Biophysical Reports

ARTICLE

Philament: A filament tracking program to quickly and accurately analyze in vitro motility assays

Ryan M. Bowser,[1] Gerrie P. Farman,[1] and Carol C. Gregorio[1,2,*]
[1]Department of Cellular and Molecular Medicine and Sarver Molecular Cardiovascular Research Program, The University of Arizona, Tucson, Arizona and [2]Cardiovascular Research Institute, Department of Medicine, Icahn School of Medicine at Mount Sinai, New York, New York

ABSTRACT In vitro motility (IVM) assays allow for the examination of the basic interaction between cytoskeletal filaments with molecular motors and the influence many physiological factors have on this interaction. Examples of factors that can be studied include changes in ADP and pH that emulate fatigue, altered phosphorylation that can occur with disease, and mutations within myofilament proteins that cause disease. While IVM assays can be analyzed manually, the main limitation is the ability to extract accurate data rapidly from videos collected without individual bias. While programs have been created in the past to enable data extraction, many are now out of date or require the use of proprietary software. Here, we report the generation of a Python-based tracking program, Philament, which automatically extracts data on instantaneous and average velocities, and allows for fully automated analysis of IVM recordings. The data generated are presented in an easily accessible spreadsheet-based, comma-separated values file. Philament also contains a novel method of quantifying the smoothness of filament motion. By fitting curves to standard deviations of velocity and average velocities, the influence of different experimental conditions can be compared relative to one another. This comparison provides a qualitative measure of protein interactions where steeper slopes indicate more unstable interactions and shallower slopes indicate more stable interactions within the myofilament. Overall, Philament's automation of IVM analysis provides easier entry into the field of cardiovascular mechanics and enables users to create a truly high-throughput experimental data analysis.

WHY IT MATTERS Philament, specifically designed for in vitro motility assays and built on an open-source platform (Python), decreases the barrier of entry to explore myofilament interactions. Philament can convert images to binary scale (threshold), find and track filaments (objects), and report on the frame to frame instantaneous velocity. This is done in a completely automated fashion, with only an initial input from the user to start the process, speeding up the analysis pipeline, and allowing for high-throughput analysis of IVM data. Finally, Philament allows users to keep the program up-to-date by updating the modules used by the Python script, greatly extending the lifespan and performance of Philament for future use.

INTRODUCTION

In vitro motility (IVM) assay is a powerful approach to evaluate the movement of fluorescently labeled filaments (usually F-actin or microtubules) on a bed of surface-immobilized motor molecules (myosin or kinesin/dynein, respectively). When attempting to decipher functional properties of striated muscle, for example, IVM allows investigators to examine a myriad of alterations in myofilament proteins (e.g., mutations, posttranslational modifications) on actomyosin interactions under precisely controlled conditions. Experimental conditions can be altered within the experimental chamber to mimic the in vivo myofilament lattice in normal conditions and how the lattice presents during other physiological conditions such as disease, exercise, and fatigue. For example, the levels of ATP/ADP/P_i that can mimic fatigue can be altered (1). Specific purified contractile proteins can be added to test their potential influence and direct roles in actin-myosin interactions; as one example, α-actinin can be added to increase frictional load mimicking increased pressure in the heart (2). Other

Submitted October 24, 2023, and accepted for publication January 25, 2024.

*Correspondence: carol.gregorio@mssm.edu

Ryan M. Bowser and Gerrie P. Farman contributed equally to this work.

Editor: Sarah Rauscher.

https://doi.org/10.1016/j.bpr.2024.100147

physiological components can be added thereby mimicking conditions within the myofilament lattice during exercise or disease progression (1). Furthermore, with the flexibility and ease of changing variables, this assay is ideally set up to create high-throughput screening assays to evaluate the impact new drugs and compounds have on actin-myosin interactions.

In general, the major drawback of the IVM assay is the inability to quickly and concisely analyze videos to extract important information such as velocity, fraction motile, and filament lengths. Most attempts to analyze filament motion in IVM assays have either utilized a point-and-click method of tracking single filaments (e.g., mTrackJ and RETRAC) (1–3) or one of a series of "batch" analysis algorithms either purchased (4–6), custom written by individual labs for use on MATLAB (7–11), or repurposed from different experimental protocols (e.g., wrMTrck (12)) to extract data (13–15). There also exists a wide array of tracking programs, whose intended usages are similar to the IVM assay. These programs include the capabilities of tracking cytoskeletal filaments within cells (16), tracking in vitro microtubule movement and analyzing kymographs (17), and deformable objects (such as cells, drosophila, or zebrafish) in a two-dimensional recording setup (18). However, these software packages are written in Java or C++, are designed and optimized for different usages, or require the user to purchase a program (MATLAB) to run the file or use out-of-date software (wrMTrck) for batch data analysis.

It is important to note that there is one program we identified, FASTrack, that was written within the Python coding language for use in analyzing the motility of filaments (19) and, as such, is freely available. However, there are some potential limitations to this program. FASTrack was written using Python 2.0 which means that it cannot capitalize on future Python optimizations (i.e., Python 2.0 support ended in 2020), and the program's command line interface heightens the learning curve for analysis.

The use of traditional batch analysis programs enables users to quickly extract data from videos without, for the most part, any biases in filament shape or motion, allowing for a more complete and quantitative analysis of the data collected. A drawback of this method to analyze data is the dependence on software to determine what is or is not a filament, and how to track motion. For the software to accurately identify a filament from a video, the images must first be thresholded (converted to binary images) to separate filaments from background noise. Once thresholded, tracking the motion of the object is performed using a "centroid fitting" routine to determine the area of the filament by placing a "dot" on the midpoint of the filament (representing its center of mass), which is then tracked from frame to frame as the filament moves. This approach allows for an accurate characterization of an object's motion, as changes in the shape of the filament are weighted proportionally, minimizing the impact of filament bending in the final calculations as it tracks its path over the coverslip. An issue that arises with traditional batch analysis programs is the loss of filament tracking due to objects momentarily overlapping. When objects being measured cross paths (overlap), previous versions of available software would immediately terminate tracking of the objects (i.e., filaments) and, depending on filtering stringency (the number of frames to qualify as a filament), remove the tracking data before the interaction. Loss of these data could potentially result in having important information left out from the experimental output on velocity and fraction motile of functional filaments since filaments crossing over is a relatively common event.

In this report, we present a new Python-based automated batch analysis program for extracting velocity and filament length information from IVM assays that we refer to as "Philament." The use of Philament speeds up analysis by a factor of 10 in comparison with wrMTrck, a tracking-only program used previously in this lab. Unlike Philament, wrMTrck does not perform thresholding and requires an input of prethresholded images, which are created using ImageJ's built-in thresholding function. Philament's enhanced thresholding and object-tracking capabilities enable the inclusion of more data from a larger number of filaments, resulting in more reliable results. Philament also allows for the comparison of average velocities to instantaneous velocities; this comparison of velocities allows for determining the possible impact treatments or reagents (e.g., purified proteins) have on the "smoothness" of motion, the standard deviation of the instantaneous velocities. Philament allows new users to quickly collect data without having to learn coding or complex image analysis software, allowing for easier entry into the field and reduced training times. Furthermore, this program can be easily updated with minimal knowledge of Python, not requiring extensive rewrites to the software to keep the program up to date (an issue we encountered with the ImageJ plugin wrMTrck our group used previously). Overall, the novel features of Philament software will allow for the use of IVM as a truly high-throughput and innovative screening mechanism for the investigation of filament-motor interactions and myofilament mechanics.

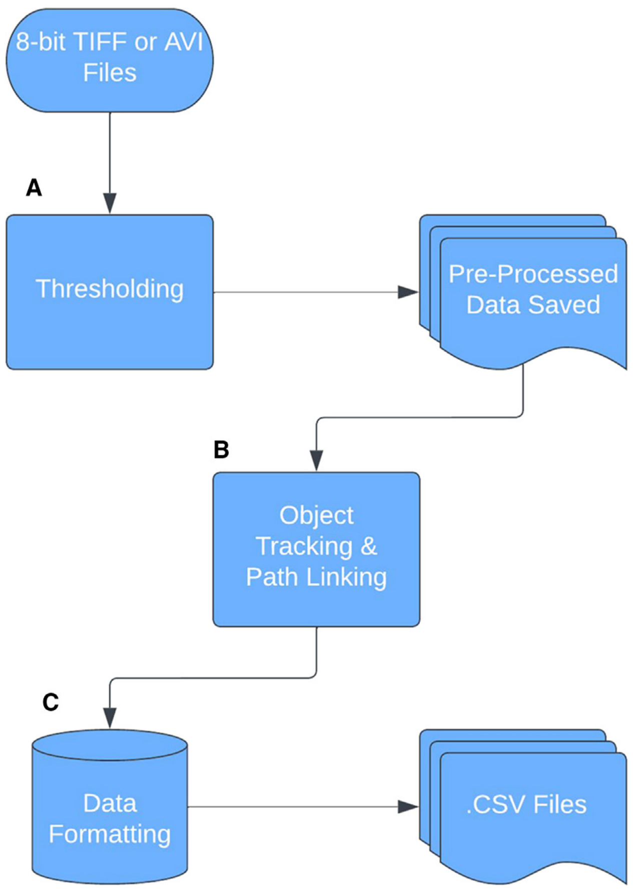

FIGURE 1 Philament data pipeline. Data are taken in as .tif or .avi image sequences and the output .csv file contains the objects' initial position, size, distance traveled, and instantaneous velocities. Philament mainly implements OpenCV, Pandas, and Trackpy packages.

METHODS

IVM

The solution composition and flow cell loading were as described in Farman et al. (15). In brief, the flow is made by laying down strips of double-sided sticky tape onto a glass slide creating two "lanes" that will be enclosed by the glass coverslip. The coverslip was previously coated with nitrocellulose to allow for myosin to bind to the surface. Adding solutions into the flow channel of the experimental chamber, gravity and capillary action will draw the solution into the flow cell. For a 22×22 mm coverslip the volume of the flow cell is $\sim$20–25 μL and is referred to here as 1 volume. To load the flow cell, first add myosin and incubate for 1 min before the addition of 1 mg/mL of BSA to coat nonmyosin-bound surfaces and to prevent nonspecific binding of the actin filaments. Unlabeled actin is then added to the flow cells to ensure that bound myosin can cycle actin properly and "trap" improperly bound myosin, which would prevent labeled actin filaments from sliding properly. Finally, the labeled actin is added to the flow cells, and motility is induced with the addition of ATP (1 mM, pH 7.3).

For the myosin dose-response curve, rabbit skeletal myosin was diluted to a concentration range of 5–250 μg/mL. The resulting data were fit to the following equation (20): $V_{obs} = V_{Max}[1 - (1 - r)^p]$, where r equals the fraction of time myosin stays bound to filamentous actin during the ATPase cycle (duty cycle) and p represents the loading concentration of myosin in the flow cell (21). All videos were collected using XCAP software (v.3.8) with a PTI IC-300 camera running through a PIXCI SV7 Dual Analog Standard Video card. The video card and software were purchased from EPIX (Buffalo Grove, IL).

Philament implementation

Philament was written in Python 3 (Python Software Foundation) (22), with a simple graphical user interface (GUI) that allows for quick changes to the thresholding and tracking parameters as well as ease of use. Currently, Philament is offered as a Python-based program capable of running on Windows or Linux/Unix machines, or by using the Windows executable program converted by using pyinstaller to appeal to those without coding experience. When using Philament as a Windows executable file, the multiprocessing feature is disabled and therefore cannot capitalize on the increased processing power and speed of multicore processors; however, in our testing, this decrease in speed was minimal.

Upon opening Philament, the user is shown a GUI (see supporting material, Fig. 1) that will allow the user to input basic information about the video(s) to be analyzed including pixel size, frame rate, and object size. Philament implements a set of Python packages designed for scientific usage, mainly Open Computer Vision (OpenCV), Trackpy, Pandas, and Numba. OpenCV is a robust and well-tested library and has been a gold standard for image classification and recognition since first released by Intel in 2000 (23). Philament utilizes the preprocessing and image handling functions (Fig. 1 *A*) from the OpenCV library, while we opted to source a tracking workflow from the Trackpy library (Fig. 1 *B*). The main factors in choosing Trackpy over other libraries such as OpenCV or Scikit-Image (24) were its ease of implementation and multiprocessing integration, which spreads tracking calculations over multiple CPU cores, thereby decreasing computational time. This increase in speed is further improved with the usage of Numba's just-in-time compiler (25), which converts high-level Python code into optimized machine code. The returned tracking data are formatted and summary statistics, such as instantaneous velocity, are calculated with the Pandas library (26) (Fig. 1 *C*), similar to how one would write formulas by hand with Microsoft Excel, but performed in fractions of a second, for hundreds of files per run. Output is saved as a comma-separated values (CSV) file, which can be opened easily with spreadsheet applications for further analysis and/or filtering. More details regarding how the videos are handled and how the data are extracted are provided in the results and discussion sections.

Statistics

Statistical analysis and figure creation were performed in GraphPad Prism 10 (GraphPad Software 225 Franklin Street. Fl. 26 Boston, MA 02110). Filament speed, fraction motile, and length filtering were calculated using Excel (see supporting material). Data are shown as mean $\pm$ SE. Statistical significance was set at $p < 0.05$ $^*p < 0.05$ $^{**}p < 0.01$ $^{***}p < 0.001$.

RESULTS

Data preprocessing/thresholding

Videos can be inputted into the program as either 8-bit Tag Image File Format (TIFF) or Audio Video Interleaved (AVI) files, and files are selected using the Windows default "file explorer window." After the desired videos are chosen, Philament begins the analysis

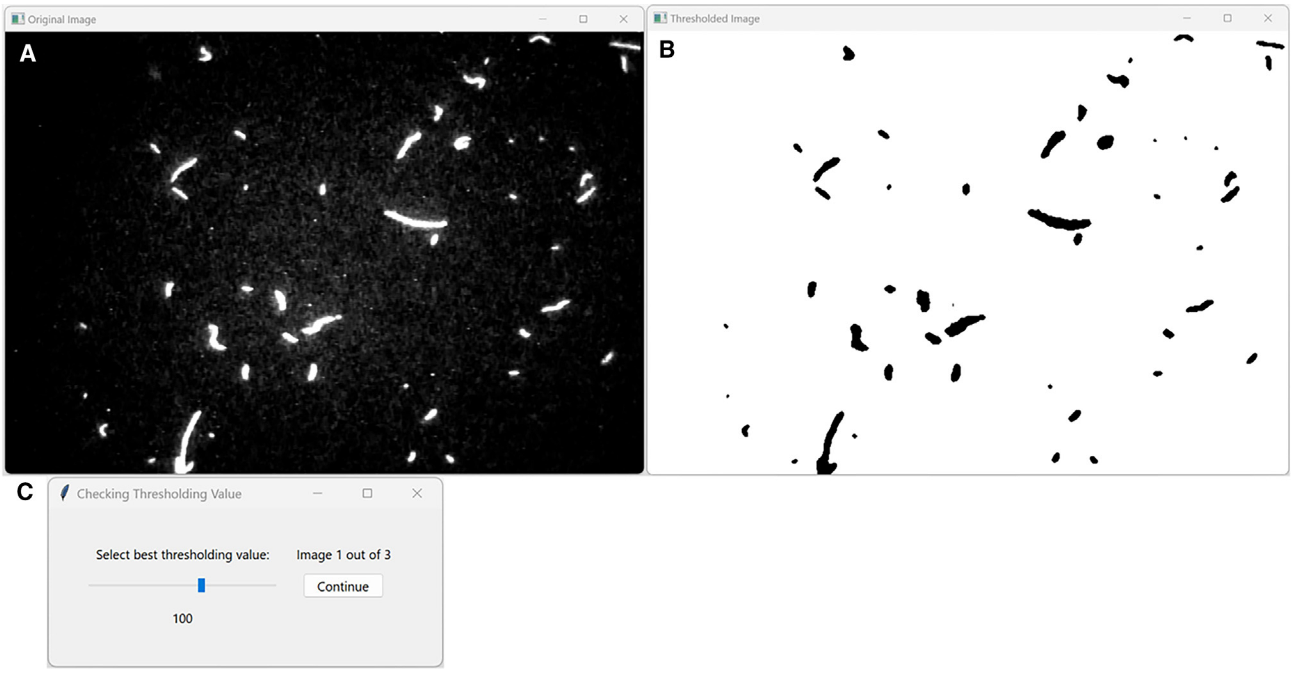

FIGURE 2 Graphical user interface (GUI) for thresholding images in Philament. When Philament is first started the user will be shown the first frame of the raw image (*A*), the automatic thresholded image (*B*), and a slider bar (*C*) that allows the user to adjust the thresholding value for that single image. In the case of multiple movies, Philament will randomly select 1 movie for every 50 movies imported (maximum 5 movies to threshold) for setting the thresholding values. When all images have been assessed a thresholded value, those values are then averaged, and that number is applied to all images that are analyzed.

pipeline (Fig. 1 *A*) by preprocessing the videos for the object-tracking algorithms. This is accomplished by applying a median blur and thresholding the frames to separate objects from the background (Fig. 2).

Median filtering (blur) is a common technique in image preprocessing, it is an edge-preserving filtering method that excels at smoothing spikey noise (also called "salt-and-peppering noise") (27). Median filtering works by finding the median value within a small predetermined area of pixels (called a search kernel) and replacing a pixel's original brightness with the median value. The size of this search kernel (5 pixels) is not adjustable from the GUI; however, if desired, it may be changed within the source code (see supporting material). The blurred frame is then thresholded, where any pixel value below the set threshold is set to the minimum brightness (0), with all other numbers set to the maximum brightness (255). This process is referred to as "binarization," and it allows for much simpler tracking because the objects (set at brightness = 255) are easily separated from the background (set at brightness = 0). This combination of image-processing techniques allows for automated batch thresholding with greatly reduced noise compared with manual thresholding via ImageJ.

In Philament, a randomized sample of videos is used to find an accurate threshold value for all files, eliminating the need to threshold every video manually. After selecting the files, Philament selects a random video and displays the processed and thresholded frame beside the raw first frame (Fig. 2). This is accompanied by a slider bar that changes the threshold value, allowing the users to pick the best value to obtain the most "clean filaments" with little background noise for the selected images. When thresholding, for every 50 videos entered into the program, one random video is selected, and the first frame is shown to the user to manually set the thresholding values. For every additional 50 videos selected for analysis, an additional video is shown to the user (with a maximum of 5 images, 250+ videos). The thresholding values chosen by the user are then saved after each image is processed, these values are averaged, and this average is used to threshold all the selected files. Once the process of determining the thresholding value is completed, Philament becomes fully autonomous and requires no further user input.

Fig. 3 illustrates the difference between the auto thresholding features of ImageJ (Fig. 3, *A* and *B.II*) versus Philament (Fig. 3, *A* and *B.III*) of a sample image (Fig. 3, *A* and *B.I*). As shown in these panels, Philament can cleanly render the particles in the movie, providing a sharper image with less background noise in comparison with automatic ImageJ thresholding. It should be noted that ImageJ can closely approximate the level of thresholding (Fig. 3 *A.IV*) seen in Philament;

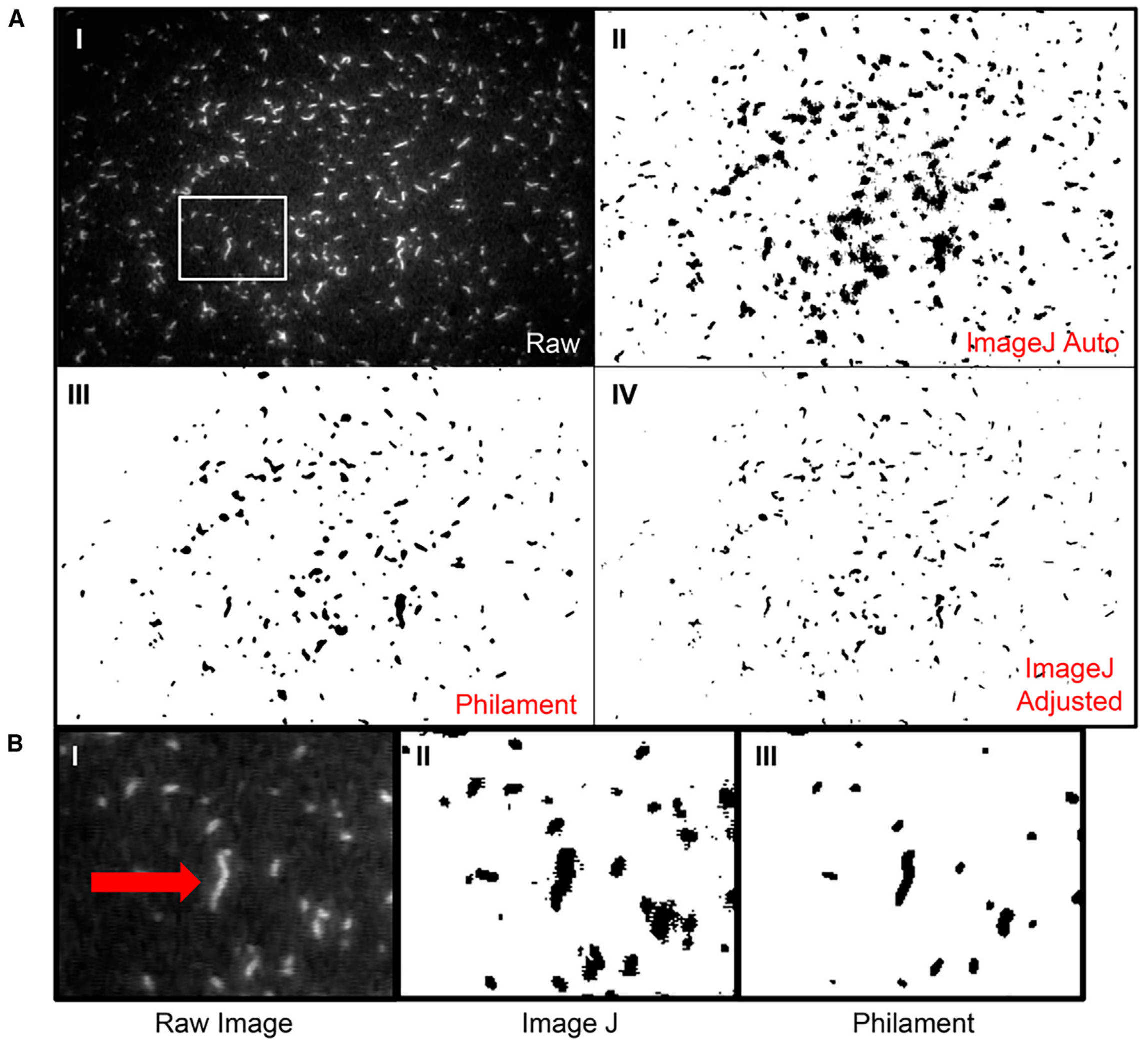

FIGURE 3 Comparison of different thresholding values for ImageJ versus Philament. Thresholding of raw image (*A.I*) by ImageJ (*A.II* auto scale, *A.IV* scaled to match Philament object sizes) demonstrates that, although ImageJ can get the thresholding close to Philament's level, this requires manual adjustment for each recording (*A.IV*). Thus, while ImageJ can slowly threshold each image to get possibly cleaner results, Philament offers the advantage of increasing the output of image processing without a great loss in data (*A.III*). (*B*) Presents the zoomed-in section of I, II, and III (*white box*) to better illustrate the difference between ImageJ and Philament's auto thresholding capabilities. (*B.II*) Demonstrates that there is significant salt and pepper noise in the image that is not visible using Philament. All downstream analyses with wrMTrck started with the hand-adjusted ImageJ thresholded recordings. Note that the filament pointed out by the red arrow is the object referred to in Fig. 4.

however, it requires longer user input, slowing down the analysis and data extraction. Also note, that thresholding by Philament (Fig. 3 *B.III*) does not record very small particles, which could be submicron-sized filaments. These submicron filaments are prone to tracking errors and are excluded from tracking via the minimum size parameter (supporting material, section object diameter).

Centroid tracking, path linking, and data processing

Once the movie has been thresholded, and thus the objects have been differentiated from the background, they can be tracked. We utilized the common centroid tracking method within Trackpy, where the circumference of the object is determined and then the "center of mass" of the object is determined. An example of this is shown in Fig. 4, *A–E*, which show the filament highlighted in Fig. 3 outlined and the center of mass of the object in each frame included. Fig. 4 *F* contains a Z-projection of these five frames showing the relative movement of the centroid between each frame. The calculation of the distance is done using basic Euclidean distance calculations and the velocity for each frame is reported (see Tables 1 and 2) based on the frames per second.

Philament's tracking module (Fig. 1 *B*) is built around an "engine" that implements the Trackpy

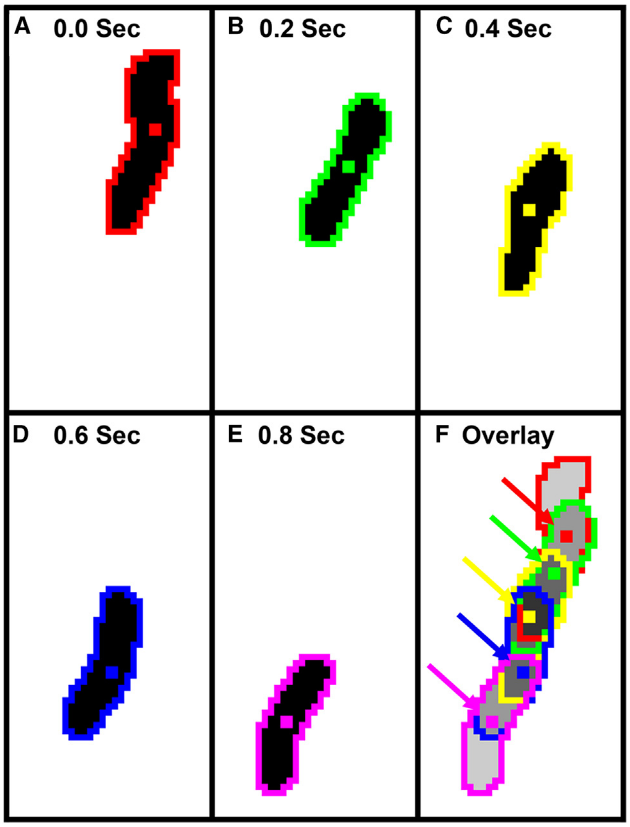

FIGURE 4 Measurement of object velocity via centroid tracking. Examination of the filament highlighted in Fig. 3 *B* reveals how the program calculates the area of the object and positions the centroid within each frame of the first five frames of the movie (*A–E*, respectively). (*F*) Demonstrates the overlay of all five frames and the position of the centroid within each frame. The distance for each frame is determined by using Euclidean distance to calculate the distance, with the assumption that the filament, and thus the centroid, traveled in a relatively straight line from one frame to the next (*colored arrows*).

package to calculate the object positions and attributes, such as size and paths of motion, while also taking advantage of multithreaded processing to decrease the time spent doing these calculations (28). The Trackpy package simplifies centroid tracking into three distinct steps using Python code. First, objects are located and defined with parameters such as minimum size and shape (Fig. 5). Second, the coordinate locations of the objects are obtained and saved as a temporary data frame. Third, the object coordinates are linked together into paths with the Crocker-Grier linking algorithm (29). At this point, the object's positional coordinates from frame to frame, as well as the object's size, and various attributes are stored in memory for use in calculating Philament's output (Table 1). As shown in Fig. 5, this improved object-tracking and path-linking algorithm produces a stark increase in the length and number of objects tracked when compared with objects tracked by wrMTrck.

TABLE 1 Raw object positional data returned by TrackPy

Frame	Particle	x	y	Mass
0	0	597.1512	6.948615	4481.702
1	0	596.34	7.233352	4124.727
2	0	596.1169	7.173213	4005.06
…	…	…	…	…
49	0	597.5523	5.953089	3637.708
0	1	307.174	284.2719	4402.076
1	1	306.3991	282.4232	4364.779
2	1	306.1741	280.5712	4504.637
…	…	…	…	…
10	1	300.9601	273.8124	3876.593
0	2	630.4716	293.7787	4507.463
1	2	641.7049	290.708	4768.711
2	2	648.5296	289.3021	4912.127
…	…	…	…	…

This is a sample of the Full Object Data Output showing how the data are compiled before being processed. Note, the values for the x and y columns are in pixels (not microns).

When comparing the paths (the linking of the motion of the centroid pointed out in Fig. 4) derived from either wrMTrck (Fig. 5 *D*) or Philament (Fig. 5 *E*) analysis to that of the averaged thresholded image (Fig. 5 *C*), the paths tracked by wrMTrck do not always line up with the movements of the filaments (Fig. 5 *D*, *red arrows*). Meanwhile, using Philament, nearly all of the motions of the filaments are accounted for by lines being drawn over the tracks (Fig. 5 *E*, *green arrows*). Note, paths drawn in Fig. 5 *E* were selected by the blue arrows to mark a single filament in each case that Philament initially "lost" but then regained. This feature of Philament, which is not possible with wrMTrck, is the ability to "remember" objects that are momentarily lost from view. If the particle is regained within a certain time frame (set in the Object Tracking Memory on the opening GUI, see supporting material), Philament resumes tracking the object. This is possible because Philament can remember the object size and, for the number of frames set, will "look" for that object within the set search radius until it has been found or the number of frames has been reached. In the path file, this is shown by a break in the colored path line indicating that the filament was "lost," and then reacquired. Philament recognizes these regained objects as the same object that was previously lost, and subsequently continues their tracking, without inserting empty instantaneous velocities, the frame to frame velocity, for the frames that were untracked. Since the lost objects do not remain stationary while they are being reacquired by Philament, the instantaneous velocity would be artificially high due to the distance covered while the program looks for the filament again. This is corrected by dividing the distance covered between the frame lost and the frame recaptured by the time it was lost, assuming the filament traveled in a straight line.

TABLE 2 Example of the final data output derived from the full object data output

A	B	C	D	E	F	G	H	I	J	K	L	M	N
Avg_Obj_Size	Std_Obj_Size	File	Particle	FirstX	FirstY	First_Frame	Avg Speed	Speed Std	Path Length	Displacement	0.2 (s)	0.4 (s)	0.6 (s)
15.8	1.28	6	0	597.15	6.95	0	0.4726	0.3569	3.4026	0.1492	0.5975	0.1606	0.4720
16.93	0.88	6	1	307.17	284.27	0	1.1655	0.7150	2.0978	1.6911	1.3931	1.2966	1.4170
18.48	2.9	6	2	630.47	293.78	0	5.3469	2.2826	12.8325	12.3824	8.0936	4.8428	4.4758
18.29	2.92	6	3	311.98	298.57	0	3.1583	2.9083	28.4248	18.8750	0.7611	1.4933	1.5407
18.94	0.35	6	4	609.47	300.83	0	6.7231	3.0530	13.4462	12.9769	6.9313	5.4590	4.6587
17.16	1.08	6	5	374.46	309.70	0	0.5199	0.3497	5.0948	0.3029	0.1370	0.1599	0.6258

Philament returns a CSV file for each condition that contains information about the object size (columns A and B), File number (C), and Particle or Filament number (D). The file also contains the position of the centroid (E and F) as well as the first frame (G) where the object was detected. Philament also reports the average speed (H), as calculated by the average of the instantaneous velocities (columns L–N abbreviated) as well as the standard deviation of the velocities (I). Finally, the program outputs the Path Length (J), which is the total movement of the filament regardless of direction and Displacement (K), which is the distance between the centroids' initial and final positions.

Similarly, for filaments that break into two separate filaments, Philament can continue tracking the original object and begin a new path for the fragment. The tracking algorithm identifies the filament fragment that most "resembles" the intact object, using object position, size, and shape as parameters. If the separation is so drastic that neither filament resembles the previously tracked object, Philament begins new paths for both objects. This functionality is also used for two separate filaments that momentarily overlap. As described above with filaments that are momentarily lost from view, Philament will "look" for the objects until they separate or until the number of frames is reached.

Data analysis, filtering

Having extracted the data from the movies, it is necessary to compile the data into a usable format that will allow for further manipulation. The tracking "engine" extracts the positional data for each tracked object (x, y position), as well as the object brightness (mass multiplied by the object size in pixels), object eccentricity (a measure of circularity), and radius of gyration of its Gaussian-like profile that is exported in the Full Data Object when selected. These raw data contain all the positional information for each frame of the video and are not processed or altered, except for sorting the data in increasing order, where particle 1 frame 1 is the first row, followed by particle 1 frame 2, etc.

Since Philament has been designed to extract data from IVM movies, no attempts were made to introduce filters to help eliminate unwanted filaments. The only filtering this program does is to eliminate filaments (referred to as particles in the program) if they are tracked for a single frame. These dropped objects and their single positional datum is still available in the full object data set but otherwise eliminated from the final compiled data sheet.

As such, the end user will have to develop a filtering system, on their own, to eliminate "bad" filaments; for example, ones that are stalled, those that are moving in a small region of space, or those that are only on screen for a small amount of time. To aid in processing the data, Philament has been designed to output the data in a spreadsheet-compatible CSV file format for ease of opening in spreadsheet applications. Philament presents the final data as shown in Table 2.

The object size (column Avg Obj Size, Std Obj Size) refers to the filament size and is reported in pixels. It is derived from the output column "mass" and refers to the brightness of the particle. This value is obtained by averaging the "mass" values from each frame the particle was tracked and dividing that number by 255 (the value of the pixel brightness after

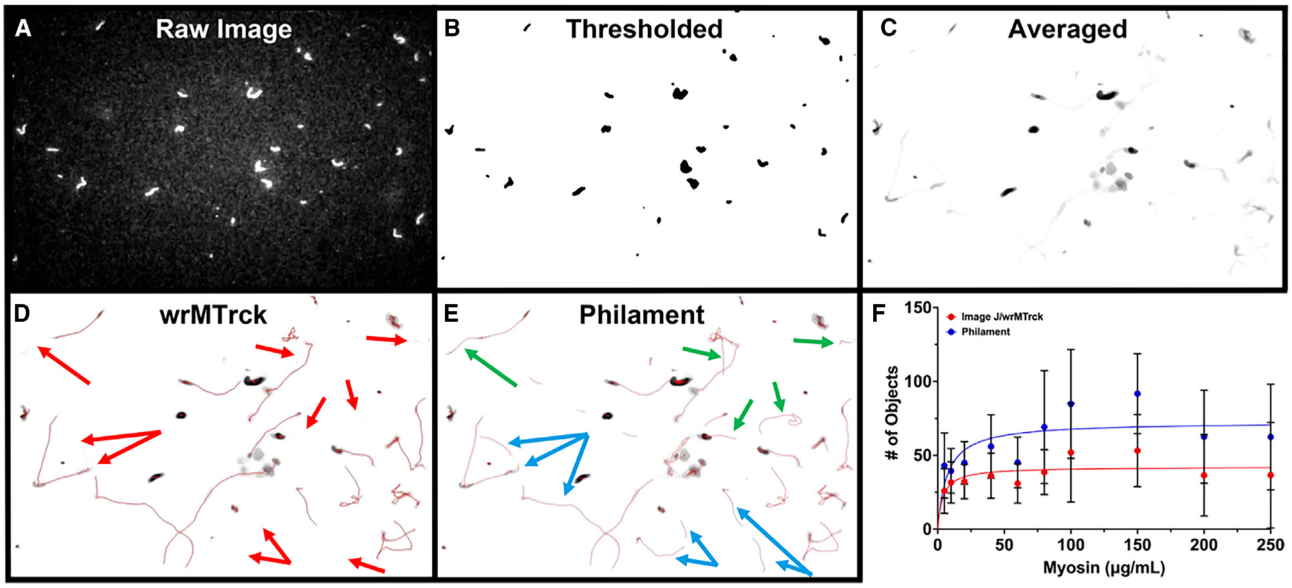

FIGURE 5 Object tracking and path generation. (*A*) Shows the first frame from a representative raw movie. (*B*) The first frame of the Philament thresholded image. (*C*) The 50-frame average of this movie. (*D* and *E*) The "Paths" files that are created by either wrMTrck (*D*) or Philament (*E*) overlaid on the 50-frame average. As can be seen, Philament is able to track more filaments and for a longer period of time than can wrMTrck (*red arrows* in *D* mark the identical filaments detected using Philament in *E*, which are marked with *green/blue arrows*). Note, the paths marked by the joined blue arrows mark the same filament showing examples of where Philament lost and then regained detection of the filament to continue to track it. Overall, the use of Philament leads to more objects being found per movie than with wrMTrck (*F*), error bars are mean ± SD.

binarization). Since filaments in Philament are considered roughly elliptical with the area of the ellipse being [a*b*pi]: a and b are one-half the length and width axes of the ellipse, respectively. Measuring, and averaging, the long axis (a) of the filaments from a sample of videos provides an averaged b*pi value which can then be used to accurately estimate the filament lengths of the filaments in pixels within the movie. This value then can be converted to microns by multiplying that by the size of the pixels for the camera being used.

The next two columns (File and Particle) refer to the movie number of the experimental condition, in this case, movie 6 out of 10, and the Particle number is the filament number. Continuing to the right the "FirstX" and "FirstY" columns refer to the centroid position of that particle on the first frame it was found. The Avg Speed and Speed Std columns are derived from the instantaneous velocities and are the average and standard deviation of all the instantaneous velocities obtained while tracking the filament. The Path Length column is the total distance the filament traveled, regardless of the direction, while the Displacement column is purely the distance of the centroids from the first frame the filament was tracked to the last frame the filament was tracked. These two columns are offered to allow users the ability to calculate the Length over Displacement (L/D) value as a means of isolating filaments that are "stuck" on a region of the coverslip. When the (L/D) value is close to 1, the filaments are traveling in a straight line. Conversely, as this number approaches 0, the filaments are going in smaller and smaller circles and may therefore need to be filtered out.

For purposes of the data reported here, we used three filters to eliminate filaments that we deemed as not acceptable for inclusion into the data set. These filters were: a frame number filter (to eliminate filaments tracked for less than 5 frames; i.e., 1/10th of total frames), an L/D filter (to eliminate filaments that were either stuck in a small loop or were moving via Brownian motion), and, lastly, a "stall" filter used to eliminate filaments that were not moving or moving very slowly. For our experimental purposes, the stall filter was chosen to be 1 pixel per second (0.139 μm/s) as the lower limit. The values of these filters can be adjusted by the user or can also include other parameters not used by our group.

Using these filters, we get the results as shown in Fig. 6, where the velocities of the Philament-analyzed movies (Fig. 6 *A*) are faster on average than the wrMTrck-analyzed movies (Fig. 6 *B*), with a more tightly clustered fraction motile (Fig. 6, *C* versus *D*). Based on the results shown in Fig. 5, this disparity in velocity and fraction motile is due to wrMTrck's inability to accurately track all the objects in the movie.

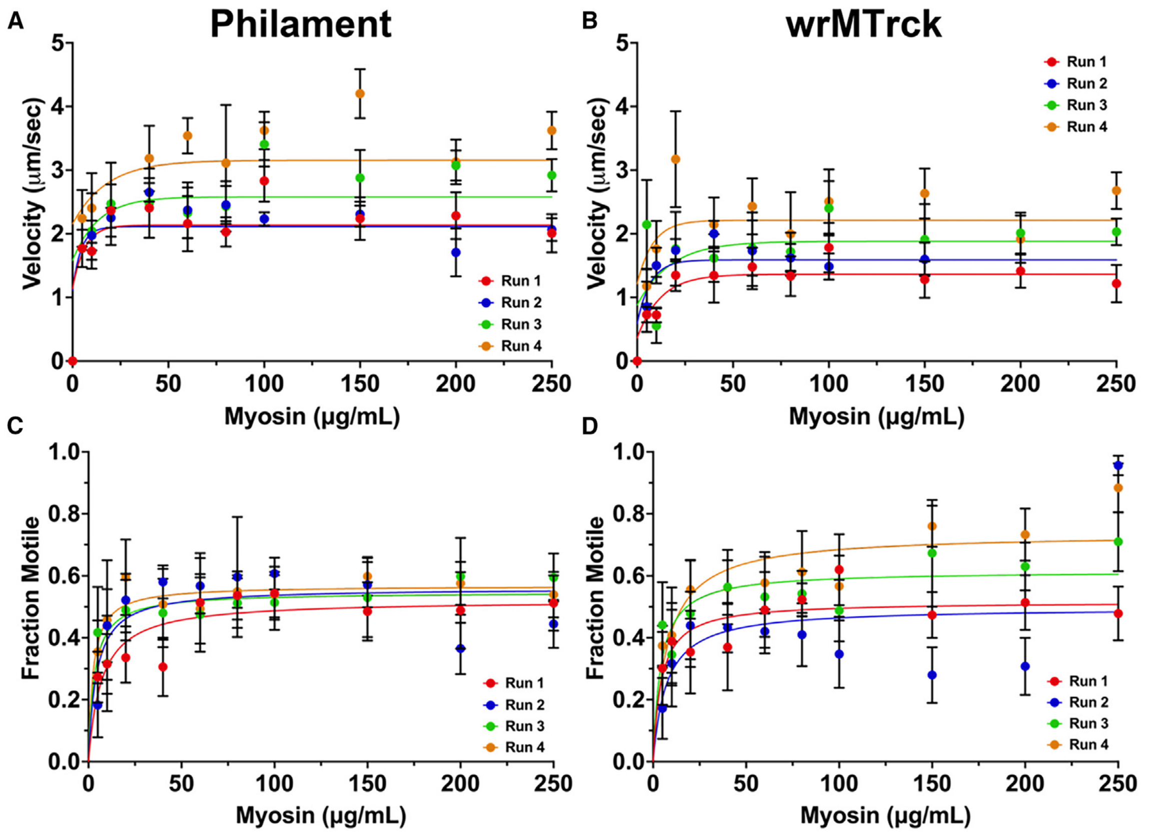

FIGURE 6 Philament reports faster filament velocities, and more uniform fraction motile versus wrMTrck. Due to enhanced object detection and tracking capabilities, Philament can capture faster average videos (*A*) than wrMTrck (*B*) while still maintaining the same trends. As shown in (*C*) (Philament) and (*D*) (wrMTrck), Philament has improved capabilities to find and track all the objects in the frame. Thus, the fraction motile is more uniform and consistent between runs providing the user more confidence in the velocity values reported. Error bars are mean ± SD.

Instantaneous velocity versus average velocity

Although previous experiments using IVM have eliminated filaments with staggered motion (frequent starts and stops) under the assumption that they were "stuck" filaments (6,7), it can be argued that the inclusion of these data is vital to accurately describe the interaction of myosin with actin. For example, in the case of myosin, because it needs to "search" for the next binding site as it walks along the actin filament, the movement of myosin along the actin filament is inherently Brownian motion in nature. In IVM analysis, for actin filaments to continue moving forward, a critical number of myosin heads must bind and pull with enough force to move the filament while simultaneously detaching previously bound myosin heads. The probability of binding is therefore dependent on a variety of factors such as the distance to the actin filament, binding site distance, the orientation of the myosin head(s) with respect to the direction of actin filament motion, and the number of myosin heads near the actin filament (30–33). Thus, anything that alters the fluidity/smoothness of actin filament motion may influence the probability of actomyosin interactions and be physiologically important. The fluidity of a filament's motion can be described as the relationship between the filament's average of the instantaneous velocity versus the standard deviation of the instantaneous velocity for each particle. As shown in Fig. 7, when plotting the standard deviation of the sliding velocity to the average velocity of each filament we observe that, as myosin concentration increases, the slope of this relationship goes down (Fig. 7 *B*). This agrees with the proposed concept that having more myosin heads available to bind and interact with actin increases the smoothness of motility, leading to a shallower slope.

Accuracy of filament tracking

To ensure that Philament can accurately locate and identify fluorescently labeled actin filaments, we initially compared the tracking ability of Philament with either hand measuring (mTrackJ) or semiautomated tracking using wrMTrck, a plugin for ImageJ. We randomly selected five movies from our portfolio of more than 3000 previously recorded movies that had a wide range of sliding velocities and filament sizes. As shown in Fig. 8, over all five movies, Philament was able to find nearly the same number of objects (filaments) as mTrackJ (Fig. 8 *A*) and report back similar velocities (Fig. 8 *B*) as the hand measurement. Conversely, wrMTrck had issues in both finding objects (~25% reduction, Fig. 8 *C*) and getting accurate velocity measurements, in comparison with hand measuring (~56% reduction, Fig. 8 *D*), indicating that Philament can find more objects and track them better than wrMTrck.

DISCUSSION

Here, we describe a new batch analysis program for the rapid analysis of IVM recordings that vastly increases the speed of the analysis of sliding filaments over a bed of motor proteins. Philament can analyze videos more than 10 times faster than wrMTrck, a powerful plugin for ImageJ that was last updated in 2011 (~5.6 vs. ~79.0 s/recording, respectively) allowing for truly high-throughput analysis of IVM videos. An easy-to-use Windows compiled version (~7 s/recording) is also available for those who prefer. Philament will open investigatory angles as users unfamiliar with coding or IVM analysis may enter the field with a lower barrier of entry and can utilize IVM to efficiently examine a plethora of mechanisms contributing to and regulating cell motility. Furthermore, Philament offers a simpler way to eliminate user bias in selecting and analyzing filaments thereby allowing for more reliable and reproducible data reporting among different users. Finally, the ability to plot the standard deviation of the instantaneous velocity to the average velocity (i.e., smoothness of motion) will allow users to directly examine the impact an intervention has on actomyosin interactions.

Overall, while there are other programs available that have more extensive features (e.g., FASTrack), we contend that the ease of use and the ability to update or customize Philament provides an appealing and compelling option for both experienced users and new users interested in IVM analysis.

Limitations of Philament

While automated program analysis software for IVM is preferred to manual tracking programs, due to its elimination of user bias in selecting filaments and the significant increase in throughput, limitations to Philament exist. The main limitation, and the most important one, is that image quality must be kept uniform for the entire data set. Since Philament does the thresholding in a batch manner, if there are videos that differ substantially with respect to intensity, that is, are brighter or dimmer than most videos being collected, Philament will have trouble analyzing the data. As such, for accuracy, attention must be given to the image brightness during the data collection, to ensure that videos have uniform brightness throughout the entire experimental run. Note that if the images have drastically different brightness, the data is not lost, since these videos can be analyzed separately.

To account for unequal brightness between files, Philament has built-in error handling and skips any files whose object paths cannot be accurately predicted from frame to frame. As path linking is a probability-based calculation, complete certainty is impossible, therefore this is another limitation. When

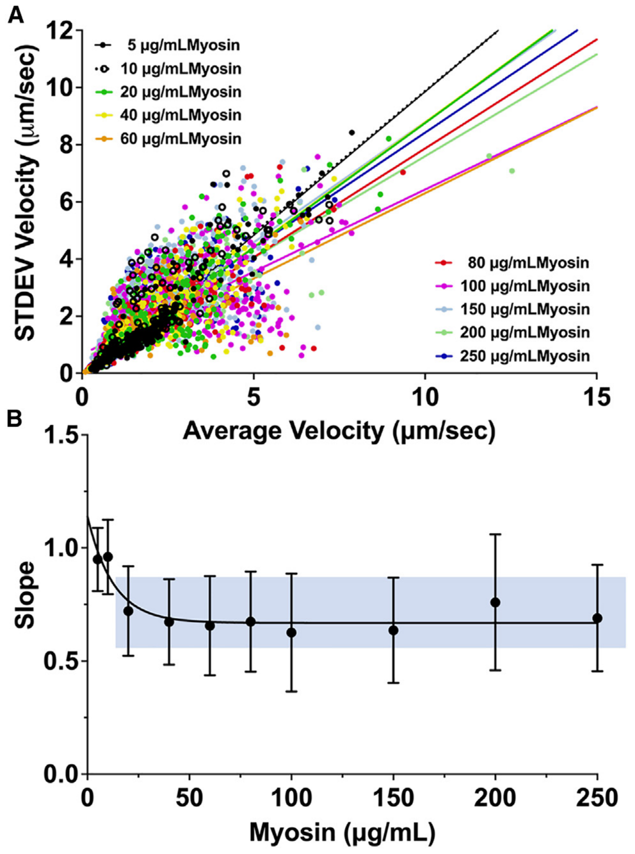

FIGURE 7 An increase in myosin concentration results in smoother-moving filaments. (*A*) Illustrates the linear relationship of one representative experimental run between the SD of the instantaneous velocity versus the average velocity of the filaments referred to here as the "smoothness of motion." In (*B*) the average of all the slopes from every video collected demonstrates that when the myosin concentration was increased to 20 μg/mL or higher, the slope of the linear comparison decreased, suggesting that the smoothness of motion increased, indicating fewer starts and stops in the motion of the filaments. Error bars are mean $\pm$ SD. All data points in the shaded area are significantly different from 5 and 10 μg/mL myosin with a *p* value of $\leq$0.0001, except for the 200 μg/mL with a *p* value of $\leq$0.01.

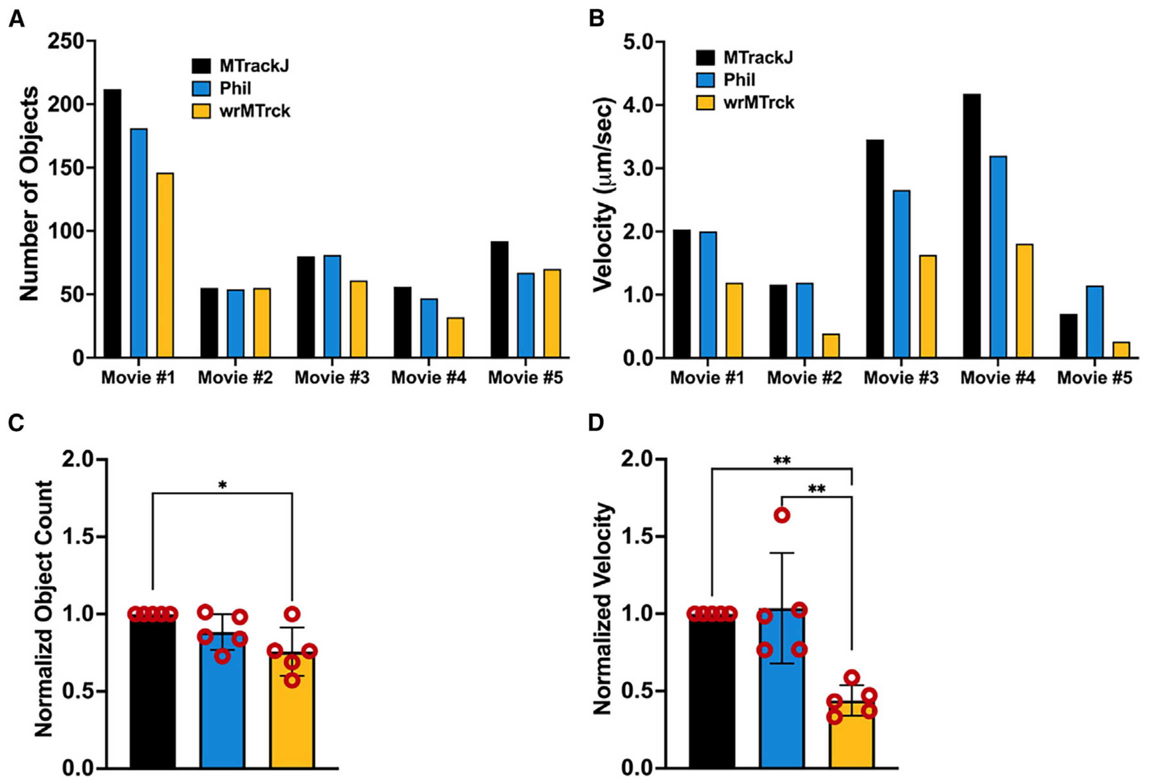

FIGURE 8 Philament offers a similar level of object capture as manual hand measurements. Five movies were analyzed to obtain the number of objects (*A*) and unfiltered velocities (*B*). As shown in (*C* and *D*), Philament (in *blue*) returns similar results as found by performing the measurements by hand (mTrackJ, *black*). Interestingly, the plugin wrMTrck (*orange*) had difficulty measuring speeds and the number of objects in the same five movies. As shown in (*C*), wrMTrck found significantly fewer filaments, compared with mTrackJ; approximately 25% fewer objects were identified while Philament identified ~11% fewer filaments. In addition, wrMTrck returned significantly slower velocities (~56%) (*D*) as compared with Philament's ~4% faster filament velocities. The values in (*A* and *B*) were normalized within each movie by dividing output velocities/no. of objects from wrMTrck or Philament by the MTrackJ values for each of the five movies. These five normalizations are grouped into (*C* and *D*). Bars = ±SD.

first using Philament, care must be taken to optimize the fitting parameters. It is recommended that users pick one high-quality recording and experimentally determine the best parameters, focusing on Object Diameter and Search Radius first, and using the "Create Path Files" option. By analyzing one file and comparing the object paths to the tracked paths by eye, users can adjust the parameters until the output paths match the true object paths. For example, if the Object Diameter is set too high, smaller filaments may be missed and will be absent in the paths file. Failure to tune parameters may result in inaccurate data. Finally, to obtain rapid throughput on the data analysis, the program was created to handle an equal number of videos per condition. While convenient for our purposes, if the goal is for high-throughput screening, the user will need to ensure each condition has the same number of videos.

SUPPORTING MATERIAL

Supporting material can be found online at https://doi.org/10.1016/j.bpr.2024.100147.

AUTHOR CONTRIBUTIONS

R.M.B. conceived and wrote the program, performed experiments, and wrote the manuscript. G.P.F. conceived and wrote the program and wrote the manuscript. C.C.G. wrote the manuscript, managed the project, and funded the project.

ACKNOWLEDGMENTS

We would like to acknowledge both Evan Tisdale and May Riggle for their guidance while learning Python and for their knowledge of software development. This research was supported by the University of Arizona Undergraduate Biology Research Program with funds from the Office of the Provost (R.M.B.), NIH R01HL123078, RO1HL164644, and R01GM120137, as well as the Czarina M. and Humberto S. Lopez Endowed Chair for Excellence in Cardiovascular Research (to C.C.G.).

DECLARATION OF INTERESTS

The authors declare no competing interests.

REFERENCES

1. Greenberg, M. J., T. R. Mealy, ..., J. R. Moore. 2010. The direct molecular effects of fatigue and myosin regulatory light chain

phosphorylation on the actomyosin contractile apparatus. *Am. J. Physiol. Regul. Integr. Comp. Physiol.* 298:R989–R996.

2. Greenberg, M. J., and J. R. Moore. 2010. The molecular basis of frictional loads in the in vitro motility assay with applications to the study of the loaded mechanochemistry of molecular motors. *Cytoskeleton (Hoboken).* 67:273–285.
3. Vikhorev, P. G., N. N. Vikhoreva, …, J. C. Sparrow. 2010. In vitro motility of native thin filaments from Drosophila indirect flight muscles reveals that the held-up 2 TnI mutation affects calcium activation. *J. Muscle Res. Cell Motil.* 31:171–179.
4. Homsher, E., M. Nili, …, L. S. Tobacman. 2003. Regulatory proteins alter nucleotide binding to acto-myosin of sliding filaments in motility assays. *Biophys. J.* 85:1046–1052.
5. Homsher, E., D. M. Lee, …, L. S. Tobacman. 2000. Regulation of force and unloaded sliding speed in single thin filaments: effects of regulatory proteins and calcium. *J. Physiol.* 524 Pt 1:233–243.
6. Homsher, E., B. Kim, …, L. S. Tobacman. 1996. Calcium regulation of thin filament movement in an in vitro motility assay. *Biophys. J.* 70:1881–1892.
7. Månsson, A., and S. Tågerud. 2003. Multivariate statistics in analysis of data from the in vitro motility assay. *Anal. Biochem.* 314:281–293.
8. Vikhorev, P. G., N. N. Vikhoreva, and A. Månsson. 2008. Bending flexibility of actin filaments during motor-induced sliding. *Biophys. J.* 95:5809–5819.
9. Vikhoreva, N. N., and A. Månsson. 2010. Regulatory light chains modulate in vitro actin motility driven by skeletal heavy meromyosin. *Biochem. Biophys. Res. Commun.* 403:1–6.
10. Korten, T., E. Tavkin, …, S. Diez. 2018. An automated in vitro motility assay for high-throughput studies of molecular motors. *Lab Chip.* 18:3196–3206.
11. Sommese, R. F., S. Nag, …, K. M. Ruppel. 2013. Effects of Troponin T Cardiomyopathy Mutations on the Calcium Sensitivity of the Regulated Thin Filament and the Actomyosin Cross-Bridge Kinetics of Human β-Cardiac Myosin. *PLoS One.* 8, e83403.
12. Nussbaum-Krammer, C. I., M. F. Neto, …, R. I. Morimoto. 2015. Investigating the spreading and toxicity of prion-like proteins using the metazoan model organism C. elegans. *J. Vis. Exp.* 52321.
13. Farman, G. P., M. J. Rynkiewicz, …, J. R. Moore. 2018. HCM and DCM cardiomyopathy-linked alpha-tropomyosin mutations influence off-state stability and crossbridge interaction on thin filaments. *Arch. Biochem. Biophys.* 647:84–92.
14. Orzechowski, M., S. Fischer, …, G. P. Farman. 2014. Energy landscapes reveal the myopathic effects of tropomyosin mutations. *Arch. Biochem. Biophys.* 564:89–99.
15. Farman, G. P., P. Muthu, and J. R. Moore. 1985. Impact of Familial Hypertrophic Cardiomyopathy-linked Mutations in the N-Terminus of the RLC on beta-Myosin Cross-bridge Mechanics. *J. Appl. Physiol.* 2014.
16. Hauke, L., A. Primeßnig, …, F. Rehfeldt. 2023. FilamentSensor 2.0: An open-source modular toolbox for 2D/3D cytoskeletal filament tracking. *PLoS One.* 18:e0279336.
17. Kapoor, V., W. G. Hirst, …, S. Reber. 2019. MTrack: Automated Detection, Tracking, and Analysis of Dynamic Microtubules. *Sci. Rep.* 9:3794.
18. Gallois, B., and R. Candelier. 2021. FastTrack: An open-source software for tracking varying numbers of deformable objects. *PLoS Comput. Biol.* 17:e1008697.
19. Aksel, T., E. Choe Yu, …, J. A. Spudich. 2015. Ensemble force changes that result from human cardiac myosin mutations and a small-molecule effector. *Cell Rep.* 11:910–920.
20. Moore, J. R., E. B. Krementsova, …, D. M. Warshaw. 2001. Myosin V exhibits a high duty cycle and large unitary displacement. *J. Cell Biol.* 155:625–635.
21. Veigel, C., S. Schmitz, …, J. R. Sellers. 2005. Load-dependent kinetics of myosin-V can explain its high processivity. *Nat. Cell Biol.* 7:861–869.
22. Van Rossum, G., and F. L. Drake. 2011. The Python Language Reference Manual. Network Theory Limited.
23. Bradski, G. 2000. The OpenCV library. *Dr. Dobb's J.* 25:120–125.
24. van der Walt, S., J. L. Schönberger; …, scikit-image contributors. 2014. scikit-image: image processing in Python. *PeerJ.* 2:e453.
25. Lam, S. K., A. Pitrou, and S. Seibert. 2015. Numba: a LLVM-based Python JIT compiler. *In* Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC. Association for Computing Machinery, Austin, Texas, Article 7.
26. McKinney, W. pandas: a Foundational Python Library for Data Analysis and Statistics. 2011.
27. Justusson, B. I. 1981. *Median Filtering: Statistical Properties*, in *Two-Dimensional Digital Signal Prcessing II: Transforms and Median Filters*. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 161–196.
28. Allan, D., T. Caswell, and C. M. van der Wel. 2019. Soft-matter/trackpy: Trackpy v0.4.2. https://doi.org/10.5281/zenodo.2019.3492186.
29. Crocker, J. C., and D. G. Grier. 1996. Methods of Digital Video Microscopy for Colloidal Studies. *J. Colloid Interface Sci.* 179:298–310.
30. Tyska, M. J., and D. M. Warshaw. 2002. The myosin power stroke. *Cell Motil Cytoskeleton.* 51:1–15.
31. Tyska, M. J., D. E. Dupuis, …, S. Lowey. 1999. Two heads of myosin are better than one for generating force and motion. *Proc. Natl. Acad. Sci. USA.* 96:4402–4407.
32. Sugiura, S., N. Kobayakawa, …, H. Sugi. 1996. Different cardiac myosin isoforms exhibit equal force-generating ability in vitro. *Biochim. Biophys. Acta.* 1273:73–76.
33. Sugiura, S., N. Kobayakawa, …, H. Sugi. 1998. Comparison of unitary displacements and forces between 2 cardiac myosin isoforms by the optical trap technique: molecular basis for cardiac adaptation. *Circ. Res.* 82:1029–1034.