WormPose: Image synthesis and convolutional networks for pose estimation in *C. elegans*

Laetitia Hebert[a], Tosif Ahamed[a], Antonio C. Costa[b], Liam O'Shaugnessy[b], Greg J. Stephens[a,b*]
[a]*OIST Graduate Univerisity, Onna, Okinawa 904-0495, Japan*
[b]*Department of Physics and Astronomy, VU University Amsterdam, 1081HV Amsterdam, The Netherlands*

An important model system for understanding genes, neurons and behavior, the nematode worm *C. elegans* naturally moves through a variety of complex postures, for which estimation from video data is challenging. We introduce an open-source Python package, WormPose, for 2D pose estimation in *C. elegans*, including self-occluded, coiled shapes. We leverage advances in machine vision afforded from convolutional neural networks and introduce a synthetic yet realistic generative model for images of worm posture, thus avoiding the need for human-labeled training. WormPose is effective and adaptable for imaging conditions across worm tracking efforts. We quantify pose estimation using synthetic data as well as N2 and mutant worms in on-food conditions. We further demonstrate WormPose by analyzing long (~ 10 hour), fast-sampled (~ 30 Hz) recordings of on-food N2 worms to provide a posture-scale analysis of roaming/dwelling behaviors.

INTRODUCTION

All animals, including humans, reveal important and subtle information about their internal dynamics in their outward configurations of body posture, whether these internal dynamics originate from gene expression [1], neural activity [2], or motor control strategies [3]. Estimating and analyzing posture and posture sequences from high-resolution video data is thus a general and important problem, and the basis of a new quantitative approach to movement behavior (for reviews see e.g. [4, 5]).

The roundworm *C. elegans*, important on its own as a model system (see e.g. [6]), provides an illustrative example, where "pose" can be identified as the geometry of the centerline extracted from worm images [7]. Even with a relatively simple body plan, identifying the centerline can be challenging due to coiling and other self-occluded shapes, Fig. 1. These shapes occur in important behaviors such as an escape response [8, 9], among mutants [10] and are a yet unanalyzed component in increasingly copious and quantitative recordings such as the Open Worm Movement Database [11].

Classical image skeletonization methods can be used to identify the worm centerline for non-overlapping shapes [7] and are employed in widely-used worm trackers because of their simplicity and speed. For coiled or self-overlapping postures, more advanced statistical models combine image features such as edges with a model of the worm's centerline [10, 12–14]. However, such image features are not always visible and are not robust to changes in noise or brightness, often requiring data-specific engineering which reduces portability. Another recent technique utilizes an optimization algorithm by searching for image matches in the "eigenworm" posture space [9], but is limited in efficacy by the slow nature of multi-dimensional image search and by the low resolving power of a comparison metric, which uses only a binary version of the raw image.

With the ability to extract complex visual information about articulated objects, methods built from convolutional neural networks (CNNs) offer a new, promising direction. CNNs are the foundation for recent, remarkable progress in markerless body point tracking [15–17], including worm posture [18, 19]. However, intensive labeling requirements by human annotators, even if assisted by technology [20], as well as the ambiguity of which or exactly how many points to label, offer a barrier to the usefulness of CNNs in posture tracking and beyond. Body point marking is challenging in the case of worm images where the annotation task is to label enough points along the worm body to reconstruct the posture. While human annotators can quickly pinpoint the extremities of the worm body, other landmarks are less obvious. In some recordings, it is even difficult to distinguish the worm head from the tail, which makes the labeling error-prone and imprecise. Furthermore, the labeling is specific to the recording conditions and can be hard to generalize across changes in resolution, organism size, background, illumination, and to rare posture configurations not specifically isolated.

We describe an algorithm, WormPose, for pose estimation in *C. elegans* containing two principal advances: (1) We create a generative model of worm shape which we combine with a new technique for producing synthetic but realistic worm images. These images are used for network training, thus circumventing the difficulty and ambiguity of human labeling, and can be easily adapted to different imaging conditions (2) We develop a CNN to reliably transform worm images to a centerline

* Corresponding author: g.j.stephens@vu.nl

curve. We demonstrate our approach using the on-food behavior of N2 and mutant worms and use our results to provide a new posture-scale analysis of roaming and dwelling behavioral states.

METHODS

Code Availability: WormPose is open-source and free with a permissive 3-Clause BSD License. The source code is available here: `https://github.com/iteal/wormpose`, and can be installed from the Python package index: `https://pypi.org/project/wormpose`.

Data Requirements: Our focus is on resolving coiled, overlapping, blurred, or other challenging images of a single worm. We assume that the input data consists of videos of a single moving worm and that most of the non-coiled frames are analyzed beforehand, for example by Tierpsy tracker [21]. For each (non-coiled) frame, we require the coordinates of equidistant points along the worm centerline, ordered from head to tail, and the worm width for the head, midbody, and tail (defined in [22]). We also use the recording frame rate. WormPose 1.0 does not detect the head of the worm, so we also expect that the labeled frames provide the head-tail position at regular intervals throughout the video.

Processing Natural Images: From a dataset described as above, we process worm images to focus on the worm object of interest. Broadly, we first segment the worm in the image and set all non-worm pixels to a uniform color. Then we either crop or extend to create a square image of uniform size with the worm in the center, cleaned of noise and non-worm objects.

The specific process of segmenting a single worm in an image can be adapted to each recording condition. For concreteness, we provide a simple OpenCV [23] implementation that is sufficient for most videos of the Open Worm Movement Database [11]. Raw images from the video are first processed by a Gaussian Blur filter with a window size of 5 pixels, and then thresholded with an automatic Otsu threshold to separate the background and the foreground. The morphological operation "close" is applied to fill in the holes in the foreground image. We use a connected components function to identify the objects belonging to the foreground. To focus on objects located at the center of the image, we crop the thresholded image on each side by an amount consisting of 15% of the size of the image. We isolate the largest blob in this cropped image as the worm object of interest. We calculate the background color as the average of the background pixels of the original image, and assign this background value to all pixels that do not belong to the worm object. All processed images are then either cropped or extended to be the same width and height, with the worm object set in the center. We set the default value for the processed image size as the average worm length of the biggest worm in the dataset, a size large enough to encompass all examples. Alternatively, the image size can be set by the user, and the images will be resized with linear interpolation, which is useful to speed up computation on large images. The minimum image size is 32×32 pixels.

Generating Worm Shapes: We generate realistic worm shape through a Gaussian Mixture Model (GMM) Fig. 2(A), which we fit to a collection of resolved body postures obtained from previous analysis [7, 9]. The use of a generative model of body shapes enables the generation of an arbitrarily large training set for the network, with shapes that respect the overall correlations between body parts while generalizing to more complex postures. We parameterize worm shape by a 100-dim vector of angles $\vec{\theta}$, formed by measuring the angle between 101 points equally spaced along the body's centerline. Uncoiled shapes were obtained using classical image tracking to extract $\vec{\theta}$ directly from images [7]. Coiled shapes were obtained in [9] by searching the lower-dimensional space of eigenworm projections ($d = 5$, obtained through Principal Component Analysis of the space of $\{\vec{\theta}\}$), to find the combination of eigenworm coefficients that best matches a given image and projecting these back into the $\vec{\theta}$ space. Using the classical image analysis results from [7] allows us to expand the space of possible $\vec{\theta}$ beyond the one captured by the first 5 eigenworms used in [9]. We use an equal population of coiled and uncoiled postures from N2 worms foraging off-food and sample uniformly according to the body curvature as measured by the third eigenworm projection, a_3. This yields a training set of $\sim$ 15000 $\vec{\theta}$ vectors. We fit the GMM through an Expectation-Maximization algorithm which finds the set of N Gaussian components that maximizes the likelihood (see e.g. [24]). The full model is parameterized by the mean and covariance of each Gaussian, and the weight associated with each Gaussian component. We assess the trade-off between model complexity and accuracy with Akaike's information criterion, which indicates that $N \sim 250 - 275$ components would be an appropriate choice, Fig.(S2). We train the GMM using `sklearn.mixture.GaussianMixture` from scikit-learn in Python [25] with 270 components.

Generating Synthetic Images: We build a synthetic image generator to produce a worm image with a specific posture and with the same appearance as a reference image, Fig. 2(B). Such synthetic images have a similar appearance to real images processed as described above.

We exclusively use classical image processing techniques, including image warping and alpha blending, to effectively bend a known worm centerline from a reference image into a different posture. The reference image is typically of a non-overlapping worm, with its associated labeled features: (1) the skeleton as a list of N_S coordinates (S_x, S_y) equidistant along the centerline, and (2) the worm width at three body points: head, midbody and tail. To create a new synthetic image we first draw a centerline $\vec{\theta}$ of size 100 from the GMM worm shape generator. We produce target skeleton coordinates $\{S_x, S_y\}$ through the transformation

$$\begin{aligned} S_x(i+1) &= S_x(i) + dS\cos\theta_i \\ S_y(i+1) &= S_y(i) + dS\sin\theta_i \end{aligned} \tag{1}$$

for $i = 1, 2 \ldots N_S$. The length element dS is determined by dividing the worm length of the reference image by $N_S - 1$ and we set the origin by centering the skeleton in the middle of the target image, Fig. 2(A, right). If needed, the target skeleton is resampled to have the same number of points N_S as the reference skeleton. We use the labeled width for the head, midbody and tail to calculate the worm width (in pixels) at all skeleton points $ww(i)$:

```
ww[0:head]=head_width
ww[head:midbody]=interp(head_width, midbody_width)
ww[midbody:tail]=interp(midbody_width, tail_width)
ww[tail:Ns-1]=tail_width
```

In a "reverse skeletonization", we take small rectangular image patches of size (l, w) from the reference image and add them along the target skeleton, Fig. 2(B). Along each skeleton we create rectangles with $\hat{l}$ oriented along the direction formed by the skeleton points i and $i+step$, and width $w(i) = w_{multiplier} \times ww(i)$. The parameter *step* determines the length l of the rectangle. For each pair of rectangles, we find the affine transformation that maps a rectangle in the reference image to a rectangle in the target image using the function *getAffineTransform* from OpenCV [23]. If *step* is too small (equal to 1), the patches will not overlap which will create discontinuities in the synthetic image, but if *step* is too large, then the patches could be larger than the amount of curvature of the worm. In practice, we set $step = 1/16 \times N_S$. We set $w_{multiplier} = 1.2$, which means the rectangle width will be larger than the actual worm width to include background pixels around the worm body.

For each pair of source-target rectangles, we use the function *warpAffine* from OpenCV [23] to project the pixels from the rectangle in the source image to the coordinates of the target rectangle in the target image. We combine the transformed patches into a single cohesive worm image by iteratively updating a mask image created from the overlapping regions. For each transform, we add the values of the new transformed image containing one patch to the current full image. We then multiply by the mask image set to 1 for non-overlapping areas and 0.5 for overlapping areas. We draw the rectangles from the worm tail so that the last rectangles will be of the worm head, as this configuration is more likely to occur naturally.

The overlapping areas combine seamlessly because of the blending, but some protrusions are still visible, especially when the target pose is very coiled. We eliminate these artifacts by masking the image with a generated image representing the expected worm outline. This mask image is created by drawing convex polygons along the target centerline of the desired worm width, complete with filled circles at the extremities. We apply a median filter with a window size of 3 to smooth the remaining noise due to the joining of the patches. Finally, all non-worm pixels are set to a uniform color: the average of the background pixels in the reference image.

To add diversity to the synthetic images, we include a set of (optional) augmentations. We translate the target skeleton coordinates by a uniform value between 0 and 5% of the image size. We vary the worm length uniformly between 90% and 110%, and the worm thickness multiplier between 1.1 and 1.3. We randomly switch the drawing order from head to tail or the contrary, so that each is equally probable. Finally, we add an extra Gaussian blur filter at the end of the process 25% of the time, with a blur kernel varying between 3% and 10% of the image size or 13 pixels, whichever is smaller.

In WormPose, the Python implementation of the image generator is optimized for speed and memory allocation. Generating a synthetic image of a large size will be slower than a smaller one. It is also faster to limit the number of reference images, as some calculation is cached. The generation is usually split into several processes, and we use a maximum of 1000 reference images per process, chosen randomly. The number of skeleton points N_S from the reference image is flexible and depends on the dataset. If N_S is too small ($N_S \lesssim 20$), the synthetic worm image will be too simplistic compared to the real images. On the other end, increasing N_S too much will decrease the performance, and the resulting synthetic image will not benefit in detail. We routinely use $50 \lesssim N_S \lesssim 100$.

Network architecture and training: For reasons ranging from motion blur to self-obscured postures, it is often difficult to discern the worm's head from the tail, such as in Fig. 3(A). Images with similar worm shape but opposite head-tail locations have quantitatively different centerlines, thus providing a challenge to network training. To handle this ambiguity, we design a loss function that minimizes the difference between the network prediction $\hat{\theta}$ and the closest of two labels: θ_a and $\theta_b = flip(\theta_a) + \pi$, representing the same overall pose but with swapped locations of the head and tail, Fig. 3(B). The output training error is the minimum of the root mean square error of the angle difference $d(\theta_1, \theta_2)$ between the output centerline $\hat{\theta}$ and the two training labels $\{\theta_a, \theta_b\}$ (Fig. 3(C)) with

$$\begin{aligned} \epsilon(\alpha, \beta) &= atan2(sin(\alpha - \beta), \cos(\alpha - \beta)) \\ d(\theta_1, \theta_2) &= \sqrt{\frac{1}{N}\sum_{i=1}^{N} \epsilon(\theta_{1i}, \theta_{2i})^2} \end{aligned} \tag{2}$$

The learned function is therefore a mapping between the input image and a worm pose without regard to head-tail location, which we determine later with the aid of temporal information.

For each dataset, we generate 500k synthetic images for training, and randomly select 10k real preprocessed images for evaluation. When training, we use a batch size of 128. We train for 100 epochs and save the model with the smallest error on the evaluation set. We use the Adam optimizer [26] with a learning rate of 0.001.

Post-Prediction: Image Error and Outlier Detection For real data, the lack of labeled data for coiled worm images means that we cannot directly evaluate the accuracy of the network predictions. Instead, we leverage our ability to generate synthetic images and apply an image error measure between the input image and the two synthetic images generated from the two possible predicted centerlines. We generate synthetic worm images representing the two predictions, using the nearest labeled frame in time as a reference image. We crop the synthetic images to the bounding box of the synthetic worm shape plus a padding of 2 pixels on each side, and apply a template matching function between this synthetic image representing the prediction and the original image. We use the *matchTemplate* function from OpenCV [23] with the normalized correlation coefficient method, which translates a template image across a source image and compute the normalized correlation c at each location. The result is a correlation map, of a size $Size(source) - Size(template) + 1$, with values ranging between $c = -1$ (perfect anti-correlation, as would occur in a pair of reversed-intensity black and white images) and $c = 1$ (perfect correlation). We use the maximum value $|c|_{\max}$ to define the image error $1 - |c|_{\max}$ and the location of $|c|_{\max}$ to estimate the predicted skeleton coordinates. Frames with an image error above a threshold value will be discarded.

To select the threshold (potentially different for each different dataset), we plot the image error distribution on a selection of labeled frames. Comparing images with their reconstructed synthetic image based on their (trusted) labels shows a distribution of low error values, Fig. S3. We select an image error threshold with a default value 0.3, which retains the majority of the predictions while removing obviously incorrect reconstructions.

Post-Prediction: Head-Tail Assignment Once the network is trained, we can predict the centerline in full video sequences, the resulting postures having a random head-tail assignment. For each image, we augment the predicted centerline $\hat{\theta}$ with the head-tail switched centerline $\hat{\theta}_{flipped} = flip(\hat{\theta}) + \pi$. We use temporal information and the labeled frames to determine the final worm pose as either one of these two centerlines, or we discard the frame entirely in low-confidence cases.

We first create segments with near-continuous poses by using an angle distance function between adjacent frames, $distance(\theta_1, \theta_2) = \frac{1}{N} \sum_{n=0}^{N-1} |\epsilon(\theta_{1n}, \theta_{2n})|$, with ϵ from Eq. 2. We start with the first frame and assign its head position randomly. We then calculate the angle distance between this centerline and the two possible options in the next frame. If the distance is higher than a threshold (we use 30°), we cannot reliably assign the head position by comparing to this adjacent frame. We calculate the distance on the following frames (maximum 0.2s in the future) until we cannot find any frame that is close enough to the last aligned frame, we then start a new time segment with a random head-tail orientation. After this first process, we obtain temporal segments with a consistent head-tail position, possibly with small gaps containing outlier results to be discarded. To increase confidence in the results, we discard segments that are too small (less than 0.2 seconds).

While the worm pose is consistent in these segments, there are still two possible head-tail orientations per segment: we use the labeled data from the non-coiled frames to pick the correct solution. We align the whole segments with the labeled data by calculating a cosine similarity between the head to tail vector coordinates of the prediction and the available labels. We finally align the remaining unaligned segments with no labels by comparing them to the neighbor segments that have been aligned before: we also calculate the cosine similarity between the head to tail vector between the two closest frames of the aligned and unaligned segment.

Post-Prediction: Interpolation For an optional post-prediction step, we interpolate small gaps (`max_gap` $= 3$ frames), using a third-order spline interpolation method. Points in the vicinity of missing frames are generally noisier. We employ a weighted interpolation method in which points in the vicinity of missing frames are assigned progressively lower weights: within a window (`frames_around` $= 3$ frames) around each gap, the weight is progressively lower starting from half the normal weight and linearly decreasing to 0 at the missing frame. Since we cannot measure the angle error in each frame directly, we estimated the ratio between the standard deviation of the angles σ_θ and that of the angular error η, using network predictions in synthetic worm images. This allows us to use the standard deviation of the angles as a scale for the magnitude of the errors, which we use to set the weights assigned to each frame. The ratio between angular error η and the standard deviation of the angles σ_θ averages at $\sim 5\%$, and we set set the weights as $w = 1/(\delta\sigma_\theta)$, where $\delta = 0.02$, such that the range of weights captures the variability in angle errors, Fig. (S1). Smaller δs result in less smooth interpolations. Segments with less than 30 consecutive resolved frames are discarded. Spline interpolation is done using the `scipy.interpolate.UnivariateSpline` function from Scipy [27]].

Post-Prediction: Smoothing For an optional post-prediction step, we smooth the angle time series using a Savitsky-Golay filter with third-order polynomials in 7 frame windows, using the `scipy.signal.savgol_filter` function from Scipy [27].

Implementation: In Fig. 4 we show a schematic of the full computational process which we implement in a Python package "WormPose", with source code: `https://github.com/iteal/wormpose` and documentation: `https://iteal.github.io/wormpose/`. We optimize for speed via intensive use of multiprocessing and also for big video files that do not fit into memory. We provide default dataset loaders: for the Tierpsy tracker [21] and for a simple folder of images. Users can add their own dataset loader by implementing a simple API: `FramesDataset` reads the images of the dataset into memory, `FeaturesDataset` contains the worm features for the labeled frames, and `FramePreprocessing` contains the image processing logic to segment the worm in images and to calculate the average value of the background pixels. A custom dataset loader is typically a Python module exposing these three objects, which then can be loaded into WormPose by the use of Python entry points. A simplified example of adding a custom dataset is available in the source code repository: `https://github.com/iteal/wormpose/tree/master/examples/toy_dataset`. We provide a tutorial notebook with sample data and an associated trained model, which can be tested in Google Colaboratory. We also include an optional interface to export results in a custom format: for the Tierpsy tracker dataset, we can export the results to the Worm tracker Commons Object Notation (WCON) format.

RESULTS

Pose estimation from wild-type and mutant worm recordings

We quantify WormPose using synthetic data as well as (N=24) wild-type N2 worm recordings and (N=24) AQ2934 mutants from the Open Worm Movement Database. The synthetic data analyzed here was not used for training and consists of 600k images. We choose N2 for general interest and AQ2934 (with gene mutation nca-2 and nRHO-1) for the prevalence of coiled shapes. For the AQ2934 dataset, we used all of the available videos. For the N2 dataset, we selected 24 videos randomly from the large selection in the Open Worm Movement Database, but with a criterion of a high ratio of successfully analyzed frames from the Tierpsy Tracker (in practice ranging between 79% to 94%). Videos where there are very few analyzed frames may signal that the worm goes out of frame, or that the image quality is so low that no further analysis is possible. Images are sampled at rate $f_s \sim 30\,\mathrm{hz}$ for $\sim 15\,\mathrm{min}$ in duration, resulting in 600k frames from each dataset. We set the image size to 128×128 pixels. We train distinct models for each dataset and then predict all images from each dataset. We show the cumulative distribution of the image error in Fig. 5(A), including typical (input and output) worm images for various error values. For additional context, we also show the image error calculated on synthetic image data not used in training. Errors in the synthetic data are larger than those for N2 worms because we have more (and more complicated) coiled postures in the synthetic data generator. In Fig. 5(B), we use our image generator to show the error in mode values for synthetic data, the only data for which we have ground truth for the centerlines. The "error worms" (worm shapes representing mode values with $\delta a_i = 0.5$) are essentially flat and we report even smaller median mode errors $\delta\tilde{a} = (0.21, 0.28, 0.21, 0.25)$ (see [9] for a comparison). We provide scripts to download these datasets from Zenodo, as well as trained models: `https://github.com/iteal/wormpose_data`.

Posture-scale analysis of roaming/dwelling behavior

We further demonstrate WormPose by exploring previously unanalyzed $N = 14$ longtime ($T \sim 10\,\mathrm{h}, f_s \sim 30\,\mathrm{hz}$) recordings of on-food N2 worms. The length of these recordings renders impractical previous coiled shape solutions [9, 10] and therefore fine-scale posture analysis of behavior.

On food-rich environments, worms typically switch between two long-lasting behaviors: a roaming state, in which worms move abundantly on the plate at higher speeds and relatively straight paths; and a dwelling state, in which worms stay on a local patch with lower speeds and higher angular speeds [28, 29]. Roaming and dwelling states can last for tens of minutes so long recordings are essential and we leverage our ability to obtain high-resolution posture tracking to explore their fine-scale behavioral details.

To identify roaming and dwelling states consistent with previous work [29], we fit a Hidden Markov Model to the centroid and angular speed averaged in 10 s windows, which yields a high speed, low angular speed state (roaming) and a high speed, low angular speed state (dwelling), Fig. 6(A). We estimate the frame-by-frame directionality of the worm's movement by subtracting the angle of the velocity vector $\psi = \tan^{-1}\left(\frac{\mathrm{v_y}}{\mathrm{v_x}}\right)$ (where v_x and v_y are the x and y components of the centroid velocity) by the overall tail-to-head worm angle on the plate Ψ, obtained by averaging the angle along the body, $\Delta\psi = \psi - \Psi$. The distribution of $\Delta\psi$ is bimodal, indicative of switching between forward ($\Delta\psi \approx 0\,\mathrm{rad}$) and reversal ($\Delta\psi \approx \pi\,\mathrm{rad}$) movement, Fig. 6(B). As in previous observations [28, 29], worms mostly move forward in the roaming state, while dwelling exhibits a larger fraction of backward locomotion.

Our high-resolution posture measurements provide a unique opportunity to dissect the fine-scale details of these long time scale behaviors. We leverage the interpretability of the eigenworm decomposition of the centerline angles [7] to assess the properties of the body wave. The first two eigenworms (a_1 and a_2) capture the undulatory motion of the worm: the angle between these two modes $\phi = -\tan^{-1}\left(\frac{\mathrm{a_2}}{\mathrm{a_1}}\right)$ is the overall phase of the body wave, while its derivative, $\dot{\phi} = \omega$ is the body wave phase velocity. The third eigenworm, a_3, captures the overall turning amplitude of the worm: $|a_3| \gtrsim 10$ correspond to Ω-like turns [7, 9]. In Fig. 6(C) we show the distribution of phase velocities ω and turning amplitudes a_3. Roaming worms typically exhibit higher body wave phase velocities in both forward ($\omega > 0$) and backward ($\omega < 0$) locomotion, Fig. 6(C, left), as well as a larger fraction of deep Ω-turns ($|a_3| \gtrsim 10$). This posture-scale analysis is somewhat contradictory to the centroid-level analysis, which characterizes dwelling as a state in which worms increase their rate of reversals and reorientation events [28, 29]. Note that the detailed posture-level analysis was only made possible through WormPose, allowing us to measure the coiled shapes as well as get a more continuous and less noisy estimate of the body wave phase velocity.

To further dissect the nature of reversal events in the roaming and dwelling states, we compute the cumulative distribution of the reversal run lengths, Fig. 6(D). Reversals in the roaming state are typically longer than in dwelling, which exhibits extremely short run lengths. The low phase velocities in dwelling also indicate that such reversals result in an insignificant translation of the worm's body. This suggests that most of the reversals measured through a centroid-based analysis in fact correspond to incoherent body motions, such as head oscillations or short retractions. Indeed, we count the frequency of body waves that travel all the way across the body, Fig. 6(E), and find that the frequency of full-body waves is extremely small in dwelling when compared to the roaming, for which coherent body movements are much more frequent. While dwelling states at the centroid level exhibit larger reversal rates, the nature of these reversals is very different from the coherent body wave reversals found during roaming. We believe that this posture-level analysis, made possible by applying WormPose to long recordings, will ultimately enable a deeper understanding of the underlying control mechanisms for roaming and dwelling behaviors.

DISCUSSION

WormPose enables 2D pose estimation of *C. elegans* by combining a CNN with a synthetic worm image generator for training without manually labeled data. Our approach is especially applicable to complex, coiled shapes, which have received less attention in quantitative analyses even as they occur during important turning behaviors and in a variety of mutants. We also introduce an image similarity, which leverages the synthetic worm generator to assess the quality of the predicted pose. Once trained, the convolution computation is fast and could enable real-time, coiled-pose estimation and feedback [30]. The computational pipeline is optimized to analyze large datasets efficiently and is packaged in an easy to use, install and extend, open-source Python package.

With common imaging resolutions, the determination of the worm's head-tail orientation is surprisingly subtle. Our approach uses the presence of labeled trusted frames from traditional tracking methods which rely on brightness changes or velocity. An appealing alternative would be to estimate the head location directly. For example, [31] uses a network to regress the coordinates of C. elegans head and tail. In addition, CNN's that estimate keypoint positions [15], [16], [17] are now widely available. However, such current general techniques applied to ambiguous worm images result in low-confidence head-tail location probabilities, especially for blurry, low-resolution or self-occluded images. Training for this task is noisy and slow to converge, suggesting that there is simply not enough visual information in a single image.

Our posture model necessitates a library of examples which we obtained from N2 worms. Some strains however have different postures such as *lon-2* or *dpy* which are longer and shorter than N2, respectively. In particular, *lon-2* can make more coils due to its longer body, and our posture model does not represent the wider variety of possible postures. Of course, we can always augment the posture library. But a more general solution is to create a physical model of the worm [32].

Our approach follows advances in human eye gaze and hand pose estimation where it is difficult to obtain accurate labeled data. 3D Computer Graphics are often employed to create synthetic images [33] with increasing realism [34]. Synthetic images for human pose estimation have also been created by combining and blending small images corresponding to the body limbs of a labeled image, to form new realistic images [35]. To bridge the similarity gap between the real and the synthetic domain, Generative Adversarial Networks (GAN) techniques alter such computer-generated images [36] or directly generate synthetic images from a source image and a target pose [37]. Models of the deformable source object (e.g. human limbs) are often encoded into such generative networks to avoid unrealistic results. Some of these ideas have been recently applied to laboratory organisms [18], including *C. elegans*, but have avoided the fundamental complexity of self-occluding shapes. Outside of the laboratory, [38] proposes an end-to-end approach to estimate zebra pose using a synthetic dataset and jointly estimating a model of the animal pose with a texture map. Another approach is to adversarially train a feature discriminator until the features from the synthetic and real domain are indistinguishable [39, 40]. In both humans and animals, we expect that the combination of physical body models and image synthesis will be important for future progress in precise pose estimation.

ACKNOWLEDGMENTS

We acknowledge funding from the Vrije Universiteit

Amsterdam and The Okinawa Institute of Science and Technology Graduate University. We thank Mathijs Rozemuller (AMOLF) for code testing and for providing a tutorial dataset, as well as Jarlath Rodgers (University of Toronto) and Kelimar Diaz Cruz (Georgia Tech) for code testing. We are also grateful for the help and support provided by the Scientific Computing section of Research Support Division at OIST.

[1] N. Niepoth and A. Bendesky, *How natural genetic variation shapes behavior,* Annual Review of Genomics and Human Genetics **21** (2020), 10.1146/annurev-genom-111219-080427.

[2] S. Musall, M. T. Kaufman, A. L. Juavinett, S. Gluf, and A. K. Churchland, *Single-trial neural dynamics are dominated by richly varied movements,* Nature Neuroscience **22**, 1677 (2019).

[3] T. Ahamed, A. C. Costa, and G. J. Stephens, *Capturing the Continuous Complexity of Behavior in C. elegans,* bioRxiv (2019), 10.1101/827535.

[4] G. J. Berman, *Measuring behavior across scales.* BMC biology **16**, 23 (2018).

[5] A. E. X. Brown and B. de Bivort, *Ethology as a physical science,* Nature Physics **14**, 653 (2018).

[6] J. M. Gray, J. J. Hill, and C. I. Bargmann, *A circuit for navigation in Caenorhabditis elegans.* Proceedings of the National Academy of Sciences of the United States of America **102**, 3184 (2005).

[7] G. J. Stephens, B. Johnson-Kerner, W. Bialek, and W. S. Ryu, *Dimensionality and dynamics in the behavior of c. elegans,* PLOS Computational Biology **4**, 1 (2008).

[8] J. L. Donnelly, C. M. Clark, A. M. Leifer, J. K. Pirri, M. Haburcak, M. M. Francis, A. D. T. Samuel, and M. J. Alkema, *Monoaminergic orchestration of motor programs in a complex C. elegans behavior.* PLoS Biology **11**, e1001529 (2013).

[9] O. D. Broekmans, J. B. Rodgers, W. S. Ryu, and G. J. Stephens, *Resolving coiled shapes reveals new reorientation behaviors in C. elegans,* eLife **5**, e17227 (2016).

[10] S. Nagy, M. Goessling, Y. Amit, and D. Biron, *A generative statistical algorithm for automatic detection of complex postures,* PLOS Computational Biology **11**, e1004517 (2015).

[11] A. Javer, M. Currie, C. W. Lee, J. Hokanson, K. Li, C. N. Martineau, E. Yemini, L. J. Grundy, C. Li, Q. Ch'ng, W. R. Schafer, E. A. A. Nollen, R. Kerr, and A. E. X. Brown, *An open-source platform for analyzing and sharing worm-behavior data,* Nature Methods **15**, 645 (2018).

[12] E. Fontaine, J. Burdick, and A. Barr, in *2006 International Conference of the IEEE Engineering in Medicine and Biology Society* (2006) pp. 3716–3719.

[13] N. Roussel, J. Sprenger, S. Hendricks Tappan, and J. Glaser, *Robust tracking and quantification of c. elegans body shape and locomotion through coiling, entanglement, and omega bends,* Worm **3**, 00 (2015).

[14] Y. Guo, L. N. Govindarajan, B. Kimia, and T. Serre, *Robust pose tracking with a joint model of appearance and shape,* (2018), arXiv:1806.11011 [cs.CV].

[15] Mathis, Alexander, Mamidanna, Pranav, Cury, Kevin M, Abe, Taiga, Murthy, Venkatesh N, Mathis, Mackenzie Weygandt, and Bethge, Matthias, *DeepLabCut: markerless pose estimation of user-defined body parts with deep learning,* Nature Neuroscience **21**, 1281 (2018).

[16] Pereira, Talmo D, Aldarondo, Diego E, Willmore, Lindsay, Kislin, Mikhail, Wang, Samuel S H, Murthy, Mala, and Shaevitz, Joshua W, *Fast animal pose estimation using deep neural networks,* Nature Methods **16**, 117 (2019).

[17] J. M. Graving, D. Chae, H. Naik, L. Li, B. Koger, B. R. Costelloe, and I. D. Couzin, *DeepPoseKit, a software toolkit for fast and robust animal pose estimation using deep learning,* eLife **8** (2019), 10.7554/elife.47994.

[18] S. Li, S. Günel, M. Ostrek, P. Ramdya, P. Fua, and H. Rhodin, *Deformation-aware unpaired image translation for pose estimation on laboratory animals,* (2020), arXiv:2001.08601.

[19] L. Wang, S. Kong, Z. Pincus, and C. Fowlkes, in *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops* (2020).

[20] K. Bates, S. Jiang, S. Chaudhary, E. Jackson-Holmes, M. L. Jue, E. McCaskey, D. I. Goldman, and H. Lu, *Fast, versatile and quantitative annotation of complex images,* BioTechniques **66**, 269–275 (2019).

[21] A. Javer, M. Currie, C. W. Lee, J. Hokanson, K. Li, C. N. Martineau, E. Yemini, L. J. Grundy, C. Li, Q. Ch'ng, W. R. Schafer, E. A. A. Nollen, R. Kerr, and A. E. X. Brown, *An open-source platform for analyzing and sharing worm-behavior data,* Nature Methods **15**, 645 (2018).

[22] A. Javer, L. Ripoll-Sánchez, and A. E. Brown, *Powerful and interpretable behavioural features for quantitative phenotyping of caenorhabditis elegans,* Philosophical Transactions of the Royal Society B: Biological Sciences **373**, 20170375 (2018).

[23] G. Bradski, *The OpenCV Library,* Dr. Dobb's Journal of Software Tools (2000).

[24] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)* (Springer-Verlag, Berlin, Heidelberg, 2006).

[25] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, *Scikit-learn: Machine learning in Python,* Journal of Machine Learning Research **12**, 2825 (2011).

[26] D. Kingma and J. Ba, *Adam: A method for stochastic optimization,* International Conference on Learning Representations (2014).

[27] E. Jones, T. Oliphant, P. Peterson, and et al., *SciPy: Open source scientific tools for Python,* (2001–).

[28] J. Ben Arous, S. Laffont, and D. Chatenay, *Molecular and sensory basis of a food related two-state behavior in C. elegans,* PLoS ONE **4**, 1 (2009).

[29] S. W. Flavell, N. Pokala, E. Z. Macosko, D. R. Albrecht, J. Larsch, and C. I. Bargmann, *Serotonin and the neuropeptide PDF initiate and extend opposing behavioral states in C. elegans,* Cell **154**, 1023 (2013).

[30] J. B. Lee, A. Yonar, T. Hallacy, C.-H. Shen, J. Milloz, J. Srinivasan, A. Kocabas, and S. Ramanathan, *A compressed sensing framework for efficient dissection of*

neural circuits, Nature Methods **16**, 126 (2019).

[31] M. R. Mane, A. A. Deshmukh, and A. J. Iliff, *Head and tail localization of c. elegans,* (2020), arXiv:2001.03981 [cs.CV].

[32] N. Cohen and T. Ranner, *A new computational method for a model of c. elegans biomechanics: Insights into elasticity and locomotion performance,* (2017), arXiv:1702.04988 [physics.bio-ph].

[33] S. Kearney, W. Li, M. Parsons, K. I. Kim, and D. Cosker, in *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2020).

[34] J. Mu, W. Qiu, G. D. Hager, and A. L. Yuille, in *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2020).

[35] G. Rogez and C. Schmid, *Image-based synthesis for deep 3d human pose estimation,* International Journal of Computer Vision **126**, 993 (2018).

[36] A. Shrivastava, T. Pfister, O. Tuzel, J. Susskind, W. Wang, and R. Webb, in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017).

[37] G. Balakrishnan, A. Zhao, A. V. Dalca, F. Durand, and J. Guttag, in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (IEEE, 2018).

[38] S. Zuffi, A. Kanazawa, T. Berger-Wolf, and M. Black, in *2019 IEEE/CVF International Conference on Computer Vision (ICCV)* (2019) pp. 5358–5367.

[39] A. Lahiri, A. Agarwalla, and P. K. Biswas, *Unsupervised domain adaptation for learning eye gaze from a million synthetic images: An adversarial approach,* (2018), arXiv:1810.07926 [cs.CV].

[40] F. Kuhnke and J. Ostermann, *Deep head pose estimation using synthetic images and partial adversarial domain adaption for continuous label spaces,* 2019 IEEE/CVF International Conference on Computer Vision (ICCV) , 10163 (2019).

[41] L. R. Rabiner, *Tutorial on Hmm and Applications,* Proceedings of the IEEE **77**, 257 (1989).

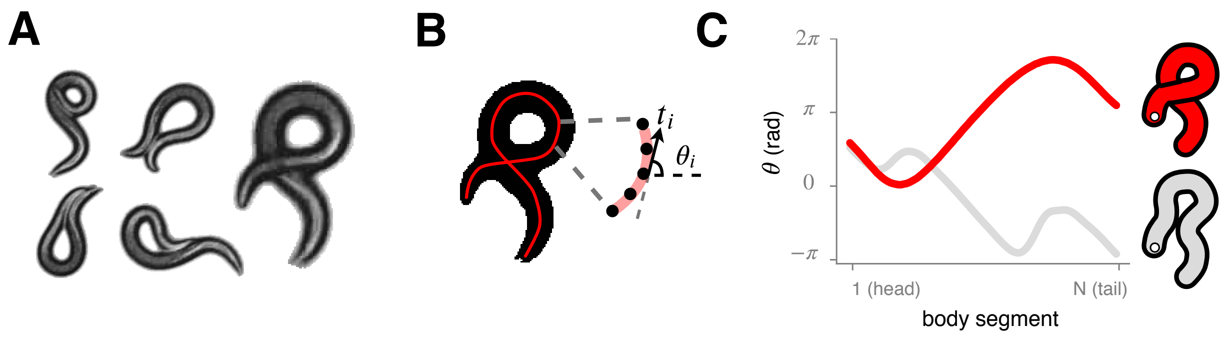

FIG. 1. The nematode *C. elegans* naturally exhibits a variety of coiled shapes which challenge the determination of the centerline posture, a fundamental component for quantitative behavioral understanding. **(A)** An exemplar collection of images displaying coiled shapes. **(B)** Instantaneous worm pose encoded as the centerline curve parameterized by tangent angles $\boldsymbol{\theta} = (\theta_1, \ldots \theta_i, \ldots \theta_N)$ ordered from head to tail. **(C)** Standard image processing techniques extract the centerline by morphological operations or image features analysis and have not been able to differentiate solutions with very different centerlines (red, grey) that occur with coiled posture. The correct centerline (red) can be determined by close visual inspection (A), however high-throughput analysis necessitates a pose estimation algorithm which is robust to fluctuations in brightness, blur, noise, and occlusion.

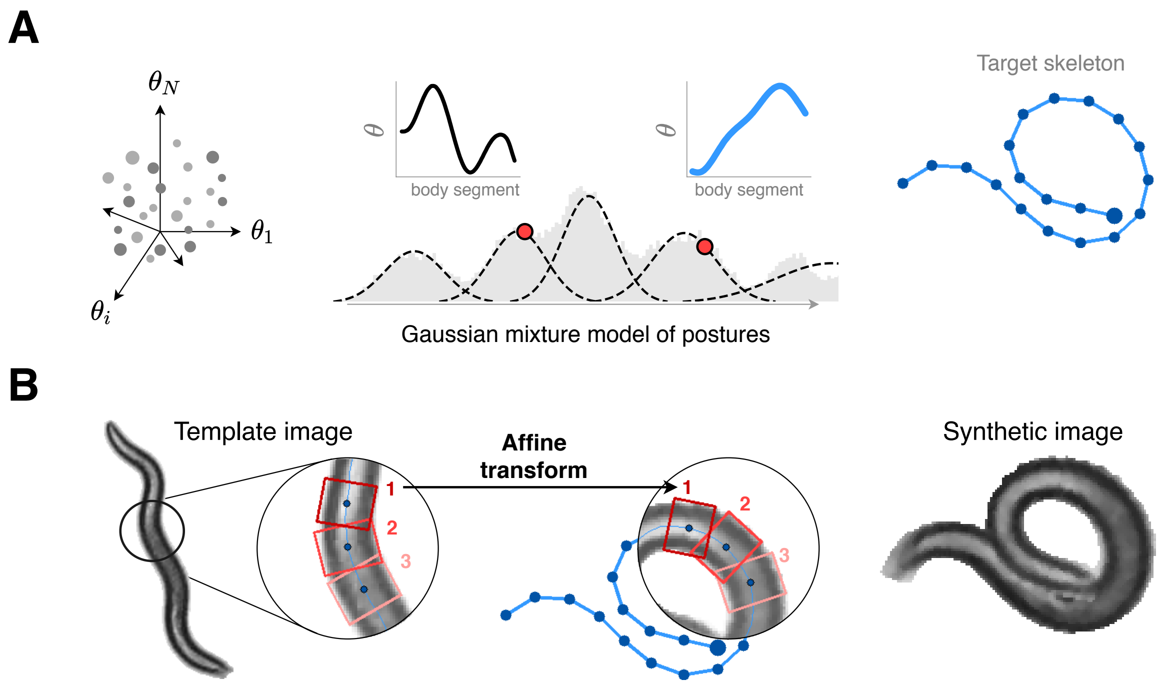

FIG. 2. We combine a generative model of worm posture with textures extracted from real video to create realistic yet synthetic images for a wide variety of natural postures, including coils, thus avoiding the need for manually-annotated training data. **(A)** We model the high-dimensional space of worm posture (left)by Gaussian mixtures (middle) constructed from a core set of previously analyzed worm shapes [9]. To each generated posture we add a global orientation (chosen uniformly between 0 and 2π), and we randomly assign the head to one end of the centerline. (right) We use the resulting centerline (angle coordinates) to construct the posture skeleton (pixel coordinates). **(B)** We warp small rectangular pixel patches along the body of a real template image (left) to the target centerline (middle), producing a synthetic worm image (right). Overlapping pixels are alpha-blended to connect the patches seamlessly. Unwanted pixels protruding from the target worm body are masked and the background pixels are set to a uniform color. Finally, the image is cleaned of artifacts through a medium blur filter.

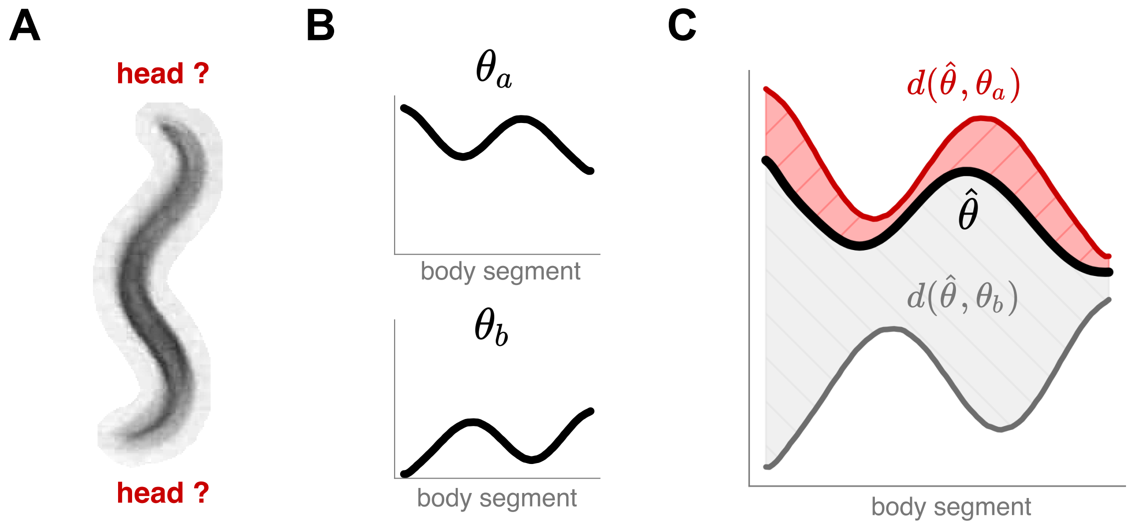

FIG. 3. We train a convolutional network to associate worm images with an unoriented centerline to overcome head-tail ambiguities which are common due to worm behaviors and imaging environments. **(A)** An example image with a seemingly symmetrical worm body. **(B)** We associate each training image to *two* possible centerline geometries, resulting in two equivalent labels: θ_a and $\theta_b = flip(\theta_a) + \pi$, corresponding to a reversed head/tail orientation. **(C)** We compare the output centerline $\hat{\theta}$ to each training centerline through the root mean squared error of the angle difference $d(\theta_1, \theta_2)$ (Eq. 2) and assign the overall error as $loss = min(d(\hat{\theta}, \theta_a), d(\hat{\theta}, \theta_b))$.

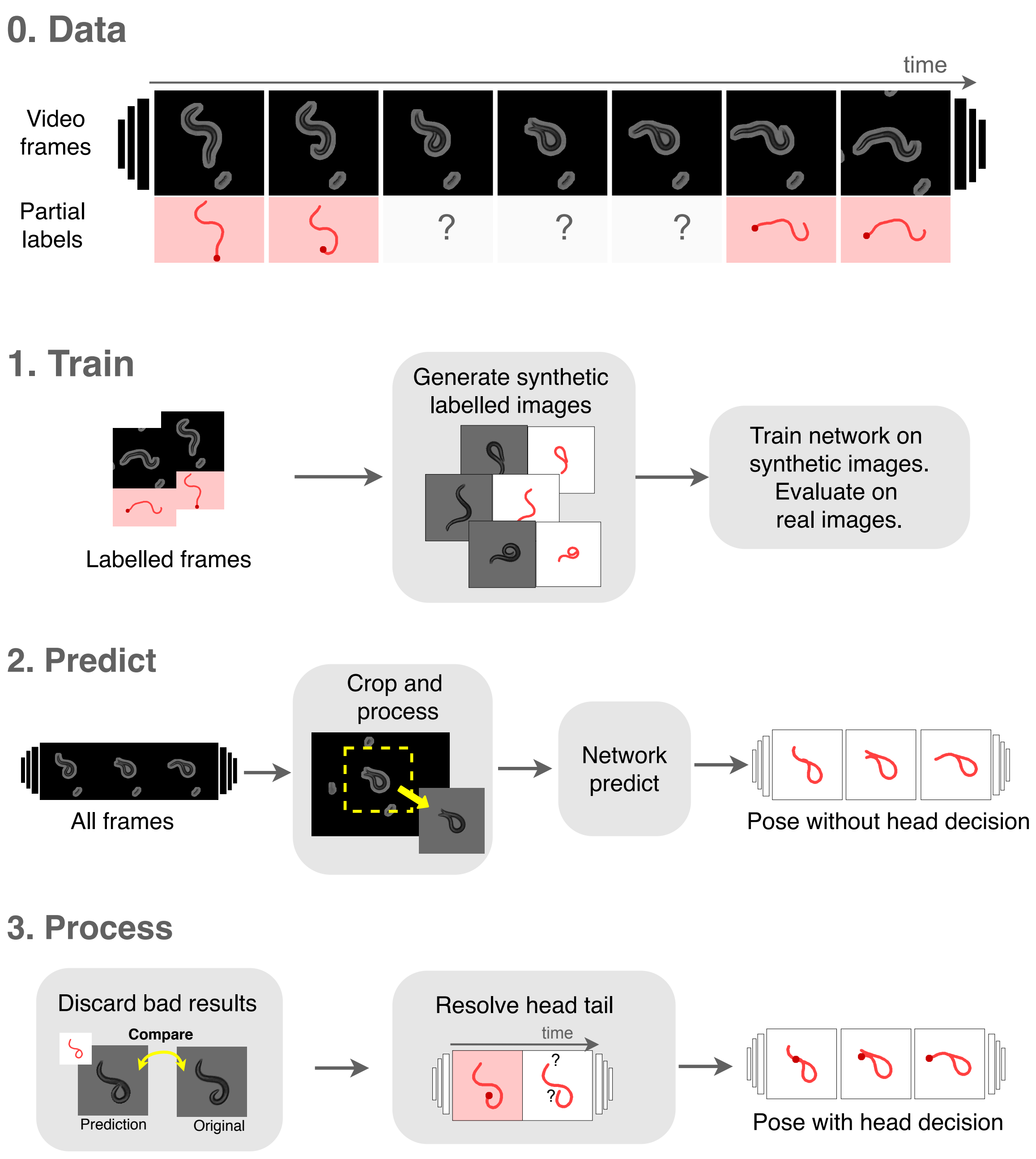

FIG. 4. The WormPose pipeline. **(0)** We use classical image processing methods to extract partial labels of simple, non-coiled postures, and then apply a CNN-based approach to complete the missing frames which result from complex images. We analyze each video recording with a three-step pipeline. **(1)** We generate synthetic data with the visual appearance of the target images but containing a wider range of postures, Fig. 2. We use this synthetic data to train a deep neural network to produce the centerline angles from a single image. During training, we periodically evaluate the network on real labeled images and keep the model that best generalizes. **(2)** We predict the entire set of target images. The images are first cropped and processed to look more visually similar to the synthetic images: background and any non-worm pixels are set to a uniform color. For each such processed image, the trained network predicts the centerline angles for both possible head-tail orientations. **(3)** Our algorithm produces a full image as output and we discard inaccurate results using a pixel-based comparison with the input image. Finally, we resolve the head-tail orientation by comparing adjacent frames. Once trained, the WormPose pipeline is rapid and robust across videos from a wide variety of recording conditions.

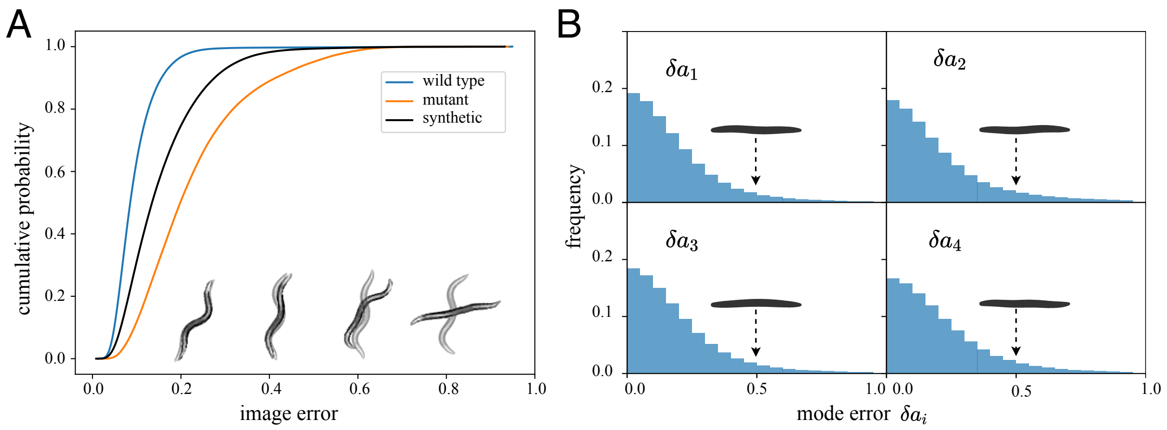

FIG. 5. Quantifying the error in pose estimation. **(A)** We show the cumulative image error of predicted images for different datasets. We predict 24 videos totaling over 600k frames from N2 wild-type and AQ2934 mutant datasets and calculate the image error between the original image and the two possible predictions, and keep the lowest value between the two. For the error calculations here we bypass the postprocessing step so no result is discarded. For interpretability we also draw representative worm image pairs for different error values and note that predictions overwhelmingly result in barely discernible image errors. On average, the N2 predictions have a lower image error than the mutant which exhibit much more coiled challenging postures. We also generate new synthetic images (using N2 as templates, 600k values) not seen during the training and predict them in the same way. The image error for the synthetic images (which generally include a higher fraction of complex, coiled shapes) is on average worse than the N2 type, but better than the mutant. **(B)** Our synthetic training approach also allows for a direct comparison between input and output centerlines, here quantified through the difference in eigenworm mode values. As with the images, the differences are also small so that even in the large-error tail of the distribution the "error worms" (worm shapes representing mode values with $\delta a_i = 0.5$) are essentially flat. The median mode errors are $\langle \delta\vec{a} \rangle_{\text{median}} = (0.21, 0.28, 0.21, 0.25)$.

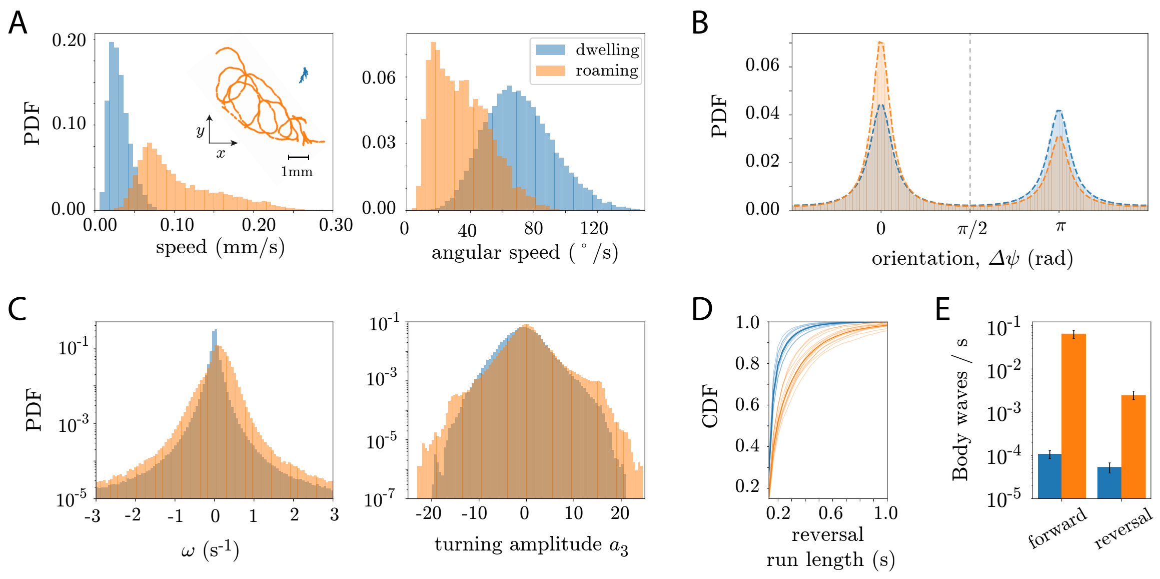

FIG. 6. Posture-scale analysis of roaming/dwelling behavior from long recordings reveals that the centroid-derived increase in the dwelling reversal rate results from incoherent body motions that do not translate the worm's body. **(A)** To align with previous definitions, we identify roaming/dwelling behavior through a Hidden Markov Model of the linear and angular speed, averaged in 10s windows, and we split the trajectory into two hidden states: a low speed, high angular speed state (dwelling, blue), and a high speed, low angular speed state (roaming, orange). (A, inset) Example 5 minute centroid trajectories in the dwelling state (blue) and the roaming state (orange). **(B)** Dwelling state exhibits a larger fraction of reversals when compared to roaming. We identify forward and backward motion using the angle between the centroid velocity vector and the tail-to-head angle obtained by averaging the body angles: $\Delta\psi < \pi/2$ for forward locomotion, $\Delta\psi > \pi/2$ for backwards. **(C-E)** Posture-scale dynamics indicate that the centroid level characterisation of roaming and dwelling states is incomplete. (C) Roaming worms exhibit a higher angular speeds in both reversal and forward motion and also a higher fraction of deep turns ($a_3 > 10$). **(D)** Cumulative distribution of reversal run lengths in the dwelling (blue) and roaming states (orange). The roaming state generally exhibits longer reversals than the dwelling state, for which reversal bouts are extremely short. Thick lines indicate the CDF for the ensemble of worms, while lighter lines are for each worm. **(E)** The rate of reversal events with complete body waves is an order of magnitude higher in the roaming state compared to dwelling.

SUPPLEMENTARY MATERIAL

Analysis of roaming and dwelling behavior: To connect to previous analysis on the roaming and dwelling behavior, we compute the worms speed and angular speed from the centroids $\vec{c} = (x, y)$ position as a function of time. To simplify the comparison with [29], we downsample the time series to 3 Hz and compute the centroid velocity as the finite difference between subsequent time points $\vec{v}(t) = \frac{\vec{c}(t+\Delta t)-\vec{c}(t)}{\Delta t}$, where $\Delta t = 1/3\,\mathrm{s}$ after downsampling. The speed is then obtained by taking the norm of the velocity vector $s(t) = |\vec{v}(t)|$, where $|.|$ represents the 2-norm. The angular speed is computed by estimating the angle between the two vectors defined from three subsequent points, which gives the change in the tangential component of the velocity. From these estimates, we obtain roaming and dwelling states by fitting a two-state Hidden Markov Model (HMM) to the speed and angular speed time traces averaged in 10 s windows (as in [29]). The model is composed of two hidden states, their stationary distributions π and Markov transition matrices P, and Gaussian emission probabilities conditioned on the current state. Fitting is performed through an Expectation-Maximization algorithm (Baum-Welch), with the emission probabilities being Gaussian distributions with a diagonal covariance matrix. The sequence of hidden states is obtained through a Viterbi algorithm. We use an open-source Python HMM package, `hmmlearn`, obtained from: `https://github.com/hmmlearn/hmmlearn`. For more info on HMMs, see [41].

Analysis on the directionality of the worms movement was done in the following way. At each time point, we estimated the tangential component of the velocity vector by $\psi(t) = \tan^{-1}(v_y(t)/v_x(t))$, and the overall tail-to-head angle by averaging the centerline angles $\Psi(t) = \langle\vec{\theta}(t)\rangle$. The worms orientation was then obtain by subtracting these two quantities $\Delta\psi = \psi - \Psi$ and normalizing them into the interval $[-\frac{\pi}{2}, \frac{3\pi}{2}]$ rad.

Posture-level analysis was performed by projecting the centerline angle time series $\vec{\theta}(t)$ into the lower dimensional space of eigenworms, $\vec{a}(t)$, using a canonical set of modes [7]. In this space, the first two eigenmodes capture the propagation of the body wave along the body. The angle between them, $\phi(t) = \tan^{-1}(\mathrm{a_2(t)/a_1(t)})$, defines the phase of the wave, while its derivative $\omega(t) = \dot{\phi}(t)$ is the phase velocity. Estimates of the phase velocity are obtained through fitting a cubic spline to ϕ, using Scipy's `scipy.interpolate.CubicSpline` [27], in order to reduce the noise in the estimation of ω. We estimated the frequency of complete body waves by finding segments in which the body wave phase ϕ did not change sign, and there is a recurrence in $\cos(\phi(\mathrm{t}))$. We make a conservative estimate of the body wave frequency by counting peaks in the time series of $\cos(\phi(\mathrm{t}))$, using the `scipy.signal.find_peaks` function of Scipy [27], with a prominence 1.95 and a minimum time between peaks of 8 frames.

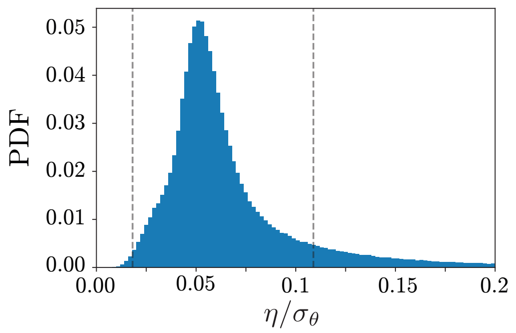

FIG. S1. Probability density function (PDF) of the ratio between the prediction error η and the standard deviation of the angles used to generated the sample images σ_θ. Since we cannot measure the angular errors directly, we use the standard deviation of the angles as a scale, and define the interpolation weights as $w = 1/(\delta\sigma_\theta)$, where $\delta = 0.02$. Since we assign progressively smaller weights at the boundary of missing frames, using $\delta = 0.02$ allows us to have a range of weights that captures the variability in angle errors (dashed lines correspond to higher and lower weights used, which fall into the lower and upper tails of the error distribution).

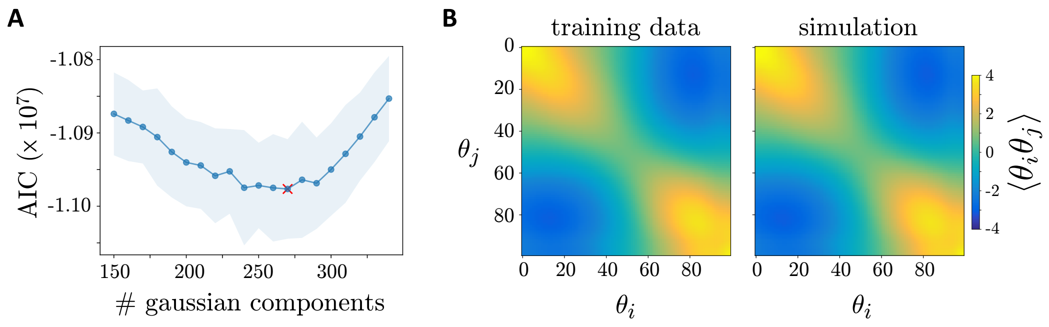

FIG. S2. Model selection assessment in the Gaussian Mixture Model (GMM) of worm shapes. **(A)** Akaike Information Criterion for GMMs with different numbers of gaussian components. The minimum is attained with $N = 270$ gaussian components. **(B)** Covariance matrix of the space of mean subtracted tangent angles $\vec{\theta}$ for the data used in training (left) and an equal number of simulated angles (right).

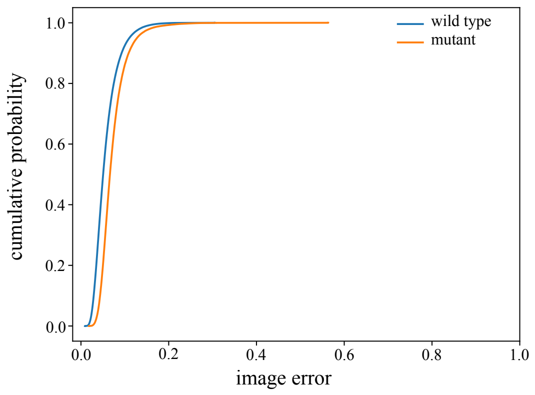

FIG. S3. The cumulative distribution of the image error for all available labeled (and thus uncoiled) frames in the N2 wild-type and AQ2934 mutant datasets.