theta^\top \operatorname{Cov}[\mathbf{x}, \mathbf{y}] \mathbf{R}_\theta \right)_{12} \right|. \tag{2}$$

where $\operatorname{Cov}[\mathbf{x}, \mathbf{y}]$ represents the 2×2 covariance matrix between $\mathbf{x}$ and $\mathbf{y}$. Conceptually, it finds the nearest angle that minimizes the absolute covariance between $\mathbf{x}$ and $\mathbf{y}$.

The software rotates by *2-Means Clustering* or *Two Gaussian Mixtures*. It determines two disjoint clusters of points in $\mathbf{X}$ using either k-means clustering (with $k = 2$) or Gaussian mixtures (with two mixtures), respectively. Let (x_1, y_1) and (x_2, y_2) represent the centroids of these two clusters such that $x_1 < x_2$. The software finds $\theta^* = \arctan\left(\frac{y_2 - y_1}{x_2 - x_1}\right)$. Conceptually, the software rotates the trace such that its cluster's centroids both fall along the x-axis.

The software rotates by *Linear Regression*. Employing least-squares linear regression, it calculates the two best-fit parameters in the line $y = mx + b$, where $y \in \mathbf{y}$ and $x \in \mathbf{x}$. For the error on m, the software uses its standard error from linear regression. Finally, it finds $\theta^* = \arctan(m)$. Conceptually, the software rotates the trace such that its best-fit line falls along the x-axis.

The software rotates by *Elliptical Regression*. It performs a singular value decomposition $\mathbf{X} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^\top$. Here, each column in $\boldsymbol{\Sigma}$ corresponds to one term in the two-variable quadratic equation, i.e. $\boldsymbol{\Sigma}$ looks similar to $\begin{pmatrix} x^2 & xy & y^2 & x & y & 1 \end{pmatrix}$. Then, the software uses the values in the most-null vector from $\mathbf{V}$ as the six best-fit parameters in the quadratic equation $Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0$. Finally, it finds

$$\theta^* = \begin{cases} 0, & B = 0 \text{ and } A < C \\ \frac{\pi}{2}, & B = 0 \text{ and } A > C \\ \frac{1}{2} \operatorname{arccot}\left(\frac{C-A}{B}\right), & B \neq 0 \text{ and } A < C \\ \frac{\pi}{2} + \frac{1}{2} \operatorname{arccot}\left(\frac{C-A}{B}\right), & B \neq 0 \text{ and } A > C \end{cases} \tag{3}$$

Conceptually, the software rotates the trace such that the major axis of its best-fit ellipse falls along the x-axis. It does not propagate error during elliptical regression.

2. Detrending

The software detrends from a selection of three algorithms. As its first two algorithms, it subtracts a high-degree polynomial or a moving average. As its third algorithm, it subtracts both of these in series, polynomial first (label as *Auto* in the GUI). Alternatively, the user

Parameter	Value	Lower	Upper
C	$\underset{I\in\mathbf{I}_k}{\text{med}}\, I$	0	∞
A	1	0	∞
x_0	$\underset{x\in\mathbf{P}_x}{\text{med}}\, x$	$\underset{x\in\mathbf{P}_x}{\min}\, x-1$	$\underset{x\in\mathbf{P}_x}{\max}\, x+1$
y_0	$\underset{y\in\mathbf{P}_y}{\text{med}}\, y$	$\underset{y\in\mathbf{P}_y}{\min}\, y-1$	$\underset{y\in\mathbf{P}_y}{\max}\, y+1$
r_x	1	0	∞
r_y	1	0	∞
θ	$\frac{\pi}{4}$	0	π

TABLE I. Parameters fit in the rotated 2D Gaussian. From left to right, the columns represent 1) parameter symbol along with its corresponding 2) initial condition and 3-4) lower then upper bounds. The 3D matrix $\mathbf{P}$ represents the x- and y-position at each pixel.

can choose not to detrend. The software does not propagate error during detrending.

The software detrends using *Polynomial Regression*. Let P_N represent the best-fit polynomial with degree N for $\mathbf{x}$ with respect to times $\mathbf{t}$. Let $x(t)$ represent position from $\mathbf{x}$ at time $t \in \mathbf{t}$. We define detrended position at time t as

$$\Delta x_N(t) \coloneqq x(t) - P_N(t) \tag{4}$$

and range function

$$R(\mathbf{x}) \coloneqq \max_{x\in\mathbf{x}} x - \min_{x\in\mathbf{x}} x. \tag{5}$$

Then the software finds the least $N^* \geq 0$ such that

$$\frac{R(\Delta x_{N^*+1}(\mathbf{t}))}{R(\Delta x_{N^*}(\mathbf{t}))} > 1 + p_\delta \tag{6}$$

for some $p_\delta \in \mathbb{R}$. Conceptually, the software increments N by one, fitting each polynomial P_N, and stops right before overfitting. It finds the lowest-degree polynomial P_{N^*} that squeezes $\Delta x_N(\mathbf{t})$ into a "small" range. Finally, it maps $\mathbf{x} \to \Delta x_{N^*}(\mathbf{t})$. We set $p_\delta = 0.01$ arbitrarily after some testing with our system. The user can tweak this value in the `Detrender` class.

The software detrends using a *Moving Average*. Let $F_{\mathcal{F}}(x)$ denote the cumulative distribution function of $\mathcal{F}[\mathbf{x}]$, where $\mathcal{F}$ denotes the Fourier transform. The software calculates a Hann-window moving average $\boldsymbol{\mu}_\mathbf{x}$ over $\mathbf{x}$ with width $NF_{\mathcal{F}}^{-1}(F^*)$ for some $N > 0$ and $F^* \in (0, 1)$. Conceptually, this estimates a "wide"-window moving average. Finally, it maps $\mathbf{x} \to \mathbf{x} - \boldsymbol{\mu}_\mathbf{x}$. We set $N = 2$ and $F^* = 0.95$ arbitrarily, erring to preserve $\mathbf{X}$. We recommend these or similar values when detrending in series with a polynomial fit. The user can tweak these values in the `Detrender` class.

E. Automatic Calculations

The software propagates error through nearly all of its processing. Exceptions, where it does not propagate error, are mentioned where relevant.

We make some conventional assumptions to propagate error. To start, we assume that errors are *small* and *normally-distributed*. Next, we assume that errors are *independent*, except during unary operations. For example, the product $(x \pm \delta x)(y \pm \delta y)$ assumes independent errors for δx and δy, but the power $(x \pm \delta x)^n$ assumes 100% correlation for δx with itself. With these three common assumptions, the software can propagate error quickly and accurately.

To implement error propagation, we created a class object `ErrorPropagator`. This class overrides all elementary operations in MATLAB, including matrix multiplication. It also overrides many common functions such as trigonometric and statistical functions. By default, it applies the min-max method to propagate error for function f. For other select functions, it applies a linear approximation. [??? make table with functions along with error method]

The user can propagate error through any scalar function, even ones not implemented explicitly in our code. To propagate error in a matrix $\mathbf{M}$ through an arbitrary single-argument function f, the user calls `ErrorPropagator.scalarFunction(M, @f)`. The function `scalarFunction` employs the min-max method to calculate error.

We implemented blob detection using the built-in `vision.BlobAnalysis` class. Blob-detecting algorithms identify clusters of bright (or dim) pixels in an image. To detect blobs, the software uses only the first video frame [??? show screenshot of Blob Detection GUI]. After detecting, the software draws these blobs as ROIs into the global editor. Automatic blob detection provides an efficient way to add many ROIs into the global editor.

Before detecting blobs, the software preprocesses the image. In order, it smooths, normalizes, and binarizes the image. To smooth the image, it subtracts a 2D moving mean using the built-in `smoothdata2` function. This diminishes local fluctuations in intensity. To normalize the image, it maps all $I \in \mathbf{I}$ similar to eq. (1) with $I_{min} = \min_{I\in\mathbf{I}} I$ and $I_{max} = \max_{I\in\mathbf{I}} I$. After preprocessing, the software can detect blobs in the image.

To detect blobs, the software binarizes the image. The user selects lower and upper bounds I_{min} and I_{max}, re-

spectively. To binarize the image, the software maps

$$I_{ij} \to \begin{cases} 1, & I_{min} \le I_{ij} \le I_{max} \\ 0, & \text{otherwise} \end{cases}$$

for all $I_{ij} \in \mathbf{I}$. With this binarized image, the software deems bright pixel clusters as blobs.

The user can tweak many blob-detection parameters. These include connectivity (4-way or 8-way), maximum number of blobs, range of blob areas, shape of blobs (rectangle or ellipse), height/width of blob shape, and inclusion of border blobs [??? reference blob-detection GUI]. With these parameters, the user can cater the detected blobs to their system.

We implemented image thresholding using a user-selected algorithm. Image-thresholding algorithms identify a characteristic intensity I^* to segment an image. To calculate I^*, the software uses only the first video frame [??? show screenshot of Thresholding GUI]. The software interpolates from I^* to I_{min} or I_{max} with the piecewise linear map

$$I_{min/max} = \begin{cases} \left(I^* - \min_{I \in \mathbf{I}} I\right) x & 0 \le x \le 1 \\ \left(\max_{I \in \mathbf{I}} I - I^*\right)(x-1) & 1 \le x \le 2 \end{cases} \tag{7}$$

where I^* represents the characteristic intensity of $\mathbf{I}$, and $I_{min/max}$ represents either I_{min} and $_{max}$. After calculating I_{min} and I_{max}, the software applies them to their corresponding ROIs in the global editor as in eq. (1). Automatic image thresholding provides an efficient way to set I_{min} and I_{max} in the global editor.

The user can tweak a few image-thresholding parameters. These include 1) an x for each of I_{min} and I_{max} and 2) an algorithm to calculate I^* [??? references to thresholding survey papers, add table]. With these parameters, the user can cater the characteristic intensities to their system.

F. Productive Shortcuts

We implemented numerous actions to work easily with ROIs. These include moving, sizing, ordering, and duplicating ROIs. Moving a ROI translates it along one of the four primary cardinal directions. Sizing a ROI either compresses or expands its size. Ordering a ROI either 1) brings it forward/backward relative to other ROIs or 2) sets its neighboring ROI as active. Duplicating a ROI creates a copy of it, including its selected algorithms and preprocessing parameters. We implemented keyboard shortcuts for all of these actions, many mimicking their corresponding actions on layers in Adobe Photoshop (shown in table II).

We implemented a minimal but very useful API. With this API, the user can run sections of our software through script. These actions include 1) importing new videos or previous sessions, 2) exporting an image from the global editor, and 3) tracking and exporting ROIs. The user can loop through and process multiple datasets. To fully automate tracking, the user must first track and process one dataset; then they import and track ROIs from this session in other videos. When processing multiple videos, the API can command the software to track all of them (except the first one) unattended.

III. RESULTS

Using our software, we find displacements for five hair bundles. First, we incorporate our blob detection and image thresholding features. Then we track all five bundles, further automatically rotating and detrending each trace. Finally, we display our results in an interactive waterfall plot.

We analyzed data of hair-bundle motion. We obtained hair cells from the inner ear of an American Bullfrog (*Rana catesbeiana*). We recorded a few hair cells, under a bright-field microscope, at $108.3(8)\,\mathrm{nm\,px^{-1}}$ and 1000 FPS. We obtained a lossless MJ2 video containing one second of 8-bit grayscale frames, each with size $256\,\mathrm{px} \times 256\,\mathrm{px}$. For all frames, we clearly see five full hair bundles. To demonstrated our software, we tracked these five bright-field ROIs.

We added ROIs using our blob-detection GUI. We excluded border blobs because they included 1) partial bundles in the bottom-left corner and 2) saturated pixels around the horizontal edges. We chose the rectangle shape with height and width both set to 32 px. We increased the relative pixel intensity to about 60%, where the blob detector identified the five hair bundles exclusively. We applied these five blobs as ROIs into our global editor.

We thresholded ROIs using our automatic-threshold GUI. We chose Otsu's method to calculate a characteristic threshold. We set $x = 1$ as in eq. (7). This effectively removed pixels with intensity below the characteristic threshold.

We changed the positive direction. We chose direction to start the left for all ROIs. We chose to rotate by the minimum-covariance method. These two choices, at least for hair bundles, roughly rotate the final traces into their directions of maximal movement.

We chose our final tracking parameters. As our tracking method, we chose the centroid method. For our system, all three methods provide similar results, but the centroid method computes fastest. As our detrending method, we chose the auto method (i.e. polynomial fit then moving average in series). For only one-second of data, our analysis does not warrant detrending. However, we detrend to follow the recommended tracking procedure for hair bundles. After choosing tracking and detrending methods, we finalized our tracking parameters.

We did not further tweak any parameters. To make the demo more easily reproducible, we chose not to man-

Command	Action
↑	Move active ROI one pixel upward
↓	Move active ROI one pixel down
←	Move active ROI one pixel leftward
→	Move active ROI one pixel rightward
Ctrl+Space	Compress all sides of active ROI one pixel inward
Ctrl+↑	Compress bottom side of active ROI one pixel upward
Ctrl+↓	Compress top side of active ROI one pixel downward
Ctrl+←	Compress right side of active ROI one pixel leftward
Ctrl+→	Compress left side of active ROI one pixel rightward
Ctrl+Shift+Space	Expand all sides of active ROI one pixel outward
Ctrl+Shift+↑	Expand top side of active ROI one pixel upward
Ctrl+Shift+↓	Expand bottom side of active ROI one pixel downward
Ctrl+Shift+←	Expand left side of active ROI one pixel leftward
Ctrl+Shift+→	Expand right side of active ROI one pixel rightward
Ctrl+Del	Remove active ROI
Ctrl+J	Duplicate active ROI
Ctrl+]	Send active ROI forward one layer
Ctrl+[	Send active ROI backward one layer
Ctrl+Shift+]	Send active ROI to front layer
Ctrl+Shift+[	Send active ROI to back layer
Alt+]	Set next ROI as active
Alt+[	Set previous ROI as active
Ctrl+Enter	Start tracking ROIs
Ctrl+S	Export image of global editor
Ctrl+R	Import ROIs from previous session
Ctrl+I	Import video from file
Ctrl+O	Open directory containing imported video

TABLE II. Shortcuts implemented in the primary GUI.

ually tweak any ROIs. However, the user can, in general, adjust parameters for each ROI independently.

We calculated traces for five ROIs. For each ROI, we calculated the centroids (with error) for each of 1000 frames with size $32\,\mathrm{px} \times 32\,\mathrm{px} = 1024\,\mathrm{px}$, producing 5000 centroids in total. Immediately after tracking, the software independently rotated and detrended each ROI. To calculate these centroids, a i9-13900K CPU took only a few seconds. We tracked 5000 positions efficiently.

We plotted five traces on an interactive waterfall plot [??? reference figure with waterfall]. The user can click traces on this plot to view them in more detail [??? include preview plot]. The traces match those from prior hair-cell literature [??? reference some prominent hair cell papers/reviews].

IV. AVAILABILITY AND FUTURE DIRECTIONS

We deposited our software at https://github.com/jmarcinik3/BundleTracker. For full features, the software requires MATLAB 2023b with the following toolboxes: Computer Vision, Curve Fitting, Econometrics, Image Processing, Optimization, Signal Processing, Statistic and Machine Learning.

To enhance the user's experience, we would like to make settings importable. In the menu bar, the user could access a settings window, allowing them to tweak and import/export settings. In the settings menu, the user could easily change color aesthetics (e.g. colormap) and default parameters (e.g. tracking method). Over time, the user could potentially tweak every aspect of the GUI to their preference.

Developers can add new algorithms to the code. For a trace, these include tracking, rotating, and detrending. For an image, algorithms include preprocessing and thresholding. Developers can introduce new processing steps, such as smoothing images when preprocessing or relocating ROIs when tracking. They can add new plots and analysis, perhaps the least explored territory of the software. Essentially, they can include their desired processing into our code.

[1] J. K. Sveen, An introduction to MatPIV v. 1.6.1, (2004).

[2] W. Thielicke and E. J. Stamhuis, PIVlab – Towards User-friendly, Affordable and Accurate Digital Particle Image Velocimetry in MATLAB, Journal of Open Research Software **2**, 10.5334/jors.bl (2014).

[3] R. Boltyanskiy, J. W. Merrill, and E. R. Dufresne, Tracking particles with large displacements using energy minimization, Soft Matter **13**, 2201 (2017).

[4] T. A. Scambos, M. J. Dutkiewicz, J. C. Wilson, and R. A. Bindschadler, Application of image cross-correlation to the measurement of glacier velocity using satellite image data, Remote Sensing of Environment **42**, 177 (1992).

[5] A. Kääb and M. Vollmer, Surface Geometry, Thickness Changes and Flow Fields on Creeping Mountain Permafrost: Automatic Extraction by Digital Image Analysis, Permafrost and Periglacial Processes **11**, 315 (2000).

[6] S. Leprince, F. Ayoub, Y. Klinger, and J.-P. Avouac, Co-Registration of Optically Sensed Images and Correlation (COSI-Corr): An operational methodology for ground deformation measurements, in *2007 IEEE International Geoscience and Remote Sensing Symposium* (2007) pp. 1943–1946.

[7] A. Messerli and A. Grinsted, Image georectification and feature tracking toolbox: ImGRAFT, Geoscientific Instrumentation, Methods and Data Systems **4**, 23 (2015).

[8] M. Fahnestock, T. Scambos, T. Moon, A. Gardner, T. Haran, and M. Klinger, Rapid large-area mapping of ice flow using Landsat 8, Remote Sensing of Environment **185**, 84 (2016).

[9] B. M. Minchew, M. Simons, B. Riel, and P. Milillo, Tidally induced variations in vertical and horizontal motion on Rutford Ice Stream, West Antarctica, inferred from remotely sensed observations, Journal of Geophysical Research: Earth Surface **122**, 167 (2017).

[10] E. Schwalbe and H.-G. Maas, The determination of high-resolution spatio-temporal glacier motion fields from time-lapse sequences, Earth Surface Dynamics **5**, 861 (2017).

[11] T. Nagy, L. M. Andreassen, R. A. Duller, and P. J. Gonzalez, SenDiT: The Sentinel-2 Displacement Toolbox with Application to Glacier Surface Velocities, Remote Sensing **11**, 1151 (2019).

[12] D. Shean, Dshean/vmap: Zenodo DOI release, Zenodo (2019).

[13] W. Zheng, W. J. Durkin, A. K. Melkonian, and M. E. Pritchard, Cryosphere And Remote Sensing Toolkit (CARST) v1.0.1, Zenodo (2019).

[14] P. How, N. R. J. Hulton, L. Buie, and D. I. Benn, PyTrx: A Python-Based Monoscopic Terrestrial Photogrammetry Toolset for Glaciology, Front. Earth Sci. **8**, 10.3389/feart.2020.00021 (2020).

[15] Y. Lei, A. Gardner, and P. Agram, Autonomous Repeat Image Feature Tracking (autoRIFT) and Its Application for Tracking Ice Displacement, Remote Sensing **13**, 749 (2021).

[16] M. Van Wyk de Vries and A. D. Wickert, Glacier Image Velocimetry: An open-source toolbox for easy and rapid calculation of high-resolution glacier velocity fields, The Cryosphere **15**, 2115 (2021).

[17] G. Pinton, J. Dahl, and G. Trahey, Rapid tracking of small displacements with ultrasound, IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control **53**, 1103 (2006).

[18] B.-I. Chuang, J.-H. Hsu, L.-C. Kuo, I.-M. Jou, F.-C. Su, and Y.-N. Sun, Tendon-motion tracking in an ultrasound image sequence using optical-flow-based block matching, BioMedical Engineering OnLine **16**, 47 (2017).

[19] A. J. Ricci, A. C. Crawford, and R. Fettiplace, Active hair bundle motion linked to fast transducer adaptation in auditory hair cells, J. Neurosci. **20**, 7131 (2000).

[20] P. Martin, D. Bozovic, Y. Choe, and A. J. Hudspeth, Spontaneous oscillation by hair bundles of the bullfrog's sacculus, J. Neurosci. **23**, 4533 (2003).

[21] D. Ramunno-Johnson, C. E. Strimbu, L. Fredrickson, K. Arisaka, and D. Bozovic, Distribution of frequencies of spontaneous oscillations in hair cells of the bullfrog sacculus, Biophysical Journal **96**, 1159 (2009).

[22] M. Beurg, J. H. Nam, Q. Chen, and R. Fettiplace, Calcium balance and mechanotransduction in rat cochlear hair cells, J. Neurophysiol. **104**, 18 (2010).

[23] A. Chaiyasitdhi, *Morphology Control of the Hair-Cell Bundle for Frequency-Selective Auditory Detection*, Ph.D. thesis, Université Paris sciences et lettres (2021).

[24] P. M. Knutsen, D. Derdikman, and E. Ahissar, Tracking whisker and head movements in unrestrained behaving rodents, J Neurophysiol **93**, 2294 (2005).

[25] J. T. Ritt, M. L. Andermann, and C. I. Moore, Embodied information processing: Vibrissa mechanics and texture features shape micromotions in actively sensing rats, Neuron **57**, 599 (2008).

[26] J. Voigts, B. Sakmann, and T. Celikel, Unsupervised whisker tracking in unrestrained behaving animals, J Neurophysiol **100**, 504 (2008).

[27] S. Venkatraman, K. Elkabany, J. D. Long, Y. Yao, and J. M. Carmena ˆ*, A System for Neural Recording and Closed-Loop Intracortical Microstimulation in Awake Rodents, IEEE Transactions on Biomedical Engineering **56**, 15 (2009).

[28] I. Perkon, A. Kosir, P. M. Itskov, J. Tasic, and M. E. Diamond, Unsupervised quantification of whisking and head movement in freely moving rodents, J Neurophysiol **105**, 1950 (2011).

[29] N. G. Clack, D. H. O'Connor, D. Huber, L. Petreanu, A. Hires, S. Peron, K. Svoboda, and E. W. Myers, Automated Tracking of Whiskers in Videos of Head Fixed Rodents, PLOS Computational Biology **8**, e1002591 (2012).

[30] J.-H. L. F. Betting, V. Romano, Z. Al-Ars, L. W. J. Bosman, C. Strydis, and C. I. D. Zeeuw, WhiskEras: A New Algorithm for Accurate Whisker Tracking, Frontiers in Cellular Neuroscience **14**, 10.3389/fncel.2020.588445 (2020).

[31] M. Guizar-Sicairos, S. T. Thurman, and J. R. Fienup, Efficient subpixel image registration algorithms, Opt. Lett., OL **33**, 156 (2008).