École Polytechnique Fédérale de Lausanne

Semester project report outline - Fly 'social reaching' behaviour

Pavan Ramdya Neuroengineering laboratory

Marianne Civit Ardevol 325056 -
Primary supervisor : Thomas Ka Chung Lam

June 14, 2024

Contents

1 Abstract

Understanding the behaviours of drosophila can provide great insight into their complex neural mechanisms and why goal directed sensorimotor transformations take place.
One of these transformations is the movement of the middle leg of fly A when fly B passes next to it. In order to better understand this, it is necessary to have repeatability in the experiment, allowing consistency and robust conclusions. Since this task becomes complicated when working with animals, it is interesting to develop a 'robotised' version of fly B which will always perform the needed movement. This experiment consists of a closed loop control of the situation with visual input from a camera and output from a robotized magnet simulating the second fly.
In this report, the computer vision part of the algorithm will be analysed and described. Its main developed functionalities are : fly position tracking, finding of the angle with respect to the head position, determining if the fly's stomach or back is being seen, tracking of the leg movement and determining which trajectory the robot can take.

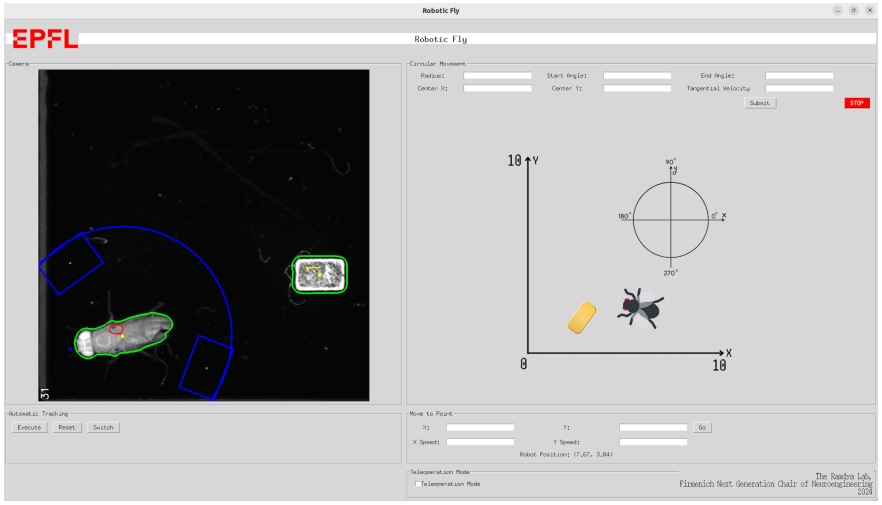


Figure 1: User interface for drosophila "social reaching" experiments

2 Introduction

The development of the Drosophila connectome provides deeper insights into the relationship between behavior and structural neuronal connectivity. This advancement enables us to observe goal-directed sensorimotor transformations and the integration of sensory modalities in the fly, revealing how these processes are linked to specific neuronal connections.

The sensorimotor transformation studied in this report, usually called ”social reaching behaviour”, is the movement of a fly’s middle leg when another fly passes next to it. The figure bellow illustrates this movement:

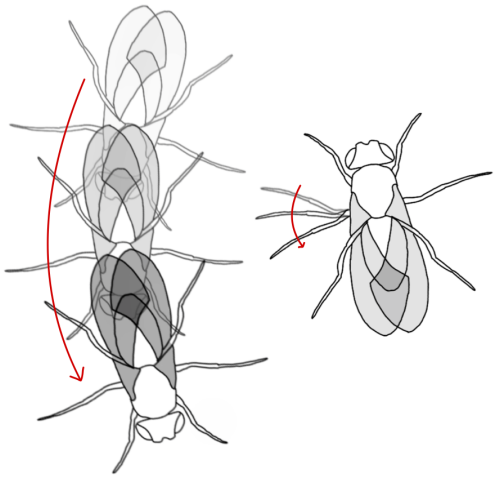


Figure 2: ”Social reaching” behaviour

Understanding this single behaviour is only the surface of a much deeper and complex element. Many questions can be explored, such as identifying which cues trigger movement, determining the precise trajectory of the middle leg and the distance needed between both flies, and understanding the neural connections related to this movement.

Many hypotheses can be made but only experiments and robust findings can conclude. In order to do so, repeatability is of crucial importance. One way to achieve this with animals such as drosphila is to replace one of them by a controlled robot. In our case, it will be the moving drosophila that will be replaced by a robotized magnet. This allows the user to control the exact movement of the magnet and to obtain results with a repeatable trajectory. Furthermore, being able to control the trajectory allows the user to find correlations between a certain trajectory type and certain movement of the leg.

When using a robotized element with a real drosophila, it is important to have a closed loop control such that the model can react to its environment. Indeed, it is necessary to obtain feedback regarding where the fly is or where it has moved to such that the robot can correctly adapt its position of trajectory. There is a constant communication between the computer vision algorithm and the controller, giving information about where the fly is at each timestep and where the magnet can move to.

Here, the camera vision input will provide information regarding all of the aspects of the real drosophila such that the control of the robot can react and move accordingly and therefore close the loop. The main information extracted from the vision input are the following:

1. Position of the real fly.

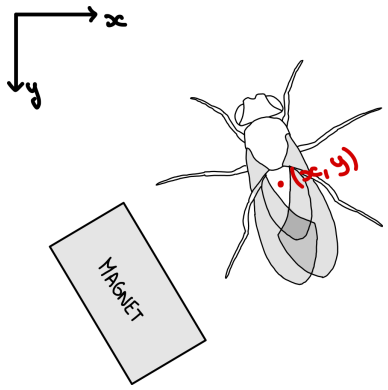


Figure 3: Fly position

   Here, the fly mostly needs to be distinguished from the magnet, and the (x, y) position of its center of mass is recorded.

2. Angle of the fly and where its head is pointing

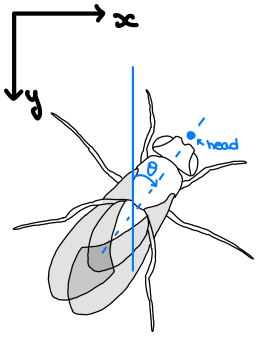


Figure 4: Angle of the head

   The blue dot indicates where the head is and the angle is given from the vertical axis to the head position.

3. Whether we are seeing the fly's back or stomach

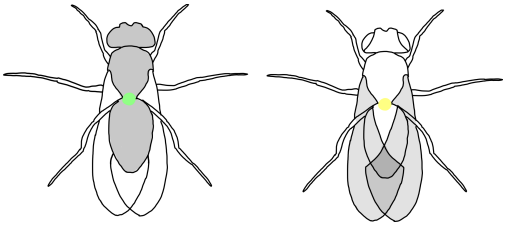


Figure 5: Which way the fly is facing

The green dot indicates that the fly is facing up (we see its stomach) and the yellow dot indicates that it is facing down (we see its back).

4. Tracking the position of the middle legs

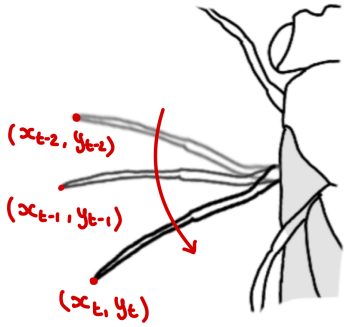


Figure 6: Tracking of the middle leg

5. Determining the maximum angle positions of the robotized fly

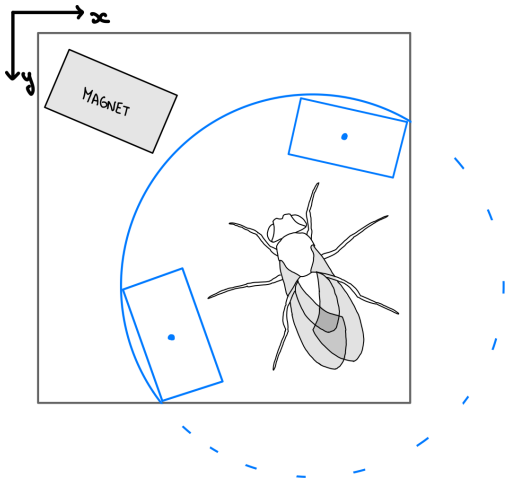


Figure 7: Maximum angle positions of the magnet

This is the information that will be provided to the magnet controller such that it knows which trajectory it can perform in order to trigger a social reaching behaviour.

This set of informations needs to be extracted during live streaming of the experiment. Indeed, the feedback controller must be able to provide information in a fast enough manner such that when the controller reacts, the information is still valid. Once this part is done, it will be integrated with a GUI and provided as input to the controller. This report focuses on the computer vision algorithm.

3 State-of-the-art

3.1 Tracking code

The visual sensor information provided by the cameras allow to provide information to the controller and move the magnet's position accordingly.

Aliaa Diab, a previous student at the EPFL already implemented a computer vision algorithm able to track up to 10 flies on video and during live streaming. The aim of that project was to implement a closed loop dual color optogenic stimulation system with free running drosophila. Therefore, the algorithm needed to be able to track flies, keep their identities and record their trajectories whilst not taking too much computational power and time.
In order to do such an algorithm, Aliaa Diab used the library OpenCV on python along with image processing techniques to detect the flies through their contours. The main process for detecting drosophilae was the following:

1. Substraction of the background
2. Gaussian blur of the image
3. Image thresholding with Otsu's algorithm
4. Contour detection
5. Contour correction if it didn't find the correct number of flies through erosion and dilatation

With this algorithm, Aliaa Diab was able to obtain up to 98% detection accuracy on recorded videos.

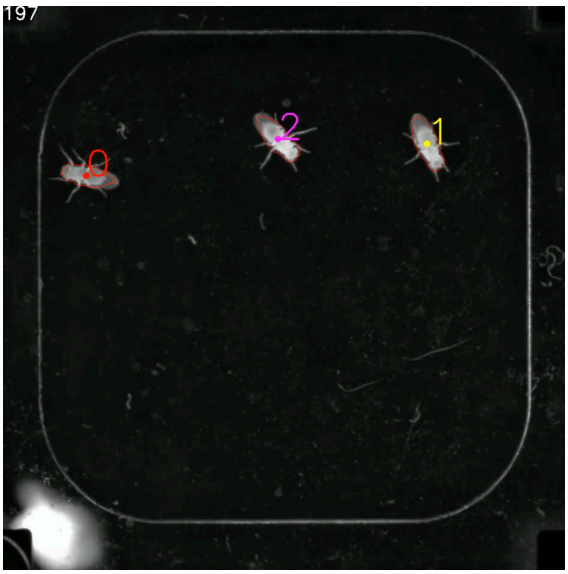


Figure 8: Aliaa Diab's tracking algorithm

Furthermore, Aliaa Diab implemented a code which would keep track of each fly's identity such that they stay the same throughout the video. This was done through a hungarian algorithm using the current position of the flies as well as the previous position and re-assigning the fly's ID to their corresponding position.

Regarding the setup of the current report's project, this code can serve as a solid starting point for the position tracking of the flies. Indeed, some aspects of this project are different and need to be integrated and adapted. These mostly concern the number of flies, the added features that need to be detected such as the fly's head position, ventral and dorsal view and most importantly the use of the robotized magnet. Adding a different object to the code will require a modification of the algorithm such that it can be detected and treated accordingly. Therefore, Aliaa Diab's code has been used as a first algorithm for tracking and has been very much modified and tuned to this specific task.

3.2 Leg tracking code

Tracking the precise location of the middle legs of Drosophila flies provides valuable insights into their trajectories and movement patterns. This detailed information can reveal the leg's position relative to neural activation and the specific movements or trajectories performed in response to the movement of a robotic magnet. In the future, it can even be envisaged to automate the detection of social reaching behaviors. Furthermore, quantifying and describing the leg's trajectory through mathematical equations can provide a robust representation of these movements, enhancing our understanding and ability to model them.

Given that the middle leg of a Drosophila fly constitutes a very small part of the entire frame captured by the camera, simple image processing techniques are inadequate for tracking its movement accurately. Consequently, employing deep neural networks can be highly effective for this application. Currently, there are few algorithms specifically designed for this purpose where the fly is viewed from below and live tracking is desired.

Some deep learning-based pose estimation interfaces are well-suited for this type of problem. One such framework is SLEAP (Social LEAP Estimates Animal Poses), which leverages deep learning to learn from provided data and track complex elements such as the Drosophila leg. According to their webpage, ”SLEAP allows you to train and use deep learning-based models that can automatically track the movements of any body parts of any number or type of animal from recorded videos. This enables precise and quantitative analysis of behavioral data”

Its main working framework is the following

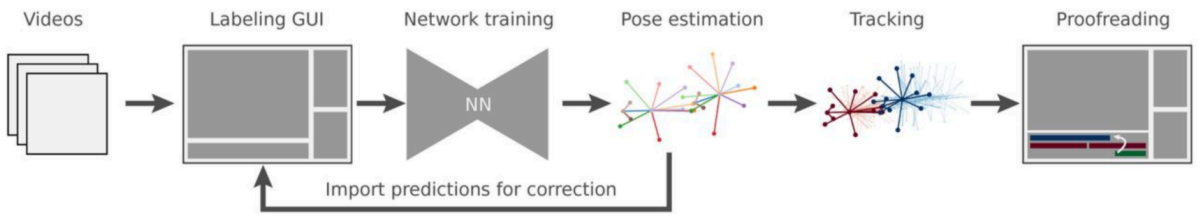


Figure 9: SLEAP framework

This deep network interface is user-friendly, well-documented, and particularly effective with animals like Drosophila, which is why it was chosen for this project. Its ease of use allows training of deep learning models using provided data and enabling precise tracking of complex body parts such as Drosophila legs.

Moreover, once trained, the deep network model can be exported for seamless integration into live stream applications. This feature enhances the versatility of SLEAP, enabling real-time tracking of Drosophila leg movements as they occur.

While SLEAP was used as the primary interface due to its tailored capabilities and proven efficacy, alternative interfaces like DeepLabCut and DeepLabStream offer additional options. These interfaces, while potentially suitable for similar tasks, vary in their specific functionalities and may provide alternative approaches to pose estimation and tracking challenges.

4 Methods

This section will go through every element that the tracking algorithm should be able detect as specified in the introduction 2 and how they have been implemented.
In the initial stages, as the robotic fly was still in development and refinement, the tracking algorithm was initially configured to monitor and track two flies simultaneously. Subsequently, as development progressed and the robotic fly (magnet) became operational, the algorithm was then modified to detect the fly and define the magnet's feasible positions around it. Furthermore, to simplify the coding process and rapidly solve problems and refine algorithms, the tracking code was initially developed for offline video tracking. This approach allowed the use of repeatable data from videos, ensuring consistent elements for testing and debugging. Once the code functioned correctly with this static data, it was then adapted for live streaming, enabling real-time tracking of the experiment.

To gain a clearer perspective of the camera's viewpoint, seeing the experiment setup can be useful:

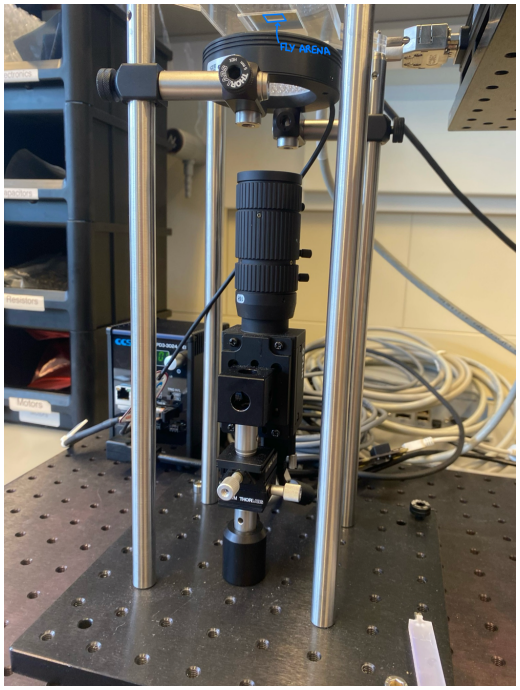


Figure 10: Camera setup from side view

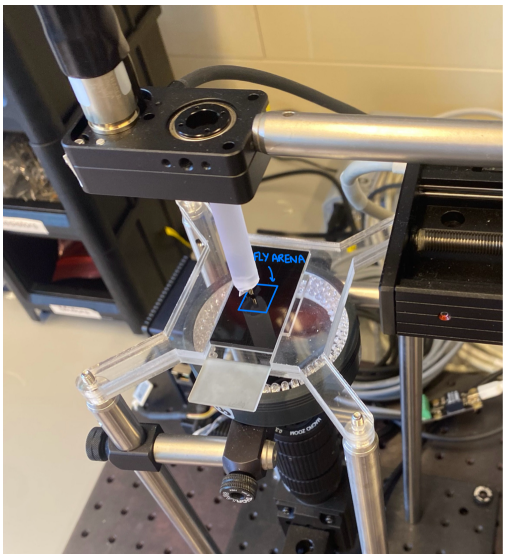


Figure 11: Camera setup from top view

As seen in figure 10, the camera is placed underneath the fly arena and captures their movement from below. The fly arena is a $1cm \times 1cm$ square where the flies can move freely. In order to keep the background consistent, a black plastic is placed on top of the arena.

4.1 Finding the position of the fly

The first task of the algorithm is to detect the fly's position. For this purpose, the code of Aliaa Diab has been used and modified to be suited to this project. The main difference is the fact that we assume to only have 2 flies, whereas in Aliaa Diab's code, the number of flies is unknown and therefore has to be found. This process could possibly introduce unnecessary errors in our case where we already know for sure how many flies will be present.
Furthermore, the background in our experiment is always black thanks to the black piece of plastic and doesn't need to be removed. This enables the code to only perform image thresholding with Gaussian blurring to detect the contours of the flies/magnet. Once the contours are detected, the two biggest ones are kept and their center position is also recorded as the (x, y) center of mass position of the fly. If the detection of at least two contours fails, a correction algorithm based from Aliaa Diab's code is applied. Usually, this happens when the two flies (or the fly and magnet) come in close contact and their contours cannot be separated. To fix this, the thresholded shapes are eroded to separate the flies and then re-dilated.

In the presence of both the fly and the magnet, it is crucial to distinguish and track these two elements accurately. The code offers two options for achieving this differentiation.
The first option involves using **user input** to identify the fly's location. This is implemented through the function `determine_fly_robot`. At the start of the experiment, the user is prompted to specify the fly's position by answering whether it is up, down, left, or right of the magnet. With this information, the algorithm then compares the positions of both contours it had found and assigns the one of the fly to the position corresponding to the user input specifications.
The second option involves **comparing the areas** of the contours, implemented via the `determine_fly_robot2` function. This method is the default in the code as it does not require user input and is generally quite robust. Since the magnet is smaller than the fly, comparing the areas usually yields an accurate prediction. In case this method fails, a button has been integrated into the GUI to allow switching the IDs between the fly and the magnet.

Tracking the position of the fly and the magnet in the code will insert visual representations of what has been found on the output frame with OpenCV's drawing functions:

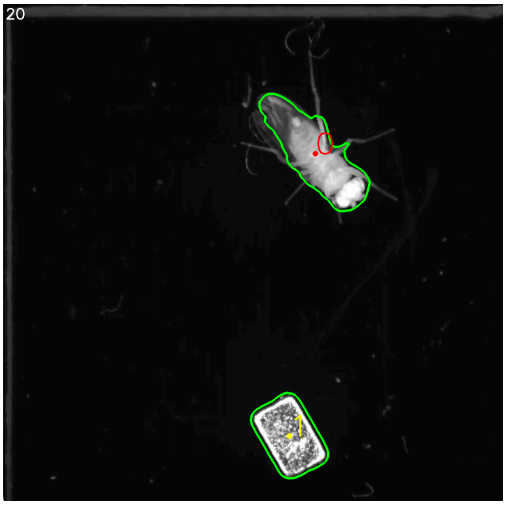


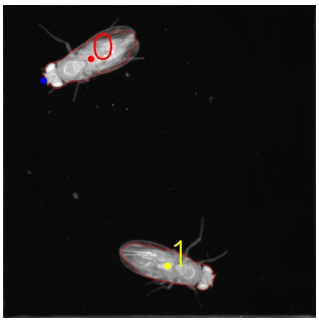
Figure 12: Contour detection of the fly and magnet and tracking

4.2 Finding the angle of the fly and where its head is pointing

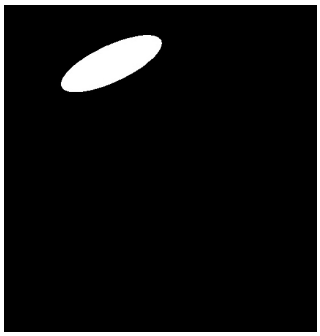
Another important aspect to determine is the fly's orientation. Identifying the direction in which the fly is pointing allows the user to predict its imminent movements and understand its positioning relative to the moving magnet. This information is crucial for analyzing the specific trajectory of the magnet, such as whether it moves from head to tail or tail to head.

The algorithm for determining the fly's orientation is implemented in the `get_fly_orientation` function. This algorithm is based on the observation that the head of the fly is always brighter than its tail (specifically, the end of the wings). Knowing this, by comparing pixel intensities, the algorithm can accurately identify the head of the fly. The steps performed are the following:

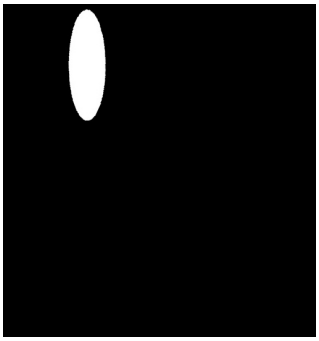
1. Approximate the fly's contour to an ellipse.
2. Create a mask of the ellipse and rotate it to the vertical axis.
3. Cut the mask in two: the quarter top part and quarter bottom part.
4. Rotate the mask back and apply it to the image (for both masks).
5. Compare the pixel intensity level in each side.



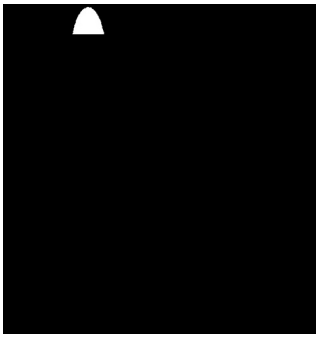
1.



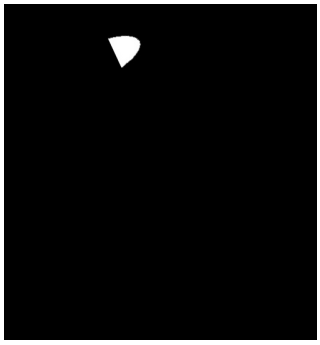
2.a



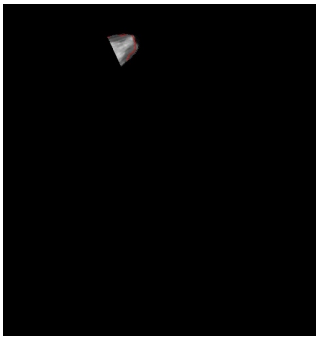
2.b



3.



4.a



4.b

In this manner, the algorithm compares the top and bottom quarters of the Drosophila, which predominantly contain the head and wings, respectively. By analyzing the pixel intensities in these regions, the algorithm can accurately distinguish the brighter head from the darker tail.
Once the head position has been found, a blue dot indicates where it is on the frame for visual feedback:

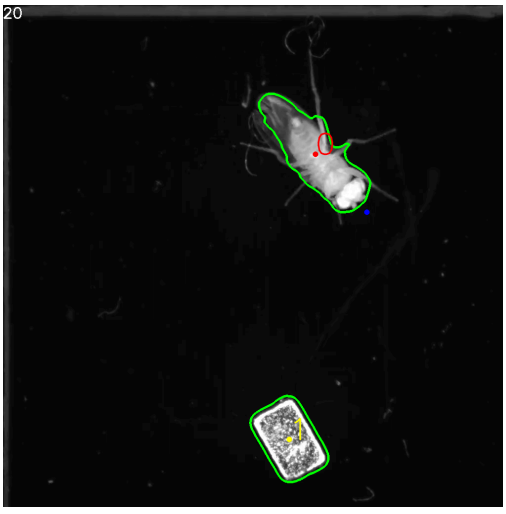


Figure 13: Found head position of the fly indicated by the blue dot

Finding the fly's head position allows to extract its angle. In the code, the angle is defined with respect to the vertical axis as below. It is based on the angle of the approximated ellipse of the contour and then adjusted depending on the head's position.

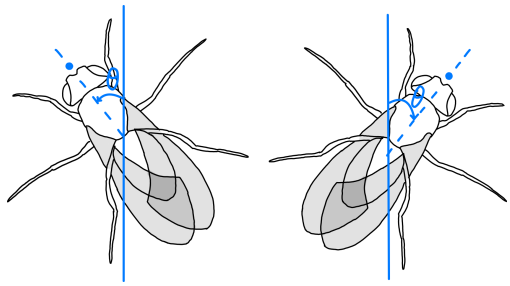


Figure 14: Angle of the head position with respect to the vertical axis

The angle ranges from $[0, 180]$ and from $[0, -180]$ with respect to the vertical axis. These values are the ones that will be used for the internal processing of the flies position. However, the angles provided to the GUI are trigonometric.

4.2.1 Making the code more robust

When using the code to detect the fly's head position, occasional jumps in the detected position could be observed. These occur due to varying light conditions as the fly moves, causing the contour algorithm to detect a smaller ellipse. Consequently, the algorithm may not fully account for the ends of the wings. When this happens, the light intensity may appear brighter at the rear of the fly, leading to prediction errors by the algorithm. Since these occurrences were sporadic and mainly manifested as sudden ”jumps” or glitches where the blue dot indicating the head would shift from the front to the rear, an algorithm has been implemented to mitigate this issue.

The idea of the algorithm is to check on the previous 10 predictions of the head position (i.e whether it was predicted in the top or bottom half of the ellipse contour). Three conditions need to be met in order to switch the current prediction for the head position:

1. More than half of the previous predictions indicate the opposite direction. This would mean that the new prediction does not make sense with the previous ones and that there has been an error.

2. The head position hasn't been corrected for more than 15 times. We assume that if the head position has already been corrected 15 times, it must be because it is indeed correct.

3. The angle of the fly isn't between $[70, 110]$ and $[250, 290]$. In these angles, it is difficult to determine what is the ”top” half and what is the ”bottom” half of the ellipse since the fly is almost completely horizontal. Switching from head to tail is very frequent in these angles and this algorithm rather introduced errors instead of resolving them.

If these three conditions are met, the predicted fly's head position is switched to the opposite one such that there is no glitching or jumping.
The presence of this algorithm can be often very useful but can also sometimes provide wrong corrections as it will be discussed in the results section of this report. The use or not of this code should be adapted to the different conditions of use of the tracking code.

4.3 Distinguishing between dorsal and ventral view

In the current setup of the algorithm, depicted in Figure 11, the fly has the capability to rotate and change its facing direction. This results in the camera capturing both dorsal and ventral views of the drosophila at different times.

Understanding whether the camera is capturing the dorsal or ventral view is crucial for accurately tracking the fly and its legs. Typically, observing the ventral view of the Drosophila is preferred because it provides clearer visibility of its legs. Having a clear view of the legs and the joints will simplify the process of tracking the precise elements of the legs and determining their trajectory. These can be observed in figure **??**.

Unlike the detection of the Drosophila's head, there is no distinct and straightforward feature extracted through image processing that can unambiguously distinguish between the fly's stomach and back. An initial algorithm was tested to differentiate these areas based on pixel intensities. However, this approach necessitated a manually set fixed threshold value to separate the two classes and is not robust to brightness changes.
As a result, the next step involves developing and training a machine learning algorithm specifically designed to recognize and classify both stomach and back regions of the fly. This approach aims to enhance robustness and accuracy, especially in varying lighting conditions, thereby improving the reliability of the fly tracking system.

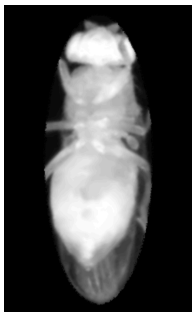
Several classification and feature extraction algorithms were extensively tested to achieve the highest possible accuracy for distinguishing between the stomach and back regions of the fly. These included random forests classifier, texture analysis, principal component analysis (PCA), support vector machines (SVM), and convolutional neural networks (CNN). From these experiments, only two classifiers demonstrated superior performance and were selected for use in the detection system: a support vector machine classifier with PCA and a transfer learning CNN model. These classifiers were chosen based on their ability to deliver the most accurate and reliable results in classifying the frames.

4.3.1 Gathering training and testing data

To create a supervised classifier for distinguishing between the stomach and back regions of the fly, labeled training data is essential. For this project, samples were collected from 17 different

videos, each approximately 1 hour in duration. The data gathering algorithm resembles the first part of the head detection algorithm. It approximates the fly's shape to an ellipse using its contour, rotates the fly vertically with the head facing upward, and crops the image to isolate the fly. Below is an illustration of a frame along with its extracted data:

Frame



Extracted data 1

Extracted data 2

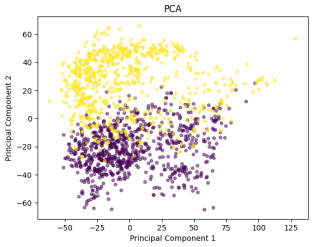
This operation is repeated on a number of frames such that $1'575$ data images where obtained for training and 457 data images for testing. They were then labeled by hand by placing them into separate folders, one for the dorsal view images and one for the ventral view images.

4.3.2 Support vector machine classifier

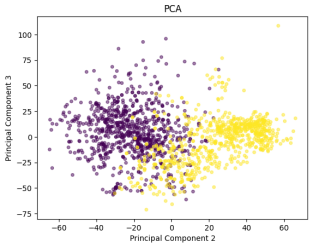
The first algorithm developed is a support vector machine classifier (SVM). This supervised classifier will try to find the best separating hyperplane between two classes in N-dimensional space. Additionally, with the help of kernel functions, the classifier is able to separate non-linear problems such as ours.

To enhance computational efficiency and extract meaningful features, a **Principal Component Analysis** (PCA) algorithm was applied to the images before classification. PCA identifies the directions in which the data exhibits the highest variance and defines vectors based on these directions. By retaining only the first eigenvectors, which capture the most significant information about the images, PCA reduces both the size of the dataset and computational costs.

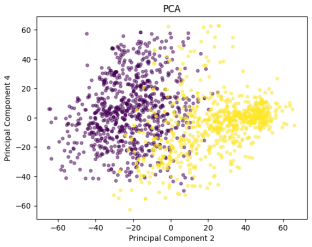
When applying a PCA to the training images with a limit of 5 components, the size of one image reduces from $(360, 220)$ which is flattened to a single array of size 79200 to 5 components. Therefore, the image is represented only by an array of 5 elements encapsulating the major variance information and reducing drastically the size of the dataset.
Visualizing some of these eigenvectors plotted against each other provides insights into whether the data exhibits separable patterns:



PCA projection on eigenvectors 1 and 2



PCA projection on eigenvectors 2 and 3



PCA projection on eigenvectors 2 and 4

From these projections, it is evident that a significant portion of the data does not overlap, suggesting separability between the classes. However, at the intersection, there is always some confusion between both classes indicating that they are not fully separable. Adding more dimensions will allow the classifier to find differences between each dimension and correctly classify the images.

Due to variations in brightness that can lead to erroneous predictions, an additional pre-processing step is implemented to mitigate these disturbances. This involves applying **histogram equalization** and row-wise **normalization** to the images.
The histogram equalization helps improve the contrast of the images and revealing the features in the flies. This is done by spreading out the most common frequency intensity value such that it is also present in other regions of the image.
Normalization along the rows of the flattened image ensures consistency in brightness levels across all images, providing a standardized reference for the algorithm. This procedure ensures that variations in brightness are interpreted consistently across the dataset. These procedures transform an image in the following manner:

Image before histogram equalization and normalisation

Image after histogram equalization and normalisation

The SVM classifier is then trained on this data with a RBF kernel allowing for non-linear classification. The results will be presented in the dedicated section.

4.3.3 Transfer learning CNN model

Another approach to training a model for image classification involves using deep neural networks and transfer learning. In this method, a pre-trained deep neural network model, which has already been trained on a large dataset of images, is adapted and fine-tuned to classify a specific set of images. This approach accelerates the training process significantly because the model has already learned foundational features from a diverse range of images.
Transfer learning is widely adopted in image classification tasks due to its robustness and high performance. By leveraging the knowledge encoded in pre-trained models, this method effectively enhances the accuracy and efficiency of classifying new images with specific characteristics.

The pre-trained model used for our classification task is the ResNet50. It is a 50 layer deep residual network which uses residual connections to learn complex features of the images. The first layers of the model are used for feature extraction and therefore the model doesn't need the use of the PCA model. The last layers are used for the actual classification of the images.

Given the variability in lighting conditions during this project, intra-class images can differ, potentially leading to misclassifications. To mitigate this issue and enhance the classifier's resilience to brightness variations, data augmentation techniques were employed. Data augmentation exposes the training process to a broader range of brightness variations, enabling the model to learn features that are invariant to brightness changes. Specifically, augmentation techniques included adjustments to brightness and contrast, ranging from 0.7 to 1.2, along with random flips of the images.

The model is then trained for a number of only 5 epochs or until the validation loss stops decreasing in order to avoid overfitting to the training data.
With this training, the classifier comes up with very accurate results which will be presented in the following section. It is therefore the recommended model to use for this task. The user is able to decide which classifier to use by setting boolean parameters to True or False in the frame processing function. Additionally, there is an option to utilize both classifiers simultaneously and retain the prediction from the classifier with the highest confidence score. This approach ensures flexibility and robustness in determining the best classification outcome for each scenario.

Once predicted, a visual output will be drawn on the frame indicating whether the algorithm is predicting a ventral or dorsal view of the drosophila. This is done through the presence of a green or yellow circle on the fly's thorax. A green circle will indicate that the ventral view is being predicted and a yellow circle indicates the dorsal view.

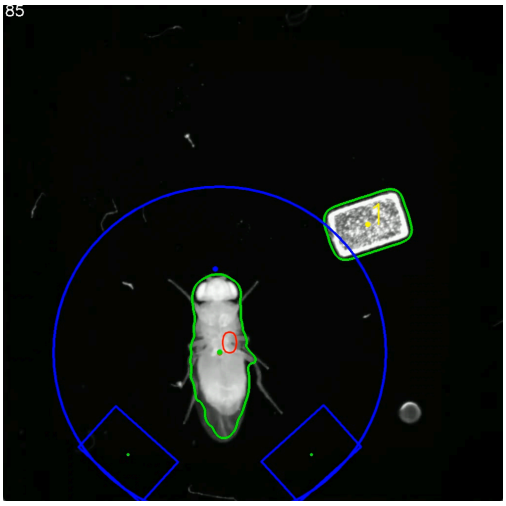


Figure 15: Ventral view being predicted shown through the presence of the green circle

4.4 Tracking the position of the fly's middle legs

Tracking the precise position of the fly's legs can provide valuable detail and information regarding the observed "social reaching behavior." In the subsequent stages, extracting and analyzing the trajectory of these movements could bring great information about this sensorimotor transformation.

Since the legs of the flies are very small, and as discussed in section 3.2 they must be tracked with a deep learning algorithm. For the purpose of this project, SLEAP was used.
Training this model involves a straightforward process: uploading a video into the software, manually labeling multiple frames and then performing training. Selecting a representative video is crucial for ensuring the training's effectiveness and the algorithm's predictive reproducibility. Therefore, a video featuring a single fly in ventral view, accompanied by a randomly moving magnet was chosen.

The classifier was specifically trained using video footage captured from the ventral view of the drosophila. This perspective is optimal because it represents the ideal positioning of the fly. In this view, the legs are more clearly visible and can be analyzed with greater precision.

The labeling of 844 frames for this model was done as shown below.

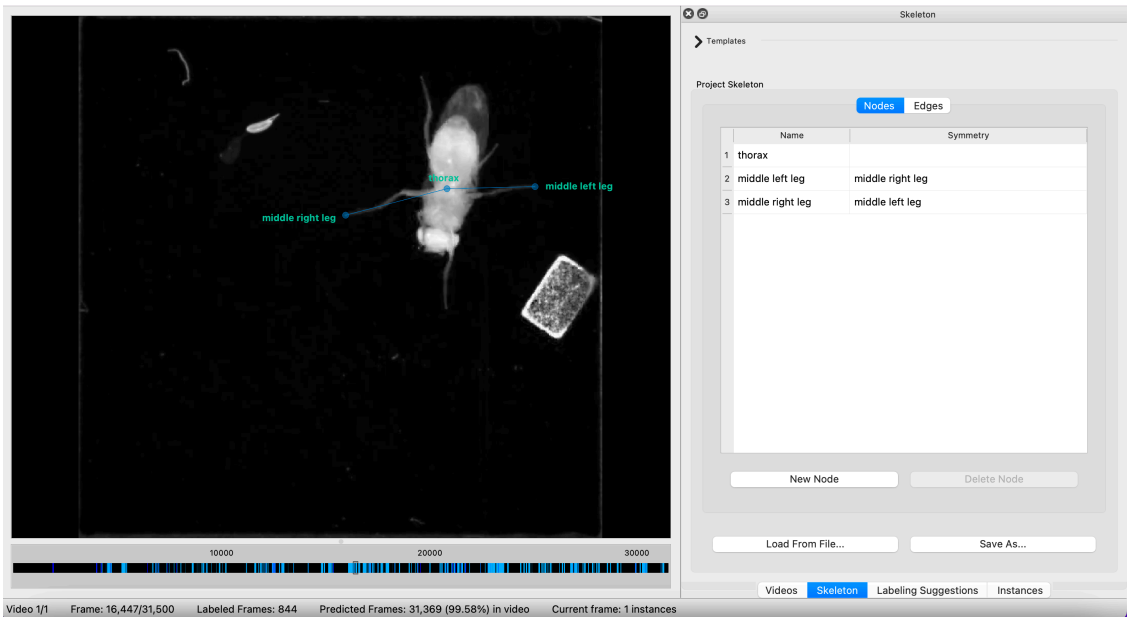


Figure 16: Labeling of a sample frame on SLEAP

Three nodes are defined : thorax, middle right leg and middle left leg with a symmetry between the middle right and middle left legs allowing the algorithm to better detect them. Labeling frames is a time-consuming task, but it can be expedited through inference. Initially, the model is trained on a small subset of frames and makes predictions on additional frames. The user then corrects these predictions and retrains the model. This iterative process continues until a sufficient number of frames are accurately labeled, enabling the model to learn and generalize effectively.

For training, the following parameters were used:

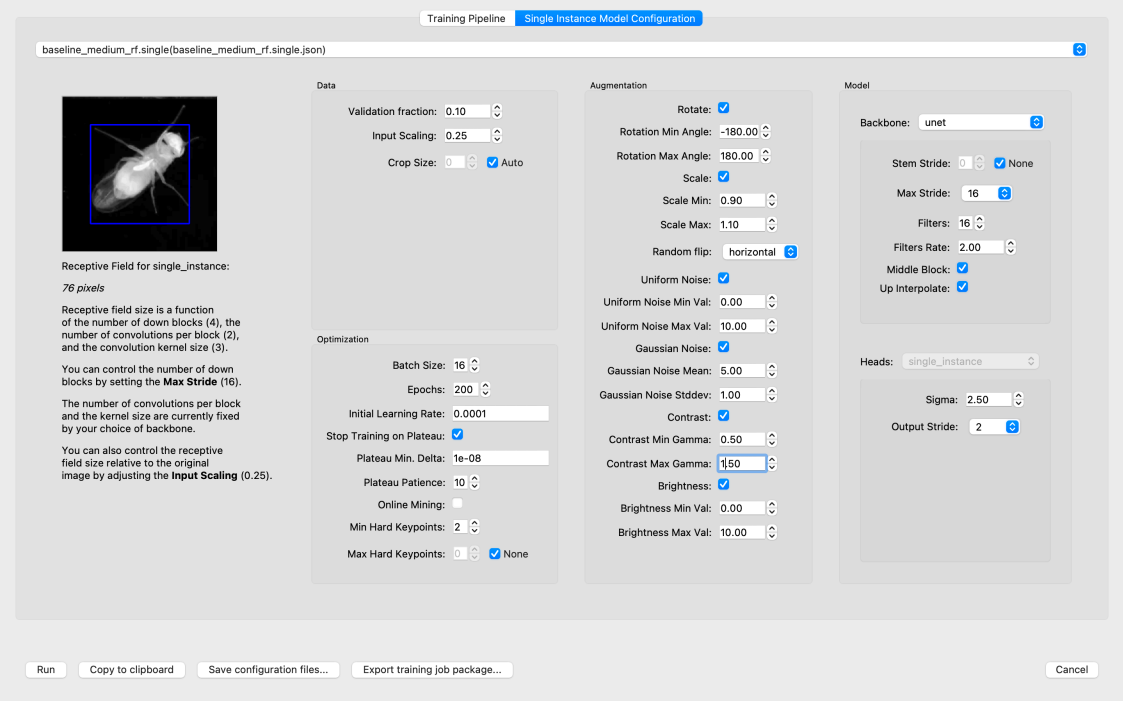


Figure 17: Parameters used for training SLEAP

With these parameters, a training package can be extracted and run on a remote computer containing a GPU. The model is then left to run for the 200 epochs or until the loss stabilizes and the training stops alone.

For prediction purposes, a computer with a GPU is also required. Consequently, deploying this model for real-time tracking isn't straightforward due to the absence of a GPU in the current setup. However, experimental videos can be recorded and analyzed later using the trained model. This approach remains highly valuable for the project, allowing experiments to be conducted, recorded, and subsequently analyzed to track leg movements and study drosophila behavior in detail.

4.5 Finding the maximum feasible positions for the robotized fly

One of the critical roles of the computer vision algorithm is to provide essential information to the magnet controller. This information includes the precise position of the robot relative to the fly, as well as the maximum start and end angles that the robot can traverse at a specified radius from the fly.

The concept of *"maximum angle"* refers to the trigonometric angle measured from the center of the fly. The magnet's movement around the fly is assumed to be circular, with a predefined radius centered on the fly. Due to the confined space of the fly's arena, the circular trajectory of the magnet may intersect the arena walls depending on the position of the drosophila. In such cases, the magnet cannot continue its circular path uninterrupted. Hence, it becomes necessary to define these maximum angles, which indicate where the circular trajectory intersects with the arena walls.

If the trajectory does not intersect the walls, the magnet can complete a full circle around the fly. These maximum angles play a crucial role in ensuring that the magnet controller operates within the constraints of the experimental setup, thereby enabling effective and controlled interaction with the drosophila.

The figure below captures this information:

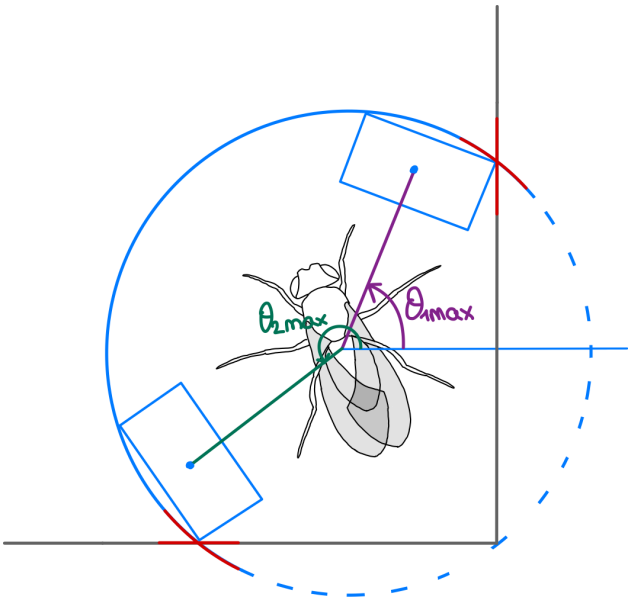


Figure 18: Maximum feasible angles for the magnet's trajectory

For this algorithm to take place, the radius of the robotized magnet's trajectory needs to be defined. It is computed as shown in figure 19:

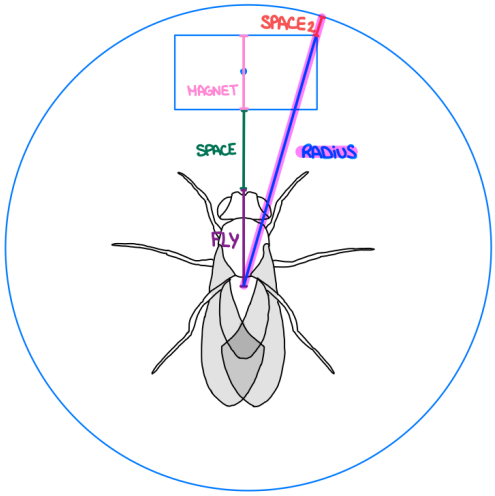


Figure 19: Maximum feasible angles for the magnet's trajectory

This radius is determined by the sizes of both the drosophila and the magnet, which are recalcu-

lated each time the program starts. Additionally, two user-defined variables, referred to as ”space” and ”space2” in Figure 19, are crucial. These two variables define the desired space between the fly and the magnet and between the magnet and the walls of the arena. The unit of measurement for these distances is specified in terms of pixels.
The calculations behind these angles are based on trigonometry, computing the radius between the drosophila and the trajectory's circle is done as explicited below. The axes used are the ones of openCV, the reference is at the top left corner with the x axis towards the right and the y axis pointing downwards.

$$\text{Vertical space fly center to magnet center: } v = \frac{fly_h}{2} + S_1 + robot_w$$

$$\text{Total radius: } R = \sqrt{v^2 + \left(\frac{robot_h}{2}\right)^2} + S_2$$

Then the (x, y) corresponding coordinates at this radius are computed for every angle $\theta \in [0, 360]$:

$$x = X_{fly} + |\sin\theta| + R$$

$$y = Y_{fly} + |\cos\theta| + R$$

Finally, these (x, y) coordinates are compared to the coordinates of the walls of the arena in order to determine if they are feasible or not. An important aspect taken into account when doing this is the fact that since this is based on the center of the magnet, the corners are not considered but should be. Therefore, for every angle tested, an extra value is added to compensate for it.

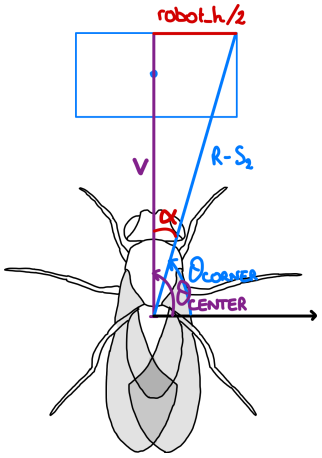


Figure 20: Maximum feasible angles for the magnet's trajectory

This extra angle is computed as

$$\alpha = \arctan\left(\frac{\frac{robot_h}{2}}{v}\right)$$

and it is added to every angle θ tested for the (x, y) coordinates.

With the inclusion of this additional angle adjustment, the algorithm can calculate the corresponding (x, y) coordinates and assess whether they fall within the confines of the arena. If the coordinates are within the arena boundaries, the angle is recorded and stored in an array. This process continues iteratively, accumulating angles that satisfy the arena's spatial constraints.

After collecting all possible angles within the arena, the algorithm evaluates the array to identify the longest continuous series of valid angles. This longest series represents the path where the magnet can travel most extensively around the fly without encountering the arena walls, ensuring optimal movement and interaction during the experiment. The two furthermost angles are then extract and provided to the magnet controller.

4.5.1 Adapting to live tracking and integration to GUI

Once the video tracking algorithm was successfully implemented for offline videos, the next crucial step was to adapt it for real-time processing to enable immediate feedback and control of the magnet based on live computer vision data.
The primary modification occurred within the `process_video` function. Initially designed to read and process all frames from a video stored in memory, it was adjusted to continuously read frames in real-time from a live video stream. Instead of buffering frames, the algorithm now processes each frame sequentially as it is captured. To facilitate the real-time frame acquisition, the implementation leverages a function called `flir-control`, developed by Thomas Ka Chung Lam. This function serves as an interface, bridging the gap between the C++ code necessary for interacting with the camera hardware and the Python code responsible for executing the video tracking algorithm. It efficiently manages the input stream from the camera, ensuring a seamless flow of frames for immediate processing and analysis.
With this algorithm, running the real-time tracking is done by calling the file `run.sh`. This file will manage the camera input to our main frame processing function file which should be explicited in the run.sh file.

Once the reading of the frames is done sequentially through the previous explanations, the rest of the frame processing is identical to the video tracking. Instead of reading frames from an array, the frames are provided in real time by the camera, one after the other.

This code is then integrated to the GUI and to the magnet controller such that the camera can provide feedback. This part was done mostly by another student, Omar Shalby and will be detailed in his report. The user interface looks like this:

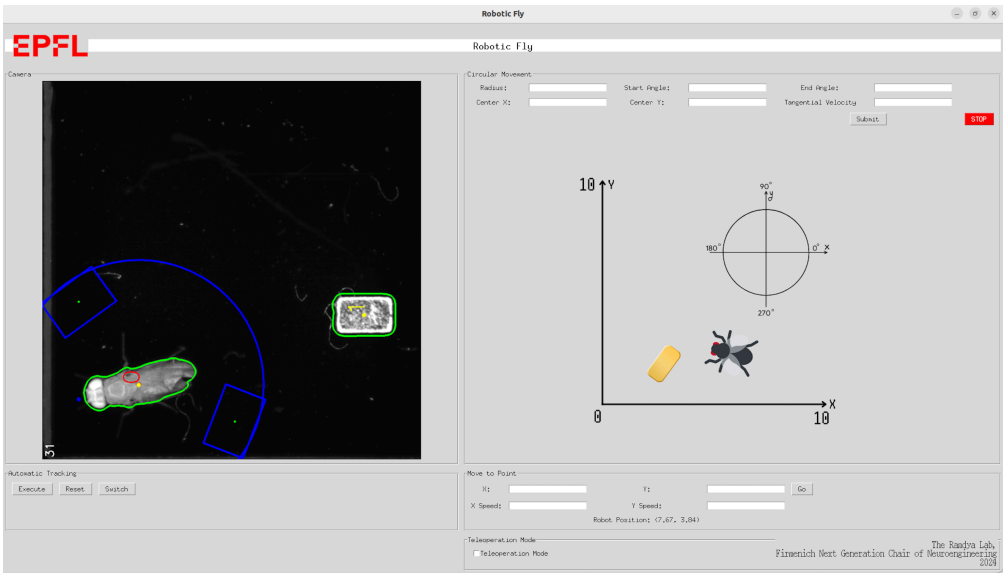


Figure 21: User interface for drosophila ”social reaching” experiments

The frame viewed in the interface provides the following information :

- Detection of the drosophila and magnet through contour drawing.
- Position of the drosophila's head position through the blue dot in front of its head.
- Whether the ventral or dorsal view is being shown through a colored dot on the center of the drosophila. If the dot is green it means that the ventral view is being seen; if it is yellow, it means that the dorsal view is being seen.
- Maximal feasible positions for the magnet around the drosophila for its circular trajectory.

Furthermore, since the radius, maximal angles and positions of the fly and magnet are provided, by clicking on the `Execute` button, the magnet will perform a circular trajectory around the fly going from one extreme angle to the other as shown in the drawing. If the user wishes to set these values manually instead, it can be done through the right side of the GUI.
Finally, if the drosophila and magnet switch identities or get assigned the wrong IDs (i.e. meaning that the blue circle with the maximum feasible poses are around the magnet instead of the fly), the `switch` button allows the user to switch IDs and the `Reset` button allows the user to re-run the function that finds the magnet and its size.

5 Results

Evaluating the performance of the tracking algorithm is crucial to having a reliable experiment from which we can deduce conclusions. This part of the report records key metrics such as accuracy, processing speed, and efficiency of the algorithm in both real-time and video tracking scenarios.

5.1 Accuracy of the tracking algorithm

5.1.1 Accuracy of the dorsal vs ventral view classifiers

The two trained classifiers discussed in Section 4.3 were optimized to achieve high performance, leveraging a large number of representative frames and specific techniques such as image normalization for SVM and data augmentation for CNN. Their performance is both quite strong but the CNN transfer learning classifier is recommended as it has a higher accuracy.

On a testing set of 255 frames, the accuracy of the PCA + SVM classifier is 98% and the accuracy of the CNN classifier is 100%. To better understand the results, one can inspect the confusion matrix of the SVM classifier.

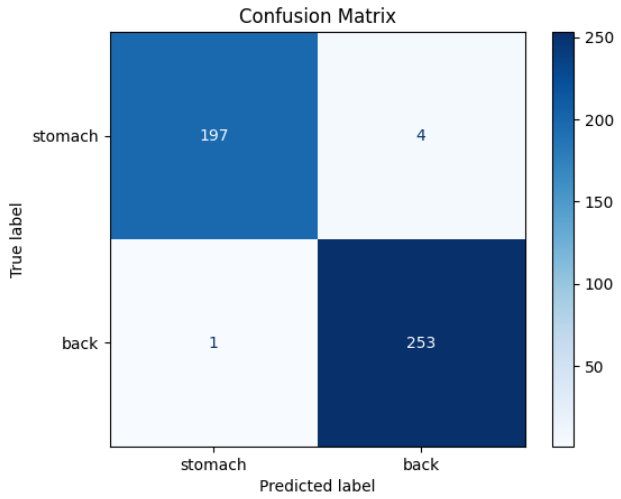


Figure 22: Confusion matrix of the SVM classifier

It is observed that there are more errors in predicting the ”stomach” (ventral) class. This pattern persists even when tracking real-time camera input, which is expected. For more explicit data, two 1-minute video recordings, totaling 2041 predicted images, were tracked using both the PCA+SVM classifier and the CNN classifier for deeper comparison. These videos capture ventral views of the flies, which have proven challenging for accurate classification.

	Total		correct		incorrect	
	number	%	number	%	number	%
SVM	2041	100	1987	97.35	54	2.65
CNN	2041	100	2024	99.17	17	0.83

Figure 23: Accuracies of the SVM and CNN classifiers compared on the same 2041 frames

These new predictions validate and support the results obtained with the test set of the classification algorithms. The accuracy of the CNN algorithm is close to 100% and higher than the one of the SVM classifier which stays at 97%. Nevertheless, both of these models provide robust predictions with few exceptions where the ventral view is seen as a dorsal view.

An important consideration is that these models were trained on female drosophila, which are larger than males. Consequently, the SVM classifier tends to almost always misclassify the ventral view of male drosophila, while it generally performs better with dorsal views. In contrast, the CNN classifier shows more resilience to size differences and can correctly classify both dorsal and ventral views of male drosophila more consistently.

5.2 Real time performance of the tracking algorithm

Analysing the real-time performance of the tracking algorithm as well as the number of frames per second that it can process can bring much information about its efficiency. The magnet's controller should be able to react in a fast enough manner such that the movement of the magnet around the fly is consistent with the drosophila's current state and location. For this reason, the frame processing has been analysed in time and broken down into different parts to understand which tasks take the longest to run.

Three types of experiments were inspected,

1. Video tracking (implying without the GUI)
2. Real-time tracking without the GUI
3. Real-time tracking with the GUI

These three types of experiments allow to understand how adding a component (i.e camera input or GUI parallel running) to the code every time impacts the efficiency of the tracking.

All experiments were run on five repetitions of 20 seconds each. The main overview of the algorithm is the following :

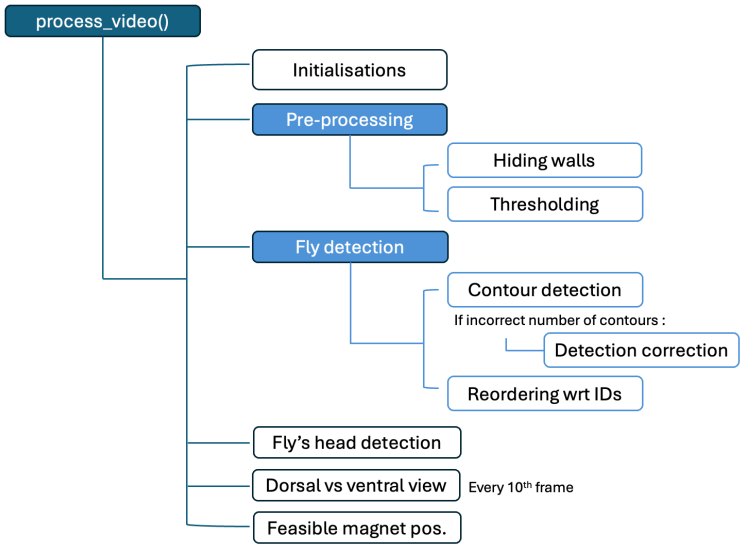


Figure 24: Outline of the frame processing algorithm

Therefore, time analysis was performed on these main sub-tasks and are reported as follows:

	Video tracking						**Real-time** no GUI						**Real-time** with GUI	
	SVM		**CNN**		**Both**		**SVM**		**CNN**		**Both**		**CNN**	
	avg	(std)	avg	(std)	avg	(std)	avg	(std)	avg	(std)	avg	(std)	avg	(std)
Average FPS	37.066	(5.823)	30.482	(10.330)	31.775	(9.974)	15.537	(2.537)	15.898	(4.243)	15.852	(4.157)	13.101	(3.415)
Whole frame processing time	20.861	(82.249)	39.641	(243.835)	39.971	(347.894)	50.425	(10.431)	52.852	(36.544)	52.555	(41.365)	19.617	(14.825)
Preprocessing time	0.976	(0.102)	1.095	(0.110)	1.051	(0.171)	3.706	(1.329)	2.494	(0.965)	2.468	(1.022)	2.365	(0.714)
Frame correction time	3.176	(0.829)	3.349	(1.336)	3.283	(1.641)	7.341	(2.081)	8.758	(2.817)	4.404	(1.127)	8.346	(3.046)
Fly detection time	0.633	(0.355)	0.690	(0.382)	0.653	(0.284)	1.943	(1.215)	1.166	(0.818)	1.559	(1.235)	1.368	(1.616)
IDs reordering time	0.156	(0.131)	0.169	(0.104)	0.163	(0.105)	0.236	(0.153)	0.228	(0.200)	0.236	(0.145)	0.236	(0.203)
Head position time	11.373	(1.978)	12.797	(2.086)	12.014	(1.915)	33.594	(7.549)	28.026	(6.667)	27.740	(5.912)	26.094	(5.485)
Ventral/dorsal time	3.192	(1.328)	156.842	(44.780)	152.021	(40.144)	7.751	(3.682)	116.891	(35.324)	138.667	(42.649)	118.912	(58.981)
Magnet feasible pos. Time	5.253	(0.650)	6.089	(0.754)	5.682	(0.681)	8.142	(2.363)	7.418	(1.895)	7.870	(2.150)	7.817	(2.237)
Drawing the magnet's pos.	0.091	(0.022)	0.095	(0.016)	0.092	(0.021)	0.131	(0.067)	0.128	(0.103)	0.127	(0.052)	0.203	(0.357)

Figure 25: Outline of the frame processing algorithm in ms

On average, the number of frames per second that can be processed by the algorithm is around 37FPS when performing tracking on a pre-recorded video and using the support vector classifier for the ventral vs dorsal view classification. When using the CNN classifier or both, this rate drops to 30FPS.

From the data in figure 26, three factors seem to be limiting the number of FPS and the rapidity of the tracking algorithm. These are: the choice of classifier for the ventral/dorsal classification, the real-time vs video tracking and the head position finding algorithm.

Indeed, the **ventral/dorsal classifiers** are crucial components of the algorithm, impacting both the accuracy and processing speed. In our setup, these classifiers are executed once every 10th

frame to balance computational load and real-time performance.
The PCA+SVM classifier typically executes in approximately 5 ms per frame, while the CNN classifier requires significantly more time, around 140 ms per frame. This stark difference in execution time —approximately 28 times longer for the CNN classifier— highlights the trade-off between accuracy and speed in classifier selection.
When examining the video tracking metrics, it becomes evident that switching from the SVM classifier to the CNN classifier nearly doubles the processing time per frame, increasing from 20 ms to 39 ms. This can be explained by the complexity of the deep model needing more time to predict as well as the use of the PCA algorithm for the SVM classifier. Indeed, using PCA will reduce the dimensionality of the prediction problem drastically and increase the classification speed of the SVM model.

However, in the real-time tracking algorithm metrics, changing from SVM to CNN does not seem to affect much the single frame processing time which stays close to 50ms or the FPS even though the classification time does vary. Two aspects can explain this inconsistency.
First of all, this classifier is only called upon once every 10th frame, therefore, the frame processing time suffers only sparsely and the long processing time isn't reflected in the overall speed.
Another reason could be the possibility that the videos used in real-time tracking include many frames that needing to be corrected or where the fly and the magnet overlap which constrained the algorithm from detecting the fly alone and calling the classifier. In this case the processing of a single frame is much faster as it doesn't need to do all of the classification work. Overall, these fast frames will be compensated by the longer frames and provide an average processing speed similar to the one of the SVM classifier.
Nevertheless, one can conclude that the PCA+SVM classifier is less computationally expensive and more time-efficient than the CNN classifier. Knowing this and the conclusions above stating that the CNN classifier is the most accurate, a time-accuracy trade-off problem arises. One has to decide if the most important aspect of the algorithm is its processing speed or its accuracy depending on the end application of the experiment.

Another computationally expensive sub-task of the algorithm is the **head position finding algorithm**. This algorithm is called upon at every frame and takes up to 60% of the frame's processing time. The reason for this is the fact that the algorithm in itself needs to create a big number of masks and applies many transformations to them as explained in section 4.2. A solution to this long processing time will be proposed in the discussion 6.

Finally, the use of the tracking algorithm in **real-time compared to video** does indeed seem to impact the frame processing time, however, using the GUI user interface or not does not change much. The frame rate decreases by half when using real time tracking going from 37FPS to 15FPS and the single frame processing time also increases.

The decrease in FPS comes from two places, the longer single frame processing time and the longer time to capture the next frame for processing (i.e the time between the end of processing frame N and the start of processing frame N+1). Indeed, using a real-time tracking algorithm means that the camera needs to capture the input frame and provide it to the algorithm in between each frame processing. Instead, the video tracking already contains all of the frames to process in an array which is much faster to access. Therefore, using a streaming camera introduces latency in capturing frames and buffering delays.
Furthermore, the single frame processing time also increases in itself and isn't impacted by the latency of the camera as it only takes into account the actual frame processing and assumes the frame is already captured. Therefore this increase in time possibly comes from CPU usage which is increased with the camera input and is less available for the actual frame processing. On top of this, it is important to take into account the fact that the real-time processing metrics have been performed on a different computer (at the ramdya lab) than the video processing metrics and could also impact the results depending on the processing speeds of the computers.

5.2.1 Leg tracking algorithm performance

The developed leg tracking algorithm represents an initial exploration into utilizing deep learning for this specific task. As detailed in Section 4.4, the model was trained using annotations from a single video encompassing approximately 850 frames. From this training, several metrics were derived, some of which are presented below. It's important to note that the training video includes frames where the fly is inadvertently crossed by the magnet, resulting in more challenging frames for prediction, such as the example shown below. This occurrence is not intended during experimental conditions, and therefore, the reported metrics may be affected, potentially underestimating the accuracy achievable under normal circumstances.

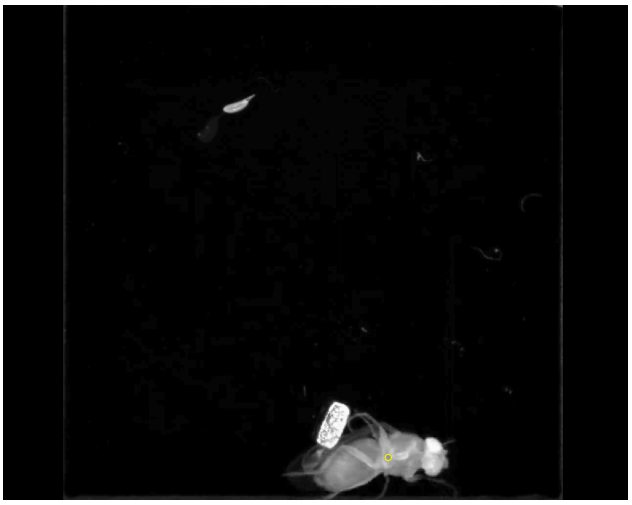
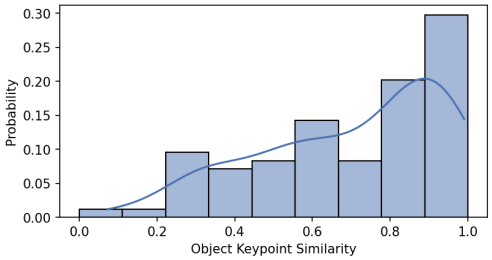


Figure 26: Outline of the frame processing algorithm in ms

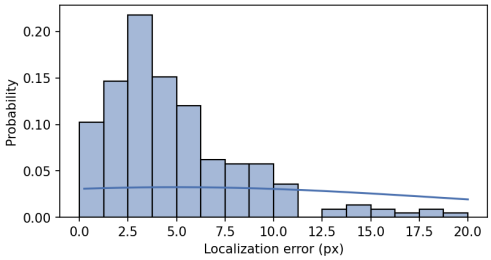
Upon reviewing the predicted videos, the model demonstrates robust performance by accu-

rately detecting leg positions in the majority of instances. Despite being trained on ventral views of flies, it effectively identifies leg positions even when the dorsal view is presented, making it versatile for both scenarios.

The model achieves a mean average precision of 44.64%, with further metrics indicating object keypoint similarity and object localization error, detailed below.



Object keypoint similarity



Localisation error

These two figure indicate that the localisation error is generally of around $2-7$ pixels meaning that the model is not extremely accurate but rather finds the general position of the leg. Furthermore, the probability of having a 100% similarity in the prediction and ground truth is rather low at 30%.

This model was trained on a female drosophila but has also been tested on males and is able to correctly predict the legs pose estimation. Even though the result metrics seem to indicate poor performance, the visual predicted video seems quite accurate. Some improvements are to be made on this side.

Similarly to training, for the prediction and inference of the model, a GPU is needed to guarantee fast predictions. As the computer used in the lab for the tracking algorithm does not contain a GPU, the model has not been adapted for live pose estimation. However, this task should be feasible with the trained SLEAP model.

6 Discussion

This project addressed several key challenges in computer vision to establish a closed-loop control system for observing drosophila behavior. Being able to have repeatable and robust experiments in research is a primordial task and introducing a robotised ”fly” allows this problem to be solved.

The primary focus of this report was on computer vision tracking of drosophila such that the user and the controller can get the information required for further processing. Results indicate that the developed tracking algorithm, integrated with the user interface and robot controller, forms a solid foundation for conducting experiments. The results have proven that a frame rate of 13FPS is to be expected when using the GUI in real time tracking and can vary depending on the computer's CPU. A typical movie is played at a frame rate of 24FPS, therefore the 13FPS is rather slower but not drastically as it cannot really be perceived when viewing the video and is sufficient for this application.

The algorithm successfully detects the drosophila's head and orientation with high accuracy as well as the detection of the ventral or dorsal view of the camera input. It offers two classifier options with an accuracy-time trade-off: the SVM classifier provides slightly faster tracking times at the expense of marginally lower performance compared to the CNN classifier, which could benefit from further optimization with less computationally intensive pre-trained networks. When performed in real time, the frame rate doesn't change much due to CPU sharing and usage.
For further improvements, the CNN classifier could be tuned and tested with other pre-trained networks that might be less computationally expensive in order to speed up the process whilst keeping the great accuracy.

Another algorithm used for detecting the drosophila's head position seems to be the second most computationally intensive task. It would be interesting to adapt it such that this time is minimised. One idea would be to compute the head position only once when the drosophila is mostly vertical and keep this head position the same until the drosophila is horizontal (i.e implying that it might change direction). When it is around the horizontal position, the head detection should however be computed at every frame (or every 10th frame) until it becomes vertical again.

Detecting the fly's legs and their movements is a very insightful task for better understanding the movement. In this project, this has been done for offline video tracking and it is shown to have a great accuracy with the ventral view of the drosophila which is the one we are mostly interested in. Further developing on this and extending it to real-time tracking with trajectory detection could reveal several aspects of the drosophila's behaviour. Looking in this direction, one could also implement a feature in the algorithm allowing to detect when the fly has performed a social reaching behaviour and tracking how many times it has occured during the video.

In conclusion to this project, the main challenges of detecting the drosophila and its many features as well as combining it with the magnet controller was mostly a success. The results show that social reaching behaviour can be triggered with this closed loop control and that the accuracy of the tracking is relatively correct. The remaining challenges are to increase the frame rate at which the videos are being processed either through the tracking algorithm or the computer processing as it is suggested through the results.

7 References

[1] Adobe. *Frame Rate*. https://www.adobe.com/creativecloud/video/discover/frame-rate.html. Accessed: 2024-06-14.

[2] AskPython Contributors. *Principal Component Analysis for Image Data*. https://www.askpython.com/python/examples/principal-component-analysis-for-image-data. Accessed: 2024-06-14.

[3] Jason Brownlee. *How to Save and Load Machine Learning Models in Python with scikit-learn*. Accessed: 2024-06-14. 2019. url: https://machinelearningmastery.com/save-load-machine-learning-models-python-scikit-learn/.

[4] PyTorch Contributors. *Transfer Learning Tutorial*. https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html. Accessed: 2024-06-14.

[5] IBM. *Support Vector Machine*. https://www.ibm.com/topics/support-vector-machine. Accessed: 2024-06-14.

[6] Trapti Kalra. *Texture Analysis with Deep Learning for Improved Computer Vision*. https://medium.com/@trapti.kalra/texture-analysis-with-deep-learning-for-improved-computer-vision-aa627c8bb133. Accessed: 2024-06-14. 2022.

[7] Nitish Kundu. *Exploring ResNet50: An in-depth look at the model architecture and code implementation*. https://medium.com/@nitishkundu1993/exploring-resnet50-an-in-depth-look-at-the-model-architecture-and-code-implementation-d8d8fa67e46f. Accessed: 2024-06-14.

[8] Author Names. "Title of the Article". In: *Sensors* 15.8 (2015), pp. 19369–19385. doi: 10.3390/s150819369. url: https://www.mdpi.com/1424-8220/15/8/19369.

[9] Author Names. "Title of the Paper". In: *arXiv e-prints* (2018). Accessed: 2024-06-14. arXiv: 1804.03142 [cs.CV]. url: https://arxiv.org/abs/1804.03142.

[10] OpenCV Contributors. *OpenCV Documentation*. https://docs.opencv.org/3.4/index.html. Accessed: 2024-06-14.

[11] scikit-learn developers. *Principal component analysis (PCA)*. https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html. Accessed: 2024-06-14.

[12] scikit-learn developers. *sklearn.svm.SVC documentation*. https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html. Accessed: 2024-06-14.

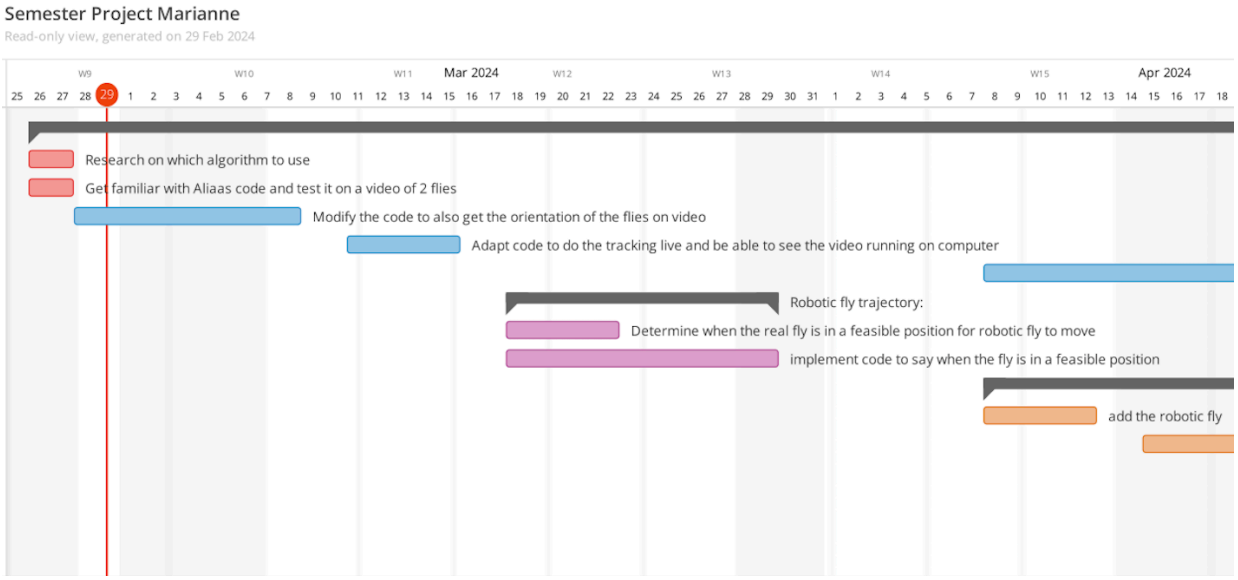
[13] SLEAP. *SLEAP: Social LEAP Estimates Animal Poses*. https://sleap.ai/. Accessed: 2024-06-14.

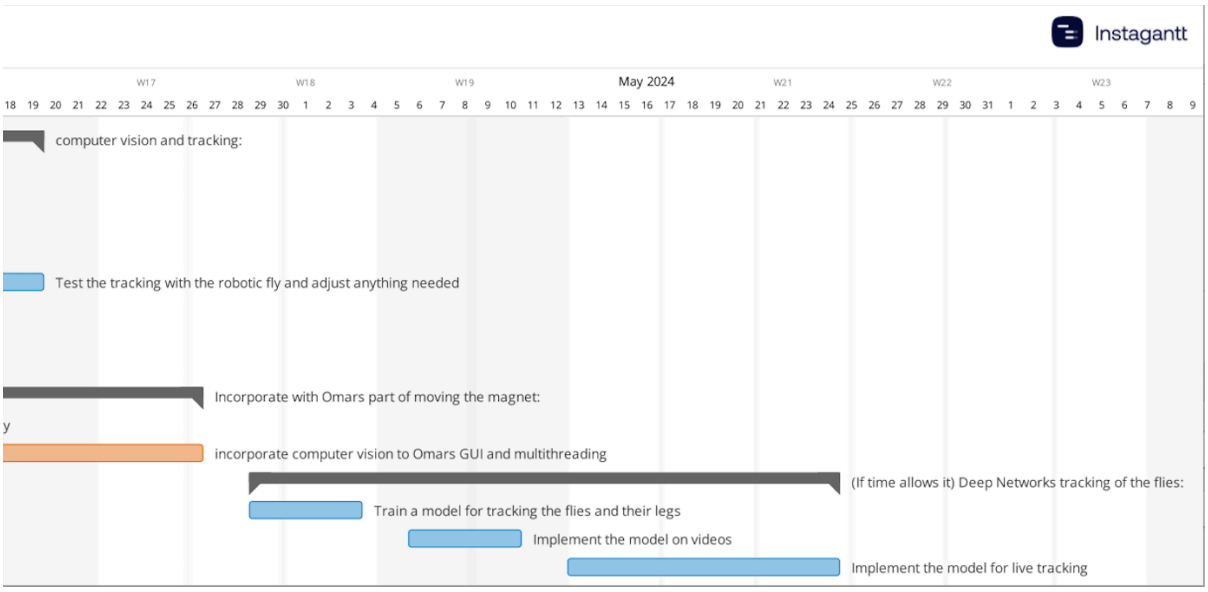
[14] Youri Stasiuk. *Pose Estimation Metrics*. https://stasiuk.medium.com/pose-estimation-metrics-844c07ba0a78. Accessed: 2024-06-14.

[15] Author unknown. *Histogram Equalization*. https://towardsdatascience.com/histogram-equalization-5d1013626e64. Accessed: 2024-06-14.

This project utilized ChatGPT for assistance in developing and refining the code, ensuring efficient implementation and optimization of algorithms. ChatGPT served as a *supportive and assistance* tool in the development process as well as a support with grammar checks and structure enhancement for the report.

8 Original Gannt chart





9 Appendix

9.1 Code to determine if a trajectory is feasible

A code is present in the algorithm that allows the user to enter a start and finish angle and to receive an output whether it is possible or not for it the magnet to actually perform this trajectory. It has not been included to the GUI but could work and be of use.

9.2 Getting data for training the classifiers

Two files are provided to get the data for generating the test dataset of the classifiers. These are the `dataAcq_func` and `get_data` files that are available for videos of 2 flies or one fly and one magnet. By running the `get_data` files, the data should be generated.