`BundleTracker`: A Graphical User Interface to Track Oscillating Objects Optically

Joseph M. Marcinik[1]

***For correspondence:**
jmarcinik@physics.ucla.edu

[1]Department of Physics & Astronomy, University of California, Los Angeles, United States

Abstract Tracking subpixel movements from microscopic objects remains a fundamental challenge in many scientific fields. To meet this challenge, we developed `BundleTracker`, an intuitive, open-source software package designed to track small-scale displacements of oscillatory objects, featuring both graphical and script-based interfaces. The software streamlines optical tracking by simultaneously examining multiple objects through three processing stages: video preprocessing, displacement tracking, and trace postprocessing. The software propagates error estimates and exports comprehensive metadata, ensuring robust and reproducible results. We demonstrate the versatility and precision of `BundleTracker` by tracking oscillatory motion in two biophysical systems and one synthetic system.

Introduction

Optical tracking remains a versatile and valuable technique for monitoring object displacements in various scientific fields. Prominently, it quantifies and visualizes fluid flow by tracking individual particles (***Sveen, 2004***; ***Thielicke and Stamhuis, 2014***; ***Boltyanskiy et al., 2017***), useful for analyzing physiological ultrasounds (***Pinton et al., 2006***; ***Todaro et al., 2012***; ***Chuang et al., 2017***) and welding mechanics (***Eriksson et al., 2013***). In geoscience, it monitors glacier movement and land deformation (***Lei et al., 2021***; ***Van Wyk de Vries and Wickert, 2021***), useful for climate studies and hazard assessments (***Van Wyk de Vries et al., 2022***). Thus, it can effectively measure motion over multiple orders of length scales, ranging from small microscopic particles to large geological formations.

Optical tracking has become more effective in light of technological advancements, including computer-vision algorithms and high-speed cameras. These advancements enable researchers to track in increasingly more challenging environments over time, with varying lighting conditions and complex object geometries. As a result, optical tracking remains indispensable for quantifying displacement and understanding dynamics across diverse scientific disciplines.

In biological systems, optical tracking is often applied when measuring displacements at microscopic scales, ranging from subnanometers to micrometers. These applications include *Caenorhabditis elegans* embryos and NEMO (NF-κB essential modulator) fibroblasts. Each system may benefit from specialized tracking algorithms or tailored processing methods, complicating efforts to develop a universal approach. Taking these issues into account, we designed tracking software that integrates multiple processing algorithms, while ensuring that new ones can be seamlessly integrated.

Some systems, particularly those involving oscillating objects, require subpixel precision when tracking displacements. Existing software packages like TrackMate (***Tinevez et al., 2017***; ***Ershov***

et al., 2022), anTraX (*Gal et al., 2020*), and WhiskEras (*Betting et al., 2020*; *Arvanitis et al., 2022*), generally track multipixel displacements effectively. These packages, however, may struggle to track subpixel displacements, as required for colloidal particle suspensions (*Crocker and Grier, 1996*) and for oscillating biological structures like hair cells (*Ricci et al., 2000*; *Martin et al., 2003*; *Fredrickson et al., 2008*; *Ramunno-Johnson et al., 2009*; *Beurg et al., 2010*; *Chaiyasitdhi, 2021*) and mosquito flagella.

Hence, we developed user-friendly software specifically for tracking subpixel displacements, implemented in MATLAB. Equipped with a graphical user interface (GUI) as well as an application programming interface (API), the software enables the user to efficiently select regions of interest (ROIs) and define processing parameters. For each ROI, the software outputs a trace with high numerical precision, limited primarily by experimental error and noise. To demonstrate the utility of the software, we tracked nanometer-scale oscillations in three optical systems.

The software batch-processes ROIs, tracking them efficiently and independently according to user-selected processing methods. It imports and reproduces previous results by saving comprehensive metadata during each tracking session. Additionally, it tracks objects in series automatically, significantly reducing the time the user typically spends rerunning or modifying code.

In this paper, we outline the capabilities of our software, from customizable processing features to tangible experimental results. In Materials and Methods, we outline the processing capabilities of this software, including preprocessing videos, tracking displacements, and postprocessing traces. In Results, we demonstrate this software through three distinct experiments: two measured biophysical systems and one forced glass-probe system. In Discussion, we discuss potential future improvements of this software along with its general applicability.

Materials and Methods

Graphical User Interfaces

We implemented user-friendly GUIs to facilitate the following tasks: 1) track subpixel displacements, 2) detect blobs as ROIs, and 3) threshold pixel intensities. The first GUI tracks batches of oscillating objects, such as collections of hair bundles. The latter two GUIs automatically detect object locations and determine preprocessing parameters, respectively. Altogether, these interfaces offer an intuitive workflow to efficiently track object displacements over time.

The `BundleTracker` GUI consists of three major components, as illustrated in ***Figure 1***. These include the global editor, region editor, and action bars (menu bar and toolbar). By partitioning the interface, the user can easily intuit where to execute their desired actions.

The global editor allows the user to manage ROIs, offering three options to create them. The user can manually draw ROIs by clicking and dragging the mouse, automatically generate them using the blob-detection GUI (see ***Figure 2***), or conveniently duplicate existing ROIs within their respective context menus. Additionally, the global editor allows the user to set default processing methods and parameter values, such as tracking algorithms or default orientation. These defaults are automatically applied to new ROIs, simplifying the setup process and saving time.

The region editor provides fine-tuned control over individual ROIs, enabling the user to adjust parameters for each ROI independently. This feature allows the user not only to customize parameter inputs precisely for each ROI, but also to compare processing inputs for one ROI. For example, the user can duplicate an ROI to test different settings, ensuring optimal tracking performance.

The action bars allow the user to perform various essential tasks. Through the menu bar, the user can 1) import videos and ROIs, 2) launch the blob-detection and image-thresholding GUIs, and 3) access interactive data visualizations. The toolbar provides four tools to select the shape for the ROIs, including a versatile freehand option. Most importantly, the toolbar contains a button to initiate the tracking process (equivalently done using the keyword shortcut shown in ***Table 2***). After tracking, the user can visualize their results using a variety of GUIs, such as the interactive waterfall plot shown in ***Figure 4***. These action bars, by consolidating key functions, allow the user

to execute tasks efficiently within the software.

To speed up processing time, we implemented optional automated GUIs for blob detection (see ***Figure 2***) and image thresholding (see ***Figure 3***). The blob-detection GUI allows the user to generate multiple ROIs quickly in the global editor. Similarly, the image-thresholding GUI allows the user to determine characteristic intensities simultaneously for all ROIs. When used together, these two automated tools significantly reduce the time spent in a single tracking session, potentially saving hours of effort.

Tracking Optical Datasets

We implemented three optional preprocessing steps, applied to each frame independently. These steps involve smoothing, filtering, and inverting pixel intensities. After each step, the software normalizes the pixel intensities within each frame of the recording. For detailed mathematical descriptions on these preprocessing techniques, refer to Preprocessing Frames.

Following the preprocessing step, the software tracks displacements of ROIs using one of three available tracking algorithms. These algorithms, which output a two-dimensional (2D) vector of positions, include calculating a centroid, fitting a 2D Gaussian, and maximizing cross-correlation. For detailed mathematical descriptions on these tracking algorithms, refer to Tracking ROIs.

After tracking, the software postprocesses traces through five operations. These steps include orienting, shifting, scaling, rotating, and detrending the traces. All of these operations, except shifting, remain optional. For detailed mathematical descriptions on these postprocessing methods, refer to Postprocessing Traces.

Automatic Calculations and Scripts

The software propagates error throughout most of its processing stages, relying on a few standard assumptions. First, it assumes errors are *small* and *normally distributed*. Second, it assumes that errors are *independent*, except during unary operations. For example, the errors δx and δy are treated as independent in the product $(x \pm \delta x)(y \pm \delta y)$. However, the error δx is treated as fully correlated with itself in the power $(x \pm \delta x)^n$. With these widely accepted assumptions, software propagates error efficiently.

The software includes automated blob detection and image thresholding. It offers blob detection using the built-in `vision.BlobAnalysis` class, providing an efficient way to add many ROIs into the global editor. It offers image thresholding from select algorithms, providing an efficient way to determine image thresholds for each ROI. For mathematical descriptions of these blob-detecting and image-thresholding algorithms, see Blob Detection and Image Thresholding, respectively.

Additionally, the software includes a simple yet powerful API, allowing the user to execute certain operations dynamically via scripts. With this API, the user can import new videos or previous sessions, export a preview image to visualize ROIs, and track (and export) ROIs. These actions enable the user to track multiple videos unattended. To fully automate tracking, the user must first track and export results from one dataset, which they can then import to inform the software how to process subsequent videos.

Results

Batch Tracking with Hair Bundles

Using our software, we tracked displacements for five hair bundles, integrating both the blob-detection and image-thresholding interfaces. We analyzed this biophysical system primarily to demonstrate the batch-tracking capabilities of the software. After defining and processing ROIs, we tracked all five bundles, automatically rotating and detrending each trace. To present these results, we displayed the five traces using two interactive visualizations included in the software.

We captured hair-bundle motion from hair cells within the inner ear of an American Bullfrog (*Rana catesbeiana*). Using a bright-field microscope, we recorded their movement at a resolution of

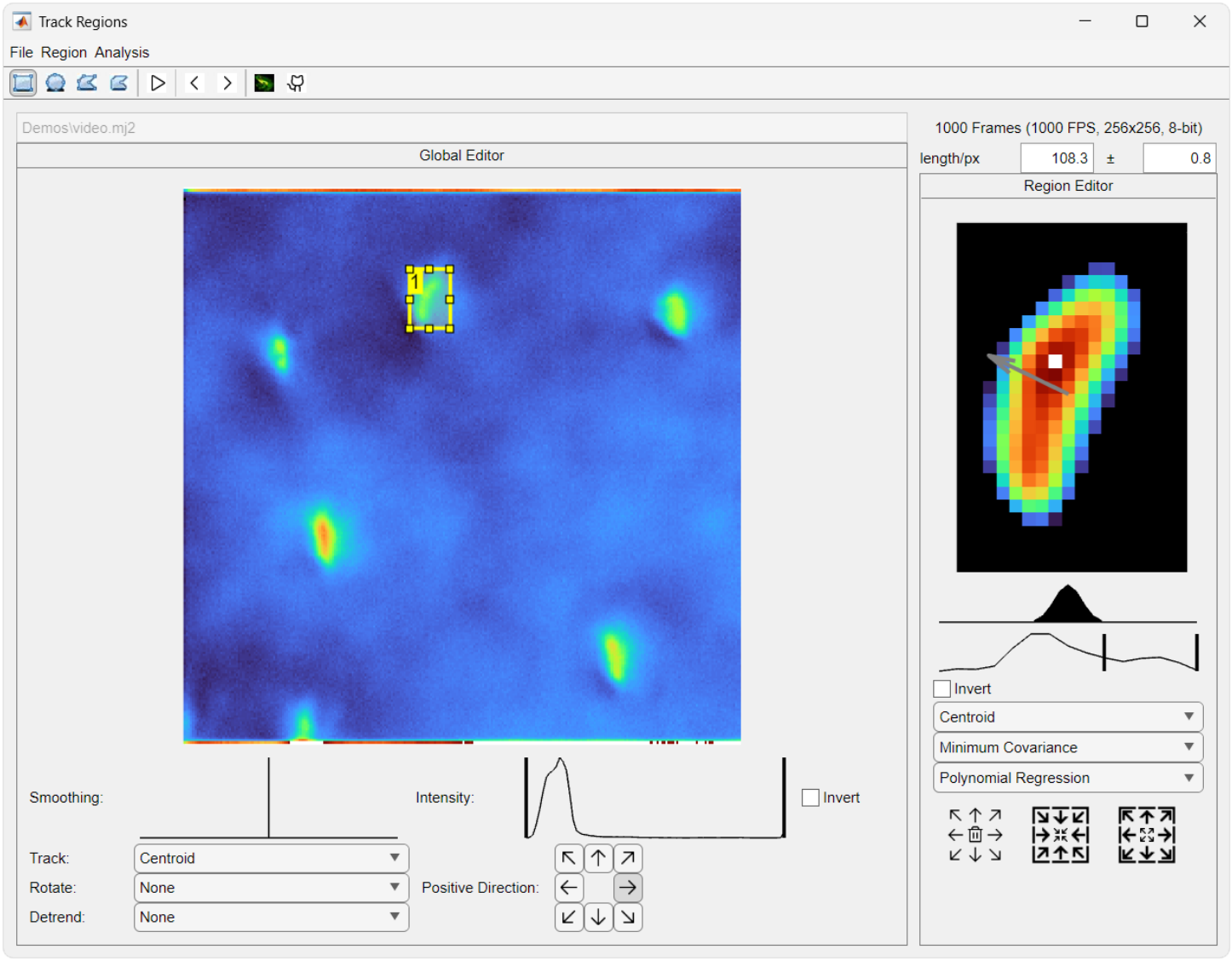

Figure 1. Screenshot of the `BundleTracker` interface, displaying a tracking session for a collection of five hair bundles from Batch Tracking with Hair Bundles. This interface consists of the action bars (top), global editor (left), and region editor (right). With the action bars, the user can execute various actions, including importing videos and starting tracking. In the global editor, the user draws and selects regions of interest. In the region editor, the user adjusts processing parameters for the selected region.

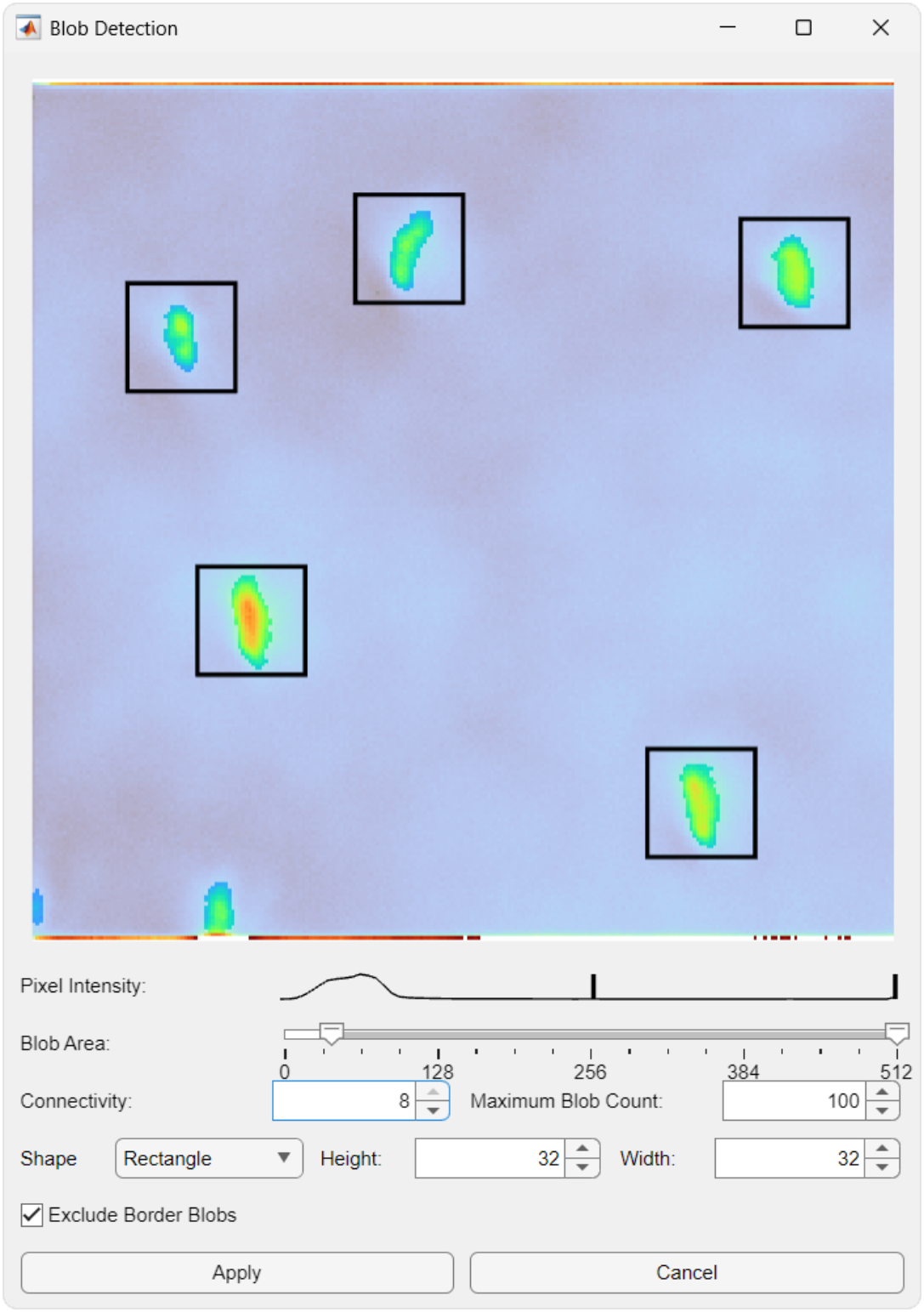

Figure 2. Screenshot of the blob-detection interface, displaying a detecting session for a collection of five hair bundles from Batch Tracking with Hair Bundles. This interface consists of a preview section (top) and controls section (bottom). In the preview section, the user can view the identified blobs along with their concomitant ROIs, $32\,\text{px} \times 32\,\text{px}$ squares in this demonstration. In the preview image, bright pixels appear saturated relative to dim pixels. With the controls, the user can adjust various parameters, refining the blob-detection algorithm to identify specific regions.

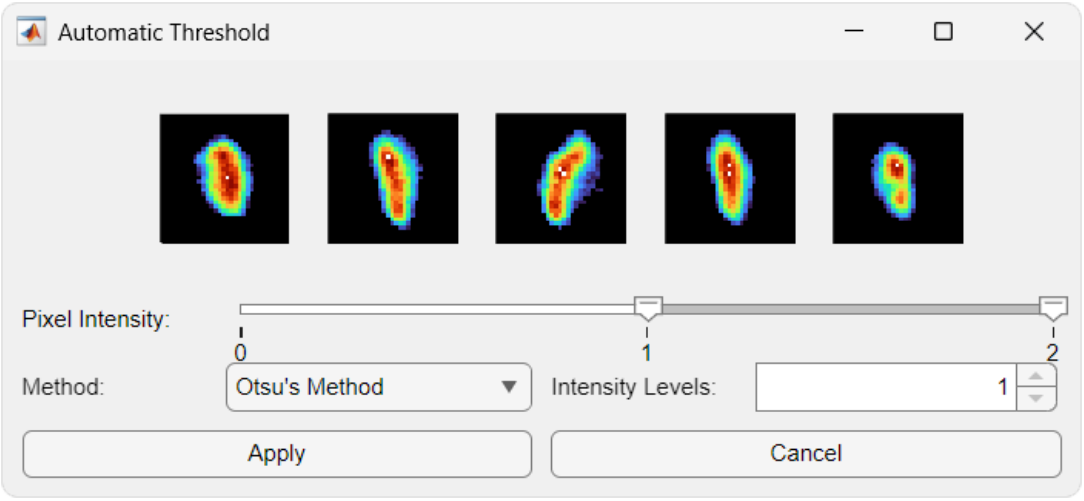

Figure 3. Screenshot of the image-thresholding interface, displaying a thresholding session for a collection of five hair bundles from Batch Tracking with Hair Bundles. This interface consists of a preview section (top) and a controls section (bottom). In the preview section, the user can view the thresholded image for each ROI. With the controls, the user can adjust pixel intensity and thresholding method, refining the image-thresholding algorithm to emphasize specific features.

$108.3(8)\,\mathrm{nm\,px^{-1}}$ and a frame rate of $1000\,\mathrm{FPS}$. We converted this video to a lossless MJ2 containing one second of 8-bit grayscale frames, each sized $256\,\mathrm{px} \times 256\,\mathrm{px}$. Throughout the video, we tracked five fully visible bundles.

We added ROIs via our blob-detection GUI. We excluded bundles that were only partially visible, such as those positioned near a corner within the field of view and those that included saturated pixels along the horizontal edges. By setting the lower-intensity to about 60%, the blob detector identified exclusively the five full bundles. After sizing these blobs as $32\,\mathrm{px} \times 32\,\mathrm{px}$ rectangles, we applied them as ROIs into our global editor.

After defining ROIs, we filtered dim pixels through our automatic-threshold GUI. To calculate a characteristic lower threshold for each ROI, we applied Otsu's method, with $x = 1$ as described in ***Equation 2***. This process optimized the dynamic contrast between the hair cells and their background.

We oriented and rotated the direction of the cells, aligning them roughly along their axis of maximal movement. After orienting their initial positive directions leftward, we rotated them automatically using the "Minimum Covariance" method.

We finalized our tracking parameters by selecting the "Centroid" method. While all three tracking methods yielded comparable results, the "Centroid" method had the fastest computing speed. To detrend, we chose the "Auto" method, which subtracts a polynomial fit and moving average. While detrending is not strictly necessary for only one second long recording of hair-bundle movement, we included it to follow the standard procedures in the field. To keep this demonstration easy to reproduce, we chose not to tweak any other parameters. However, the user can fine-tune the settings for each ROI independently if desired.

We calculated centroids with concomitant error estimates for all five ROIs. The software then rotated and detrended each trace automatically. Using an i9-13900K CPU, the software tracked and processed all 5000 of these centroids in a few seconds. This resulted in five hair-bundle traces, each consisting of 1000 displacements.

We plotted these traces using two interactive interfaces. The first interface, a waterfall plot (illustrated in ***Figure 4***), allows the user to zoom in on individual traces by clicking on them. These traces closely matched those from prior hair-cell studies (***Martin et al., 2003***; ***Ramunno-Johnson et al., 2009***; ***Roongthumskul et al., 2011***; ***Meenderink et al., 2015***). The second interface (shown in

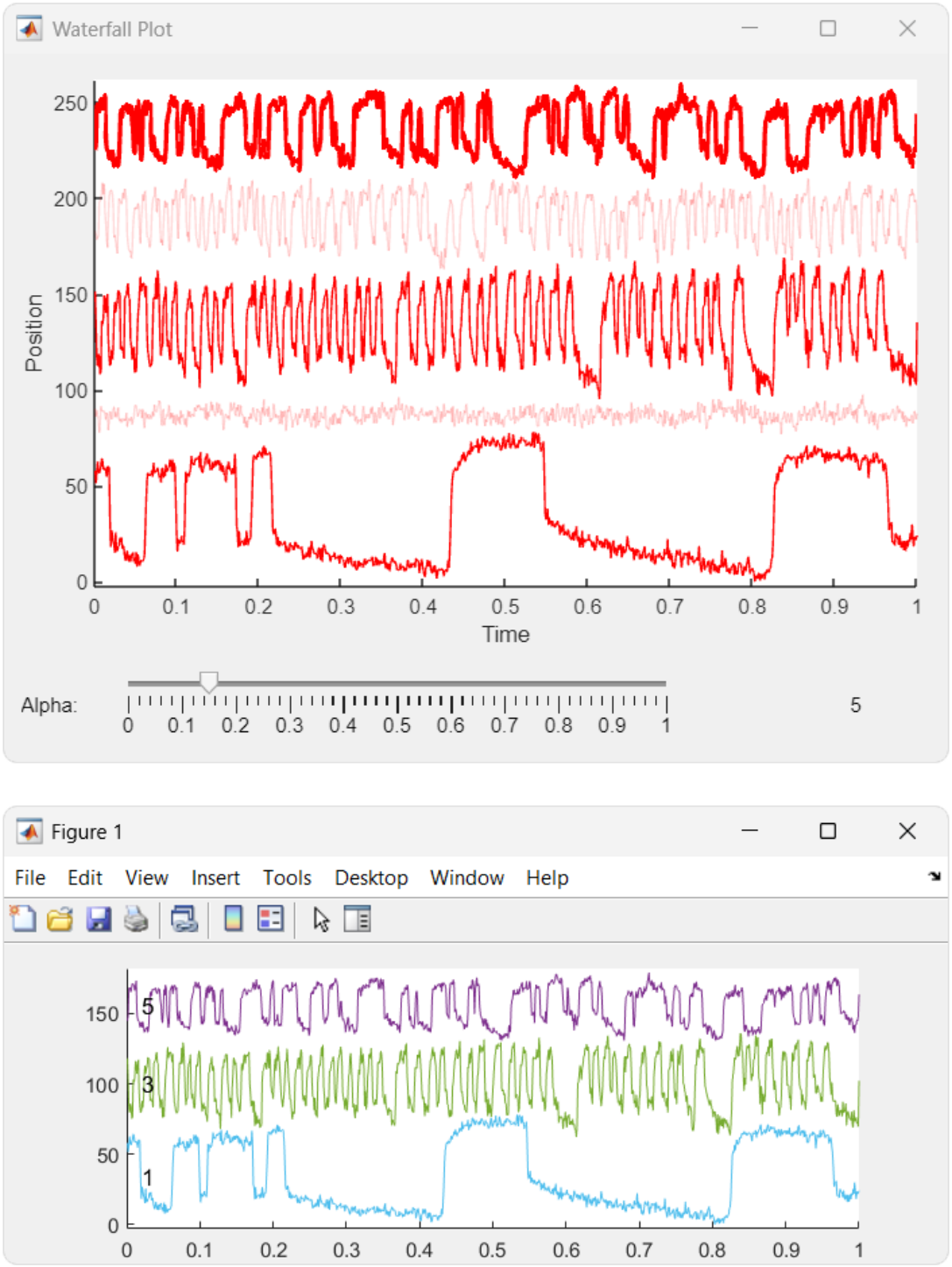

Figure 4. Screenshot of the waterfall interface, displaying traces for a collection of five hair bundles from Batch Tracking with Hair Bundles. This interface consists of two windows. The top window displays all traces from the selected tracking session. These traces have implicit indices starting from 1 for the bottommost trace, incrementing upward. The bottom window previews a subset of traces, selected manually by clicking on traces in the top window. The bottom window also displays corresponding indices for the selected traces.

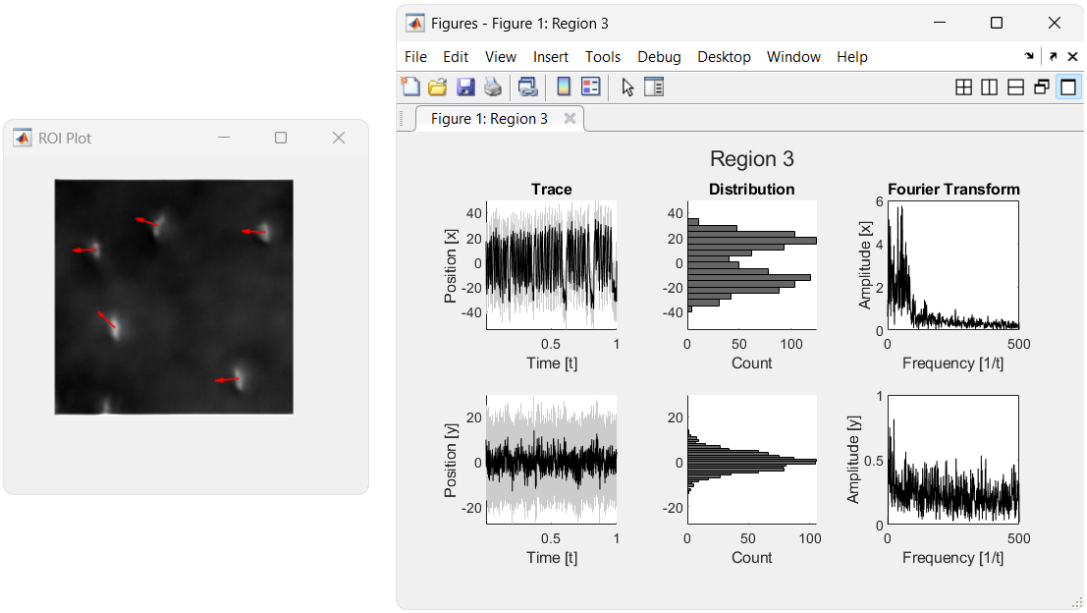

Figure 5. Screenshot of the analysis GUI, displaying a collection of five hair bundles from Batch Tracking with Hair Bundles. This interface consists of two windows. The left window displays all tracked bundles, with their resultant direction indicated by the arrows. The right window displays three plots for each component of the selected bundle (top and center in this demonstration), selected manually by clicking on bundles in the left window. From left to right, these plots show displacement over time, distribution of displacements, and Fourier transform over frequency.

Figure 5) enables the user to click on an ROI, opening a pop-up window that displays displacement over time, distribution of displacements, and Fourier transform. Together, these interfaces offer a fast and user-friendly way to conduct preliminary analyses on the tracked objects.

Two-Dimensional Tracking with a Mosquito Flagellum

Using our software, we tracked displacements of one flagellum on a southern house mosquito (*Culex quinquefasciatus*). We analyzed this biophysical system primarily to demonstrate the 2D-tracking capabilities of the software. We began by measuring its motion under a bright-field microscope, recording at a resolution of $416(12)\,\mathrm{nm\,px^{-1}}$ and a frame rate of $2000\,\mathrm{FPS}$. We converted this video to a lossless MJ2 containing over three seconds of 16-bit grayscale frames, each sized $512\,\mathrm{px} \times 512\,\mathrm{px}$. We created one ROI by manually drawing a $241\,\mathrm{px} \times 227\,\mathrm{px}$ rectangle around the tip of the flagellum.

We performed three preprocessing steps on the flagellum. First, we smoothed each frame with a $6\,\mathrm{px}$ Gaussian kernel. Next, we filtered out bright background pixels using an upper-intensity threshold of approximately 20%. To finish preprocessing, we inverted the pixel intensities in each image, further enhancing the contrast between the flagellum and its background.

We tracked displacements using the "Maximum Cross-Correlation" method. We then oriented the initial positive direction as rightward and rotated the trace with the "Elliptical Regression" method, aligning the elliptical motion along its major axis. To complete the postprocessing, we detrended the trace using the "High-Pass Filter" method.

Finally, we visualized the results by plotting the 2D motion as a joint probability density of displacements. The entire process, including computationally intensive smoothing, took only about two minutes. The resultant joint distribution (shown in ***Figure 6***) showed the full 2D trajectory of the flagellum, demonstrating that the software can be used to track complex spatial trajectories, as well as oscillatory motion in one direction.

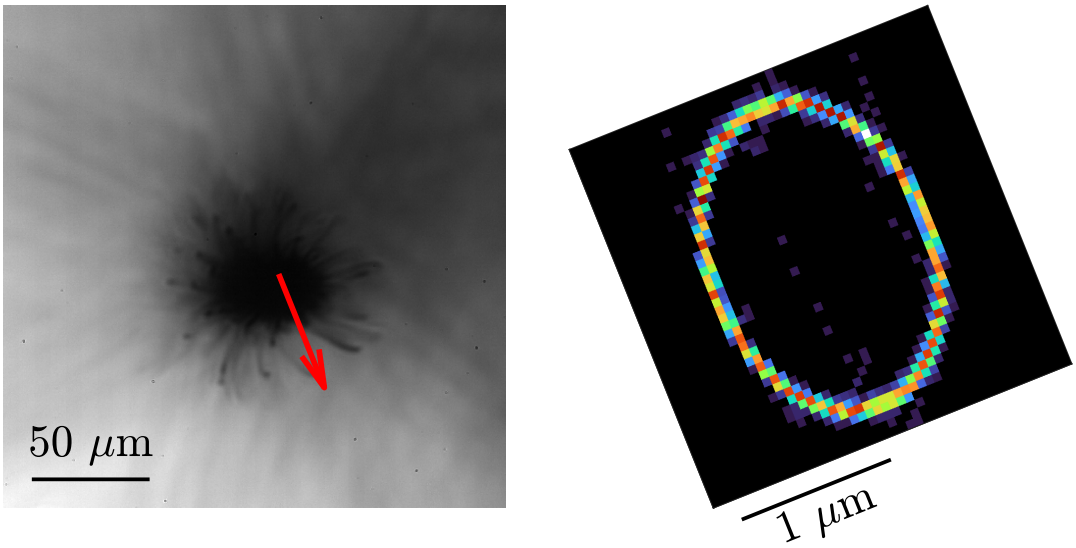

Figure 6. Joint distribution of displacements for a mosquito flagellum, as described in Two-Dimensional Tracking with a Mosquito Flagellum. The grayscale image (left) displays a screenshot of a mosquito flagellum. The red arrow points toward the $+x$-direction of the resultant trace, with the $+y$-direction oriented $90\,\mathrm{deg}$ counter-clockwise from this $+x$-direction. The heatmap (right) displays a joint probability density of displacements for this flagellum.

Precision Tracking with a Forced Probe

Using our software, we tracked the displacement of a glass probe stimulated by a $10\,\mathrm{Hz}$ sinusoidal signal. We analyzed this synthetic system primarily to demonstrate the subpixel precision of the software. We drove the probe at six different amplitudes (peak-to-peak) for voltage via a piezoelectric device, i.e. inclusively from $100\,\mathrm{mV}$ to $600\,\mathrm{mV}$ in $100\,\mathrm{mV}$ steps (***Peng and Ricci, 2016***). While applying these forces, we recorded the motion of the probe under a bright-field microscope, at a resolution of $82.9(7)\,\mathrm{nm\,px^{-1}}$ and a frame rate of $4000\,\mathrm{FPS}$. We converted this video to a lossless MJ2 containing over $1\,\mathrm{min}$ of 16-bit grayscale frames, each sized $128\,\mathrm{px} \times 128\,\mathrm{px}$. We defined one ROI by manually drawing a $35\,\mathrm{px} \times 30\,\mathrm{px}$ rectangle around the tip of the probe.

We performed three preprocessing steps on the probe. First, we smoothed each frame with a $11\,\mathrm{px}$ Gaussian kernel. Next, we filtered out bright background pixels with an upper-intensity threshold of roughly 43%. Finally, we inverted pixel intensities in each image, further increasing the contrast between the probe and its background.

We tracked displacements using the "Centroid" method. We oriented the positive direction as rightward, matching the direction of the stimulus. As the final postprocessing step, we detrended the trace using the "Polynomial Regression" method.

To evaluate the precision of the tracking software, we calculated the frequency of the probe. The entire process took less than $5\,\mathrm{min}$, with most of the computational time spent preprocessing. The probe oscillated with an amplitude around $30\,\mathrm{nm}$, increasing with the voltage supplied to the piezoelectric device. We computed the power spectral densities (PSDs) of the probe at each stimulated voltage, using the Bartlett method (***Bartlett, 1950***) with window durations of roughly $1\,\mathrm{s}$. Across all six voltages, the PSD consistently revealed a peak at $10.00(1)\,\mathrm{Hz}$, successfully recovering the original frequency of the stimulus, as illustrated in ***Figure 7***.

Discussion

`BundleTracker` allows the user to track optical datasets efficiently. After inputting a grayscale video, the user defines fixed ROIs and adjusts processing parameters. Once the user finalizes these ROIs, the software calculates a trace for each ROI independently, saving sufficient metadata to ensure

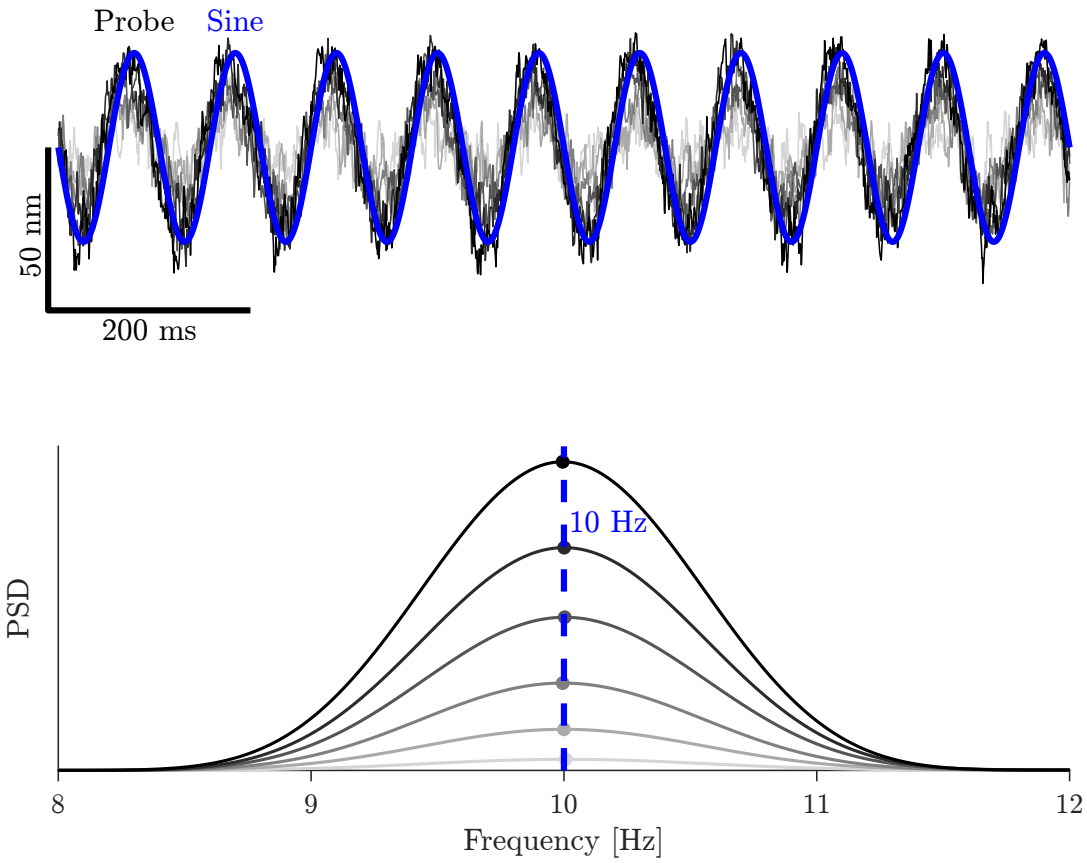

Figure 7. Trace and power spectral density (PSD) of displacements for stimulated glass probe from Precision Tracking with a Forced Probe. Each grayscale color corresponds to one of six stimulus amplitudes from 100 mV to 600 mV in 10 mV steps. Top: The grayscale lines display displacement traces of the glass probe during a 10 Hz stimulus. The blue line indicates the ground-truth phase from the 10 Hz sinusoidal signal. Bottom: The solid lines display PSDs corresponding to the grayscale traces. Along each line, the marker shows the frequency and power at peak power for the corresponding PSD. The vertical blue line indicates 10 Hz, the ground-truth frequency of the stimulus.

that the results remain reproducible.

The software predominantly tracks objects that remain within a fixed ROI. For example, it effectively tracks cells undergoing small-scale oscillations and any objects experiencing gradual drift. However, it does not precisely track dense collections of distinct objects. For example, it does not suitably track compact crowds of moving people (***Silveira Jacques Junior et al., 2010***) nor intersecting paths of particle collisions (***Pecora, 2018***; ***Gal et al., 2020***; ***Berghouse et al., 2024***).

This software excels at tracking hair cells in the auditory system. These sensory cells enable animals to process critical information about sounds in their environment. As a key component of the auditory process, hair cells transduce mechanical deflections from sound waves into electrical signals, oscillating to reliably detect subnanometer movements. Over the past few decades, hearing research has advanced significantly (***Hudspeth, 2008***, ***2014***), focusing on the molecular components inside the inner ear (***Martin et al., 2003***). As single specimens can house thousands of hair cells (***Thorne and Gavin, 1984***; ***Wright et al., 1987***), researchers often record batches of these cells within a single video. Moreover, they frequently record multiple experiments in the same region, varying the cellular conditions around the cells to capture their diverse behaviors (***Strimbu et al., 2012***; ***Meenderink et al., 2015***; ***Lin and Bozovic, 2020***). Processing these extensive datasets manually can be both time-consuming and labor-intensive, but the batch-tracking features in the software streamline the process, reducing the time and effort required to analyze these recordings.

To support future expansion, we wrote the code with a modular and object-oriented architecture, ensuring that the software remains readily extendable. With feasible effort, developers can add new processing algorithms or modify existing ones. Additionally, they can fine-tune all numerical parameters to their specific needs, with more flexibility than the GUI elicits alone.

Acknowledgments and Availability

We thank Martín Toderi for recordings of hair-bundle oscillations, Justin Faber for videos of mosquito flagellar motion, and Dima Vaido for videos of stimulated glass-probe movement. We thank Dolores Bozovic for help revising this manuscript and providing information on hair cells. We feel grateful toward the Bozovic lab members for tediously testing early versions of the software. They generously suggested improvements, especially during the first few months of development, where the algorithms and layout underwent frequent unexpected changes.

The software is available for MATLAB 2023b on GitHub under `jmarcinik3/BundleTracker`, including the demonstration video from Batch Tracking with Hair Bundles.

References

Arvanitis P, Betting JHLF, Bosman LWJ, Al-Ars Z, Strydis C. WhiskEras 2.0: Fast and Accurate Whisker Tracking in Rodents. In: Orailoglu A, Jung M, Reichenbach M, editors. *Embedded Computer Systems: Architectures, Modeling, and Simulation* Cham: Springer International Publishing; 2022. p. 210–225. doi: 10.1007/978-3-031-04580-6_14.

Bartlett MS. Periodogram Analysis and Continuous Spectra. Biometrika. 1950; 37(1/2):1–16. doi: 10.2307/2332141.

Berghouse M, Miele F, Perez LJ, Bordoloi AD, Morales VL, Parashar R. Evaluation of Particle Tracking Codes for Dispersing Particles in Porous Media. Sci Rep. 2024 Oct; 14(1):24094. doi: 10.1038/s41598-024-75581-0.

Betting JHLF, Romano V, Al-Ars Z, Bosman LWJ, Strydis C, Zeeuw CID. WhiskEras: A New Algorithm for Accurate Whisker Tracking. Frontiers in Cellular Neuroscience. 2020; 14. doi: 10.3389/fncel.2020.588445.

Beurg M, Nam JH, Chen Q, Fettiplace R. Calcium Balance and Mechanotransduction in Rat Cochlear Hair Cells. J Neurophysiol. 2010; 104(1):18–34. doi: 10.1152/jn.00019.2010.

Boltyanskiy R, W Merrill J, R Dufresne E. Tracking Particles with Large Displacements Using Energy Minimization. Soft Matter. 2017; 13(11):2201–2206. doi: 10.1039/C6SM02011A.

Chaiyasitdhi A. Morphology Control of the Hair-Cell Bundle for Frequency-Selective Auditory Detection. PhD thesis, Université Paris sciences et lettres; 2021.

Chuang BI, Hsu JH, Kuo LC, Jou IM, Su FC, Sun YN. Tendon-Motion Tracking in an Ultrasound Image Sequence Using Optical-Flow-Based Block Matching. BioMedical Engineering OnLine. 2017 Apr; 16(1):47. doi: 10.1186/s12938-017-0335-x.

Crocker JC, Grier DG. Methods of Digital Video Microscopy for Colloidal Studies. Journal of Colloid and Interface Science. 1996 Apr; 179(1):298–310. https://linkinghub.elsevier.com/retrieve/pii/S0021979796902179, doi: 10.1006/jcis.1996.0217.

Eriksson I, Powell J, Kaplan AFH. Melt Behavior on the Keyhole Front during High Speed Laser Welding. Optics and Lasers in Engineering. 2013 Jun; 51(6):735–740. doi: 10.1016/j.optlaseng.2013.01.008.

Ershov D, Phan MS, Pylvänäinen JW, Rigaud SU, Le Blanc L, Charles-Orszag A, Conway JRW, Laine RF, Roy NH, Bonazzi D, Duménil G, Jacquemet G, Tinevez JY. TrackMate 7: Integrating State-of-the-Art Segmentation Algorithms into Tracking Pipelines. Nat Methods. 2022 Jul; 19(7):829–832. doi: 10.1038/s41592-022-01507-1.

Fredrickson L, Cheng A, Strimbu CE, Bozovic D, Arisaka K. The Use of a CMOS Camera to Resolve Nanometer Displacements of Hair Cell Stereocillia in the Bullfrog Sacculus. In: *Imaging, Manipulation, and Analysis of Biomolecules, Cells, and Tissues VI*, vol. 6859 SPIE; 2008. p. 296–302. doi: 10.1117/12.764186.

Gal A, Saragosti J, Kronauer DJ. anTraX, a Software Package for High-Throughput Video Tracking of Color-Tagged Insects. eLife. 2020 Nov; 9:e58145. doi: 10.7554/eLife.58145.

Guizar-Sicairos M, Thurman ST, Fienup JR. Efficient Subpixel Image Registration Algorithms. Opt Lett, OL. 2008 Jan; 33(2):156–158. doi: 10.1364/OL.33.000156.

Hudspeth AJ. Making an Effort to Listen: Mechanical Amplification in the Ear. Neuron. 2008 Aug; 59(4):530–545. doi: 10.1016/j.neuron.2008.07.012.

Hudspeth AJ. Integrating the Active Process of Hair Cells with Cochlear Function. Nat Rev Neurosci. 2014; 15(9):600–614. doi: 10.1038/nrn3786.

Lei Y, Gardner A, Agram P. Autonomous Repeat Image Feature Tracking (autoRIFT) and Its Application for Tracking Ice Displacement. Remote Sensing. 2021 Jan; 13(4):749. doi: 10.3390/rs13040749.

Lin CHJ, Bozovic D. Effects of Efferent Activity on Hair Bundle Mechanics. J Neurosci. 2020 Mar; 40(12):2390–2402. doi: 10.1523/JNEUROSCI.1312-19.2020.

Martin P, Bozovic D, Choe Y, Hudspeth AJ. Spontaneous Oscillation by Hair Bundles of the Bullfrog's Sacculus. J Neurosci. 2003; 23(11):4533–4548. doi: 10.1523/jneurosci.23-11-04533.2003.

Meenderink SWF, Quiñones PM, Bozovic D. Voltage-Mediated Control of Spontaneous Bundle Oscillations in Saccular Hair Cells. J Neurosci. 2015 Oct; 35(43):14457–14466. doi: 10.1523/JNEUROSCI.1451-15.2015.

Pecora C. Particle Tracking Velocimetry: A Review. PhD thesis, University of Washington; 2018.

Peng AW, Ricci AJ. Glass Probe Stimulation of Hair Cell Stereocilia. In: Sokolowski B, editor. *Auditory and Vestibular Research: Methods and Protocols* New York, NY: Springer; 2016.p. 487–500. doi: 10.1007/978-1-4939-3615-1_27.

Pinton GF, Dahl JJ, Trahey GE. Rapid Tracking of Small Displacements with Ultrasound. IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control. 2006 Jun; 53(6):1103–1117. doi: 10.1109/TUFFC.2006.1642509.

Ramunno-Johnson D, Strimbu CE, Fredrickson L, Arisaka K, Bozovic D. Distribution of Frequencies of Spontaneous Oscillations in Hair Cells of the Bullfrog Sacculus. Biophysical Journal. 2009 Feb; 96(3):1159–1168. doi: 10.1016/j.bpj.2008.09.060.

Ricci AJ, Crawford AC, Fettiplace R. Active Hair Bundle Motion Linked to Fast Transducer Adaptation in Auditory Hair Cells. J Neurosci. 2000; 20(19):7131–7142. doi: 10.1523/jneurosci.20-19-07131.2000.

Roongthumskul Y, Fredrickson-Hemsing L, Kao A, Bozovic D. Multiple-Timescale Dynamics Underlying Spontaneous Oscillations of Saccular Hair Bundles. Biophys J. 2011 Aug; 101(3):603–610. doi: 10.1016/j.bpj.2011.06.027.

Sahoo PK, Soltani S, Wong AKC. A Survey of Thresholding Techniques. Computer Vision, Graphics, and Image Processing. 1988 Feb; 41(2):233–260. doi: 10.1016/0734-189X(88)90022-9.

Sezgin M, Sankur B. Survey over Image Thresholding Techniques and Quantitative Performance Evaluation. JEI. 2004 Jan; 13(1):146–165. doi: 10.1117/1.1631315.

Silveira Jacques Junior JC, Musse SR, Jung CR. Crowd Analysis Using Computer Vision Techniques. IEEE Signal Processing Magazine. 2010 Sep; 27(5):66–77. doi: 10.1109/MSP.2010.937394.

Strimbu CE, Fredrickson-Hemsing L, Bozovic D. Coupling and Elastic Loading Affect the Active Response by the Inner Ear Hair Cell Bundles. PLoS One. 2012; 7(3):e33862. doi: 10.1371/journal.pone.0033862.

Sveen JK. An Introduction to MatPIV v. 1.6.1. . 2004; .

Thielicke W, Stamhuis EJ. PIVlab – Towards User-friendly, Affordable and Accurate Digital Particle Image Velocimetry in MATLAB. Journal of Open Research Software. 2014 Oct; 2(1). doi: 10.5334/jors.bl.

Thorne PR, Gavin JB. The Accuracy of Hair Cell Counts in Determining Distance and Position along the Organ of Corti. The Journal of the Acoustical Society of America. 1984 Aug; 76(2):440–442. doi: 10.1121/1.391136.

Tinevez JY, Perry N, Schindelin J, Hoopes GM, Reynolds GD, Laplantine E, Bednarek SY, Shorte SL, Eliceiri KW. TrackMate: An Open and Extensible Platform for Single-Particle Tracking. Methods. 2017 Feb; 115:80–90. doi: 10.1016/j.ymeth.2016.09.016.

Todaro MC, Choudhuri I, Belohlavek M, Jahangir A, Carerj S, Oreto L, Khandheria BK. New Echocardiographic Techniques for Evaluation of Left Atrial Mechanics. Eur Heart J Cardiovasc Imaging. 2012 Dec; 13(12):973–984. doi: 10.1093/ehjci/jes174.

Van Wyk de Vries M, Bhushan S, Jacquemart M, Deschamps-Berger C, Berthier E, Gascoin S, Shean DE, Shugar DH, Kääb A. Pre-Collapse Motion of the February 2021 Chamoli Rock–Ice Avalanche, Indian Himalaya. Natural Hazards and Earth System Sciences. 2022 Oct; 22(10):3309–3327. doi: 10.5194/nhess-22-3309-2022.

Van Wyk de Vries M, Wickert AD. Glacier Image Velocimetry: An Open-Source Toolbox for Easy and Rapid Calculation of High-Resolution Glacier Velocity Fields. The Cryosphere. 2021 Apr; 15(4):2115–2132. doi: 10.5194/tc-15-2115-2021.

Wright A, Davis A, Bredberg G, Ülehlová L, Spencer H. Hair Cell Distributions in the Normal Human Cochlea: A Report of a European Working Group. Acta Oto-Laryngologica. 1987 Jan; 104(sup436):15–24. doi: 10.3109/00016488709124972.

Zack GW, Rogers WE, Latt SA. Automatic Measurement of Sister Chromatid Exchange Frequency. J Histochem Cytochem. 1977 Jul; 25(7):741–753. doi: 10.1177/25.7.70454.

Mathematical Notation

We introduce notation to effectively communicate our algorithms. Let $\mathbf{I}$ denote a three-dimensional (3D) matrix of pixel intensities in a grayscale video. In $\mathbf{I}$, the first two indices correspond to position, while the third index corresponds to time.

We denote trace positions as a 2D column vector $\mathbf{X} = [\mathbf{x}, \mathbf{y}]$, where $\mathbf{X}_k \in \mathbb{R}^2$ specifies position at frame k. To rotate a trace counter-clockwise by angle θ, we apply the rotation matrix $\mathbf{R}_\theta = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$.

For statistical operations, we denote the expected value and standard deviation, of the matrix $\mathbf{M}$ along axes x_i ,as $\langle \mathbf{M} \rangle_{x_i}$ and $\sigma_{x_i}[\mathbf{M}]$, respectively. When omitting this subscript, the operation applies over the entire matrix. We denote the Fourier transform of $\mathbf{x}$ as $\mathcal{F}[\mathbf{x}]$.

Defining and Tracking ROIs

The software streamlines tracking for optical datasets by supplying an intuitive GUI for defining ROIs and for adjusting processing settings. The user begins by importing a video. The software supports grayscale videos, including 8-bit and 16-bit MJ2 (Motion JPEG 2000) as well as 8-bit AVI (Audio Video Interleave). If importing an RGB video, the user must first convert to grayscale, which they can accomplish using the built-in `VideoReader` and `VideoWriter` classes.

After importing a video, the user defines ROIs. They can manually draw each ROI as one of four supported shapes: rectangle, ellipse, polygon, and freehand. Alternatively, they can automatically generate ROIs using the blob-detection GUI. Once defined, the user can manually adjust the size and position of each ROI.

Once the user finalizes their ROIs, they command the software to batch-process them. Within each ROI, the software performs three steps sequentially: preprocess frames, track displacements, and postprocess traces. Throughout these steps, the software propagates error estimates unless otherwise noted. To ensure reproducible results, the software automatically saves comprehensive metadata from every session, enabling the user to adjust parameters or rerun sessions later with consistent results.

Preprocessing Frames

We implemented three optional preprocessing steps: smooth, filter, and invert. The software preprocesses each frame independently, normalizing pixel intensities after each step.

For smoothing, the user specifies a window width, which indicates the size of a 2D normalized Gaussian window. Using the built-in `smoothdata2(method="gaussian")` function, the software applies this as a Gaussian blur to the image. If the width equals zero, then smoothing produces no effect.

For filtering, the user sets lower and upper bounds I_{min} and I_{max}, respectively, for pixel intensities. The software filters the image by mapping each pixel intensity $I \in \mathbf{I}$ according to

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

If $I_{min} = 0$ and $I_{max} = 1$, then filtering produces no effect.

For inverting, the user chooses whether to invert pixel intensities. If selected, then the software maps $\mathbf{I} \mapsto -\mathbf{I}$. If not, then it skips inverting.

To help the user visualize preprocessing effects, the software displays pixels for the first preprocessed frame using a three-tier colormap. This colormap assigns distinct values to undersaturated, oversaturated, and standard pixel ranges, enabling the user to easily identify saturation issues. This display updates in realtime as the user adjusts preprocessing parameters.

Parameter	Significance	Value	Lower	Upper
z_0	Offset	$\underset{I\in\mathbf{I}_k}{\text{med}}\, I$	0	∞
A	Amplitude	1	0	∞
μ_x	Center (x)	$\underset{x\in\mathbf{P}_x}{\text{med}}\, x$	$\underset{x\in\mathbf{P}_x}{\min}\, x-1$	$\underset{x\in\mathbf{P}_x}{\max}\, x+1$
μ_y	Center (y)	$\underset{y\in\mathbf{P}_y}{\text{med}}\, y$	$\underset{y\in\mathbf{P}_y}{\min}\, y-1$	$\underset{y\in\mathbf{P}_y}{\max}\, y+1$
σ_x	Standard deviation (x)	1	0	∞
σ_y	Standard deviation (y)	1	0	∞
θ	Angle	$\frac{\pi}{4}$	0	π

Appendix 0—table 1. Parameters fit in the rotated 2D Gaussian. From left to right, the columns represent 1) parameter symbol along with its corresponding 2) initial condition as well as 3-4) lower and upper bounds. The 3D matrix $\mathbf{P}$ represents the x- and y-position at each pixel.

Tracking ROIs

After preprocessing, the software tracks the position of each ROI. Currently, it implements three tracking algorithms: Centroid, 2D Gaussian and Maximum Cross-Correlation. As a result, the software outputs a 2D vector of positions $\mathbf{X}$ for each ROI.

Centroid

The software finds displacement by calculating a centroid of pixel intensities. It first estimates experimental error from standard deviations in pixel intensity. It then propagates these errors to calculate centroids along with their concomitant error.

To estimate error $\delta I(I)$ at pixel intensity $I \in \mathbf{I}$, the software generates an error map $\delta I : [0,1] \to \left[0, 1/2\right]$. Using least-squares regression, the software fits a power law $y = Ax^m$ between $x \in \langle \mathbf{I} \rangle_t$ and $y \in \sigma_t[\mathbf{I}]$. Asserting this relationship, the software approximates $\delta I(I) \approx AI^m$ and maps $I \mapsto I \pm \delta I(I)$.

The software then calculates centroids weighted by pixel intensity. Using normalized weights $\mathbf{w} \propto \mathbf{I}$, the resultant centroid $\mathbf{X}_k$ acts as the position at frame k.

2D Gaussian

The software finds displacement by fitting a 2D Gaussian function. It fits pixel intensities in frame k with respect to their positions. Employing the Levenberg-Marquardt algorithm, it calculates seven best-fit parameters in a rotated 2D Gaussian with the form

$$f(\mathbf{x}) = z_0 + A\exp\left(-[\mathbf{R}_\theta(\mathbf{x}-\boldsymbol{\mu})]^\top \boldsymbol{\Sigma}[\mathbf{R}_\theta(\mathbf{x}-\boldsymbol{\mu})]/2\right) \text{ where } \boldsymbol{\Sigma} = \text{diag}(\boldsymbol{\sigma})$$

These parameters include 1-2) center $\boldsymbol{\mu}$, 3-4) standard deviation $\boldsymbol{\sigma}$ and 5) angle θ, along with 6) offset z_0 and 7) amplitude A of the Gaussian. We show the initial guesses for these parameters in ***Table 1***. Using nonlinear least-squares estimates, the software calculates a 95% confidence interval, using half this range as the error for each parameter. The resultant $\mathbf{X}_k = \boldsymbol{\mu}$ acts as the position at frame k.

Maximum Cross-Correlation

The software finds displacement by maximizing cross-correlation between frames. The software assumes null displacement for the first frame ($k = 1$). For each subsequent frame ($k \geq 2$), it calculates $\mathbf{X}_k = \arg\max_{\Delta\mathbf{x}} \left\langle \mathbf{I}_{ij1}(\mathbf{x}_{ij})\mathbf{I}_{ijk}(\mathbf{x}_{ij} - \Delta\mathbf{x}) \right\rangle_{ij}$ to determine displacement. Conceptually, the software identifies the spatial shift $\Delta\mathbf{x}$ at which pixel intensities $\mathbf{I}_{ijk}$ in frame k achieve their maximum cross-correlation relative to those $\mathbf{I}_{ij1}$ in the first frame. To improve accuracy, the software upsamples each frame by a factor of 100 before maximizing cross-correlation (***Guizar-Sicairos et al., 2008***). This reduces positional error to as little as $1/100^{\text{th}}$ of a pixel.

Postprocessing Traces

We implemented five postprocessing operations on the traces. Listed in their performed order, these steps include orienting, shifting, scaling, rotating, and detrending. Out of these, all except shifting are optional.

The software orients traces by necessarily reflecting and optionally redirecting them. Before orienting, the traces start downward and rightward as displayed in the global editor, with $\mathbf{x}$ and $\mathbf{y}$ pointing directly rightward and downward, respectively. The software begins by reflecting $\mathbf{y} \mapsto -\mathbf{y}$ so that the trace conventionally points up-rightward. In this paper, the component $\mathbf{y}$ hereafter remains $90\,\mathrm{deg}$ counter-clockwise relative to $\mathbf{x}$. The user can further redirect the trace by choosing an angle θ_0 for $\mathbf{x}$, either 1) selecting from the eight cardinal directions or 2) dragging the arrow in the region editor. The software then rotates $\mathbf{X} \mapsto \mathbf{R}_{\theta_0}^{\top}\mathbf{X}$, leaving the trace unrotated if $\theta_0 = 0$.

The software mandatorily shifts traces. To do so, it subtracts the mean position by mapping $\mathbf{X} \mapsto \mathbf{X} - \langle \mathbf{X} \rangle_t$.

The software optionally scales traces. The user can scale by selecting a scale factor A. The software then scales traces by multiplying $\mathbf{X} \mapsto A\mathbf{X}$, leaving the trace unscaled if $A = 1 \pm 0$.

The software optionally rotates traces using one of six algorithms described in Rotating Traces. Based on the selected algorithm, it computes an angle θ^* and rotates $\mathbf{X} \mapsto \mathbf{R}_{\theta^*}^{\top}\mathbf{X}$ counter-clockwise. Additionally, it further ensures that the larger-amplitude component aligns with the x-direction, rotating $\mathbf{X} \mapsto \mathbf{R}_{\frac{\pi}{2}}^{\top}\mathbf{X}$ if $\sigma[\mathcal{F}[\mathbf{x}]] < \sigma\left[\mathcal{F}[\mathbf{y}]\right]$. The software skips this rotation step if the user selects "None".

The software optionally detrends traces using one of three algorithms described in Detrending Traces. Based on the selected algorithm, it removes the gradual drift in the trace, acting independently on each component. The software skips this detrending step if the user selects "None".

Rotating Traces

The software offers six algorithms to rotate traces, each designed for different scenarios. For most algorithms, the software calculates errors using four-fold cross-validation on subsets of $\mathbf{X}$. Looping over these subsets, the software calculates angles $\boldsymbol{\theta}^*$, estimating $\delta\theta^* = \sigma[\boldsymbol{\theta}^*]$ as its error unless otherwise noted.

Minimum Covariance

This algorithm minimizes the covariance between $\mathbf{x}$ and $\mathbf{y}$ by solving

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

where $\mathrm{Cov}[\mathbf{x}, \mathbf{y}]$ represents the 2×2 covariance matrix between $\mathbf{x}$ and $\mathbf{y}$. Conceptually, this rotation finds the angle that minimizes the absolute covariance, aligning the trace such that $\mathbf{x}$ appears most dissimilar to $\mathbf{y}$.

2-Means Clustering or Two Gaussian Mixtures

These algorithms cluster points in $\mathbf{X}$ into two groups using either k-means clustering (with $k = 2$) or Gaussian-mixture models (with two components). After finding two clusters, the software finds $\theta^* = \arctan\left(\frac{y_2 - y_1}{x_2 - x_1}\right)$, where (x_1, y_1) and (x_2, y_2) denote the centroids of the clusters such that $x_1 < x_2$. Conceptually, this method rotates the trace such that these centroids align with the x-axis.

Linear Regression

This algorithm fits a line $y = mx + b$ to the trace using least-squares regression. It then takes the slope m, along with its standard error, to find $\theta^* = \arctan(m)$. Conceptually, this rotation aligns the best-fit line along the x-axis.

Elliptical Regression

This algorithm performs singular value decomposition on $\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top$. Here, each column in $\mathbf{\Sigma}$ corresponds to one term in the quadratic equation $Ax^2 + Bxy + Cy^2 + Dx + Ey + F = 0$, i.e. $\mathbf{\Sigma} \sim \begin{pmatrix} x^2 & xy & y^2 & x & y & 1 \end{pmatrix}$. The software equates these coefficients with values in the most-null vector from $\mathbf{V}$. It then calculates

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

Conceptually, this rotation aligns the major axis of the best-fit ellipse with the x-axis. Unlike the other rotation algorithms, the software skips propagating error during elliptical regression.

2D Gaussian

This algorithm fits a rotated 2D Gaussian to pixel intensities contained in the ROI. After fitting a Gaussian to intensities in the first frame, the best-fit angle serves as θ^*. For more details on fitting this Gaussian, see 2D Gaussian. Conceptually, this rotation aligns the nearest semi-axis of the Gaussian along the x-axis.

Detrending Traces

The software offers four algorithms to detrend traces. As its first three algorithms, it subtracts a high-degree polynomial, subtracts a moving average, or applies a high-pass filter. As its fourth algorithm, it combines the polynomial and moving-average methods in series. (labeled as "Auto" in the GUI). It applies the selected method to each displacement component independently. Alternatively, the user can opt to leave the trace unaltered. The software skips propagating error when detrending.

Polynomial Regression

This algorithm fits a polynomial P_N of degree N to $\mathbf{x}(t)$ as a function of time $t \in \mathbf{t}$. For all $t \in \mathbf{t}$, the software detrends the position as $\Delta x_N(t) \coloneqq x(t) - P_N(t)$. It then determines the smallest degree N^* that satisfies

$$\frac{R\left(\Delta x_{N^*+1}(\mathbf{t})\right)}{R\left(\Delta x_{N^*}(\mathbf{t})\right)} > 1 + p_\delta \text{ where } R(\mathbf{x}) \coloneqq \max(\mathbf{x}) - \min(\mathbf{x})$$

for some $p_\delta \in \mathbb{R}$. By default, we set $p_\delta = 0.01$ arbitrarily after some testing, but the user can tweak this value in the `Detrender` class. To subtract drift, the software maps $\mathbf{x} \mapsto \Delta x_{N^*}(\mathbf{t})$. Conceptually, it avoids overfitting drift by finding the lowest-degree polynomial that squeezes $\Delta x_N^*(\mathbf{t})$ into a "small" range.

Moving Average

This algorithm computes a Hann moving average $\boldsymbol{\mu}_\mathbf{x}$ over $\mathbf{x}$. It dynamically estimates the window width as $NF_{\mathcal{F}}^{-1}(F^*)$, where $F_{\mathcal{F}}(f) \propto \int \mathcal{F}[\mathbf{x}]\,\mathrm{d}f$ denotes the cumulative distribution function of $\mathcal{F}[\mathbf{x}]$, with $N > 0$ and $F^* \in (0, 1)$. We set $N = 2$ and $F^* = 0.95$ arbitrarily, erring to preserve $\mathbf{X}$ and retain drift, but the user can tweak these in the `Detrender` class. To subtract drift, the software maps $\mathbf{x} \mapsto \mathbf{x} - \boldsymbol{\mu}_\mathbf{x}$. Conceptually, it subtracts a wide-window moving average from the original trace.

High-Pass Filter

This algorithm applies a high-pass filter to the trace. The software calculates PSDs using the built-in `pwelch` function. With a PSD, the software identifies the dominant, nonzero peak frequency, and it sets the cutoff frequency to $\omega_{peak}/2$. To balance removing drift with preserving signal, we chose

this factor of $1/2$ arbitrarily. Using this cutoff, the software applies a high-pass filter to the trace using the built-in `highpass("Steepness"=0.5)` function. Conceptually, this filter removes the low-frequency components, assumed to represent drift, while mostly preserving the high-frequency components, assumed to contain signal.

Automating Features

Error Propagation

To facilitate 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. For most functions, this class propagates error using the min-max method, which optimizes numerical precision. For select functions, it applies a linear approximation, which improves computational speed.

The user can propagate error through any scalar function. To propagate error for a matrix $\mathbf{M}$ through an arbitrary single-argument function f, the user calls `ErrorPropagator.scalarFunction(`$\mathbf{M}$`,` `@`f`)`, which calculates error using the min-max method.

Blob Detection

The software assimilates blob detection using the built-in `vision.BlobAnalysis` class, which identifies clusters of bright pixels in an image. To detect blobs, the software uses only the first video frame. It then inserts these blobs as ROIs in the global editor.

Before detecting blobs, the software preprocesses the image by smoothing, normalizing, and binarizing. To smooth the image, it subtracts a 2D moving mean using the built-in `smoothdata2` function, diminishing local fluctuations in pixel intensity. To normalize, the software maps all $I \in \mathbf{I}$ similar to ***Equation 1*** with $I_{min} = \min(\mathbf{I})$ and $I_{max} = \max(\mathbf{I})$. After smoothing and normalizing, the software binarizes the image. To binarize the image, the user selects lower and upper bounds I_{min} and I_{max}, respectively. The software then maps

$$I \mapsto \begin{cases} 1, & I_{min} \leq I \leq I_{max} \\ 0, & \text{otherwise} \end{cases}$$

for all $I \in \mathbf{I}$, highlighting bright-pixel clusters as blobs.

The user can customize eight blob-detection parameters to suit their dataset (illustrated in ***Figure 2***). These parameters include connectivity (4-way or 8-way), maximum number of blobs, range of blob areas (lower and upper bounds), shape of blobs (rectangle or ellipse), size of blob (width and height dimensions), and inclusion of border blobs. By adjusting these parameters, the user can refine blob detection to identify the objects contained in their specific dataset.

Image Thresholding

The software incorporates image thresholding using a user-selected algorithm, which identifies a characteristic intensity I^* to segment an image. To calculate I^*, the software uses only the first video frame. The software further interpolates threshold intensities by piecewise linearly mapping

$$I_{min/max}(x) = \begin{cases} [I^* - \min(\mathbf{I})]x + \min(\mathbf{I}) & 0 \leq x \leq 1 \\ [\max(\mathbf{I}) - I^*]\,(x-1) + I^* & 1 \leq x \leq 2 \end{cases} \tag{2}$$

where $I_{min/max}$ corresponds to I_{min} or I_{max} in ***Equation 1***. After calculating I_{min} and I_{max}, the software applies these thresholds to their corresponding ROIs.

The user can customize three image-thresholding parameters (illustrated in ***Figure 3***). These parameters include 1-2) an x for each I_{min} and I_{max} as well as 3) an algorithm to calculate I^* (***Zack et al., 1977***; ***Sahoo et al., 1988***; ***Sezgin and Sankur, 2004***). By adjusting these options, the user can refine characteristic intensities to highlight the desired features of their specific system.

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

Appendix 0—table 2. Shortcuts implemented in the `BundleTracker` interface.

Efficient Shortcuts

We implemented several features to streamline working with ROIs: moving, resizing, reordering, and duplicating. Moving an ROI shifts it along the four cardinal directions. Resizing it adjusts its dimensions, either compressing or expanding it. Reordering it changes its layer relative to other ROIs, bringing it either forward or backward. Duplicating it creates a copy of its pixel location, region shape, and processing settings. To speed up workflow, the software provides keyboard shortcuts for all of these actions, many mimicking their corresponding actions on layers in Adobe Photoshop (shown in ***Table 2***).

The software includes a minimal but powerful API, allowing the user to partially run it via scripts. This script-based interface provides key actions: 1) importing new videos or previous sessions, 2) exporting a preview image of the global editor, and 3) tracking and exporting ROIs. Using these actions, the user can efficiently loop through and process multiple datasets. To fully automate tracking, they process an initial session and import its metadata into other datasets, enabling them to track all subsequent videos unattended.