Research Internship Project

Developing Genotype-Conditioned Artificial Drosophila Larvae Behavioural Data

December 4, 2024

Author : Soumodeep HOODATY

Promotion : 2024

L'enseignant Referent IP Paris : Eric MOULINES

Supervisors : Tihana JOVANIC, Jean-Baptiste MASSON

Effective Internship Dates: 17/07/2024 to 22/08/2024

Name of Organization : Institut des Neurosciences Paris-Saclay

Address : 151 Route de la Rotonde, 91400 Saclay

Acknowledgements

I want to express my deepest gratitude to my supervisors, Tihana Jovanic and Jean-Baptiste Masson, for their invaluable support and guidance throughout the writing of this report. Their expertise and willingness to assist were instrumental in completing this project.

I am also profoundly thankful to Francois Laurent for his immense help in handling all the technical problems and helping me set up this project.

Additionally, I extend my heartfelt thanks to my professors at Institut Polytechnique de Paris for their unwavering support and teaching. Their knowledge and encouragement have been vital in my academic journey and in completing this report.

Contents

1 Introduction

Why Drosophila larva?

The central nervous system can produce a range of behaviours composed of various types of motor actions. With recent advancements in genetics, large-scale automated behavioural data collection, and machine learning, we can now causally link behaviours to their underlying neural substrates and decipher neural mechanisms. In certain animals, such as Drosophila larvae, this mapping can be done on an unprecedented scale, involving millions of specimens and individual neurons. This allows for the identification of neural circuits responsible for specific behaviours. High-throughput screening efforts are precious as they connect the activation or suppression of particular neurons to behavioural patterns in millions of animals. This generates a rich dataset that explores how various motors respond to external stimuli. Studying this insect also helps explain complex mechanisms in other species, including humans, which can shed light on neurodegenerative disorders like Parkinson's and Alzheimer's.

Classifying different behaviours into distinct actions is an efficient process for reducing the dimension of behavioural actions and thus allowing their analysis. My host labs have developed automated procedures for classifying behaviour and an application for re-annotating experimental data to train new classifiers. Recent research, however, has shown that certain larva lines exhibit notable departures from recognised behaviour categories. As a result, redefining current actions or retraining the classifier to detect these novel behaviours correctly will be necessary.

The current project's objective is to devise a simulation framework to generating synthetic data that includes essential traits of larvae discovered during trials. The technique will enable the generation of additional edge case samples and retraining to strengthen the classifier. Additionally, it will be utilised as brief generative programmes to produce new kinds of actions with predetermined properties (such as a variance in the head angle or a motion velocity norm). Another potential use case is testing several experimental techniques on virtual larvae before carrying out the actual experiment. An autonomous process previously created in the lab for tracking larval behaviour will be used to assess the technique's accuracy.

2 Context and motivation of the internship

Motivation

The necessity for machine learning arises from the lab's various experiments on Drosophila larvae. These experiments involve control and experimental groups where specific neurons are inactivated or activated. Different stimuli are then applied to observe the larvae's responses. While a neuroscientist might visually detect qualitative changes between the control and experimental groups, many quantitative differences require statistical and computational analysis to evaluate accurately. This is where machine learning becomes essential. It allows for the detection and visualization of the various quantitative changes (for example, transitional probabilities of actions) and subtle qualitative changes, which are not readily apparent through direct observation.

Drosophila larvae are instrumental in researching behaviour and neural computation. Advanced genetic techniques allow precise manipulation of neurons, linking neural activity with behaviour. However, analyzing extensive and noisy data requires robust methods.

Recent studies (REF, lehmen et al, BioRxiv 2023, De Tredern et al, BioRxiv 2023) indicate deviations in larval behavior, challenging traditional perceptions. We suggest a Bayesian Program Synthesis (BPS) approach to address this. BPS will generate synthetic data mirroring experiments, capturing population-level parameters, and replicating behavioural traits across genotypes. It will use various generative models to simulate larval behaviour under specific conditions. By applying this synthetic data to behaviour analysis pipelines, we aim to enhance transfer learning from annotated datasets, improving our understanding of larval behaviour and neural mechanisms.

There are various genetic lines on which we need to conduct experiments, with every line having its unique features to map its behaviour. Thus, this calls for a universal tagging tool model which would help us accommodate these differences.

The Laboratories

Neural Circuits & Behavior Team

Located in Paris-Saclay Institute of Neuroscience, the team's objectives are to exploit the advantages of the exquisite genetic tools available in Drosophila for manipulation and monitoring of neuronal activity at single neuron level and the numerical simplicity of the larval nervous system that allows the reconstruction of synaptic connectivity across the nervous system using large-scale electron microscopy to study the structure and function of the neural circuits for sensorimotor decisions and action sequences. It prioritises these three goals:

1. Ascertain how contextual information affects decisions taken concerning behavioural choices.
2. Identify the brain-wide mechanism that underlies the competitive relationships between several actions.
3. Identify the circuit mechanism that produces longer innate sequences

Decision and Bayesian Computation - Epimethe Laboratory

The Neural Circuits & Behavior team is collaborating closely with Decision and Bayesian Computation laboratory - Epimethe (Institut Pasteur, INRIA, CNRS, UPC Paris), whose primary focus is on biological decision-making through an interdisciplinary approach combining statistical physics, machine learning, and experimental biology.

Physiological and behavioural properties of Drosophila larvae

Drosophila melanogaster goes through several phases throughout its life cycle, including embryo, larva, pupa, and adult fly. The project's data comes from studies conducted on larvae. This section provides a brief introduction to help you better understand how the project is set up.

The white, translucent larva of D. melanogaster resembles a worm-like burrower. At 25°C, the insect typically remains in this phase for 96 hours before developing into a pupa; the exact length of this stage depends on the temperature and other environmental conditions.

Drosophila larvae are particularly appealing for neuroscience research since their central nervous systems (CNS) contain 10,000 fewer neurons than those of adult humans, which is a significant difference,

The larva's body can be divided into 11 segments: 3 thoracic and 8 abdominal, each segment comprising a group of longitudinal and transverse muscles. The larva uses peristaltic wave-like motions, which are the outcome of successive contractions of various muscles, to alter its location. The exterior soft cuticula and inner cellular epidermis make up the body covering.

While the visual sense system is highly developed in adult flies, it is not in the larvae, who interact with their surroundings primarily through somatosensation (mechanosensation, temperature nociception, etc. and well-developed gustatory and olfactory centres. However, all sensory neuronal classes found in adult D. melanogaster remain in larvae.

Adult flies display various behaviours, from basic motor skills to intricate social interactions. In contrast, larvae often exhibit inquisitive foraging behaviour (looking for sufficient food to undergo metamorphosis), with primitive social behaviour, learning and memory, along with some defensive or evasive behaviour when confronted with potentially threatening situations.

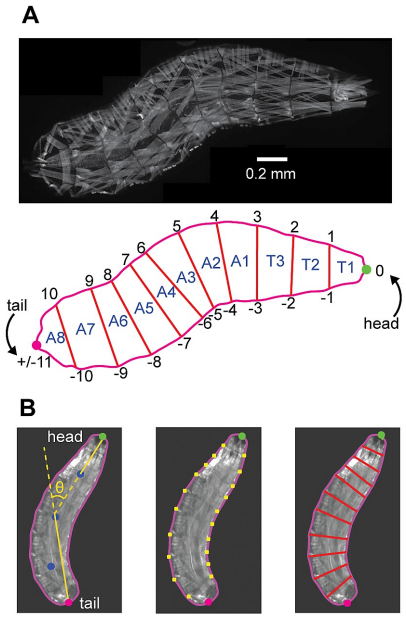


Figure 1: Segmentation of Drosophila larva

A few terms and definitions

1. **Sensory Cue:** A signal from the environment detected by the sensory organs and processed by the nervous system.

2. **Motor Outputs:** Actions or movements generated in response to sensory cues. In the case of Drosophila larvae, this includes various behaviours like rolling, crawling, turning, etc.

3. **Nervous System:** The network of neurons and other cells that process sensory information and control behaviour. It includes sensory processing, motor control, and intermediate areas involved in decision-making and action selection.

4. **Behavioral Choice:** The decision-making process determines which action to perform in response to a sensory cue. This involves selecting one behaviour while suppressing others to ensure a coherent response.

5. **Mutually Exclusive Actions:** Actions that cannot be performed simultaneously, such as turning left and right.

6. **Competitive Interactions:** Neural mechanisms ensure only one action is chosen and executed by suppressing alternative actions. These interactions can be distributed across different areas of the nervous system.

7. **Action Stability and Transitions:** Once an action is initiated, it must be maintained until an appropriate transition to the next action occurs. Understanding how the nervous system controls these transitions is crucial for comprehending behavioural sequences.

8. **Probabilistic Responses:** The idea that the same stimulus might evoke different responses in different individuals or even in the same individual at different times indicates variability and flexibility in behaviour.

9. **GAL4/UAS System:** A genetic tool used in Drosophila research to control gene expression. GAL4 is a yeast transcription factor that binds to UAS (Upstream Activating Sequence) to drive the expression of target genes in specific neurons.

10. **Behavioral Screen:** A systematic method to observe and record the behaviours of organisms in response to specific stimuli. It involves screening various Drosophila lines to identify neurons involved in particular behaviours.

11. **Electron Microscopy (EM) Reconstruction:** A technique to visualize the fine structure of neurons at a high resolution, used here to map the connectivity patterns of neurons.

12. **Ventral Nerve Cord:** The part of the nervous system in Drosophila larvae analogous to the spinal cord in vertebrates, transmitting information between the brain and the rest of the body.

13. **Optogenetics:** A technique that uses light to control neurons genetically modified to express light-sensitive ion channels, allowing precise control of neural activity.

Behavioral Dictionary

Six stereotypical actions form the larval behavioural dictionary: crawl, bend (all turning actions), stop (not moving), hunch (fast head retraction), back crawl (crawling backwards), and roll (fast lateral rolling action with random change of diretions). These actions are represented by letters A–F in plots and tables. Actions needed for analysis are inferred using supervised and unsupervised machine-learning techniques.

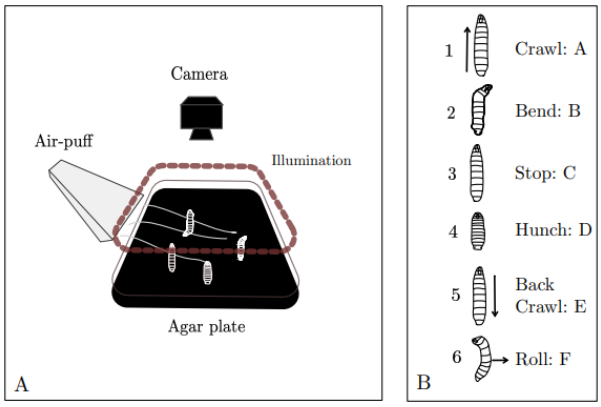


Figure 2: A: Visual Representation of Experiment B: The six defined actions

The team has developed a Python package pytrxmat to manipulate these files more easily. The team has utilized the following dictionary of lavrae's actions:

During experiments, larvae are recorded for 5 minutes. The recordings are then directly processed by machine learning algorithms to extract coordinates of the larvae's contour and spine, features associated with each action, larvae's length, and time stamps. The larvae can then be plotted in 2D space, as shown in the figure below.

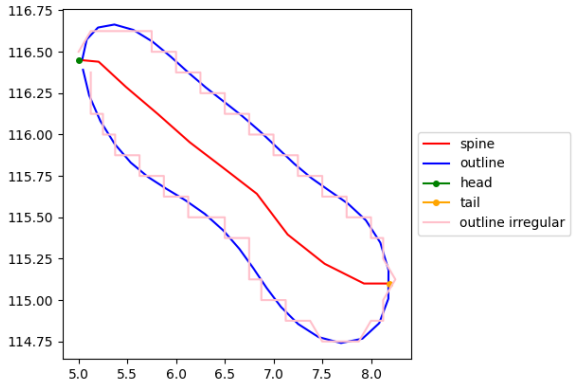


Figure 3: Visual representation of outline smoothening

For early development, we concentrate on one feature for each action and larva length.We take the asymmetry factor (also called head angle) as the main measure for asymmetric actions (e.g., bend and roll) and the motion velocity norm for the other actions (e.g., run, back, and hunch).

Behavioral Experiments

In the "new activation screen" experiment, about 30-70 larvae were separated from food using a 20% sucrose solution for 10 minutes, rinsed, and placed in a 23 cm^2 behaviour rig with 4% agar. Larval behaviour was recorded with a DALSA Falcon 4M30 camera for 120 seconds. At 30 and 75 seconds, 15-second pulses of 660 nm red light (4 $\mu W/mm^2$, Philips Lumileds) were applied. Larvae were tracked in real-time using Multi-Worm Tracker (MWT) software, rejecting objects tracked for less than 5 seconds or moving less than one body length. MWT provided contour and spine coordinates over time without storing raw videos.

We will use the t15 sample data to perform all tasks. All of the optogenetic lines have similar features, and the code is valid for all of them. We plan to write the code transcendingly to run it on those lines as well.

Objective

The main goal of creating the synthetic data is to generate uncommon samples in the experiments. These samples will subsequently be used to retrain the classifier—which is currently in use in the lab—to strengthen it. Modelling the behaviour of larvae under specific conditions would be another potential use, allowing for the prediction of experiment results before actual experimentation. We do not intend to control larval interactions in this version. Nonetheless, we must control temporal dependencies in movement patterns and consider how stimuli affect the decision between an action and a trajectory. We also want to be able to "by hand" restrict the generated data (for instance, accurately measure the animal's length or the parameters of an action, like its average speed).

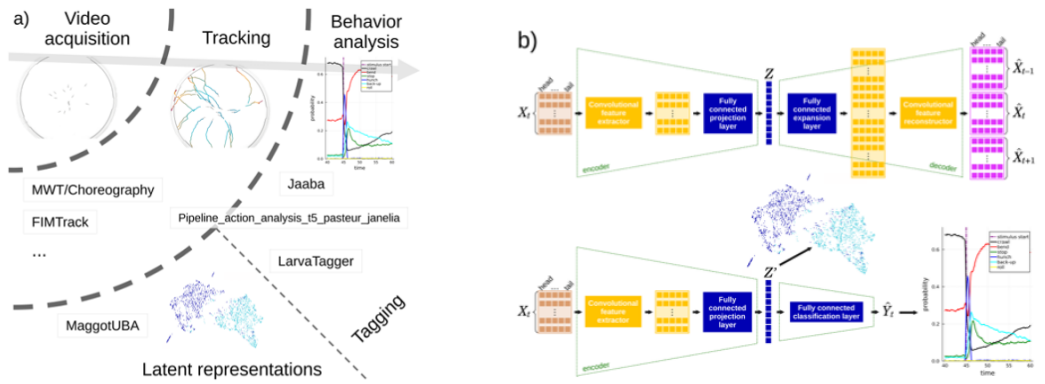
We need to set up distinct processes to create action sequences and coordinates. When generating a sequence, the result should be an action chain with the type, duration, and characteristics of each action (e.g., "run" for 10 seconds at an average speed of 5 mm/s, then "cast" for 1 second at a variation of 35 degrees in head angle, etc.). When generating coordinates, the model should produce realistic larvae's contour and spine coordinates for every time frame, considering limitations like the larva's length, duration of action, average speed, and other characteristics.

The major challenge I will be tackling is to implement the regularisation of the shape of the larva and check if applying the Fourier decomposition (seen later) directly to recorded contours works better than using it after regularisation.

3 LarvaTagger - Tagging Software

LarvaTagger is a behaviour tagging tool for Drosophila larvae. It performs both manual and automatic tagging, with the possibility of training new taggers using a pre-trained neural network and labelled/tagged "ground-truth" data. The LarvaTagger project is divided into multiple sub-projects to support different use cases. For example, the user interface (UI) is provided by LarvaTagger.jl, available at `https://gitlab.pasteur.fr/nyx/larvatagger.jl`. The LarvaTagger.jl repository is the main entry point of the whole project and its documentation. However, the automatic tagging logic is functionally independent from the UI.

Tracking Drosophila larvae is challenging because of the lack of structure in the 2D shape of a larva when seen from above. Furthermore, the choice of the number of larvae per behavioural assay represents a trade-off between increasing statistics and minimising crossing events. Indeed, the larvae have highly deformable bodies and are challenging to properly differentiate and track when they interact while being filmed at low resolutions (recent progress has been made at high resolution). All the tracking solutions we know of require precise calibration and may fail to detect moving objects with minor variations of experimental parameters.



(a) Visual Representation of Larva Tagging (b) MaggotUBA Autoencoder

Figure 4: In **(a)** tools and approaches to go from video data to behaviour representations: the moving objects in video data (an example frame is shown in the top left corner) are tracked with tools such as MWT or FIMTrack (see text for references). Behaviour analysis uses tracking data (top, middle; colours show time), with tools such as Jaaba, Pipeline pasteur janelia or LarvaTagger. Here, Pipeline pasteur Janelia is identified primarily as a tagging tool but also includes a head-tail reorientation procedure of the tracking domain. Behaviour is analyzed at the population level, typically in discrete actions (e.g. count or frequency data like in the top right illustration). Alternatively, behaviour can be analyzed using continuous representations in latent spaces. Sequences of postures are projected in a low dimensional latent space as depicted in the bottom illustration of a cloud of 2D points (one point = one sequence) with the colour encoding different actions. MaggotUBA is specifically designed for this approach, and LarvaTagger's automatic tagging backend is based on MaggotUBA. In **(b)** integration of MaggotUBA (top diagram), an autoencoder that extracts features of the continuous behaviour in a self-supervised fashion. A sequence of postures X_t is compressed into a low-dimensional latent representation Z from which a longer sequence, including past and future postures in addition to the input X_t, is reconstructed. Training the autoencoder allows for learning behavioural features that compress the dynamics in the latent space. The encoder features are reused in combination with a classification stage in the MaggotUBA-based tagger (bottom diagram) to learn a discrete behavioural dictionary. The MaggotUBA-based tagger can be used to generate both types of behavioural readouts.

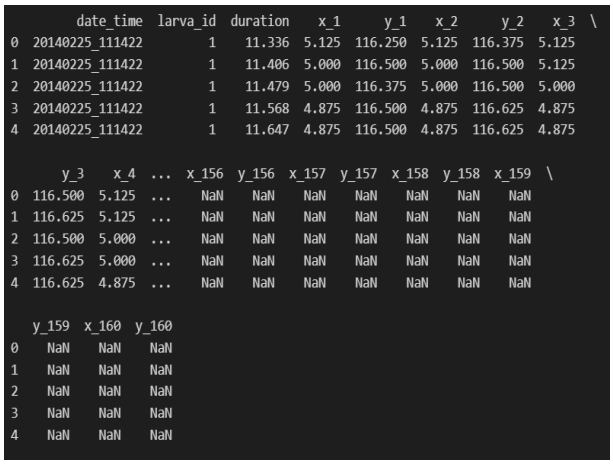
4 Data

Indexing

We require generating an index of the database as it is too large to work on at once. We take a portion of the data and perform all the tasks on that. We assume that the rest of the data would provide us with similar results. However, we plan to run the simulations on the rest of the data as well using the cluster system. With the index, we can have the desired features as captured by the Multi Worm Tracker software.

Structure

The data stored in the `.spine` and `.outline` files are in a particular format. The first column is `date_time`, which represents the date and time the experiment took place. The second column is the `larva_id`, which helps us uniquely identify the larva acting. The third column is `duration`, which gives us the time stamp at which the set of coordinates represents the spine/outline of the larva. These are followed by the x and y coordinates of the larva's spine/outline points.



	date_time	larva_id	duration	x_1	y_1	x_2	y_2	x_3	\
0	20140225_111422	1	11.336	5.125	116.250	5.125	116.375	5.125	
1	20140225_111422	1	11.406	5.000	116.500	5.000	116.500	5.125	
2	20140225_111422	1	11.479	5.000	116.375	5.000	116.500	5.000	
3	20140225_111422	1	11.568	4.875	116.500	4.875	116.625	4.875	
4	20140225_111422	1	11.647	4.875	116.500	4.875	116.625	4.875	

	y_3	x_4	...	x_156	y_156	x_157	y_157	x_158	y_158	x_159	\
0	116.500	5.125	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
1	116.625	5.125	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
2	116.500	5.000	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
3	116.625	5.000	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
4	116.625	4.875	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	

	y_159	x_160	y_160
0	NaN	NaN	NaN
1	NaN	NaN	NaN
2	NaN	NaN	NaN
3	NaN	NaN	NaN
4	NaN	NaN	NaN

Figure 5: An example of a `.outline` file

All the attributes associated with the different `larva_id` are stored in `.trx` files. For example, the bending angles of the tail, head, motion velocity, action being performed, and duration. As of now, we will be taking into consideration:

- **numero_larva _num** : Same as the larva ID
- **average_larva_length** : which stores the average larval length (spine)
- **average_motion_velocity_norm**: Stores the average velocity with which the larva is acting.
- **variance_head_angle**: stores the variance of the distribution of the angles made by the head.
- **duration**: stores the total duration for which the action is being performed.
- **start_t**: stores the timestamp at which the larva starts the action
- **end_t**: stores the timestamp at which the larva ends the action

We proceed to create a merged dataframe which includes the action required to render an animation of the larva based on the coordinates.

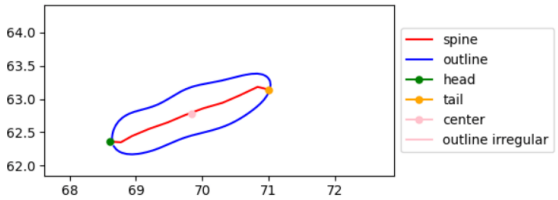


Figure 6: Animated larva (`larva_id = 2`)

5 Generation of synthetic larva coordinates

Important classes and methods

We use the experimental data to generate synthetic larva coordinates and actions. We set the minimum number of actions to be performed at 2 and the maximum

at 8. This can be changed as and when desired.

We first define a list of actions under which we will classify the larva's movements. As seen before, we have six defined actions: cast, hunch, roll, run, and back. We then define the number of points on the spine and that of the outline of the larva (which should be twice the number of points on the spine, on either side, plus the two points for the head and the tail).

Next, we design two classes based on the parameters that will be used to index the database and the features that will be extracted. `IndexQueryParameters` defines the parameters based on which we will be querying the database. For now, we have kept it limited to: **avg_length**, which gives us the average length of the larva, **avg_velocity** giving us the average velocity with which the larva moves, **var_angle** which provides us with the angle between the lines passing through the head and the centre of mass of the larva and the tail and the centre of mass, and **duration** which is the duration for which the larva performs the said action. The next class, `IndexResultItem` defines the features that we want to extract from the database: the unique larva ID given by **numero _larva_num**, the start and end time of the action, provided by **start_t** and **end_t** respectively. We would also require the spine and outline coordinates files.

We will use these classes to query the database according to our needs. We store the index in a **`.json`** file, which is used to help generate the synthetic data we desire.

We create a class `preprocess_curve` that will help us interpolate sequences of coordinates so that all sequences have the same number of points. It contains two functions: `interpolate_sequence`, which returns an interpolated list with any length as desired. We use the `numpy.interp` function. The second function `smooth_sequence` uses **Fast Fourier Transformations** and **Inverse Fast Fourier Transformations** to smoothen a sequence of points. We take the number of harmonics as 7, as discussed earlier.

```
numpy.interp(x, xp, fp, left=None, right=None, period=None)
```

returns the one-dimensional piecewise linear interpolant to a function with given discrete data points (xp, fp), evaluated at x.

Now we will define two important files, `coordinates_frame.py` and `coordinates_frame_sequence.py`, which will culminate in `coordinates_compiler.py`. Then, that would be used in `sequence_generator.py` which will finally help us generate synthetic larva coordinates which we do in `generate_larva_coordinates.py`.

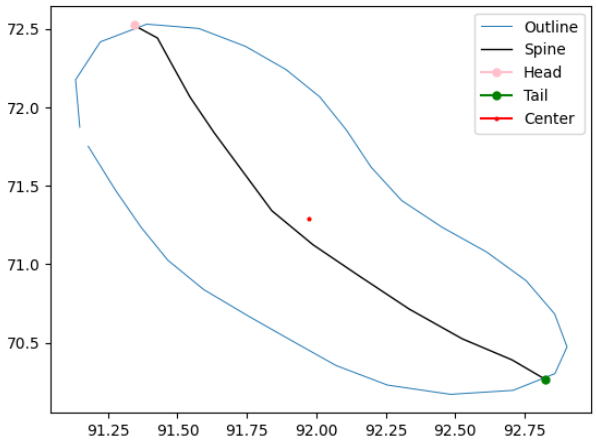


Figure 7: Visualisation of discontinuity during transition of actions ($9^{th} second$)

Limitations and Problems

- The index file generated is too large (5.5 gigabytes), and the `index.json` file generated is broken. However, it works for small portions of the database.
- There is a clear discontinuity between the actions observed in the video above. This is taken care of to some extent using the interpolation of frames, which we will see next.

6 Interpolation of Frames

We employ frame interpolation to resolve the issue of discontinuity, albeit only for larvae with similar (regular) shapes.

The idea is to take two frames during the transition of one action to the other and generate new frames in between to smooth the transition.

The `interpolate` function generates a sequence of `CoordinatesFrame` instances, interpolating between a given first and last frame. This sequence includes the first and last frames, with intermediate frames calculated to ensure the maximum shift between consecutive frames does not exceed a specified value. This is useful for creating smooth transitions or animations between two states.

```
def interpolate(
    first_frame: CoordinatesFrame, last_frame:
        ↪ CoordinatesFrame, maximum_shift=0.05
) -> CoordinatesFrameSequence:
    """
    Returns a sequence of frames with first and last
        ↪ frame defined and interpolated frames
        ↪ in-between.
    The number of intermediate frames is determined
        ↪ automatically in accordance to the maximum
        ↪ shifting distance.

    The value of maximum shift has been determined
        ↪ experimentally.
    """
```

Several helper functions are used for element-wise operations on lists of floats:

- `element_wise_sum(list1, list2)`: Sums corresponding elements of `list1` and `list2`.
- `element_wise_difference(list1, list2)`: Subtracts corresponding elements of `list2` from `list1`.
- `multiply_by_constant(the_list, constant)`: Multiplies each element of `the_list` by `constant`.

The differences between the outlines and spines of the first and last frames are calculated using `element_wise_difference`:

```
    x_outline_diff =
        ↪ element_wise_difference(last_frame.x_outline,
        ↪ first_frame.x_outline)
    y_outline_diff =
        ↪ element_wise_difference(last_frame.y_outline,
        ↪ first_frame.y_outline)
    x_spine_diff =
        ↪ element_wise_difference(last_frame.x_spine,
        ↪ first_frame.x_spine)
    y_spine_diff =
        ↪ element_wise_difference(last_frame.y_spine,
        ↪ first_frame.y_spine)
```

The maximum distance shift between the first and last frames is computed, and the number of frames to interpolate is determined:

```
    max_distance =
        ↪ first_frame.get_maximum_distance_shift(last_frame)
    n_of_frames_to_interpolate =
        ↪ int(math.ceil(max_distance / maximum_shift)) - 1
```

Intermediate frames are generated by advancing a certain percentage towards the last frame:

```
    frames_interpolated = []
    for i in range(n_of_frames_to_interpolate):
        percentage_to_advance_by = (i + 1) /
            ↪ n_of_frames_to_interpolate

        new_x_spine = element_wise_sum(
            first_frame.x_spine,
            multiply_by_constant(x_spine_diff,
                ↪ percentage_to_advance_by),
        )

        new_y_spine = element_wise_sum(
            first_frame.y_spine,
            multiply_by_constant(y_spine_diff,
                ↪ percentage_to_advance_by),
        )

        new_x_outline = element_wise_sum(
```

```
            first_frame.x_outline,
            multiply_by_constant(x_outline_diff,
                ↪ percentage_to_advance_by),
        )

        new_y_outline = element_wise_sum(
            first_frame.y_outline,
            multiply_by_constant(y_outline_diff,
                ↪ percentage_to_advance_by),
        )
```

The function returns a `CoordinatesFrameSequence` containing all the interpolated frames:

```
    return CoordinatesFrameSequence(frames_interpolated)
```

We use the `interpolate` function in the `CoordinatesCompiler` class to help generate the smooth sequence of frames. Note that we try to limit the number of frames in a way (which can be done manually) not to make it obvious that we have performed interpolation. Usually, the number of interpolated frames that work best is 4 to 5.

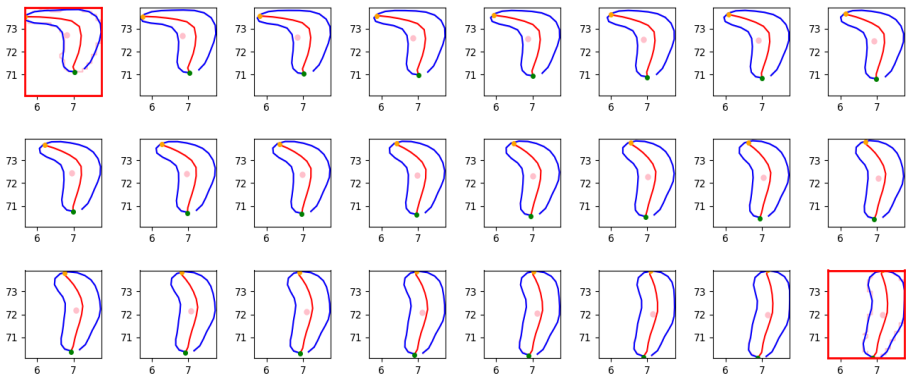


Figure 8: An example of interpolated frames. The frames with red boundaries indicate the original frames between which the rest of the frames were generated.

We fix the head point and the mid point and shift the other coordinates.

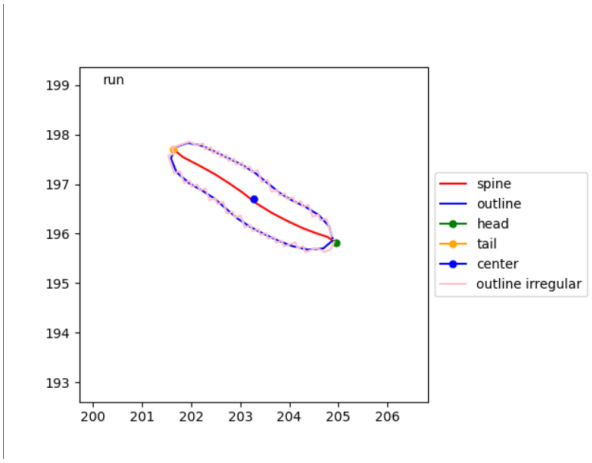
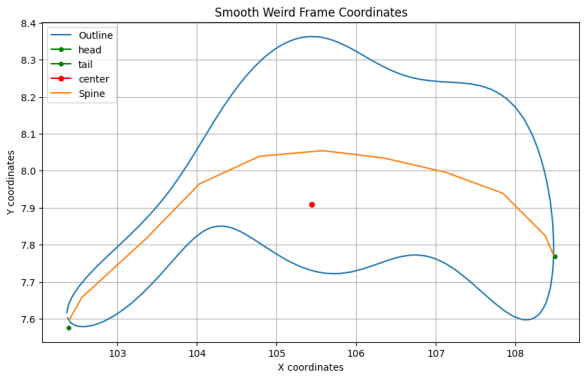


Figure 9: An example of a synthetically generated and interpolated action sequence

Limitation

The major limitation we encountered is that the shape of the larvae changes from one `larva_id` to another. This would mean that the new larva we create changes its shape while performing actions, which is impossible. Hence, there is a great need for a regularization class which could be utilized, especially to regularize the recorded shape of the larva, which has abnormal shapes. Two primary reasons we get abnormal larva shapes is that as the length of the larva increases, its shape tends to stray away from the norm. In addition, poor experimental conditions, such as improperly drying the larvae while conducting the experiments, may lead to such shapes.



(a) Shape maybe due to poor experimental conditions

(b) Shape due to large length of larva (6mm)

Figure 10: Examples of anomalous larvae shape

7 Regularization of Larva Shape

We first took up the example larva (figure 10(a)), and tried to regularise its shape using the following method:

- Divide the spine into four parts using three points:
  - Midpoint
  - Quarter point (1/4 × number of spine points)
  - Third quarter point (3/4 × number of spine points)
- Draw a line passing through each point, parallel to the Y-axis.
- Collect the outline points near the outline's intersection and each parallel line.

- For each of the three spine points, select the outline point at the maximum distance from the spine point to form the new outline.
- Include a few points near the head and the tail in the new outline to make it more realistic.

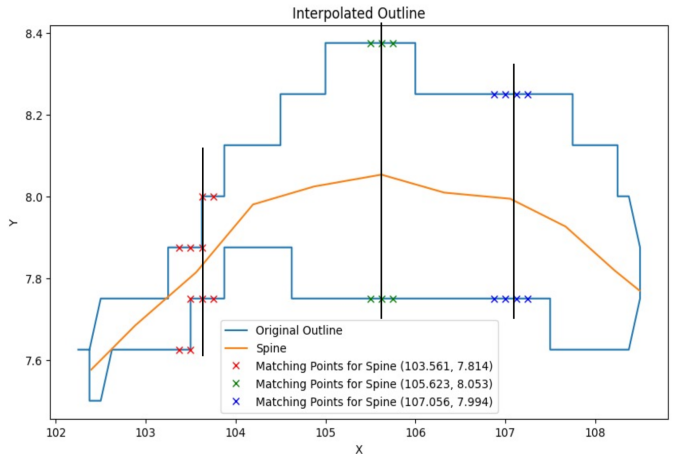
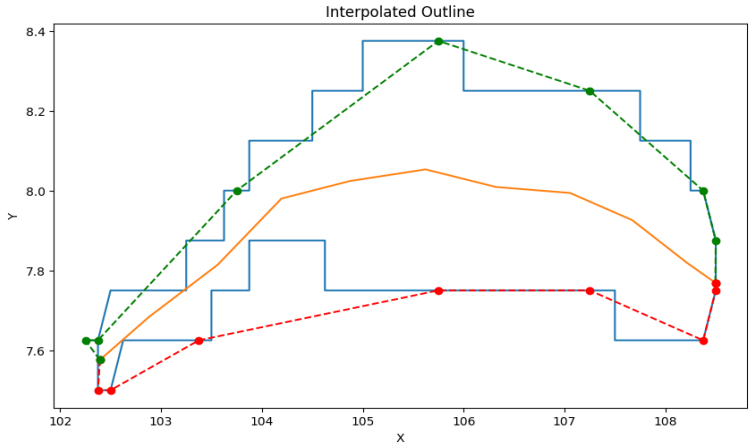


Figure 11: Lines parallel to Y-axis passing through the chosen spine points

Next, after adding a few points near the head and tail of the larva, we join the points to get a regularised shape:



Although this somewhat improves the outline contour, it is far from what we want to achieve.
Hence, we will try and implement the physics-based regularisation of shape as mentioned in the next section.

Physics informed regularization of larva shape

Recording larvae with a wide-field view allows analyzing up to 100 larvae per plate, but it results in lower-resolution images. The large-scale experiments often included not perfectly dried larvae, leading to abnormal contour shapes. The impurities of the Agar further contributed to these irregularities. Such abnormalities risk misclassifying larval behaviour and introducing bias into statistical analyses. A regularization procedure using physics-informed Bayesian inference was developed to address this, ensuring accurate larval shape representation.

Preprocessing

Multi-Worm Tracker extracts contours with various points depending on the larvae's size in each frame. We denote this contour by $f(i) = (x(i), y(i))$ for $i \in 1, 2, \ldots, N_{tracking}$ with $N_{tracking}$ the number of points on the contour. The shape was regularized by fixing the number of contour points to N = 50 coupled with a low-pass filtering. In particular, the contours were generated by retaining the K lowest modes of the Fourier decomposition of the recorded contour,

$$S_n = a_0(f) + \sum_{k=0}^{K}[a_k(f)cos(\tfrac{kn2\pi}{N}) + b_k(f)sin(\tfrac{kn2\pi}{N})]$$

with a_k and b_k being the Fourier coefficients,

$$a_k = \sum_{i=1}^{N_{tracking}} \frac{f(i)}{N_{tracking}} cos(\tfrac{2\pi ki}{N_{tracking}})$$

$$b_k = \sum_{i=1}^{N_{tracking}} \frac{f(i)}{N_{tracking}} sin(\tfrac{2\pi ki}{N_{tracking}})$$

The shape was reconstructed with the K = 7 lowest harmonics, a number chosen empirically to prevent discontinuities. This first reconstruction ensured screen-scale regularisation of larvae contours regardless of their variability in size and shape.

Simplified physics model of larva

We employed a minimal yet effective 2D physics model to model a larva's dynamic shape. This model represents the larva as an elastic contour with active membrane

energy. The total energy E of the larva is the sum of its kinetic energy E_k, surface energy E_S, and bending energy E_b:

$$E = E_k + E_S + E_b. \tag{1}$$

The kinetic energy is given by:

$$E_k = \int_\Omega \frac{1}{2}\rho v(s)^2\, ds = \sum_{i=1}^{N} \frac{m}{2}\left(\frac{[x_i(t) - x_i(t-\Delta t)]^2 + [y_i(t) - y_i(t-\Delta t)]^2}{(\Delta t)^2}\right), \tag{2}$$

where Ω is the contour's surface, ρ is the surface density, $v(s)$ is the speed at point s, m is the total mass of the membrane, and Δt is the time lapse between images.

The surface energy is given by:

$$E_S = \int_\Omega K\, ds^2 = \sum_{i=1}^{N} K\left((x_i - x_{i-1})^2 + (y_i - y_{i-1})^2\right), \tag{3}$$

where K is the elastic modulus.

The bending energy is given by:

$$E_b = \int_\Omega 2k(C - c)^2\, ds = \sum_{i=1}^{N} 2k\theta_i^2, \tag{4}$$

where k is the bending modulus, C is the mean curvature, c is the spontaneous curvature defined by $c\hat{\mathbf{n}} = \frac{d\hat{\mathbf{t}}}{ds}$ with $\hat{\mathbf{n}}$ as the unit normal vector and $\hat{\mathbf{t}}$ as the unit tangent vector of the contour in curvilinear coordinates. This curvature equals $c_i = \theta_i$, and the mean curvature is set to zero for a discrete point.

Inference

To infer the regularized shape $\Sigma = \{M_1', M_2', \ldots, M_N'\}$ of the larva, Bayesian inference was utilized. The posterior distribution is given by:

$$P(\Sigma|S) \propto P(S|\Sigma)P(\Sigma), \tag{5}$$

where $P(S|\Sigma)$ is the likelihood of the model and $P(\Sigma)$ is the prior that incorporates our physical model:

$$P(\Sigma) \propto e^{-[E_S(\Sigma)+E_k(\Sigma)+E_b(\Sigma)]}. \tag{6}$$

The likelihood ensures the inferred contour closely matches the recorded contour according to a quadratic loss:

$$P(S|\Sigma) = e^{-\frac{1}{\lambda}\sum_i \|M_i - M_i'\|^2}, \tag{7}$$

where $\lambda = \frac{1}{2}$.

Thus, the log-posterior distribution is:

$$\log P(\Sigma|S) = E_S + E_k + E_b - \frac{1}{\lambda}\sum_i \|M_i - M_i'\|^2. \tag{8}$$

The hyperparameters (mass m, curvature coefficient k, and elastic modulus K) were set such that the three energy terms have similar weights: $m = 1$, $k = 1$, and $K = 5$. To address high-curvature anomalies, the kinetic energy was capped using $E_k^{\text{eff}} = \tanh\left(\frac{E_k}{\sigma}\right)$ with $\sigma = 100$. The maximum a posteriori (MAP) regularized contour Σ was inferred using stochastic gradient descent. Note that while we rely solely on the MAP of the shape in downstream analysis and not on the full posterior distribution, it is accessible using Markov Chain Monte Carlo sampling if necessary.

I will implement this using Python to create a universal regularisation class for the larva.

8 Plan of Action

- The current action plan is to implement the physics-based regularisation using Python and check if we can mimic the original results.
- I would also like to improve the previous model by increasing the number of spinal points we consider to see if it is valid.

9 Conclusion

We are developing and implementing a method to generate synthetic behavioural data for Drosophila larvae, incorporating key characteristics observed in experiments. The primary aim is to enhance the robustness of existing classifiers and enable more accurate identification of various larval behaviours, including novel and edge-case actions. We are trying to address the challenges of generating realistic larval shapes and transitions between actions using Bayesian inference, frame interpolation, and regularisation techniques.

Implementing the interpolation of frames using Python has improved the accuracy of the generated data. However, limitations such as the large index file size and the need for shape regularization indicate areas for further refinement. The proposed next steps include enhancing the regularization process.

Overall, the project demonstrates the potential of synthetic data generation to improve the performance of machine learning classifiers in behavioural studies. The methodologies developed here can be adapted for various applications, including pre-experimental protocol testing and the analysis of neurodegenerative diseases. Continued refinement and testing will ensure these tools can be effectively integrated into broader research frameworks, contributing to neuroscience and behavioural biology advancements.

10 Bibliography

- Jovanic, T. (2020). Studying neural circuits of decision-making in Drosophila larva. Journal of Neurogenetics, 34(1), 162–170.
- Blanc, A., Laurent, F., Barbier–Chebbah, A., Cocanougher, B. T., Jones, B. M., Hague, P., Zlatic, M., Chikhi, R., Vestergaard, C. L., Jovanic, T., Masson, J., & Barre, C. (2024). Statistical signature of subtle behavioural changes in large-scale behavioural assays. bioRxiv (Cold Spring Harbor Laboratory).
- Swierczek, N. A., Giles, A. C., Rankin, C. H., & Kerr, R. A. (2011). High-throughput behavioral analysis in C. elegans. Nature Methods, 8(7), 592–598.
- Ohyama, T., Jovanic, T., Denisov, G., Dang, T. C., Hoffmann, D., Kerr, R. A., & Zlatic, M. (2013). High-Throughput analysis of Stimulus-Evoked behaviors in drosophila larva reveals multiple Modality-Specific escape strategies. PloS One, 8(8), e71706.
- Masson, J., Laurent, F., Cardona, A., Barré, C., Skatchkovsky, N., Zlatic, M., & Jovanic, T. (2020). Identifying neural substrates of competitive interactions and sequence transitions during mechanosensory responses in Drosophila. PLOS Genetics, 16(2), e1008589.
- Bishop CM. Pattern Recognition and Machine Learning (Information Science and Statistics). Springer; 2011.
- Tan, F. H. P., PhD, & Azzam, G. (2017). Drosophila melanogaster: Deciphering Alzheimer's Disease. Malaysian Journal of Medical Sciences, 24–24(2), 6–20.
- Muñoz-Soriano, V., & Paricio, N. (2011). Drosophila Models of Parkinson's Disease: Discovering relevant pathways and novel therapeutic strategies. Parkinson's Disease, 2011, 1–14.
- Lehman, M., Barré, C., Hasan, M. A., Flament, B., Autran, S., Dhiman, N., Soba, P., Masson, J., & Jovanic, T. (2023). Neural circuits underlying context-dependent competition between defensive actions inDrosophilalarva. bioRxiv (Cold Spring Harbor Laboratory).
- De Tredern, E., Manceau, D., Blanc, A., Sakagiannis, P., Barre, C., Sus, V., Viscido, F., Hasan, M. A., Autran, S., Nawrot, M., Masson, J., & Jovanic, T. (2023). Feeding-state dependent modulation of reciprocally interconnected

inhibitory neurons biases sensorimotor decisions inDrosophila. bioRxiv (Cold Spring Harbor Laboratory).

- Experiments with Drosophila for Biology Courses
- Li, H., Kroll, J. R., Lennox, S. M., Ogundeyi, O., Jeter, J., Depasquale, G., & Truman, J. W. (2014). A GAL4 driver resource for developmental and behavioral studies on the larval CNS of Drosophila. Cell Reports, 8(3), 897–908.
- Gowda, S. B. M., Salim, S., & Mohammad, F. (2021). Anatomy and neural pathways modulating distinct locomotor behaviors in drosophila larva. Biology, 10(2), 90.
- A Parvathi, V., a, A Amritha, S., a, & A Paul, F. (2009). WONDER ANIMAL MODEL FOR GENETIC STUDIES - Drosophila Melanogaster –ITS LIFE CYCLE AND BREEDING METHODS – A REVIEW. In Sri Ramachandra University, Sri Ramachandra Journal of Medicine: Vol. II (Issue 2, p. 33).
- PLOS ONE. (n.d.). Segmentation of Drosophila larva. Figshare.
- pytrxmat. (2023, November 16). PyPI.
- Lehmann, K. S., Hupp, M. T., Jefferson, A., Cheng, Y., Sheehan, A. E., Kang, Y., & Freeman, M. R. (2023). Astrocyte-dependent local neurite pruning and Hox gene-mediated cell death in Beat-Va neurons. bioRxiv (Cold Spring Harbor Laboratory).
- De Tredern, E., Manceau, D., Blanc, A., Sakagiannis, P., Barre, C., Sus, V., Viscido, F., Hasan, M. A., Autran, S., Nawrot, M., Masson, J., & Jovanic, T. (2023b). Feeding-state dependent modulation of reciprocally interconnected inhibitory neurons biases sensorimotor decisions inDrosophila. bioRxiv (Cold Spring Harbor Laboratory).

11 Appendix

Index Generation

Generate Index

```
python3 generate_index.py [-h] path_to_database_directory
output_filename
```

A program for generating index for a file database with larvae's experimental data

Positional arguments:

- `path_to_database_directory`
- `output_filename`

Options:

- `-h, -help` show this help message and exit

Execution

```
python3 src/generate_index.py t15sample index.json
```

JSON Data


```
[
  {
    "action": "run",
    "average_larva_length": 4.619,
    "average_motion_velocity_norm": 0.006,
    "variance_head_angle": 0.0,
    "duration": 0.327,
    "entities": [
      {
        "numero_larva_num": 1.0,
        "start_t": 35.318,
        "end_t": 35.645,
        "spine_file": "t15sample/... .spine",
        "outline_file": "t15sample/... .outline"
      }
    ]
  },
  {
    "action": "run",
    "average_larva_length": 4.7,
    "average_motion_velocity_norm": 0.005,
    "variance_head_angle": 0.002,
    "duration": 0.635,
    "entities": [
      {
        "numero_larva_num": 1.0,
        "start_t": 81.242,
        "end_t": 81.877,
        "spine_file": "t15sample/... .spine",

        "outline_file": "t15sample/... .outline"
      }
    ]
  }
]
```

Exploratory Data Analysis

Transitional probability between actions

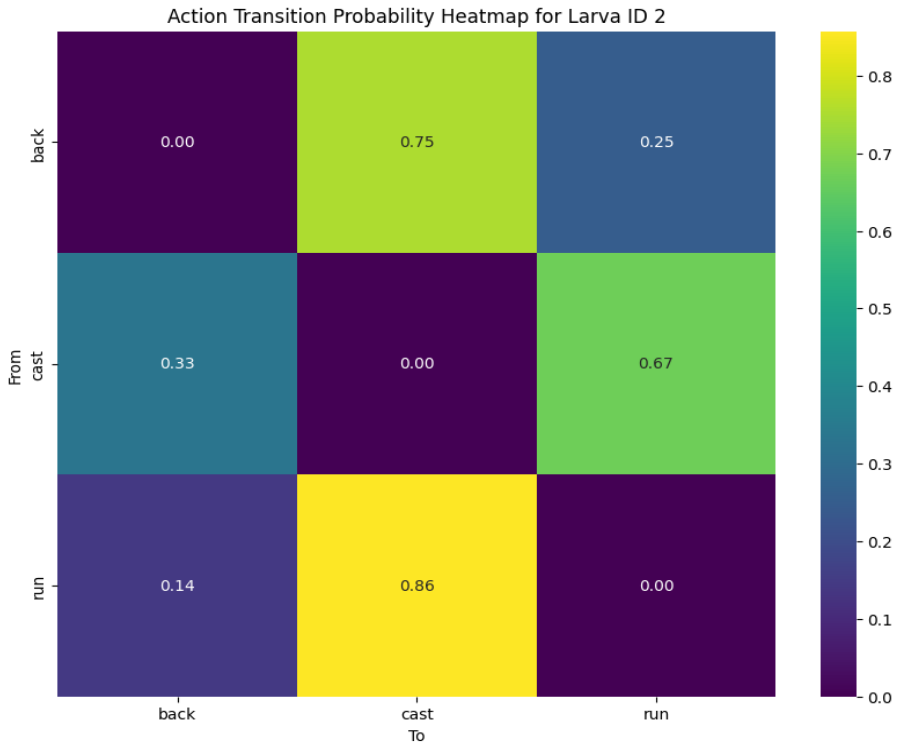


Figure 12: Pearson's heatmap of transitional probabilities between any two actions for `larva_id = 2`

A few feature distributions

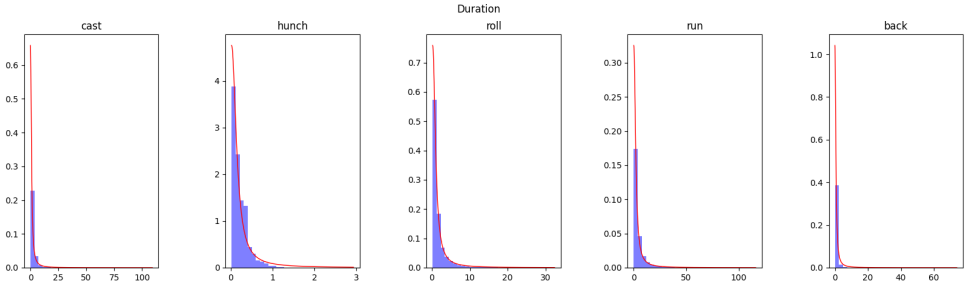


Figure 13: Distribution of average duration per action

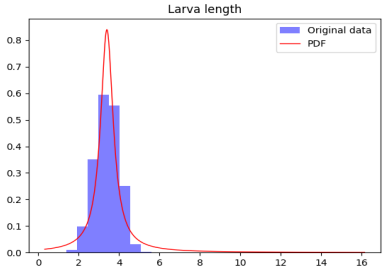


Figure 14: Distribution of average larva length

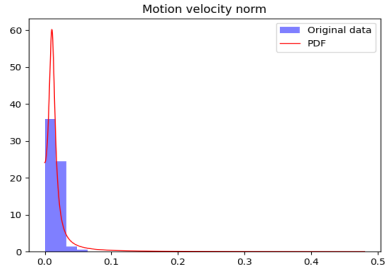


Figure 15: Distribution of average average motion velocity norm

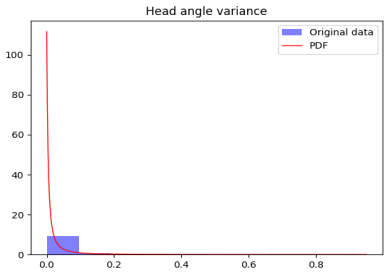


Figure 16: Distribution of variance of head angle

Filtering NaN values

```
    def filter_out_nan(values):
    return [x for x in values if not math.isnan(x)]
```

Interpolation and Smoothening Functions

```
    def interpolate_sequence(original_list, new_length):
    original_length = len(original_list)
    x = np.arange(original_length)
    new_x = np.linspace(0, original_length - 1,
       ↪ new_length)

    return np.interp(new_x, x, original_list).tolist()
```

```
    def smooth_sequence(values, n_harmonics=7):
    """
    Returns values points smoothed by performing FFT and
       ↪ inverse FFT on original values and taking the
    first n_harmonics (7 by default).
    Nan values are filtered out.
    """
    original_length = len(values)

    values = filter_out_nan(values)

    # Center signal to avoid shifting after performing
       ↪ FFT and inverse FFT
    signal_values_centered = values - np.mean(values)
    rft = np.fft.rfft(signal_values_centered)
```

```
    rft[n_harmonics:] = 0

    smoothed_values = np.fft.irfft(rft) + np.mean(values)

    # Interpolate to restore the original length of the list
    return interpolate_sequence(smoothed_values.tolist(), original_length)
```

Sequence generator class

```
    """
A program to generate a sequence of actions according to transition probabilities found in real data.
"""

from typing import List, Union
import numpy as np
import random
from query_index import query
from model_action import ACTION_TYPES
from action_generator import ActionGenerator
from coordinates_compiler import CoordinatesCompiler
from model_action import ActionType
from model_index import IndexQueryParameters


def random_number_normal_distribution(
    min: float, max: float, mean: float = 0, std: float = 1
) -> float:
    while True:
        value = np.random.normal(mean, std)
        if min <= value <= max:
            return value


def random_number_uniform_distribution(min: float, max: float) -> float:
    return np.random.uniform(min, max)

```

```
28
29 def random_duration_s(min: float = 2, max: float = 20) ->
   ↪ float:
30     return random_number_normal_distribution(min, max,
          ↪ (max + min) / 2)
31
32
33 def random_action_type() -> ActionType:
34     return random.choice(ACTION_TYPES)
35
36
37 class SequenceGenerator:
38     def __init__(self, path_to_index_file: str) -> None:
39         self.path_to_index_file = path_to_index_file
40
41     def __exists_in_database(
42         self,
43         action: ActionType,
44         duration: float,
45         larva_length: float,
46         motion_velocity_norm: float,
47         variance_head_angle: float,
48     ) -> bool:
49         params: IndexQueryParameters = {
50             "action": action,
51             "avg_length": larva_length,
52             "avg_velocity": motion_velocity_norm,
53             "var_angle": variance_head_angle,
54             "duration": duration,
55         }
56
57         similar_entities = query(
58             self.path_to_index_file,
59             params,
60             atol=0.1,  # Allow some tolerance interval
61         )
62
63         return len(similar_entities) > 0
64
65     def __previous_action_type(
66         self,
67         action_generators: List[ActionGenerator],
68     ) -> Union[ActionType, None]:
```

```
69          if len(action_generators) == 0:
70              return None
71
72          return action_generators[-1].action_type
73
74      def generate(
75          self, min_n_actions=2, max_n_actions=8,
               ↪ time_interval_ms=80
76      ) -> CoordinatesCompiler:
77          """
78          Generates a random sequence of actions and
               ↪ returns an object with compile() method to
79          retreive spine and contour coordinates.
80          """
81
82          n_actions = int(
83              random_number_uniform_distribution(min_n_actions,
                   ↪ max_n_actions)
84          )
85
86          larva_length =
               ↪ random_number_uniform_distribution(2, 6)
87
88          motion_velocity_norm =
               ↪ random_number_uniform_distribution(0.001,
               ↪ 0.1)
89          variance_head_angle =
               ↪ random_number_uniform_distribution(0.001,
               ↪ 0.1)
90
91          print("\n# of actions in a sequence:", n_actions)
92
93          action_generators: List[ActionGenerator] = []
94
95          while len(action_generators) < n_actions:
96              action_type = random_action_type()
97              duration = random_duration_s()
98
99              if (
100                 self.__exists_in_database(
101                     action_type,
102                     duration,
103                     larva_length,
```

```
                    motion_velocity_norm=
                    motion_velocity_norm,
                    variance_head_angle=
                    variance_head_angle,
                )
                and self.__previous_action_type
                (action_generators) != action_type
            ):
                print(f"\nNext action: {action_type},
                   ↪ duration: {duration}s")
                action_generators.append(
                    ActionGenerator(
                        action_type,
                        duration,
                        larva_length,
                        motion_velocity_norm,
                        variance_head_angle,
                        self.path_to_index_file,
                    )
                )

        return CoordinatesCompiler(action_generators)
```

Action Generator class

```
    from query_index import query
import random
from database import read_coordinates_file
from model_action import ActionType
from model_index import IndexQueryParameters
from coordinates_frame_sequence import
   ↪ CoordinatesFrameSequence
from coordinates_frame import CoordinatesFrame


class ActionGenerator:
    def __init__(
        self,
        action_type: ActionType,
        duration: float,
        larva_length: float,
```

```
        motion_velocity_norm: float,
        variance_head_angle: float,
        path_to_index_file: str,
    ) -> None:
        self.action_type = action_type
        self.duration = duration
        self.larva_length = larva_length
        self.motion_velocity_norm = motion_velocity_norm
        self.variance_head_angle = variance_head_angle
        self.path_to_index_file = path_to_index_file

    def __select_random_exemplar(self, atol: float):
        """
        Selects random exemplar with similar features from the database.
        """
        params: IndexQueryParameters = {
            "action": self.action_type,
            "avg_length": self.larva_length,
            "avg_velocity": self.motion_velocity_norm,
            "var_angle": self.variance_head_angle,
            "duration": self.duration,
        }

        similar_entities = query(
            self.path_to_index_file,
            params,
            atol=atol,  # Allow some tolerance interval
        )

        if len(similar_entities) == 0:
            return None

        random_entity = random.choice(similar_entities)
        random_entity.update(params)  # Adding initial paramateres to the larva found
        return random_entity

    def generate(self, atol=0.1, time_interval_ms=80) -> CoordinatesFrameSequence:
        """
        Generates coordinates of larva according to the action type and parameters specified.
```

```
        """
        random_exemplar =
           ↪ self.__select_random_exemplar(atol)

        spine_coordinates =
           ↪ read_coordinates_file(random_exemplar
        ["spine_file"])
        outline_coorinates =
           ↪ read_coordinates_file(random_exemplar
        ["outline_file"])

        spine_coordinates = spine_coordinates[
            (
                spine_coordinates["t"].between(
                    random_exemplar["start_t"],
                    random_exemplar["end_t"],
                    inclusive="both",
                )
            )
            & (spine_coordinates["larva_num"] ==
               ↪ random_exemplar["numero_larva_num"])
        ]

        outline_coorinates = outline_coorinates[
            (
                outline_coorinates["t"].between(
                    random_exemplar["start_t"],
                    random_exemplar["end_t"],
                    inclusive="both",
                )
            )
            & (outline_coorinates["larva_num"] ==
               ↪ random_exemplar["numero_larva_num"])
        ]

        frames = [
            CoordinatesFrame(
                x_s,
                y_s,
                x_o,
                y_o,
                x_s[0],
                y_s[0],
```

```
                x_s[-1],
                y_s[-1],
                str(self.action_type),
            )
            for x_s, y_s, x_o, y_o in zip(
                spine_coordinates.filter(regex="^x\d*").
                values.tolist(),
                spine_coordinates.filter(regex="^y\d*").
                values.tolist(),
                outline_coorinates.filter(regex="^x\d*").
                values.tolist(),
                outline_coorinates.filter(regex="^y\d*").
                values.tolist(),
            )
        ]

        return CoordinatesFrameSequence(frames,
            ↪ time_interval_ms)
```