

Computational Methods for Behavior Analysis

Thesis by
Eyrun Eyjolfsdottir

In Partial Fulfillment of the Requirements for the
degree of
Doctor of Philosophy

CALIFORNIA INSTITUTE OF TECHNOLOGY
Pasadena, California

2017
Defended September 16, 2016

ABSTRACT

Behavioral scientists strive to decode the functional relationship between sensory input and motor output of the brain, which requires quantitative measurement of animal behavior. Artificial intelligence researchers aim to build intelligent systems, capable of understanding, predicting, and generating behavior. Our research lies on the intersection of the two fields; our goal is to automate measurement of animal behavior and to model their sensory-motor relationship using machine learning.

We have developed a tool that tracks the pose of multiple fruit flies and aims to maintain their identity throughout a video. It outputs motion trajectories that can be used to quantify behavioral differences between individuals, for example by comparing histograms of velocities and wing angles. We show that the tool also works well on non-fly-like animals such as zebrafish larvae.

Embedded in these motion trajectories are temporal patterns that constitute *actions*. We developed two supervised learning frameworks for action detection: a sliding window framework and a structured output framework. Both frameworks learn to classify actions from motion trajectories and expert annotated action intervals. Our results show that the simpler sliding window framework achieves better results in spite of being much faster to train, reaching 90% of human performance.

Supervised learning requires a lot of training data which involves time consuming and painstaking annotation. To alleviate that we have built a semi-supervised neural network framework that, in addition to classifying actions, learns to predict how an animal will move next given its motion and sensory inputs so far. Our model achieves as good results as its supervised counterpart with only half of the expert labels. In addition, we show that motion prediction can be used to generate convincing simulations of fruit fly behavior and handwritten text, and that our model learns to represent high level information, such as identity, when trained unsupervised.

Although developed for animal behavior, our methods are general and could be applied to other motion data. We hope that this thesis demonstrates the value of studying animal behavior for the development of artificial intelligence.

PUBLISHED CONTENT AND CONTRIBUTIONS

- [Asa+14] Kenta Asahina, Kiichi Watanabe, Brian J Duistermars, Eric Hoopfer, Carlos Roberto González, Eyrún Arna Eyjólfsdóttir, Pietro Perona, and David J Anderson. “Tachykinin-expressing neurons control male-specific aggressive arousal in *Drosophila*”. In: *Cell* 156.1 (2014), pp. 221–235. URL: <http://www.sciencedirect.com/science/article/pii/S0092867413015365>.
E.E. developed fly tracking software used for the behavior analysis.
- [Eyj+14] Eyrún Eyjólfsdóttir, Steve Branson, Xavier P Burgos-Artizzu, Eric D Hoopfer, Jonathan Schor, David J Anderson, and Pietro Perona. “Detecting social actions of fruit flies”. In: *Computer Vision—ECCV 2014*. Springer, 2014, pp. 772–787. URL: https://link.springer.com/chapter/10.1007/978-3-319-10605-2_50.
E.E. made algorithmic contributions, collected data, designed the experiments, and analyzed the results.
- [Eyj+17] Eyrún Eyjólfsdóttir, Kristin Branson, Yisong Yue, and Pietro Perona. “Learning recurrent representations for hierarchical behavior modeling”. In: *ICLR 2017* (2017). URL: <https://openreview.net/forum?id=BkLhzHtlg¬eId=BkLhzHtlg>.
E.E. designed the algorithms, prepared the data, designed the experiments, and analyzed the results.
- [Lim+14] Rod S Lim, Eyrún Eyjólfsdóttir, Euncheol Shin, Pietro Perona, and David J Anderson. “How food controls aggression in *Drosophila*”. In: *PloS one* 9.8 (2014), e105626. URL: <http://journals.plos.org/plosone/article?id=10.1371/journal.pone.0105626>.
E.E. wrote code for analyzing the data.

TABLE OF CONTENTS

Abstract	iii
Published Content and Contributions	iv
Table of Contents	v
List of Illustrations	vi
Chapter I: Introduction	1
1.1 Behavior analysis and its automation	1
1.2 Contributions and thesis outline	2
Chapter II: Tracking	3
2.1 Evaluation	3
2.2 Detection	9
2.3 Tracking	14
2.4 Pose estimation	21
2.5 Behavior quantification	23
Chapter III: Action detection	26
3.1 Background	27
3.2 Fly-vs-Fly	29
3.3 Feature representation	30
3.4 Sliding window classification	35
3.5 Structured output classification	36
3.6 Experiments and analysis	38
3.7 Discussion	46
Chapter IV: Behavior modeling	50
4.1 Background	51
4.2 Model	52
4.3 Applications	56
4.4 Data	57
4.5 Experiments and analysis	60
4.6 Discussion	69
Chapter V: Conclusion	71
Bibliography	73

LIST OF ILLUSTRATIONS

<i>Number</i>	<i>Page</i>
2.1 Tracker summary	3
2.2 Tracker comparison	4
2.3 Identity accuracy	6
2.4 Pose accuracy	8
2.5 Performance on zebrafish larvae	9
2.6 Background computation	10
2.7 Chamber detection	12
2.8 Fly detection	13
2.9 Tracking procedure overview	14
2.10 Stitching of tracklets	18
2.11 Pose estimation	21
2.12 Heat map of individual's location	23
2.13 Heat map of relative position	24
2.14 Histogram of wing angles and body lengths	24
3.1 Synoptic table of action datasets	28
3.2 Fly-vs-Fly dataset	30
3.3 Actions of fruit flies	31
3.4 Action statistics	32
3.5 Feature representation	33
3.6 Feature distribution for actions	33
3.7 Performance measures	39
3.8 Frame vs. bout discrepancy	40
3.9 Human performance	41
3.10 Human comparison	41
3.11 Method comparison on Fly-vs-Fly	42
3.12 Performance clustering and feature importance	43
3.13 Method comparison on CRIM13	44
3.14 Confusion matrix for CRIM13	45
3.15 Results on Boy meets boy	47
3.16 Results on Aggression	48
3.17 Results on Courtship	49

4.1	Model architecture	53
4.2	Data overview	58
4.3	Data representation	59
4.4	Classification performance	61
4.5	Effect of diagonal connections cost	62
4.6	Motion prediction performance	63
4.7	FlyBowl simulation	64
4.8	Fly-vs-Fly simulation	65
4.9	Handwriting simulation	65
4.10	SynthFly simulation	66
4.11	Fly-vs-Fly discovery	67
4.12	Handwriting discovery	68
4.13	Unsupervised action and actor classification	69

Chapter 1

INTRODUCTION

1.1 Behavior analysis and its automation

Behavior can be defined as the coordinated responses to external and/or internal stimuli [LLF09]. It is a function of the brain that involves interaction with a dynamic environment and can be perceived at various scales [Gom+14].

Behavioral scientists strive to understand the functional relationship between stimuli and response and how neural systems mediate these relations [Moo02]. Ethologists study the natural behavior of animals, e.g. how it is composed of interconnected actions, while neuroscientists and psychologists study behavior in a controlled environment where they can measure and manipulate neural activations and environmental stimuli. These studies require measuring (describing and/or quantifying) behavior in order to derive correlations or causal relationships between behaviors over time or between behavior and stimuli. Traditionally, behavior has been measured by manual observation but in recent years scientists are turning towards automated systems for more objective and precise measurements, allowing for significantly increased throughput [AP14].

Artificial intelligence (AI) researchers are also concerned with behavior. Like behavioral scientists, they are interested in finding a functional relationship between the input and output of the brain, but not necessarily the underlying physiological processes. In fact, AI systems may eventually outperform humans at any given task. One branch of AI is working on machine understanding and prediction of human activity, which can be applied to surveillance, assisted living, and sports analytics. Another branch focuses on generating intelligent behavior, for applications such as self driving vehicles, robots, and virtual assistants.

The two groups of scientists can benefit from each other. Advances in computer vision, natural language processing, and machine learning enable automatic measurement of animal behavior from video. Data collected for behavioral science experiments, of animals behaving naturally or with a controlled stimuli, for hours or days at a time, can be extremely useful for AI research; obtaining human data of the same caliber is impossible due to privacy and ethical restrictions, and thus emulating animal intelligence is a promising direction for AI.

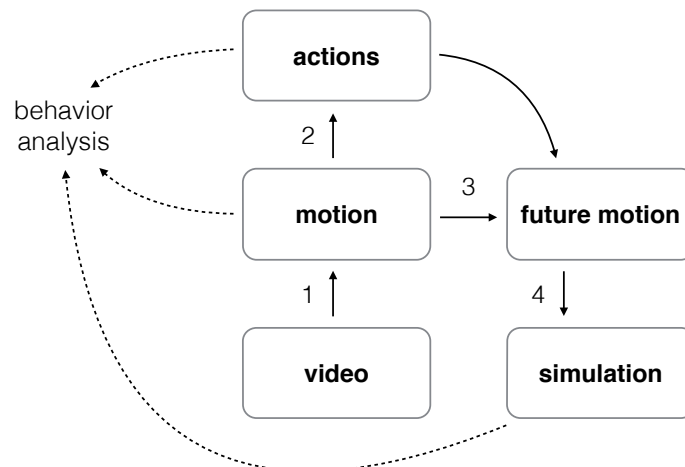
1.2 Contributions and thesis outline

In this thesis I present my work towards automating behavior analysis, which involves: 1) computing motion from video, 2) detecting and classifying actions, 3) predicting future motion, and 4) simulating behavior. The work is organized into three main chapters, each describing one or more of my contributions:

Chapter II describes a system that tracks multiple fruit flies in a video, keeps track of their identity, and outputs motion trajectories describing their pose at every frame. The results demonstrate that these features are useful for quantifying fly behavior.

Chapter III compares two action detection frameworks that learn from expert provided labels to detect and classify actions in motion sequences, allowing for semantic analysis of behavior. It introduces the Fly-vs-Fly dataset which contains videos of fruit fly interactions and is used for measuring performance of the tried algorithms.

Chapter IV proposes a neural network architecture that simultaneously classifies actions and predicts motion. Learning the two objectives simultaneously requires fewer expert labels to train good action classifiers. The structure of this architecture facilitates discovery of high level behavioral phenomena. In addition, motion prediction can be used to simulate behavior, which may become useful for understanding the underlying mechanism of behavior.



Chapter 2

TRACKING

FlyTracker is a system that aims to track the pose (position, orientation, size, wing- and leg positions) of multiple flies and maintain their identities throughout a video. In addition to trajectories, it outputs features (such as velocity, distance to walls, wing angles) useful for behavior analysis. FlyTracker comes with an interface that lets users: 1) calibrate the system on their experimental setup, 2) track all flies in videos, 3) visualize results and correct errors, and 4) annotate behavior of individual flies. It has been tested on videos containing up to 20 individuals, recorded at 25-200 frames per second, and with resolution of 6-60 pixels per body length.

In Section 2.1 we evaluate the performance of FlyTracker, and in Sections 2.2-2.4 we describe in detail the computer vision algorithms behind it which can be divided into: detection, tracking, and pose estimation. FlyTracker is available at www.vision.caltech.edu/Tools/FlyTracker.

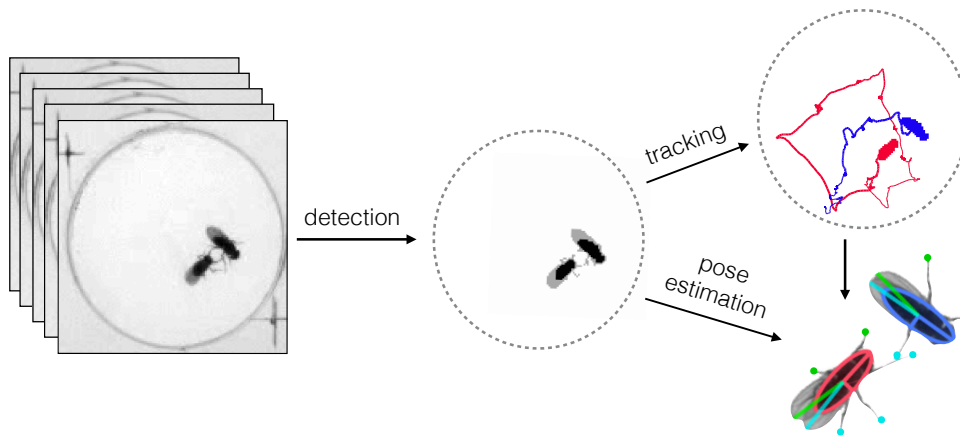


Figure 2.1: Three stages of FlyTracker: **detection** (extract fly bodies and chamber boundaries), **tracking** (connect body components across frames), and **pose estimation** (fit body, wing, and leg components to detected flies).

2.1 Evaluation

We evaluate FlyTracker in the following ways: first, we qualitatively compare it against other tracking tools, second, we compare its identity accuracy with the state of the art, and third, we measure its pose accuracy.

Related work

CADABRA [Dan+09] and Ctrax [Bra+09] are two of the first available fly tracking systems, they were released around the same time but serve different purposes. CADABRA is tailored towards fly pairs and outputs the detailed pose of each fly as well as their mutual position and orientation. Ctrax, on the other hand, tracks multiple flies and uses motion continuity and segmentation of overlapping bodies to keep track of their identity. Both tools have been used extensively for fly behavior analysis, i.e. in the work of [WA10; Wan+11; Lim+14; OZR11; Ram+15], to list a few. IdTracker [Pér+14] is a more recent software that tracks any type of animal, its main strength is re-identification which allows it to keep track of individuals throughout the video and recover from mistakes. It does not attempt to separate bodies that are merged due to overlapping but instead re-identifies them based on an appropriately computed ‘fingerprint’ after they have been separated again.

FlyTracker combines the strengths of each of the above. It tracks multiple flies and extracts a detailed pose of each fly at every frame, which it uses to re-identify flies when motion trajectories are ambiguous. Like Ctrax it resolves touching bodies, and similar to idTracker it uses appearance features to re-identify flies. Its appearance features are less elaborate than those of idTracker, but they are also less computationally expensive and do not make assumptions about lighting or resolution. Figure 2.2 summarizes the difference between the different tracking softwares. A more comprehensive list can be found in [Del+14] which includes software that is commercial, tailored to different animals, or requires an elaborate setup (i.e. multiple cameras or specialized hardware).

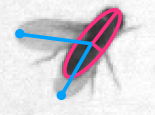
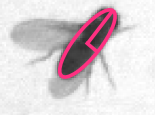
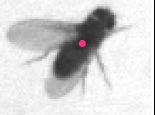
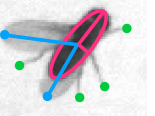
	CADABRA	Ctrax	idTracker	FlyTracker
# animals	2	>20	>20	>20
resolves overlap	yes	yes	no	yes
stitches gaps	yes	no	yes	yes
pose				

Figure 2.2: Comparison of different tracking tools. **Resolves overlap** refers to whether individual bodies are segmented apart when bodies are overlapping. **Stitches gaps** refers to whether trajectory identities are re-assigned when trajectories end and start again (i.e. due to flies leaving field of view or merged bodies).

Identity accuracy

We measure the identity accuracy on three different videos, shown in Figure 2.3, and compare the performance of FlyTracker with idTracker which also aims to keep track of subject's identity. **8-well flies** contains 8 fly pairs, each in a separate chamber, with 405 pixels on each fly's body. This video has two main challenges, 1) when flies are close to chamber walls their reflection can be mistaken for a true fly, and 2) during aggressive behaviors the flies are in close contact and sometimes jump quickly away from each other. However, for each fly pair one of the flies has one wing clipped so that their identities can be determined by a human annotator. FlyTracker automatically detects chamber boundaries, and therefore treats this video as 8 separate tracking problems, but for comparison with idTracker we also include results where the video is treated as a single chamber with 16 flies. **Fly bowl** contains 20 flies, half of which are female, with 108 pixels on each fly's body. The main challenge of this video is that multiple male flies often chase a female at the same time, with bodies of 2 or more flies frequently touching. **Zebrafish** contains 5 zebrafish larvae with 610 pixels covering each body on average, and the main challenge in this video is that fish can swim over each other.

Figure 2.3 compares the two tracking systems in terms of their tracking time (on OSX with 4 cores and 8GB memory), the percent of frames that a fly was not detected (%missed), the percent of frames that a fly was misclassified (%wrong), and how many identity swaps (#swaps) would be necessary for correct assignment (ignoring misidentifications shorter than 1 second). The percent misclassified and number of swaps exclude undetected frames. In addition, we visualize identity assignments to give intuition for the meaning of these errors.

The results show that globally idTracker performs better, as the percent of misidentified fly-frames is lower, but locally FlyTracker performs better, as the percent of missed detections (when animals are touching) is lower, and the number of swaps needed to correct identities is lower on average. On the bowl video, idTracker fails to recover identities. This may be due to a few reasons: 1) although the authors report good results for as low as 50 pixels per body, they recommend 150 pixels for good performance, 2) they report that identity recovery can be compromised when crosses are ubiquitous, which is the case with this video, and 3) the video is backlit and might therefore not have enough detail on the fly body for fingerprinting.

In conclusion, the two systems seem to be complementary; if one cares about keeping track of animal identities, but not about their pose or their location when touch-

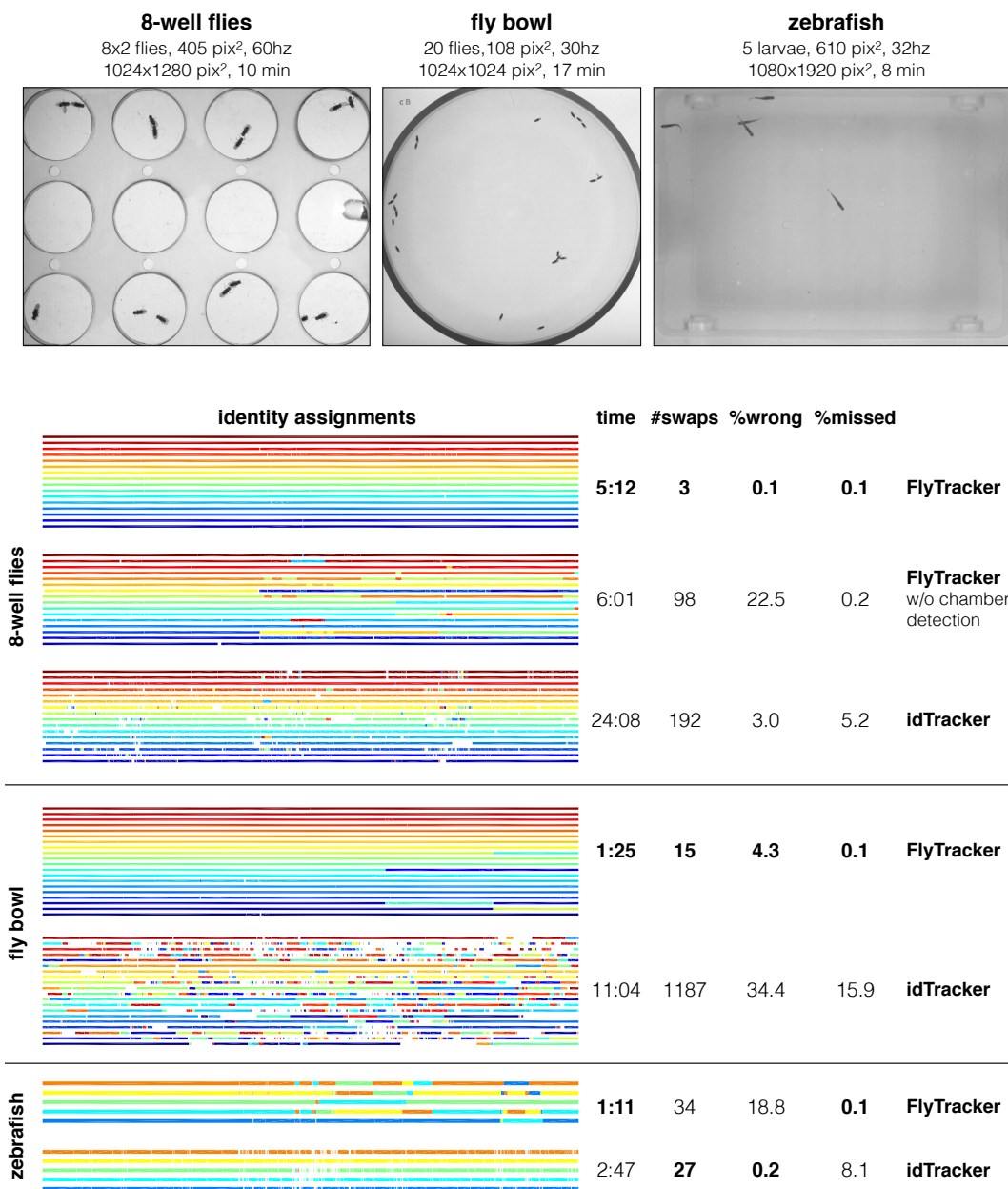


Figure 2.3: **Top:** Videos used to measure identity correctness. **Bottom:** Comparison of identity accuracy between FlyTracker and idTracker. Identity assignments are visualized with time on the x-axis, ground truth fly identity on the y-axis, and color represents the predicted identity of a fly.

ing, idTracker is better (with the caveat that relatively high resolution is required). If one cares about position when animals are in contact and/or the pose of the animals, FlyTracker is a better choice. In addition, FlyTracker is considerably faster than idTracker, especially when animals frequently touch or overlap.

Pose accuracy

In order to measure pose accuracy, we collected ground truth annotations for the 8-well fly video described in the previous section. An annotator was presented with randomly selected frames, zoomed in on a randomly selected fly, and asked to click on its head, the tip of its abdomen, both sides of its body, the two wing tips, and on the tips of all visible legs. In order to measure accuracy during difficult cases, additional frames were sampled where flies are touching and where their bodies overlap. The annotator was asked ignore frames where she could not determine the pose of the fly.

Figure 2.4 plots the pose errors for 1) flies not touching, 2) flies touching, and 3) flies overlapping, and compares ground truth pose to the pose estimated by FlyTracker, showing five smallest and five largest body pose error for each case, measured as a function of position-, orientation-, and length error, and the precision and recall of the wings and legs. We matched wings and legs using a one to one matching with a maximum threshold, and the wing- and leg position errors are reported only for matched legs and wings. Precision refers to the percentage of detected wings/legs that matched with a ground truth wing/leg, and recall to the percentage of ground truth wings/legs that were matched with a predicted wing/leg. Results are based on 510 annotated fly-frames, 150 of which flies are touching, and 110 of which they overlap.

When flies are not touching the median errors are relatively small. Position and length errors are generally around 0.1 mm, orientation error less than 1.5 degree, and both wing and leg precision is greater than 95% while wing recall is 94% and leg recall 83%. When flies overlap, orientation and length errors increase to 2.4 degrees and 0.16 mm, and wing and leg recall drops down to around 70%.

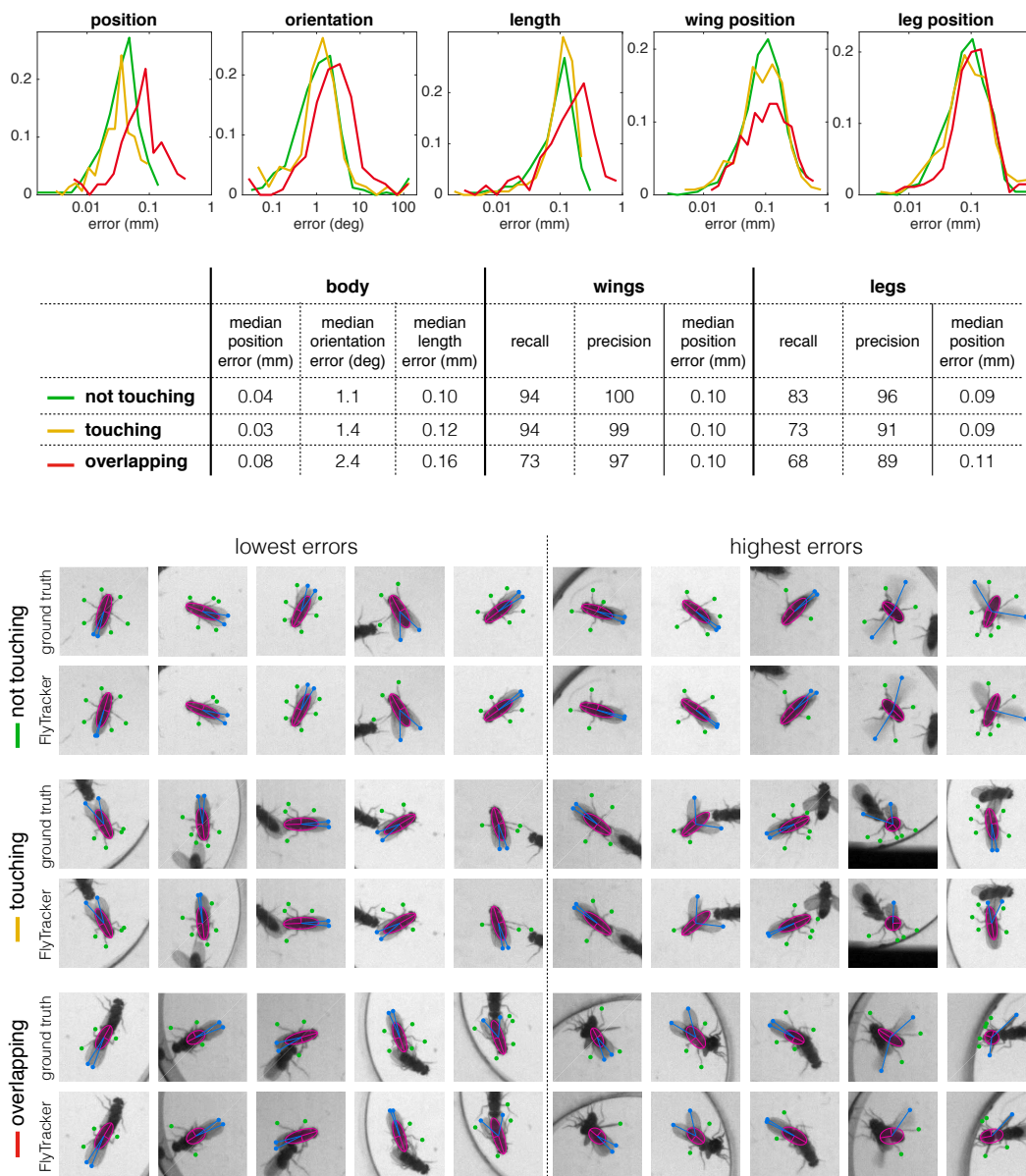


Figure 2.4: **Top:** histogram of pose errors plotted separately for flies not touching (green), touching (yellow) and overlapping (red). **Middle:** median errors for body, wings, and legs, and precision-recall for wing and leg detection. **Bottom:** Five lowest/highest pose errors shown shown for the three cases.

Performance on other animals

FlyTracker is optimized for tracking fruit flies. It makes use of constraints specific to a fly (elliptical body, 2 wings, 6 legs) to fit a skeletal representation to its body. However, it can in theory track any type of animal as long as the background is constant throughout the video and the animals are all of similar size. We showed in Figure 2.3 the identity performance on a video containing 5 zebrafish, and Figure 2.5 shows that the results also look promising when considering segmentation of overlapping fish and pose. The zebrafish larvae do not have wings or legs and their body is non-rigid, but the tracker fits an ellipse to the thicker part of their body and a leg to its tail. The shape of the ellipse and the relative position of the "leg" could be used to infer the curvature of the fish.

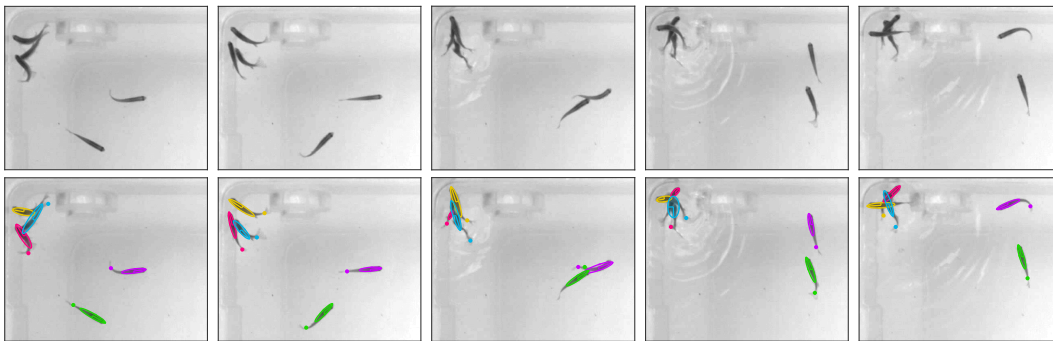


Figure 2.5: Zebrafish larvae tracked using FlyTracker. This example shows a case where identity swap was successfully disambiguated.

2.2 Detection

Input videos are assumed to have rigid camera-chamber setup with constant lighting and no shadows cast from flies, such that a background image can be estimated and used to detect flies via background subtraction. This also requires that the animals are not stationary for the entire duration of the video. A single video may contain several experiments, separated by chamber walls, and in order to reduce complexity of the tracking problem chambers boundaries are detected automatically and flies tracked independently within each chamber. In this section we describe the background computation algorithm, the chamber detection algorithm, and finally the fly detection algorithm.

Background estimation

Two common background computation methods [Ben+08] are 1) taking the mean of all images in a video, which is susceptible to ghost effects when flies stay in place over extended periods of time; and 2) taking the median over all pixels, which is subject to noise and makes the assumption that pixels are not covered by foreground at least half of the time. Our approach combines these two ideas to obtain a smooth background image without ghosting effects, assuming each area of the background is visible in at least one frame.

The background image is computed by iteratively adding weighted images to a background sum. First, a rough background is estimated and at each successive iteration a randomly selected image is added, weighted such that areas believed to belong to the foreground contribute a negligible amount. This is described in Algorithm 1 which takes as input a T -frame video, $\{I_i : i \in [1, T]\}$, and a minimum n and maximum N number of frames to use for background computation, and outputs background image, B , and weight sum, S , which indicates how many images were used to compute the value for each pixel. Figure 2.6 shows steps of the algorithm for an example video.

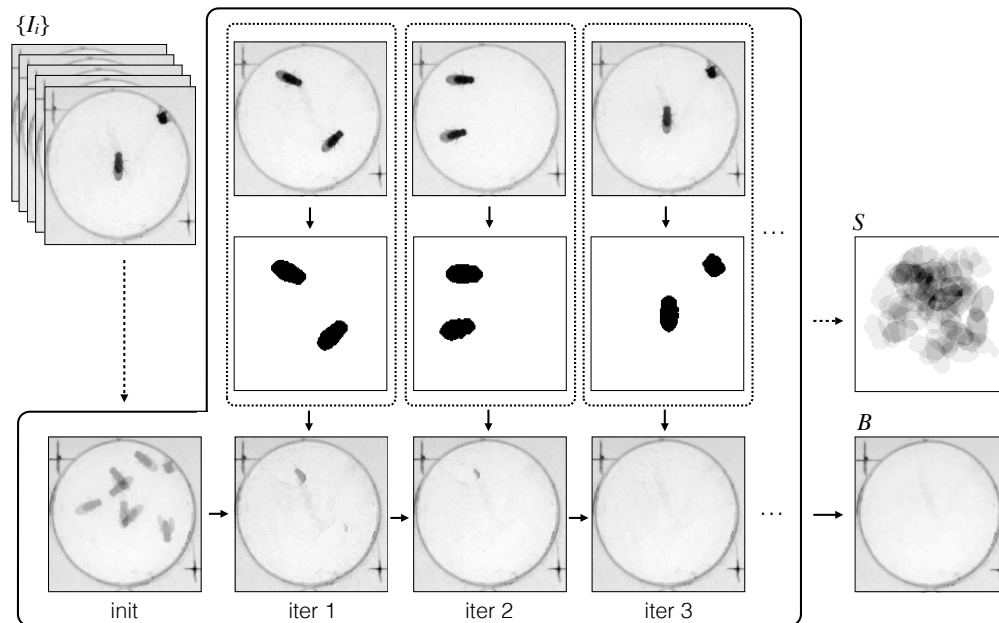


Figure 2.6: Weighted background computation shown for an example video. At each iteration, frame I_i , weighted by its estimated foreground mask, is added to the background weight sum. Here, the final output was obtained after 33 iterations.

Algorithm 1 WeightedBackground

Input: $\{I_i : i \in [1, T]\}, n, N$ **Output:** B, S

```

1:  $B = \text{mean}(\{I_i : i \in 1 : \lfloor T/n \rfloor : T\})$            ▶ rough background estimate
2:  $\tau = \text{inferThresh}(I_{\lfloor T/2 \rfloor} - B)$                  ▶ foreground threshold
3:  $B_S = \mathbf{0}$                                            ▶ initialize background sum
4:  $S = \mathbf{0}$                                            ▶ initialize sum weights
5: for  $i \in \text{randi}(T, N)$  do                               ▶ loop through  $N$  random frames
6:    $W = (I_i - B) < \tau$                                    ▶ background mask
7:    $B_S = B_S + I_i .* W$                                    ▶ add weighted image to background
8:    $S = S + W$                                            ▶ update weight sum
9:    $\alpha = S == 0$                                        ▶ mask for zero weight pixels
10:   $B = B_S ./ S .* (1 - \alpha) + B .* \alpha$              ▶ update background
11:  if  $\min(S) > n$  AND  $\text{converged}(B)$  then break

```

Chamber detection

Detecting chamber boundaries can improve tracking quality as it provides extra constraints for matching flies and it facilitates computation of a fly's proximity to chamber walls which can be informative of the fly's behavior. Algorithm 2 takes as input background, B , background weight sum, S , number of chambers, N , and (optionally) chamber parameters. It outputs chamber centers, C , and the optimal chamber parameters (radius if circular, width and height if rectangular). The algorithm uses the gradient of B to compute an edge mask, and determine the chamber shape: if gradient orientation distribution has 4 modes, the chambers are inferred to be rectangular or otherwise circular. It thresholds S to obtain a cumulative foreground mask. It then loops through a range of possible chamber sizes (determined based on image size), create an edge filter with positive weights on the boundary and negative weights inside it, and a foreground filter with positive weights inside the boundary and negative outside it, and convolves the masks with the respective filters and sum their response. The optimal parameter is that producing the highest maximum response. Finally, from the total response, R , chambers are selected from the top local maxima such that they do not overlap. Figure 2.7 shows the input, output, and intermediate variables for an example video.

Algorithm 2 ChamberDetect**Input:** B, S, N , (param)**Output:** $\{C_i : i \in [1, N]\}$, param

```

1:  $G, O = \text{imGradient}(B)$                                 ▶ magnitude and orientaiton of gradient
2:  $\tau = \text{mean}(G) + \text{std}(G)$ ;                            ▶ gradient threshold
3:  $E = G > \tau$                                           ▶ edge mask
4:  $F = S < \text{prctile}(S, 99)$                             ▶ cumulative foreground mask
5: if margin < 4 then
6:    $r_{max} = 0$ , param = []                                ▶ initialize parameters
7:    $P = \text{getParameterRange}(O, N)$                     ▶ infer chamber parameter range
8:   for  $p \in P$  do                                     ▶ loop through range
9:      $K_E, K_F = \text{generateFilters}(p)$                  ▶ edge and foreground filters
10:     $r = \max(\text{conv}(E, K_E) + \text{conv}(F, K_F))$          ▶ max total response
11:    if  $r > r_{max}$  then  $r_{max} = r$ , param =  $p$      ▶ update parameters
12:   $K_E, K_F = \text{generateFilters}(\text{param})$ 
13:   $R = \text{conv}(E, K_E) + \text{conv}(F, K_F)$                  ▶ response image
14:   $C = \text{getTopResponses}(R, N, \text{param})$                ▶ top N chamber locations

```

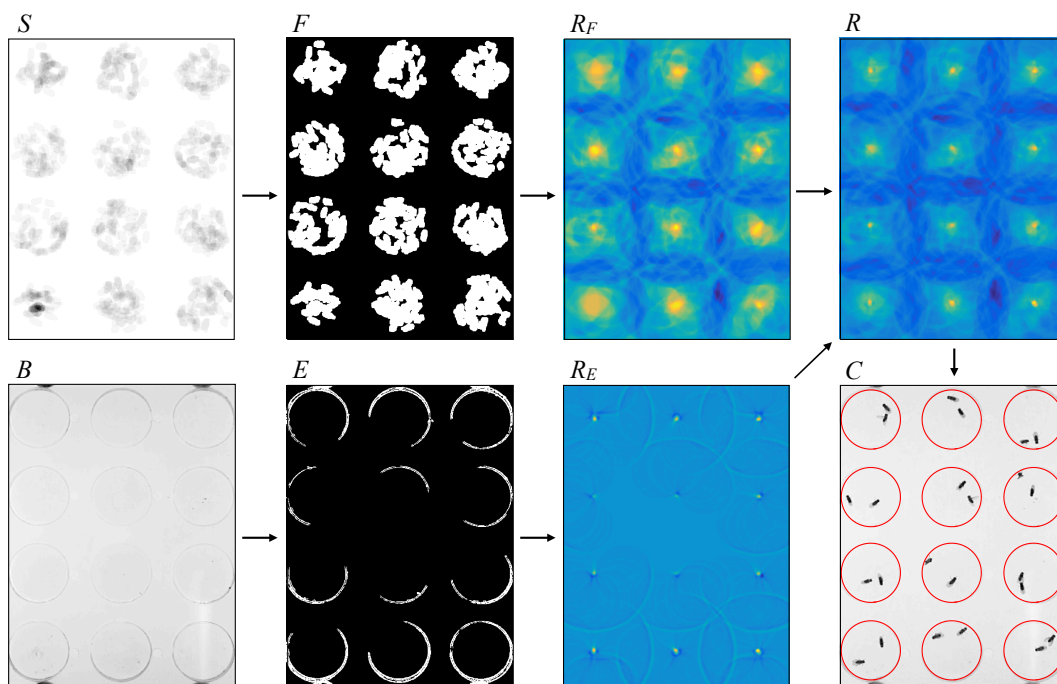


Figure 2.7: Chambers detected from background edges and foreground estimates cumulated during background computation.

Fly detection

Using the same principle as in background computation, where moving objects are located using a rough background estimate, we subtract background, B , from each image to detect flies. The background subtracted image (signal) is normalized with the background itself to account for non-uniform environments: the signal strength of a fly walking on top of a dark area is weaker than a signal of a fly on top of a bright area, as shown in Figure 2.8. The foreground image is thresholded to obtain a foreground mask (weak threshold, τ_f) and a body mask (strong threshold, τ_b) and from the masks we extract connected components (using $(\pm 1, \pm 1)$ pixel neighborhood). This is described in Algorithm 3 which takes as input a video and a background image, and outputs, for each frame, i , a list of foreground objects, $\{f_i^j : j \in [1, n_i]\}$, and body objects, $\{b_i^j : j \in [1, m_i]\}$. Each object is defined in terms of a pixel list, centroid, and area, and foregrounds point to the list of bodies it contains (children), and bodies to their corresponding foreground (parent); if two or more flies are touching, their bodies are the children of a single foreground. Tracking can be done for each chamber individually, considering only detections within each chamber's boundary.

Algorithm 3 FlyDetect

Input: $\{I_i : i \in [1, T]\}, B$

Output: $\{f_i : i \in [1, T]\}, \{b_i : i \in [1, T]\}$

```

1: for  $i \in [1, T]$  do                                     ▶ loop over frames
2:    $I_f = (I_i - B) ./ \max(0.1, B)$                        ▶ normalized foreground image
3:    $M_f = I_f < \tau_f$                                      ▶ foreground mask
4:    $M_b = I_f < \tau_b$                                      ▶ body mask
5:    $M_b = \text{dilate}(\text{erode}(M_b))$                        ▶ eliminate noise
6:    $\{f_i^j : j \in [1, n_i]\} = \text{conncomp}(M_f)$           ▶ foreground objects
7:    $\{b_i^j : j \in [1, m_i]\} = \text{conncomp}(M_b)$           ▶ body objects
8:    $\text{removeInvalid}(f_i, b_i)$                              ▶ too small / big / skinny
9:    $\text{assignRelationship}(f_i, b_i)$                        ▶ assign bodies to foregrounds

```

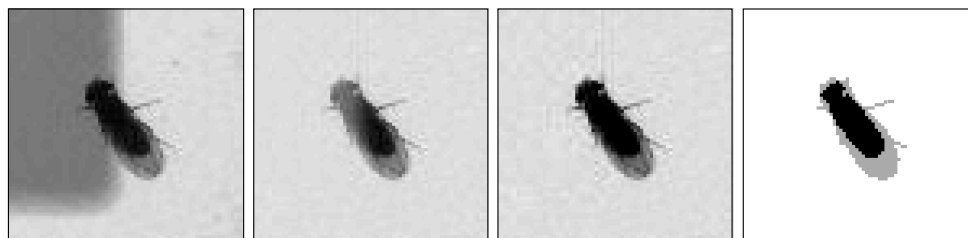


Figure 2.8: Background subtraction: $I_i \rightarrow (I_i - B) \rightarrow I_{fg} \rightarrow (M_f + M_b)$. In the last image, black pixels represent body and gray \cup black pixels represent foreground.

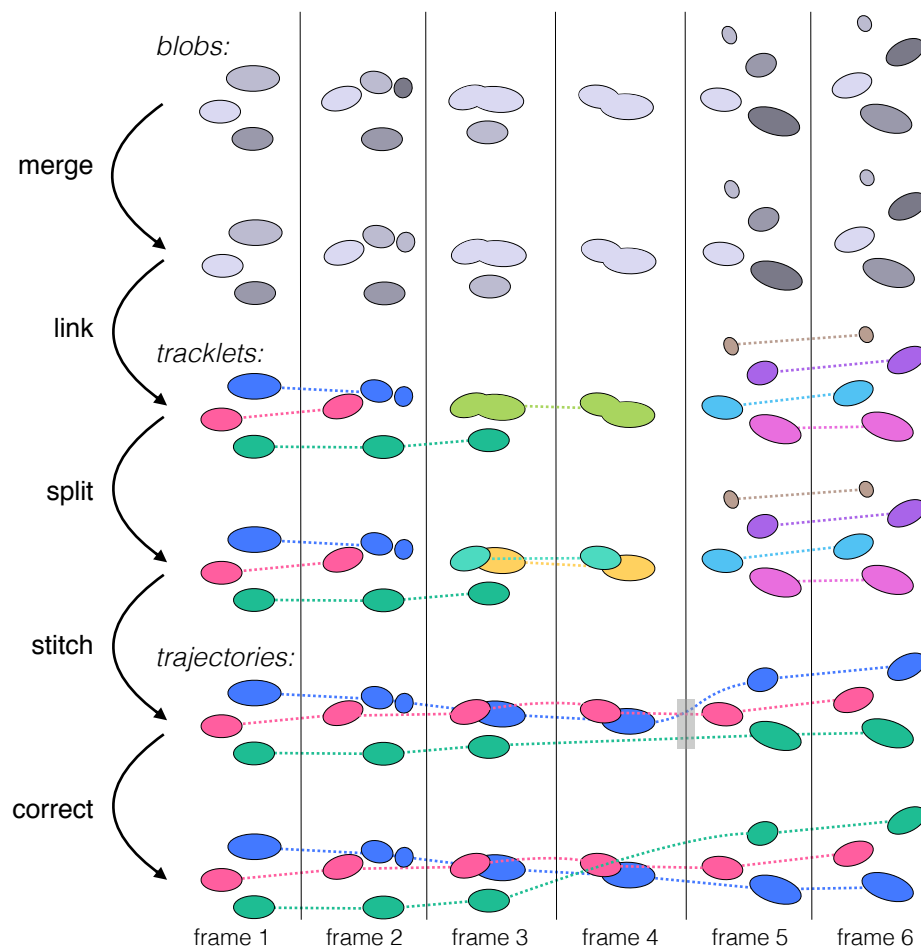


Figure 2.9: A synthetic example demonstrating the tracking procedure. The top row shows blob detections (colored in a shade of grey according to their identity within the frame) which include anomalies such as split, merged, missing, and false blobs. Our tracking algorithm is split into the following steps: 1) **merge** small blobs with nearest sibling, 2) **link** blobs across frames unless matching is ambiguous, 3) **split** tracklets containing more than one fly, 4) **stitch** tracklets into full length trajectories, and 5) **correct** potential identity swaps based on appearance.

2.3 Tracking

Tracking by detection involves matching detected bodies between consecutive frames to obtain contiguous trajectories. If detection were perfect this would be a fairly simple task. However, common problems include: a) merged detections when subjects overlap, b) missed detections caused by motion blur or subjects out of field of view, c) false detections due to noise, d) split detections due to over-segmentation, and e) mismatching due to discontinuous motion of subjects. Our tracking algorithm, summarized in Figure 2.9, is designed to handle these challenges.

Merging

From now on we will refer to detected body, b_i^j , as a *blob*, as it may not represent a self-contained body but rather multiple bodies, body fragment, or noise. We want to split blobs that contain more than one fly, and merge blobs that belong to a single fly, but solving both issues simultaneously is a circular problem: if two blobs in frame i match to one blob in frame $i + 1$, should the blobs in frame i be merged or the blob in frame $i + 1$ be split? Frames $i - 1$ and $i + 2$ could help disambiguate that, however, further merging or splitting of blobs in surrounding frames, as well as missing and false blobs, can further complicate things. By merging small blobs with nearby blobs (whose body they potentially belong to), the problem reduces to splitting only. A blob's survival is based on its *cardinality*, c , which measures how many fly bodies a blob is believed to contain. Blobs with 0-cardinality are merged with their closest sibling (with smallest distance between closest pixels).

Algorithm 4 Merge

Input: $\{b_i : i \in [1, T]\}$

Output: $\{b_i : i \in [1, T]\}$

- 1: $m_a = \text{median}_{i,j}(b_i^j.\text{area})$ ▷ expected size of blob
 - 2: **for** $b \in \{b_i^j\}$ **do**
 - 3: $b.c = \text{round}(b.\text{area} / m_a)$ ▷ blob cardinality
 - 4: **if** $\text{isLargestChild}(b)$ **then** $b.c = \max(b.c, 1)$ ▷ keep largest / only child
 - 5: **if** $b.c == 0$ **then** $\text{mergeWithClosestSibling}(b)$
-

Linking

Ultimately, the goal is to match blobs between consecutive frames and link them together from frame 1 to T to form *trajectories* for individual flies. We start by linking the blobs into shorter trajectories, *tracklets*, whose matches are un-ambiguous. Matching is considered ambiguous when a) many blobs match to one, b) one blob matches to many, c) swapping matches results in low cost increase, or d) cost of matching is too high. Algorithm 5 takes as input the list of blobs at each frame, and outputs a list of N tracklets, $\{t_i : i \in [1, N]\}$. For n blobs in frame i , and m blobs in frame $i + 1$, it builds an $n \times m$ cost matrix, C , defined in terms of blob overlap and distance. It uses the Hungarian matching algorithm [Kuh55] to assign blobs from frame i to blobs in frame j , then greedily allows blobs to swap to a high cardinality blobs if the match is cheaper and both cardinality constraints are satisfied, and finally removes assignments that are too expensive or ambiguous. Remaining matches are joined to form tracklets.

Algorithm 5 Link

Input: $\{b_i : i \in [1, T]\}$
Output: $\{t_i : i \in [1, N]\}$

```

1:  $N = |b_1|$                                 ▶ current number of tracklets
2:  $inds = [1, N]$                             ▶ current tracklet indices
3: for  $i \in [1, N]$  do
4:    $t_i = \text{init}(b_1^i, 1, 0)$             ▶ start tracklet at frame 1 with 0 matches
5: for  $i \in [1, T]$  do                    ▶ loop through frames
6:    $A = \text{MATCH}(b_i, b_{i+1})$             ▶ assignments
7:    $A^* = \text{removeOneMany}(A)$             ▶ one-to-one assignments
8:   for  $j, k \in \text{find}(A^* == 1)$  do
9:      $t_{inds(j)}.append(b_{i+1}^k)$       ▶ append to existing tracklet
10:  for  $j \in \text{find}(\text{sum}(A^*, 1) == 0)$  do
11:     $c = \text{sum}(A(j, :))$                 ▶ number of matches from tracklet
12:     $t_{inds(j)}.end(i, c)$             ▶ end tracklet at frame  $i$  with  $c$  matches
13:     $inds = inds \setminus inds(j)$       ▶ remove index from current indices
14:  for  $k \in \text{find}(\text{sum}(A^*, 2) == 0)$  do
15:     $c = \text{sum}(A(:, k))$                 ▶ number of matches to tracklet
16:     $t_{N+1} = \text{init}(b_{i+1}^k, i + 1, c)$  ▶ start tracklet at frame  $i + 1$  with  $c$  matches
17:     $N = N + 1$ 
18:     $inds = inds \cup (N + 1)$           ▶ add index to current indices
19:
20: function  $\text{MATCH}(b_i, b_{i+1})$ 
21:    $C = \text{COST}(b_i, b_{i+1})$             ▶ match cost
22:    $n, m = \text{size}(C)$ 
23:    $C' = \mathbf{1}((n + m)) * \max(C)*2$       ▶ add high-cost dummy nodes ...
24:    $C'(1 : n, 1 : m) = C$                 ▶ ... for non-matches
25:    $A' = \text{hungarian}(C')$               ▶ one-to-one matching
26:    $A' = \text{greedySwap}(C', A', b_i.c, b_{i+1}.c)$  ▶ swap to cheaper if allowed
27:    $A = A'(1 : n, 1 : m)$                 ▶ remove dummy matches
28:    $A(C > \tau_C) = 0$                   ▶ un-assign invalid matches
29:    $A(\text{ambiguous}(A, \tau_{dC})) = 0$       ▶ un-assign ambiguous matches
30:   return  $A$ 
31:
32: function  $\text{COST}(b_i, b_{i+1})$ 
33:   for  $b^j \in \{b_i\}, b^k \in \{b_{i+1}\}$  do
34:      $C_o(j, k) = 1 - (b^j \cap b^k) / (b^j \cup b^k)$  ▶ overlap cost
35:      $C_d(j, k) = \text{distance}(b^j, b^k)$  ▶ distance cost
36:   return  $C_o * \lambda + C_d$           ▶  $\lambda \simeq$  fly length

```

Splitting

Some of the tracklets may contain more than one fly (in the case where flies overlap). The next step is to determine the cardinality of each tracklet, similar to what we did with blobs before. The cardinality is determined based on how many tracklets match to and from the tracklet, as well as blobs-cardinality within the tracklet, and is set such that cardinality is consistent between matching tracklets. Once the algorithm has determined tracklet-cardinalities, it split blobs of tracklets with cardinality $c > 1$ by fitting a mixture of c 2-dimensional Gaussians to the blob pixels using Expectation-Maximization [DLR77]. It then set the cardinality of all blobs to 1 and rerun the linking algorithm to obtain $M \leq N$ 1-fly tracklets.

Algorithm 6 Split

Input: $\{t_i : i \in [1, N]\}, \{b_i : i \in [1, T]\}$

Output: $\{t_i : i \in [1, M]\}$

- 1: $\{c_i\} = \text{computeCardinality}(\{t_j : j \in [1, N]\})$
 - 2: **for** $j \in \text{find}(\{c_i\} > 1)$ **do** ▷ split high cardinality tracklets
 - 3: **for** $b_i^k \in t_j.\text{blobs}$ **do**
 - 4: $\{b^l : l \in [1, c_j]\} = \text{splitBlob}(b_i^k, c_j)$ ▷ fit mixture of c_j Gaussians
 - 5: $b_i.\text{remove}(b_i^k)$ ▷ remove blob from frame i
 - 6: $b_i.\text{add}(\{b^l : l \in [1, c_j]\})$ ▷ add blobs to frame i
 - 7: **for** $b \in \{b_i^j\}$ **do** $b.c = 1$ ▷ set all cardinalities to 1
 - 8: $\{t_i : i \in [1, M]\} = \text{Link}(\{b_i : i \in [1, T]\})$ ▷ re-link all blobs
-

Stitching

The next step is to stitch together the tracklets to form full trajectories for each fly (similar to linking blobs to tracklets) and discard tracklets containing noise. The tracklets may be separated because of ambiguous or expensive matches, missing blobs, or noise blobs. The first step of stitching is to identify *sub-problems*, or intervals of overlapping gaps. Between sub-problems there exist exactly as many *confident* tracklets as there are flies, and a sub-problem involves all tracklets whose start or end overlaps with the sub-problem's interval. This is visualized in Figure 2.10 A). A tracklet's confidence is a function of its duration and distance traveled.

For each sub-problem, the objective is to find a path from each confident tracklet entering the problem to each confident tracklet exiting the problem. The cost of matching tracklets is composed of three factors: **Time cost**, C_T , represents the time between the end of tracklet i to the beginning of tracklet j , **space cost**, C_S , is computed between the blob at the end of tracklet i and the blob at the beginning

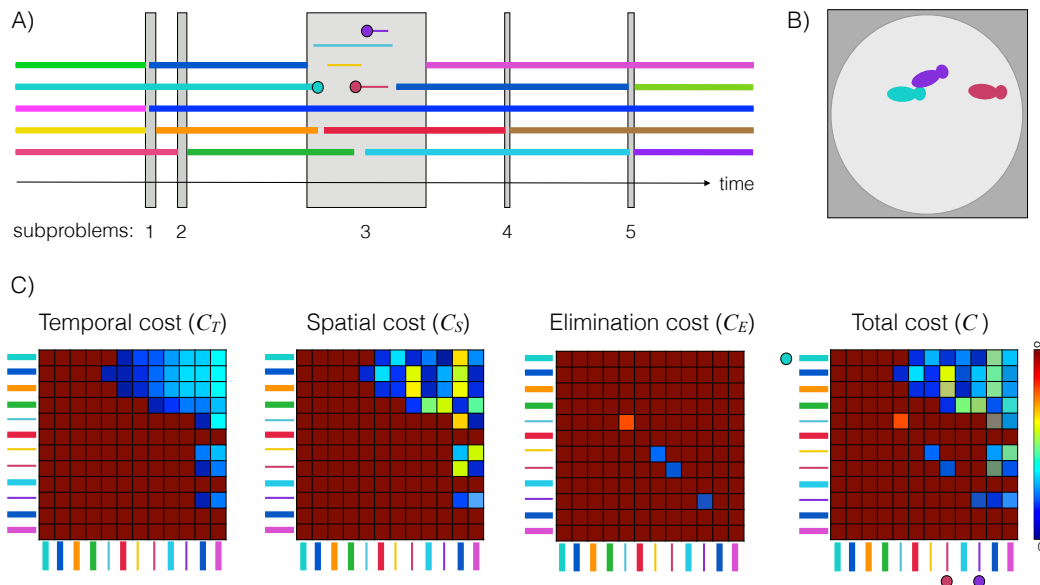


Figure 2.10: A) Tracklets arranged greedily. Overlapping gaps define a subproblem. Note that only tracklets whose boundaries overlap with the subproblem window belong to that subproblem. B) Location of blobs at time indicated by circles in A). C) Cost matrices for subproblem 3. The total cost shows that it is more optimal to match the cyan tracklet to the purple one rather than the pink one.

of tracklet j , using their predicted location at the frame between the tracklets, and **elimination cost**, C_E , is the cost of eliminating tracklet i based on its duration, distance, and confidence. Space and time costs between tracklet i and j are infinite if j starts before i ends, and all confident tracklets have infinite elimination cost. The tradeoff between time and space cost is such that when duration between tracklets is low, space cost weighs higher, and when duration is high the time cost dominates the total cost (because flies may have traveled arbitrarily far).

This can be solved as a one-to-one matching problem, again using the Hungarian algorithm, with a guaranteed path between the confident tracklets at the beginning and end of the subproblem (a confident tracklet cannot connect with a self-eliminating tracklet, because, by definition, self-eliminating sequences connect *from* themselves *to* themselves). All tracklets that match to themselves can be discarded as noise.

Trajectories are constructed the same way as tracklets were from blobs, by linking together the matched tracklets. This time everything is matched, even if matching is ambiguous, but all ambiguous matches are flagged such that they can later be automatically or manually corrected.

Algorithm 7 Stitch**Input:** $\{t_i : i \in [1, M]\}, n_{flies}$ **Output:** $\{t_i : i \in [1, n_{flies}]\}, F$

```

1:  $S = \text{FINDSUBPROBLEMS}(\{t_i : i \in [1, M]\})$ 
2:  $A_G = \mathbf{0}$  ▷ Global,  $M \times M$ , assignment matrix
3: for  $s \in S$  do
4:    $C = \text{COMPUTECOST}(\{t_i : i \in s.inds\})$  ▷ cost of matching tracklets
5:    $A = \text{hungarian}(C)$  ▷ one-to-one assignment
6:    $F = [F \text{ flag}(A, \tau)]$  ▷ add ambiguous matches to flags  $F$ 
7:    $A_G(s.inds, s.inds) = A$  ▷ update global assignment
8:  $\{t_i : i \in [1, n_{flies}]\} = \text{removeAndLink}(\{t_i : i \in [1, M]\}, A_G)$ 
9:
10: function  $\text{COMPUTECOST}(\{t_i : i \in [1, n]\})$ 
11:    $C_T = C_S = C_E = \infty(n, n)$ 
12:   for  $i \in [1, n]$  do
13:     for  $j \in [1, n]$  do
14:       if  $\text{start}(j) > \text{end}(i)$  then
15:          $C_T(i, j) = \text{start}(j) - \text{end}(i)$  ▷ temporal cost
16:          $C_S(i, j) = \text{spaceCost}(t_i, t_j)$  ▷ spatial cost
17:       if  $i$  unconfident then
18:          $C_E(i, i) = \text{elimCost}(\text{votes}, \text{dur}, \text{dist})$  ▷ elimination cost
19:        $W = 1 / (1 + \exp(-(C_T + 1)))$  ▷ time-space tradeoff
20:        $C_T = \text{sqrt}(C_T)$ 
21:        $C = \min(W .* C_S + (1 - W) .* C_T, C_E)$  ▷ total cost
22:   return  $C$ 
23:
24: function  $\text{FINDSUBPROBLEMS}(\{t_i : i \in [1, M]\})$ 
25:    $count = \mathbf{1}$  ▷ 1 x T vector
26:    $\{c_i : i \in [1, M]\} = \text{confidence}(\{t_i : i \in [1, M]\})$  ▷ tracklet confidence
27:   for  $i \in \text{find}(c_i > 0)$  do
28:      $count(t_i.start + 1 : t_i.end - 1) = count(t_i.start + 1 : t_i.end - 1) + 1$ 
29:    $S = \text{conncomp}(count < n_{flies})$  ▷ framerate for each subproblem
30:   for  $s \in S$  do ▷ assign trajectories to subproblems
31:      $s.inds = \text{findBoundyOverlap}(s, \{t_i : i \in [1, M]\})$ 
32:   return  $S$ 

```

Automatic identity correction

Finally, potential identity swaps are disambiguated using appearance features such as body width, length, and area, and length of wings. But first, in order to obtain such a description of each fly, pose extraction (described in the next section) is performed on the detection and tracking outputs. Each identity swap involves two flies (if more flies are involved, they are treated as independent swaps). Using appearance features from frame 1 to the swap frame, f , a binary classifier (logistic regression on binned appearance features) is trained and applied to the appearance features from f to the next swap frame for each fly. The output of the classification at every test frame is aggregated into a probability matrix. If the probability of swapping exceeds a threshold, then blobs from f onward are swapped between the two trajectories, and flags are updated accordingly.

In cases where fly appearances vary significantly, for example when comparing male vs. female or when one of the flies has its wing clipped, auto-identification works well, but in cases where flies are very similar, ambiguous matches may need further correcting. The FlyTracker interface allows the user to loop through potential swaps and manually correct them.

Algorithm 8 AutoCorrect

Input: $\{t_i : i \in [1, n_{flies}]\}, F$

Output: $\{t_i : i \in [1, n_{flies}]\}$

```

1: for  $a, b, f \in F$  do                                ▶ trajectories  $a$  and  $b$  at frame  $f$ 
2:    $X_a = \text{shapeFeat}(t_a, 1, f - 1), Y_a = \mathbf{0}$       ▶ extract features from start...
3:    $X_b = \text{shapeFeat}(t_b, 1, f - 1), Y_b = \mathbf{1}$       ▶ ... until swap
4:    $X_{tr} = [X_a, X_b], Y_{tr} = [Y_a, Y_b]$            ▶ training data
5:    $f_a = \text{nextFlag}(F, a, f), f_b = \text{nextFlag}(F, b, f)$ 
6:    $X_a = \text{shapeFeat}(t_a, f + 1, f_a), Y_a = \mathbf{0}$     ▶ extract features from swap...
7:    $X_b = \text{shapeFeat}(t_b, f + 1, f_b), Y_b = \mathbf{1}$     ▶ ... until next swap
8:    $X_{te} = [X_a, X_b], Y_{te} = [Y_a, Y_b]$            ▶ test data
9:    $\text{model} = \text{train}(X_{tr}, Y_{tr})$                    ▶ train binary classifier
10:   $S = \text{predict}(X_{te}, \text{model})$                    ▶ apply classifier
11:   $p = \text{aggregate}(S)$                                ▶  $p(i, j), i, j \in \{a, b\}$ 
12:  if  $p(a) = b - p(a) = a > \tau$  then              ▶ if confidently swap
13:     $t_a, t_b, F = \text{swap}(t_a, t_b, f, F)$           ▶ update trajectories and flags at  $f$ 

```

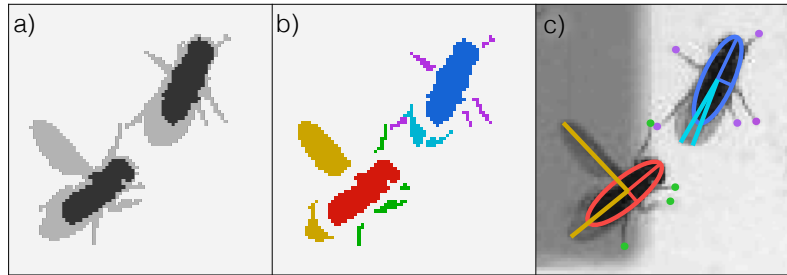


Figure 2.11: a) Foreground (gray) and body (black), b) wings, legs, and body segments, c) flies parameterized with body ellipse, wing segments and leg positions.

2.4 Pose estimation

In addition to tracking individual flies, a skeleton is fit to each fly such that their locomotion can be measured as well as their whereabouts. This is done by segmenting the masks output from detection into body-, wing-, and leg pixels, and fitting an oriented ellipse to the body pixels, a line segment to the wing pixels, and points to the tips of the leg pixels, as visualized in Figure 2.4 and described in Algorithm 9.

For each body object output by the detection algorithm, a Gaussian ellipse is fit to the list of pixels it contains by computing the mean and covariance over the pixel locations. The orientation of the ellipse is the angle between the image x-axis and the major axis length of the fly, but it may be flipped by 180 degrees to enforce consistency with 1) fly's wing positions, 2) velocity, or 3) surrounding frames, depending on which one is most trusted.

Next, the foreground objects are used to extract legs and wings: each foreground's mask is eroded and dilated such that "skinny" regions of the mask disappear. Connected components of pixels that disappeared are considered leg candidates. Leg pixels and the dilated body masks and then subtracted from the foreground mask, and connected components of remaining pixels for candidate wings. Validity of legs and wings, and to which fly they belong (when multiple bodies contained in foreground) is determined using size/pose constraints. Each body is assigned at most 6 legs and 2 wings, and if a fly has only one wing component it is split along its major axis to form two wing components. The position of wing/leg segments is represented by the location of the pixel furthest away from the body ellipses center, and are ordered based on their signed angle with respect to the fly's major axis.

From this representation, features such as velocity, wing angles, and distance to other flies, which are useful for quantifying a fly's behavior, are computed.

Algorithm 9 ExtractPose

Input: $\{f_i : i \in [1, T]\}, \{b_i : i \in [1, T]\}, \{t_i : i \in [1, n_{flies}]\}$
Output: $\{t_i : i \in [1, n_{flies}]\}$

```

1: for  $b_i^j \in \{b_i : i \in [1, T]\}$  do
2:    $b_i^j.(centroid, ori, axes) = \text{fitEllipse}(b_i^j)$            ▶ fit ellipse to each body
3: for  $t \in \{t_i : i \in [1, n_{flies}]\}$  do                 ▶ disambiguate orientation using...
4:    $t = \text{bwdDisambiguateOri}(t)$            ▶ ... velocity- and backwards consistency
5: for  $i \in [1, T]$  do
6:   for  $f \in f_i$  do                                       ▶ loop through foregrounds in frame
7:      $M_F^{ed} = \text{dilate}(\text{erode}(M_F))$            ▶ remove "skinny" regions
8:      $M_L = M_F - M_F^{ed}$                                        ▶ leg mask
9:      $cc_L = \text{conncomp}(M_L)$                                        ▶ leg candidates
10:     $M_W = M_F - M_L - \text{dilate}(M_B)$            ▶ wing mask
11:     $cc_W = \text{conncomp}(M_W)$                                        ▶ wing candidates
12:    for  $b \in f$  do                                       ▶ loop through bodies in foreground
13:       $b.ori = \text{fwdDisambiguateOri}(b.t)$ 
14:       $b.wings = \text{assignWings}(b, cc_W)$ 
15:       $b.legs = \text{assignLegs}(b, cc_L)$ 
16:
17: function ASSIGNWINGS( $b, cc$ )
18:    $trusted = \text{strictConstraints}(b, cc)$            ▶ large, same side, 1 resting body
19:   if  $trusted$  then
20:      $b.ori = \text{disambiguateWing}(b, cc)$            ▶ adjust ori to fit wing positions
21:      $valid = \text{constraintsWings}(b, cc)$            ▶ shape and pose constraints
22:     if  $|valid| == 1$  then                                       ▶ if wings merged as one
23:        $cc, valid = \text{splitMiddle}(b, cc(valid))$            ▶ split along major axis
24:      $wings = \text{fitVector}(cc(valid), b.centroid)$  ▶ use furthest pixel from body
25:     return  $wings(1 : \max(2, |wings|))$            ▶ return 2 best candidates
26:
27: function ASSIGNLEGS( $b, cc$ )
28:    $valid = \text{constraintsLegs}(b, cc)$            ▶ shape and pose constraints
29:    $legs = \text{fitVector}(cc(valid), b.centroid)$  ▶ use furthest pixel from body
30:   return  $legs(1 : \max(6, |legs|))$            ▶ return 6 best candidates

```

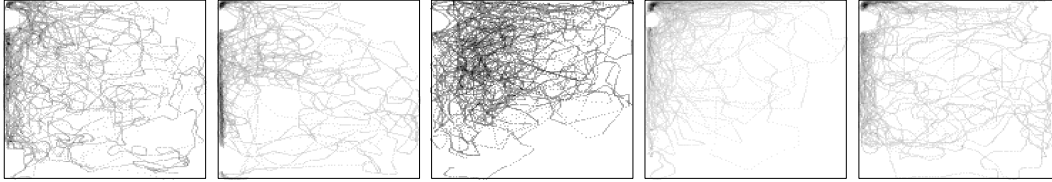


Figure 2.12: Heat map for the position of each larva in the zebrafish video. Here we can see that the larvae are attracted to one corner of the arena, and that individual 3 explores more than the others.

2.5 Behavior quantification

Global measurements Having extracted motion trajectories from videos, one can begin to quantify behavior. Global measurements of the data, such as histograms or heat maps for an individual or a group can be very informative about their behavior. For example, Figure 2.12 shows a heat map for each larva in the zebrafish video, from which it can be inferred that the north-west corner of the chamber was attractive to the group. It can also be seen that individual number 3 spent more time exploring the chamber than the others. Another example that involves a heat map computed from the position of animals is shown in Figure 2.13 which sums up the relative position of other individuals with respect to each individual. Here we can see that females (bottom two rows) have individuals located behind them and males (top two rows) generally have other individuals in front of them. This is indicative of a chasing behavior between males and females. One interesting thing to note here is that male number 5 does not seem to chase females.

Figure 2.14 shows global measurements that have to do with the pose of flies rather than their location. On the left it shows a histogram of wing angles, plotted for each individual in the fly bowl video. It has three prominent modes, one where the angle is low, which is common for both males (colored in shades of green-blue) and females (colored in shades of purple-pink), one around 15 degrees which is common for two of the female flies, and one that is close to 90 degrees that is occasionally reached by males. In the middle of Figure 2.14 we show a histogram of the long axis of the body ellipse for each fly, and it is clear that females are significantly larger than males, which can be a useful feature for automatic behavior analysis. On the right we have pictured a heat map of leg locations with respect to the fly's pose, for all flies in the 8-well flies video. Perhaps not surprisingly, there are six modes around the fly. One might use this information to look all frames where legs are between the two front modes, which might indicate flies grooming.

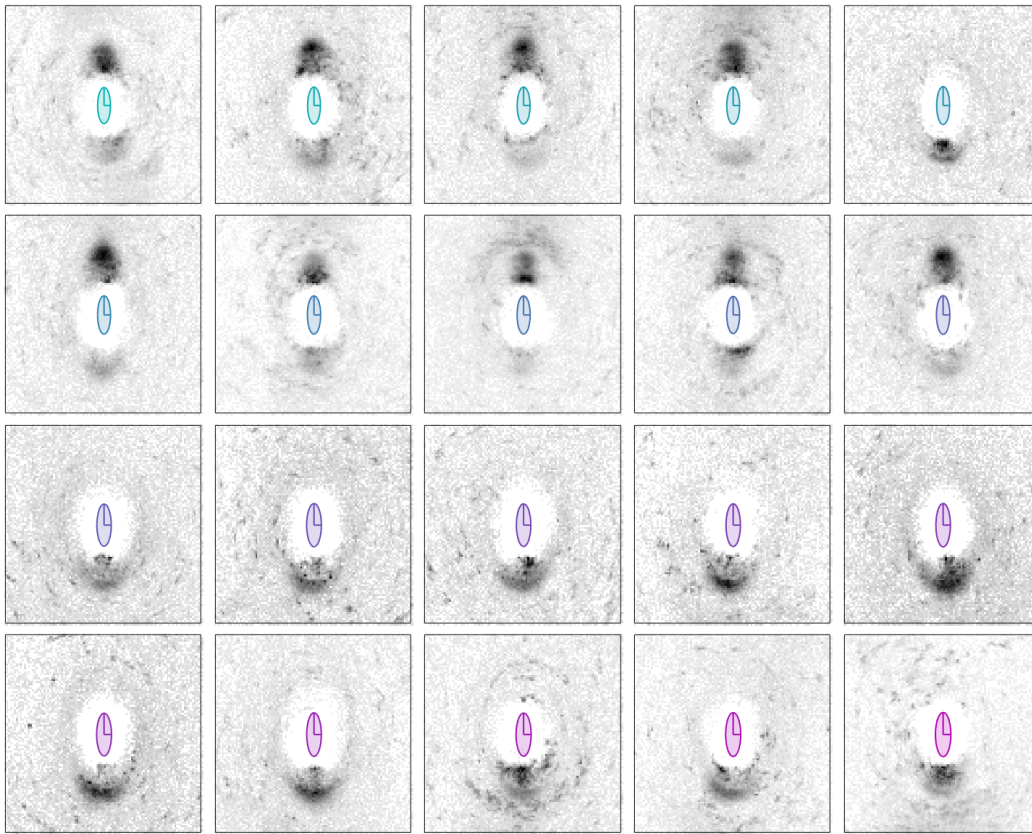


Figure 2.13: Heat map for the position of flies relative to each fly in the fly bowl. The top two rows are male flies and the bottom two are females. Here it is clear that females are chased by males, except that male 5 seems to also be a chaser.

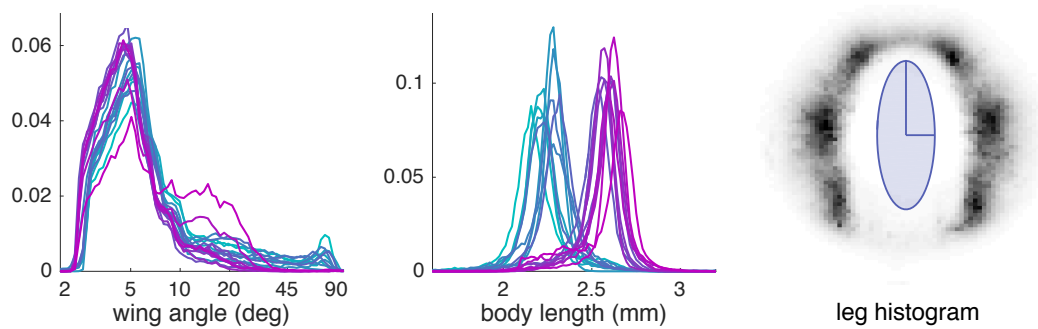


Figure 2.14: **Left:** Histogram of wing angle for each fly in the fly bowl. We see that males (colored in shades of blue) sometimes have close to a 90 degree angle, while females do not. **Middle:** Histogram of body length of each fly in the fly bowl. The two modes represent males and females. **Right:** Heat map of leg positions of flies in the 8-well flies video.

Local measurements One might also be interested in measuring how a fly's pose evolves over time or in the spatio-temporal patterns in its pose, which constitute *actions*. This could be done by specifying thresholds for features, such as wing angle or velocity, and searching for all frames where a feature or a combination of features exceeds its threshold. For more intricate patterns, one could specify that feature 1 has to be above a threshold for a certain duration, followed by feature 2 exceeding its threshold within a certain duration, and so on. However, this kind of manual approach is limited by the user's notion of which features are informative of an action, and may need iterative fine tuning to eliminate false detections or missed action instances. An automatic and more robust solution for learning such patterns is described in the following chapter.

Chapter 3

ACTION DETECTION

Machine understanding of human behavior is potentially the most useful and transformative application of computer vision. It will allow machines to be better aware of their environment, enable rich and natural human-machine interaction, and it will unleash new applications in a number of industries including automotive, entertainment, surveillance and assisted living. Development of automated vision systems that can understand human behavior requires progress in object detection, pose estimation, tracking, action classification and detection, and activity analysis. Progress on the latter (actions and activities) is hampered by two difficulties. First, tracking and pose estimation is very difficult in humans due to variation in clothing, the amount of occlusion in natural environments and in social conditions, and the sheer complexity and number of human body motions. Second, it is difficult (both technically and legally) to film large numbers of humans acting spontaneously while they perform interesting activities. As a result, human action datasets are small and unrepresentative, especially when social behavior is concerned.

A good strategy for computer vision researchers to make progress on behavior analysis is to shift their attention to the simpler world of laboratory animals [Bur+12a]. We collaborate with behavioral neurobiologists who are interested in measuring and analyzing behavior across genotypes, in order to understand the link between genes, brains, and behavior. One of their most popular model organism is the fruit fly, *Drosophila melanogaster*: it is easy to care for, has a fast life cycle, and exhibits a wide range of behaviors despite having merely 10^5 neurons. Through this collaboration we have put together a large annotated dataset of fruit flies interacting spontaneously in controlled environments. This dataset allows us to study natural actions and develop insight into how to represent, segment and classify them. If our effort is successful, we can both advance the state of the art in human action analysis and provide biologists with tools for automatic labeling of actions, enabling them to do experiments at a scale which would otherwise be extremely expensive or impossible.

In this chapter we describe methods for detecting actions of fruit flies from their trajectories obtained using FlyTracker. The main contributions of our study are:

- 1) We consider two different action detection architectures: sliding window detectors and structured output detectors. By comparing five variants of the two architectures on our dataset, we find that sliding window detectors outperform the structured output detectors, in spite of being orders of magnitude faster.
- 2) We describe *bout features* that extract statistical patterns from frame-level features over an interval of time, and emphasize the similarities of bouts within an action class. Our experiments show that actions cannot be well detected using frame-level features alone, and that bout features improve performance by 28%.
- 3) We discuss pitfalls of measures commonly used for benchmarking action detection in continuous video and demonstrate which measures are most suitable, suggesting a protocol for comparing the performance of different algorithms.
- 4) We introduce Caltech Fly-vs-Fly Interactions (Fly-vs-Fly for short), a dataset containing 22 hours of fruit flies interacting spontaneously and sporadically. It comes with complete labeling of 10 actions, annotated by neurobiologists, and a second layer of annotations that can be used as a reference point for action detection performance. Along with the videos and annotations we publish a number of time-varying trajectory features, computed from the tracked pose (position, orientation, wing angles, etc.) of the flies. The dataset is available at www.vision.caltech.edu/Video_Datasets/Fly-vs-Fly.

3.1 Background

Datasets A large number of human action datasets have been published. KTH [SLC04] and Weizmann [Gor+07] are early contributions that have been extensively used, but they are very small and consist of pre-segmented clips of acted actions. Hollywood 2 [MLS09], Olympic Sports [NCF10], HMDB51 [Kue+11], and UCF-101 [SZS12], contain pre-segmented clips of natural actions, making them suitable for action classification, but not for detection and segmentation of actions, while UT-interactions [RA10] contains continuous social interactions that are acted. VIRAT [Oh+11] contains hours of continuous video of humans behaving naturally and intermittently, lending itself well to action detection research; however, the pose of the subjects cannot yet be robustly tracked and the human motion that can be explored is limited; furthermore, VIRAT does not contain social actions. HumanEva [SB06], HDM05 [Mül+07], TUM Kitchen [TBB09], CMU MMAC [De +09], and CAD-60/120 [Sun+12; KGS12] are continuous and come with fully tracked skeletons which makes them useful for analyzing a range of human motions; however,

Dataset	Year	#Citations	Duration	#Actions ⁺	Natural	Social	Continuous	Articulated pose
KTH	2004	1634	3 hours*	6	x	x	x	x
Weizmann	2005	986	5 minutes*	10	x	x	x	x
HumanEva	2006	583	22 minutes	6	x	x	✓	✓
HDM05	2007	107	3 hours	70	x	x	✓	✓
TUM Kitchen	2009	98	1 hour*	13	x	x	✓	✓
CMU MMAC	2009	97	6 hours*	16**	x	x	✓	✓
Hollywood 2(1)	2009	436(1327)	20 hours	12	✓	✓	x	x
Olympic Sports	2010	196	2 hours*	16	✓	x	x	x
UT Interactions	2010	41	20 minutes	6	x	✓	✓	x
HMDB51	2011	137	2 hours	51	✓	✓	x	x
VIRAT	2011	91	29 hours	23	✓	x	✓	x
UCF-101(50,11)	2012	40(57,477)	27 hours	101	✓	✓	x	x
CAD-60/120	2011/13	175/34	2 hours*	22**	x	x	✓	✓
UCSD mice	2005	1458	2 hours	5	✓	x	x	x
Honeybee	2008	61	3 minutes	3	✓	x	✓	x
Home-cage	2010	60	13 hours	8	✓	x	✓	x
CRIM13	2012	15	37 hours	12	✓	✓	✓	x
Fly-vs-Fly	2014	-	22 hours	10	✓	✓	✓	✓

*estimated upper limit, **sub-activities (actions/verbs), *excluding the null category

Figure 3.1: Synoptic table of action datasets shown in chronological order, grouped by human vs. animal. Properties desired for detecting realistic social actions from articulated pose are highlighted in green.

these datasets are small, and their actions are acted. Figure 3.1 compares the detail of the mentioned datasets.

Publicly available datasets of animal behavior video are Honeybee Dance [Oh+08], UCSD mice [Dol+05], Home-cage behaviors [Jhu+10a], and CRIM13 [Bur+12a]. The latter two are suitable for action detection, containing long videos of spontaneous mouse actions, but both are parameterized with only the tracked centroid of the subject and spatial-temporal features. A large and well annotated dataset containing unsegmented, spontaneous, social actions, that includes tracking of articulated body motion has not yet been published. Our dataset aims to fill that place.

Action detection A common approach to action detection is frame-by-frame classification, where each frame is classified based on features extracted from the frame itself, or from a time window around it: [Dan+09] detected actions of fruit flies using manually set thresholds on frame-level features, along with nearest neighbor comparison; [Bur+12a] used boosting and auto-context on sliding windows for detecting actions between mice; and [Kab+12] also use window based boosting for detecting actions of fruit flies in their interactive behavior annotation tool, JAABA. More sophisticated approaches globally optimize over possible temporal segmenta-

tions, outputting structured sequences of actions: [Jhu+10a] used an SVMHMM, described in [A+03], for detecting actions of single housed mice; [HLD11] used a multi-class SVM with structured inference for segmenting the dance of the honey-bee; and [Shi+11] used a discriminative semi-Markov model for segmenting human actions.

We implemented three variants of the above approaches, specifically comparing a sliding window SVM detector against two structured output SVM detectors, expecting the latter to improve frame-wise consistency and better capture structured actions. For reference, we compare our results with the methods described in [Kab+12] and [Bur+12a] and with the performance of trained novice annotators.

3.2 Fly-vs-Fly

In collaboration with biologists we have collected the Fly-vs-Fly dataset, which contains a total of 22 hours (1.5m frames recorded at 200Hz and 2.2m frames at 30Hz) of 47 pairs of fruit flies interacting. The videos are organized into three subsets, each of which was collected for a different study:

Experimental setup

Boy meets boy is designed to study the sequence of actions between two male flies, whose behaviors range from courtship to aggression. The flies are placed in a 4x5 cm² chamber with a food patch in its center and walls coated with Fluon, constraining the flies to walking on the floor [Hoy+08]. It contains six 20 minute videos recorded at 200Hz with 12 pix/mm (24 pixels covering the 2mm fly body length).

Aggression contains videos of two hyper aggressive males [Hoo+15] and is used to quantify the effect of genetic manipulation on their behavior. The flies are placed in a circular 16mm diameter chamber with uniform food surface [Asa+14]. It consists of ten 30 minute videos recorded at 30Hz with 8 pix/mm.

Courtship videos contain a female and a male, which in some cases are wild type and in the rest are so-called hyper courters [Hoo+15]. This set of videos was used to study how genetic manipulation affects male courtship behavior. It consists of 31 videos recorded with the same chamber and video settings as Aggression.

The filming setups for these experiments are shown in Figure 3.2.

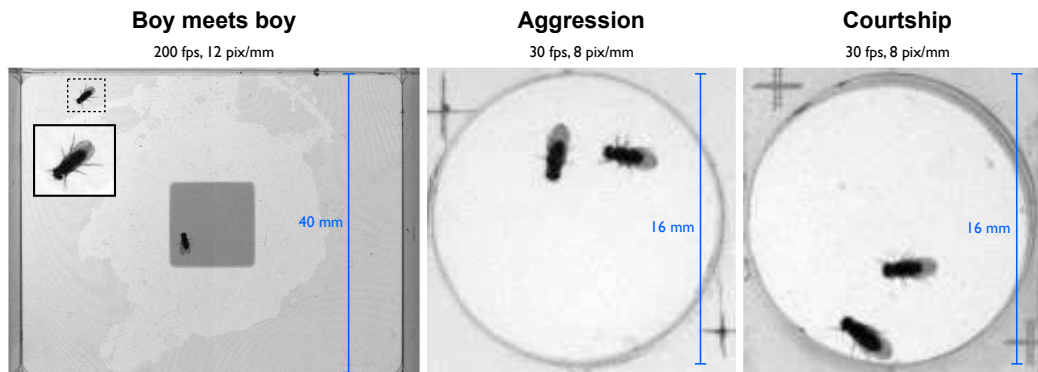


Figure 3.2: Experimental setup for Boy meets boy, Aggression, and Courtship.

Annotations

The entire dataset was annotated by, or under the supervision of, biologists, with 10 action classes that have been identified for the study of fruit fly interactions [Che+02; Hal94; Dan+09]: *wing threat*, *charge*, *lunge*, *hold*, and *tussle* (aggressive), *wing extension*, *circle*, *copulation attempt*, and *copulation* (courtship), and *touch* (neutral). Each action is described and visualized in Figure 3.3.

Annotating a video involves finding all time intervals that contain an action of interest, also referred to as action *bouts*, and requires recording the start frame, end frame, and class label of each detected bout. The dataset is annotated such that actions can overlap, for instance tussling usually includes lunging, wing threat sometimes includes a charge, and wing extension and circling tend to overlap. Each action class takes up less than 2% of the total frames in a video, apart from touch (7%) and copulation (57%), and some classes have substantial intraclass variation, both in terms of duration and appearance. Figure 3.4 summarizes the dataset.

The Fly-vs-Fly dataset, including videos, trajectory features, and annotations, is available at www.vision.caltech.edu/Video_Datasets/Fly-vs-Fly.

3.3 Feature representation

For action classification, data representation is half the challenge. An ideal classifier is invariant of any intra-action variation, but to train such a classifier a complex model or large amounts of training data may be needed. Alternatively, this invariance can be encoded into the features.

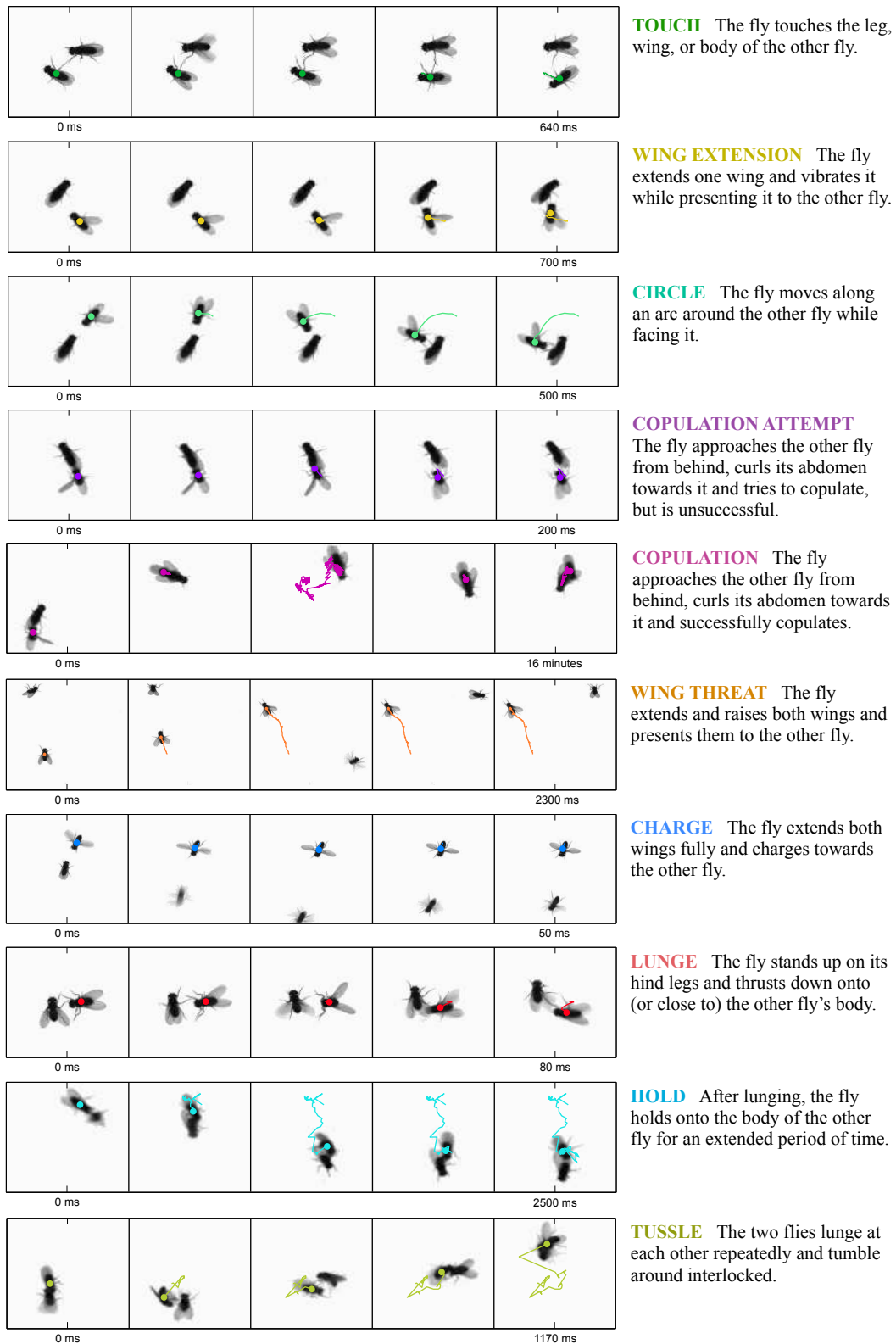


Figure 3.3: An example and description of each action in Fly-vs-Fly.

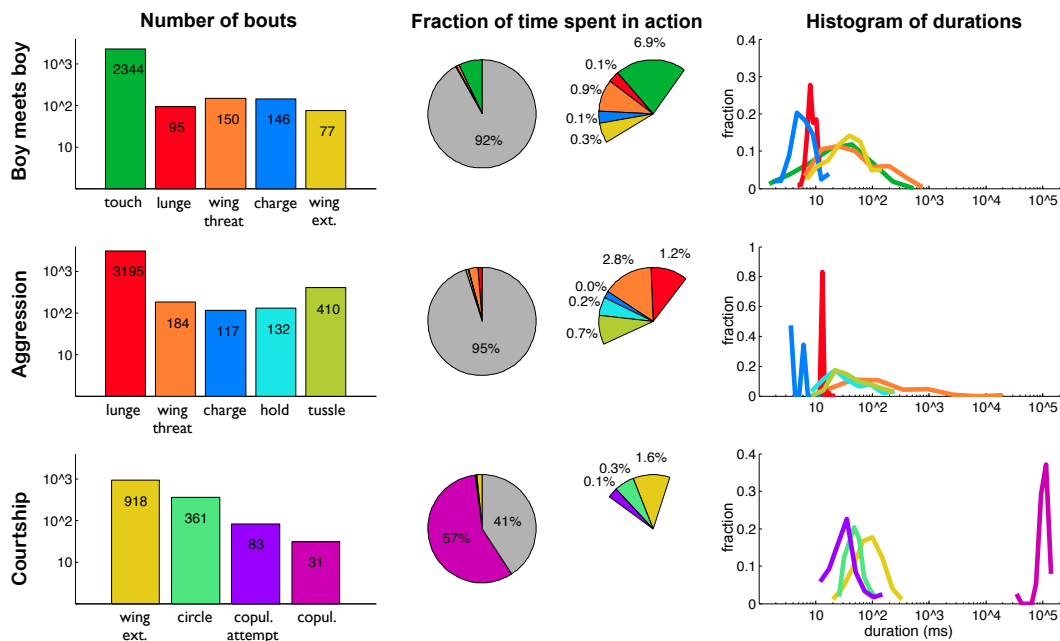


Figure 3.4: Action statistics: *Left*: Number of bouts for each action. *Center*: Fraction of time a fly spends in each action, where the gray area represents the grab-bag category *other*. *Right*: Distribution of bout durations for each action class.

Frame features

From the tracking output, described in Chapter 2, we derive a set of features that are designed to be invariant of the absolute position and orientation of a fly, and relate its pose to that of the other fly. The features (illustrated in Figure 3.5) can be split into two categories: individual features which include the fly’s *velocity*, *angular velocity*, *min and max wing angles*, *mean wing length*, *body axis ratio*, *foreground-body ratio*, and *image contrast* in a window around the fly; and relative features which relate one fly to the other with *distance between* their body centers, *leg distance* (shortest distance from its legs to the foreground of the other fly), *angle between*, and *facing angle*. Analysis of the feature distributions showed that the velocities, wing angles, and foreground-body ratio are better represented by their log values, becoming more normal distributed. Figure 3.6 shows the distribution of each feature, for all actions in the Boy meets boy sub-dataset, giving an idea of which features are important for which action. In addition, we take the first two time derivatives of each feature, resulting in a feature space of 36 *per-frame* features. The features are computed from the reference frame of each fly, yielding two asymmetrical feature vectors, as the actions we consider are always involve one fly ‘performing’ the action.

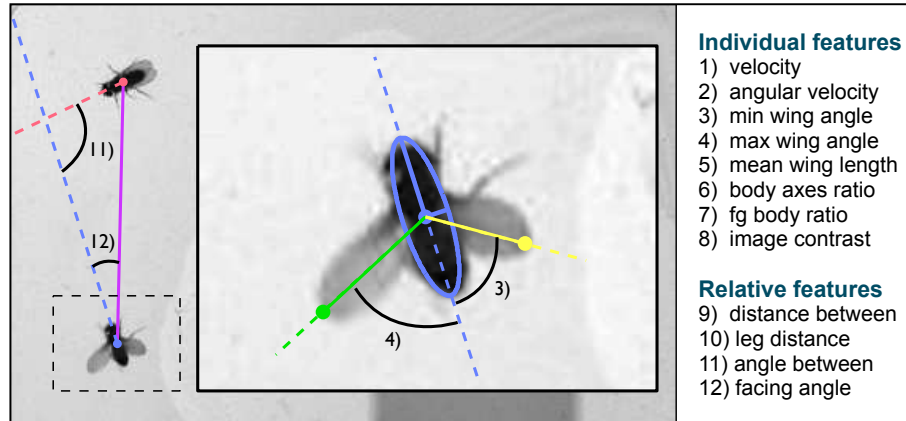


Figure 3.5: Illustration of features derived from the tracked fly skeletons. Individual features describe the fly's motion and are invariant of its absolute position and orientation, and relative features relate the pose of the fly to that of the other fly.

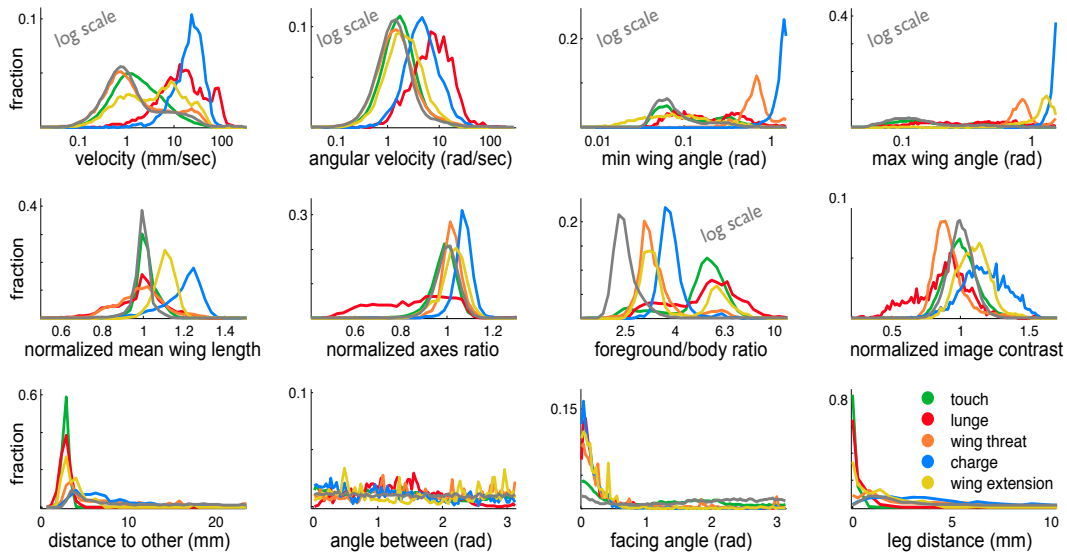
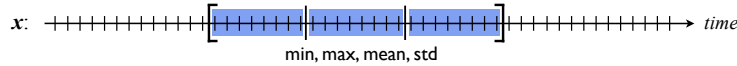


Figure 3.6: Frame-wise feature distribution for the actions of the Boy meets boy sub-dataset, and the grab-bag action *other* shown in gray. Here we see which features are important to each action, for instance, velocity is high during lunge and charge, min wing angle is high during charge and wing threat, and max wing angle is high during wing extension.

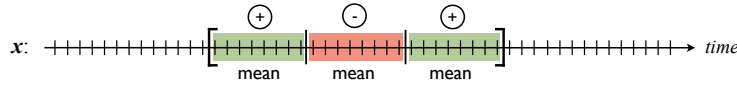
Bout features

Although the actions look fairly separable in the per frame features, adding context of the surrounding frames may significantly improve classification. We define a number of bout-level features that are designed to extract statistical patterns from an interval and emphasize the similarities of bouts within an action class, independent on bout duration. The following bout features, $\psi_k(x, t_{start}, t_{end})$, are functions of sequence x and interval $[t_{start} t_{end}]$:

Temporal region features capture statistics of frame-level features over subintervals, and emphasize patterns within an action composed of r subactions. They can be expressed as: $\{\text{op}(x(t_{start} + (i - 1)\delta t : t_{start} + i \delta t - 1))\}_{i \in \{1, \dots, r\}}$, where $\delta t = (t_{end} - t_{start} + 1)/(r - 1)$, $r \geq 1$, and $\text{op} \in \{\text{min}, \text{max}, \text{mean}, \text{std}\}$.



Harmonic features are meant to capture harmonic actions and can be expressed as: $\sum_{i=1}^r (-1)^i \text{mean}(x(t_{start} + (i - 1)\delta t : t_{start} + i \delta t - 1))$, where $\delta t = (t_{end} - t_{start} + 1)/(r - 1)$ and $r \geq 1$.



Boundary features emphasize the change in features at the start and end of a bout, and help with locating boundaries. For a fixed δt , they can be expressed as: $\text{mean}(x(t_{start/end} : t_{start/end} + \delta t)) - \text{mean}(x(t_{start/end} - \delta t : t_{start/end}))$.



Bout change features capture the difference in features between the beginning and end of a bout, expressed as: $x(t_{end}) - x(t_{start})$.

Global difference features compare the mean of a bout to global statistics of data, expressed as: $\text{mean}(x(t_{start} : t_{end})) - \text{op}(x)$, where $\text{op} \in \{\text{min}, \text{max}, \text{mean}\}$.

Histogram features represent the normalized distribution of each feature within the bout, expressed as: $\text{hist}(x(t_{start} : t_{end}), \text{bins})$, where bins are extracted from the training data, such that an equal number of frames falls into each bin.

In our experiments we use three temporal region splits, $r \in \{1,2,3\}$, and set the number of histogram bins to be 2^3 , resulting in a total of $K = 48$ bout functions. With K bout functions applied to each of the N per-frame features, the feature representation for a bout ends up being a $D = KN$ dimensional vector, ψ .

3.4 Sliding window classification

In this paper we focus on detection by exhaustive classification, in particular we compare two different architectures: *Sliding window classification* which refers to classifying fixed size windows that move frame-by-frame over a video sequence, and *structured output classification* which refers to detection by optimizing over all possible segmentations of a sequence into actions. Both schemes involve a training algorithm that learns an action classifier from n labeled sequences, $\{(x_i, y_i)\}_{i \in \{1, \dots, n\}}$, and an inference algorithm that takes a new sequence x and predicts $y := \{y^j\} = \{(s^j, e^j, c^j)\}$, where y^j is the j th bout in the segmentation of x , s^j and e^j mark the start and end of the bout and c^j is its class label. We treat the problem of detecting different actions as disjoint detection problems, mainly because the data that we are interested in has many overlapping actions.

Our sliding window implementation has 4 main components: a *training* algorithm that learns a classifier from labeled sequences, a *classifier* module, an *inference* algorithm that predicts labels for unseen sequences, and a *post processing* module that promotes continuity in the prediction labels.

Training: The training algorithm converts each sequence of input labels, $\{y_i\} = \{(s^j, e^j, c^j)_i\}$, to indicator vectors, $\{z_i\}$, that specify whether a frame belongs to an action or not. It extracts normalized bout features over fixed sized windows surrounding each frame of all sequences, obtaining high dimensional data points whose labels are the same as those of the frames around which the windows were placed. With this data it trains a classifier using a bootstrapping scheme that overcomes memory limitations that may be associated with large data, and allows us to indirectly optimize with respect to performance measures that involve the number of predicted positives. At each iteration it learns a classifier from a subset of the data, using a learning algorithm suitable for the classifier type, applies it to all of the data and adds misclassified samples to the training set - repeating until the desired performance measure stops increasing.

Inference: The inference algorithm extracts bout features from a window around each frame in x , normalized with statistics from the training data, and classifies each

window using the classifier obtained from the training step. The resulting sequence of scores is thresholded to obtain an action indicator vector, \hat{z} , whose connected components make up the predicted label sequence, \hat{y} , assigning each bout the label, start frame, and end frame of its component.

Post processing: Classifying a sequence frame-by-frame often results in noisy labels, that is, within a bout of an action a few frames may be just below a threshold and therefore split the bout into multiple bouts. To account for this we fit an HMM to the scores to achieve smoother transitions: we convert scores to posterior probabilities, $P(x(t)|z(t) = 1) := 1/(1+\exp(-\text{score}(t)))$, $P(x(t)|z(t) = 0) := 1 - P(x(t)|z(t) = 1)$, compute prior probabilities, $P(z(1) = c)$, and transition matrix, $P(z(t+1) = c_i|z(t) = c_j)$, from the training data, and run the Viterbi algorithm [Vit67] to find the most probable frame-wise sequence of actions.

Classifier: The classifier module consists of a binary classifier and its associated learning algorithm. For comparison with our structured SVM implementation, we choose to use a linear SVM classifier, learnt using the LIBLINEAR implementation described in [Fan+08]. The classifier can be substituted by any other binary classifier, such as boosting, regression, neural net, or a generative model.

This approach can be converted to a frame-based detector, by simply substituting the bout features around a frame with its per-frame features.

3.5 Structured output classification

Structured output detectors differ from sliding windows in that they optimize over all possible segmentations of a sequence into action intervals, finding the best start and end frame of all bouts, allowing for varying sized intervals.

Structured SVM

We extend the structured SVM [Tso+05] to train a model that can be utilized for segmenting sequences into actions, by defining a *score function*, $f(x, y)$, which assigns high scores to good segmentations, and a *loss function*, $\mathcal{L}(y, \hat{y})$, which penalizes poor segmentations.

Training: The goal is to learn the weights w of a score function from a given training set, such that for each training example the score of the true segmentation y_i is higher than the score of any other segmentation y by at least $\mathcal{L}(y_i, y)$. If these constraints cannot be satisfied, a hinge loss is suffered. To learn these weights we

use the primal structured SVM objective:

$$w^* \leftarrow \arg \min_w \|w\|^2 + C \frac{1}{n} \sum_{i=1}^n \left(\max_y [f(x_i, y) + \mathcal{L}(y_i, y)] - f(x_i, y_i) \right),$$

which we minimize using a cutting plane algorithm [Tso+05] that iteratively finds the most violated constraint: $\hat{y} = \arg \max_y [f(x_i, y) + \mathcal{L}(y_i, y)]$. Searching over all possible segmentations is intractable, but since our score- and loss functions are linear in the bouts of y , dynamic programming [Bel56] can solve for the optimal y .

Score function: We define a score function $f(x, y)$, which measures how well y segments x into actions and can be represented as the sum of a bout score, unary cost, transition cost, and duration cost, over all bouts in the segmentation:

$$f(x, y) = \sum_{(s^j, e^j, c^j) \in y} [w_{c^j} \cdot \psi(x, s^j, e^j) - \lambda(c^{j-1}, c^j) - \gamma(c^j, s^j, e^j) - \tau(c^j)].$$

Weights w_{c^j} are used to calculate the score for a bout of class c^j , $\tau(c^j)$ is the cost of detecting a bout of class c^j , $\lambda(c^{j-1}, c^j)$ is the cost of moving from action c^{j-1} to c^j , and $\gamma(c^j, s^j, e^j)$ is the cost of spending $e^j - s^j + 1$ frames in action c^j . These terms are inspired by a hidden semi Markov model, comparable to [Shi+11].

Loss function: The loss function penalizes discrepancies between ground truth segmentation y and a predicted segmentation \hat{y} , and should be constructed such that a small loss indicates satisfactory results. We define it as:

$$\mathcal{L}(y, \hat{y}) = \sum_{(s, e, c) \in y} \frac{\ell_{fn}^c}{e - s + 1} \left(\bigcap_{\hat{y}, \hat{c} \neq c} (s, e) \right) + \sum_{(\hat{s}, \hat{e}, \hat{c}) \in \hat{y}} \frac{\ell_{fp}^{\hat{c}}}{\hat{e} - \hat{s} + 1} \left(\bigcap_{y, c \neq \hat{c}} (\hat{s}, \hat{e}) \right),$$

where $\bigcap_{\hat{y}, \hat{c} \neq c} (b, e)$ is the number of frames in \hat{y} intersecting with $[b, e]$ with different action class $\hat{c} \neq c$, ℓ_{fn}^c is the cost for missing a bout of class c , and $\ell_{fp}^{\hat{c}}$ is the cost for incorrectly detecting a bout of class \hat{c} . This loss function softly penalizes predictions where the start or end of the bout is slightly incorrect. On the other hand, since the loss is normalized by the bout duration, it effectively counts the number of incorrectly predicted bouts and, unlike a per-frame loss, long actions are not deemed to be more important than short ones.

Inference: Given a score function, $f(x, y)$, and an input x , the optimal segmentation can be found by solving $\hat{y} = \arg \max_y f(x, y)$. Again, similarly to the learning phase, searching over all possible segmentations is intractable but we can solve for y using dynamic programming.

Semi-structured SVM

This approach is a hybrid of the sliding window SVM and the structured SVM; its inference algorithm optimizes over possible segmentations of a sequence, using dynamic programming, but the classifiers are trained using a linear SVM on fixed bouts from the training set, similar to [HLD11].

Training: We extract bout features from the positive bouts, $\{(s^i, e^i)\}_i$, for each sequence x_i in the training set, and from randomly sampled negative bouts. We consider a bout as negative if its intersection with a positive bout is less than half of their union, so that large intervals containing positive bouts and small intervals that are parts of a positive bout are still considered as negatives. Inference involves considering all possible intervals of any duration as potential action bouts, however training on all such possible intervals would be intractable. Instead, we generate a limited number of randomly sampled negatives and use a bootstrapping training process that gradually adds useful negative samples. At each iteration we train a classifier on the current training data, run inference with the learnt classifier, and add falsely detected positives to the set of negative training samples - repeating until no new false positives are detected.

Inference: Here the goal is the same as in the structured SVM approach, to find the optimal segmentation of a new input sequence x , $\hat{y} = \arg \max_y f(x, y)$, but with a simpler score function: $f(x, y) = \sum_{(s^j, e^j, c^j) \in y} w_{c^j} \cdot \psi(x, s^j, e^j)$. Again, we solve this using dynamic programming. We speed up the inference by setting upper limits on the duration of an action, which we obtain from the training set.

3.6 Experiments and analysis

Measures

The performance measure used to compare algorithms should favor desirable predictions; in the case of action detection for behavior analysis it is important that there are few false hits and misses compared to the number of true action instances, which becomes difficult the more sparsely actions occur in the data. We have generated a synthetic ground truth sequence with five sporadic action classes, and two different prediction sequences, to demonstrate the difference between three common measures: a *confusion matrix*, *ROC curves*, and *precision-recall curves*. This comparison shows that precision-recall most effectively emphasizes the large performance discrepancy between the two predictions (see Figure 3.7).

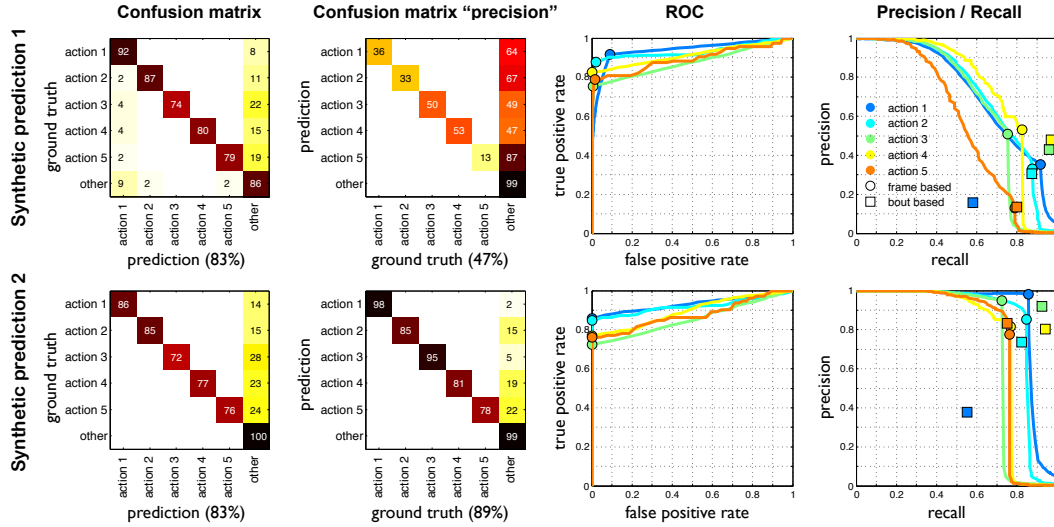


Figure 3.7: Confusion matrices and ROC are unreliable diagnostics for assessing experimental results. Each row shows the result of a different synthetic experiment. The confusion matrices (first column) and the ROC (third column) suggest that both experiments yield the same result and hide the large difference in the number of false detections. This fact is revealed by the “precision” confusion matrix (second column) and by the precision-recall curve (fourth column). The last column also shows how bout-wise and frame-wise measurements can differ.

Precision-recall curves, used for measuring detection performance for a single class, plot *precision* against *recall*, favoring minimum number of false positives and false negatives with respect to the number of positives. **ROC** curves are similar but instead of precision they plot the *false positive rate*, which places little emphasis on false positives when negatives take up vast majority of the frames. A **confusion matrix**, used in multi-class classification, is a square matrix whose entry (i, j) represents the fraction of ground truth instances of class i that are predicted as class j , and is commonly summarized by its diagonal mean. However, its diagonal effectively measures the recall of each class and fails to emphasize false positive instances which get absorbed into the grab-bag class *other*. To account for this, one must also look at the ‘dual’ confusion matrix, where entry (i, j) represents the fraction of predicted instances of class i that belong to class j according to ground truth, in which case the diagonal effectively measures the precision of each class. We conclude that “precision” and “recall” confusion matrices are good measures for multi-class detection problems, where classes are mutually exclusive, but for experiments such as ours, where classes overlap and false positives are expensive, precision-recall curves are the best performance measurement tools.

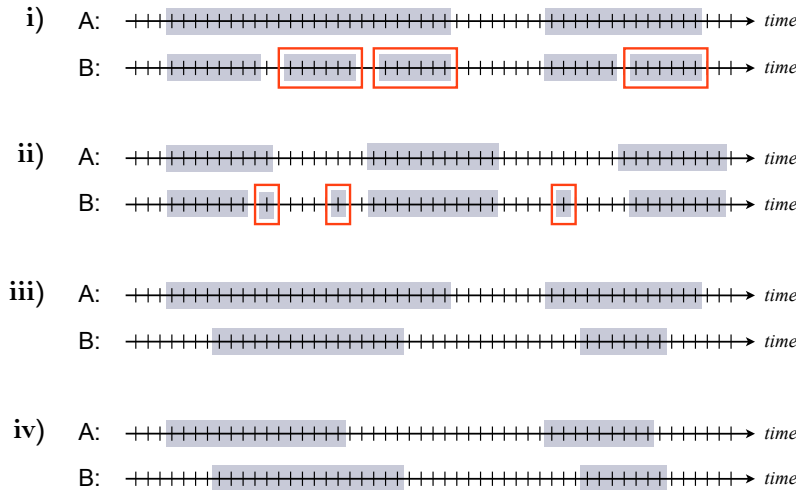


Figure 3.8: Examples of frame-bout performance discrepancies. Red squares denote missed/false detections, depending on whether A or B is ground truth. A lower bout than frame wise performance can be due to scenarios such as i) and ii), and lower frame than bout wise performance due to scenarios such as iii) and iv)

For behavior analysis, correctly counting the number of action instances is equally important as correctly measuring the duration spent in an action, hence we must also measure the **bout-wise performance**. To do that we use an overlap criteria, that deems a ground truth bout (s_g, e_g, b) and predicted bout (s_p, e_p, b) to match only if, $\frac{\min(e_g, e_p) - \max(s_g, s_p)}{\max(e_g, e_p) - \min(s_g, s_p)} > threshold$. If multiple bouts fit that criteria, we match the one with the highest ratio. Figure 3.7 shows that there can be large discrepancies between frame-wise and bout-wise performance. This is the case when predicted bouts are more fragmented than ground truth bouts, or when bouts are consistently predicted to be shorter than, or offset from, the ground truth (see Figure 3.8).

In order to rank different methods we combine precision and recall into a single value using the *F-score*, defined as $F_\beta = (1 + \beta^2) \cdot \frac{precision \cdot recall}{\beta^2 \cdot precision + recall}$, which for $\beta = 1$ represents the harmonic mean that favors balanced precision-recall combinations. To further combine bout-wise and frame-wise performance we define the *F*-score* as the harmonic mean of F1-frame and F1-bout.

Human performance

We trained novice annotators to learn to detect actions in the Fly-vs-Fly dataset, by showing them a subset of annotated movies, having them annotate another subset and providing them with feedback such that they could adjust their detection criteria. Once trained, they re-annotated a large portion of the test data, enough to give

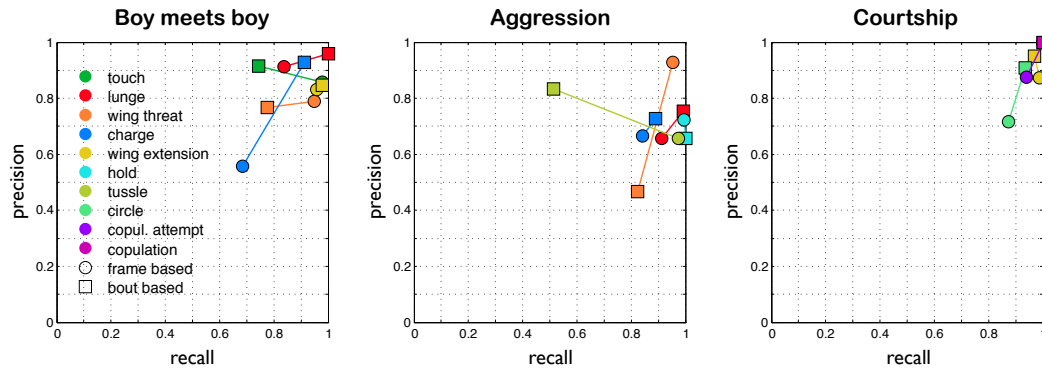


Figure 3.9: Human performance measured in terms of frame based (circles) and bout based (squares) precision-recall

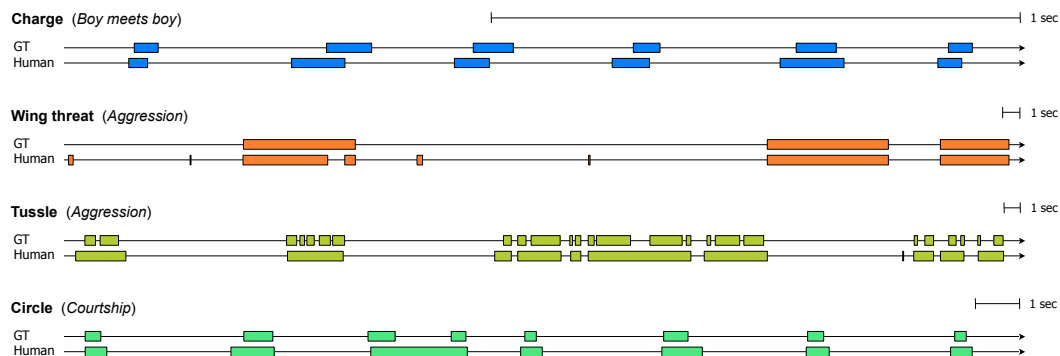


Figure 3.10: Segmentation samples of actions with high frame-bout performance discrepancy. Here GT refers to experts and Human to trained annotators.

an idea about the difficulty of detecting each action. Overall, the trained annotators achieved best performance on the Courtship sub-dataset, which they described as being easier to annotate than the other two sub-datasets, with actions seemingly less ambiguous. Figure 3.9 shows the bout- and frame wise precision-recall for each action in Fly-vs-Fly, and Figure 3.10 explains bout-frame performance discrepancies. The human performance is a good indicator for what to expect from automatic detection algorithms; we do not expect perfection, due to action ambiguity and imperfections in ground truth annotations, but ideally they should achieve at least as good a performance as humans.

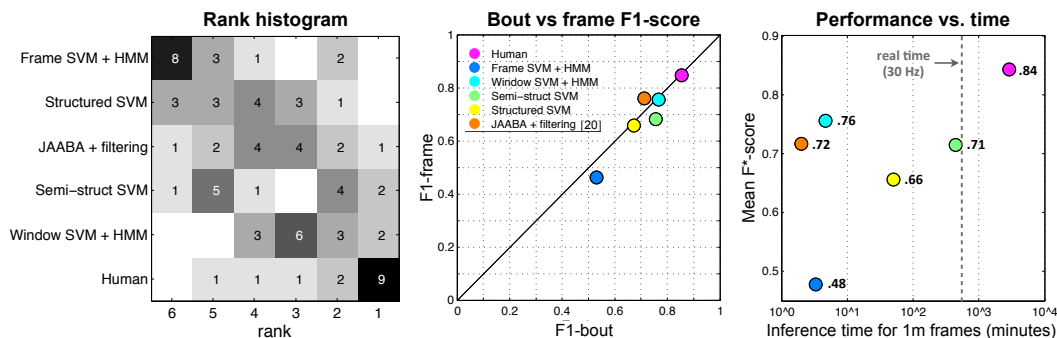


Figure 3.11: Method comparison on the Fly-vs-Fly dataset. *Left*: Histogram of method ranks over all actions, based on their F*-score, ordered by mean rank. *Center*: Comparison of F1-scores of each method, averaged over all actions. *Right*: F*-score of each method as a function of inference time.

Method comparison

Here we explore how a window based SVM compares to structured, and semi-structured SVMs, which we find very interesting as they all make use of linear classifiers and the same bout features, but differ in their training and inference procedures. In addition, we compare them with a frame based SVM to get a sense for how much bout features contribute to performance, and to JAABAs back-end [Kab+12], another window based detector, for comparison with methods currently deployed in action detection systems.

Each method’s free parameters were optimized using a subset of the training data for validation, and we found that HMM post processing improved the mean F*-score of the window- and frame based SVMs by 11% and 3% respectively. For comparison with JAABA we trained detectors by substituting their boosting classifier implementation into the learning and inference modules of our window based framework. JAABA as presented in [Kab+12] does not include post processing, but here we apply a box filter suggested on their project website for a fair bout-wise performance comparison, improving its mean F*-score by 6%.

To measure the performance of our action detectors, we computed bout- and frame-wise precision, recall and F1-scores, and the F*-score which can be used to rank the different methods. These measures, broken down for each behavior in Figures 3.15-3.17, show considerable variation in method rank depending on the action. Here we summarize the results in a detector rank histogram (Figure 3.11), which shows the number of times each detector achieved each rank and orders methods according to

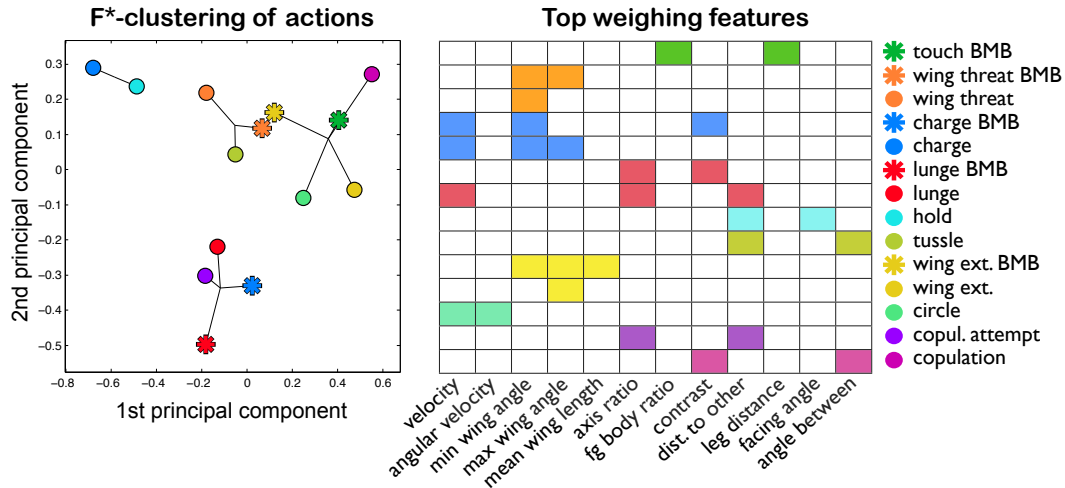


Figure 3.12: *Left*: clustering of actions based on F^* -score of all methods. *Right*: top weighted features determined by the trained SVM detectors.

their mean rank. For a finer resolution view of how the methods line up we show the mean F1-scores, averaged over all actions, and the mean F^* -score as a function of time it takes to run the detector on 1 million frames. This view mostly preserves the rank observed in the rank histogram, but it also shows that most methods cluster around 70% performance, apart from humans at 84% and frame based SVM at 48%. In addition, it shows that the window based methods perform slightly better than the structured output counterparts, in spite of being orders of magnitude faster.

These summary measures abstract away information about performance patterns between actions that may give insights into the different types of actions. To explore that, we cluster actions based on their F^* -score for each method, by applying principal component analysis to the F^* -matrix, and fitting k-means to the dimension-reduced matrix, splitting actions into 4 groups. Figure 3.12 shows that this clustering groups together lunge, charge, and copulation attempt, which all share the characteristic of being short and concise but poorly captured by the frame based detector, and, as one might expect, it groups actions (wing threat and wing extension) from different sub-datasets together. From the learnt detectors of the three different SVM approaches we found that the window based detector made most use of the bout statistics and histogram features, while structured ones used boundary dependent features to a similar extent, and that the top per-frame features used by all methods are those listed in Figure 3.12, showing that each feature is the highest contributing feature to at least one action.

Performance on CRIM13

Finally, to give a better idea of where these methods place within state of the art, we test the top ranked detector on the most recently published animal dataset, CRIM13, and compare our results with those presented in [Bur+12a]. Actions in CRIM13 are non-overlapping, and the detection problem is treated as multi-class. To make a similar comparison we convert our binary action detectors to a single multi-class detector by fitting them to an HMM with 13 states. By shifting output scores of individual binary classifiers, before converting them to posterior probabilities, we can trade off the performance of different classes. We obtain the optimal shift-parameters by greedily maximizing w.r.t. the diagonal mean of the “recall” confusion matrix, to match the measure used in [Bur+12a], and since we are interested in high precision-recall combination, we also optimize w.r.t. the mean F1-score of the “recall” and “precision” matrix diagonals. Figure 3.14 shows the confusion matrices produced for each of our optimization criteria, and Figure 3.13 shows our results compared with those presented in [Bur+12a]. We ran our algorithm only on tracking features (TF) provided with the CRIM13 dataset, obtaining performance just above the top results reported in [Bur+12a], which includes spatial-temporal features (STF), and 3.2% higher than their performance on tracking features alone. Optimizing w.r.t. F1-score results in approximately 6% F1-performance gain over the “recall” optimization.

Method	mean recall	mean F1
Boosting (TF) + Autocontext [1]	58.30%	-
Boosting (TF + STF) + Autocontext [1]	61.20%	-
Window SVM+HMM (recall shift)	61.66%	40.76%
Window SVM+HMM (F1 shift)	45.42%	47.22%

Figure 3.13: Comparison of the window based SVM to the methods used in [Bur+12a], showing performance on the CRIM13 test dataset.

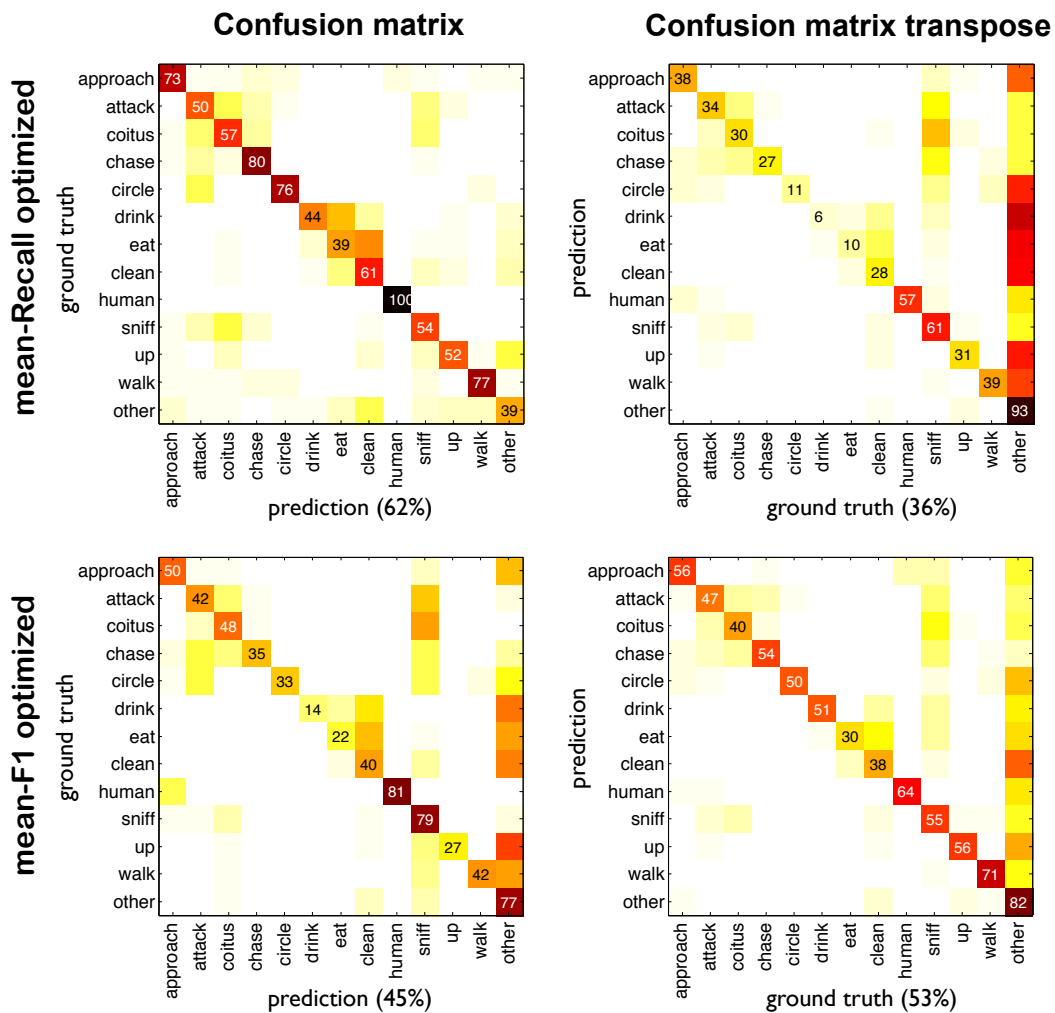


Figure 3.14: Confusion matrices for Window SVM + HMM on the CRIM13 test dataset. *Left*: performance optimized w.r.t. the diagonal mean of confusion matrix. *Right*: performance optimized w.r.t. “recall” and “precision” confusion matrices.

3.7 Discussion

We collected a large dataset of fruit fly videos that, with its natural and sporadic interactions and rich set of articulated pose features, fills a gap in existing datasets. We developed a framework for comparing action detection performance, showing that precision and recall are the best suited measures for evaluating detection algorithms, and that results should be reported both in terms of bout- and frame-wise performance. Using these measures, we showed that bout features highly improve performance upon frame-level features. We compared sliding window classifiers to the more sophisticated structured output detectors, and found that window based classifiers outperformed their structured counterparts, despite having much lower time complexity. This was surprising to us as the structured output methods allow for elastic sized windows which should better capture structure within bouts. A caveat is that the more complex actions in our dataset have low duration variation, therefore fixed sized window classifiers with good bout features may suffice. Our results also show (Figures 3.15-3.17) that the structured output methods suffer from over-segmenting long bouts of actions that do not have much structure, which leads to a lower bout-wise performance. We believe this may be overcome by incorporating higher order Markov terms in the score function, and will explore that in future work. In our experiments, the top performing algorithm, window SVM+HMM, reached 90% of human performance (76% compared to 84% F*-score) and matches the performance of the best published method on CRIM13.

This performance does not come at a low cost, each of the methods presented require a large number of annotated training data which can be difficult to obtain. In addition, annotators are not always in agreement with one another, especially in the cases where actions look ambiguous. Training an action detector on ambiguous actions may negatively affect the performance of the algorithm. In the next chapter we address both of these problems, we propose an algorithm that requires significantly fewer labeled data and can be trained on partially labeled sequences.

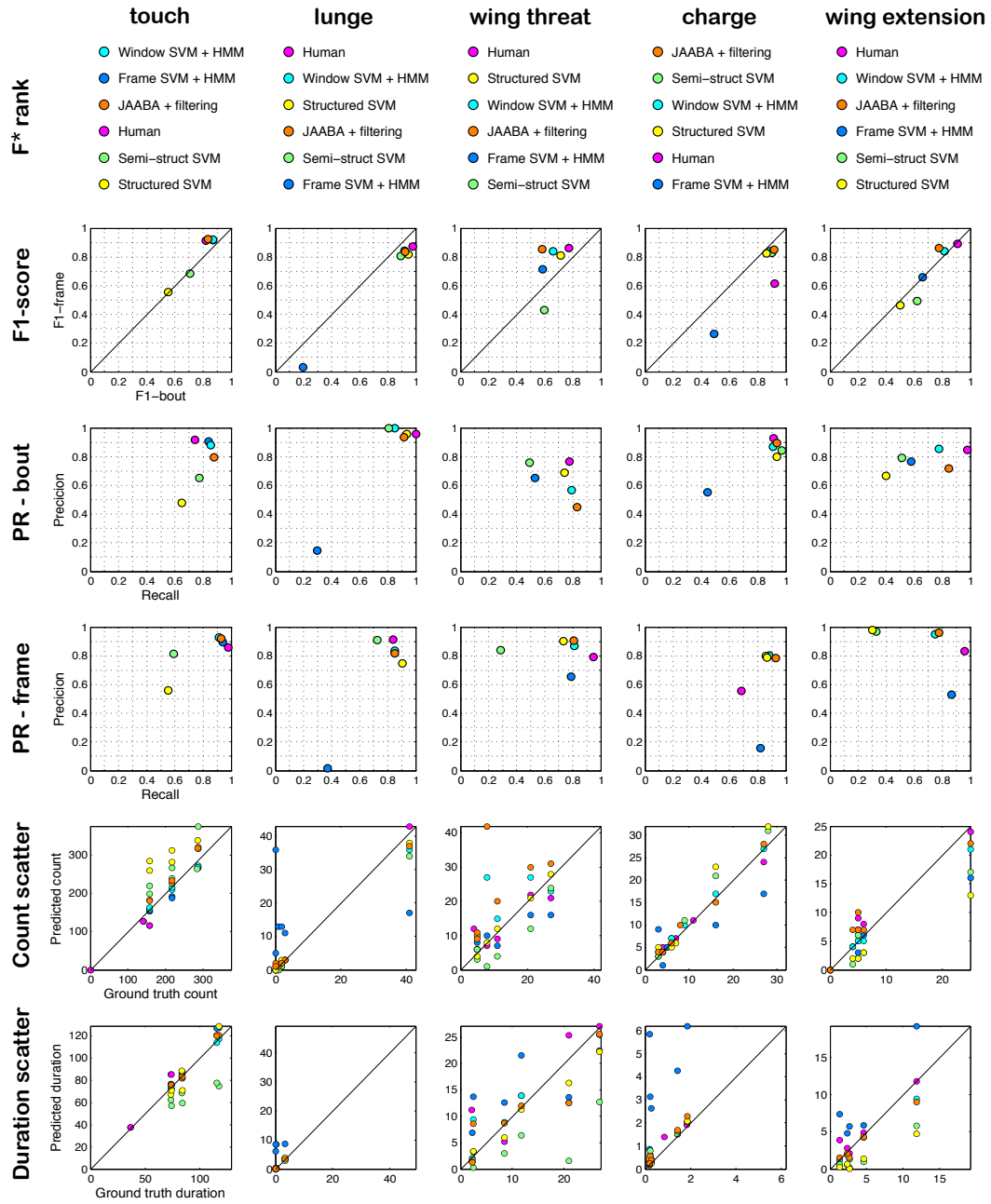


Figure 3.15: Results on Boy meets boy

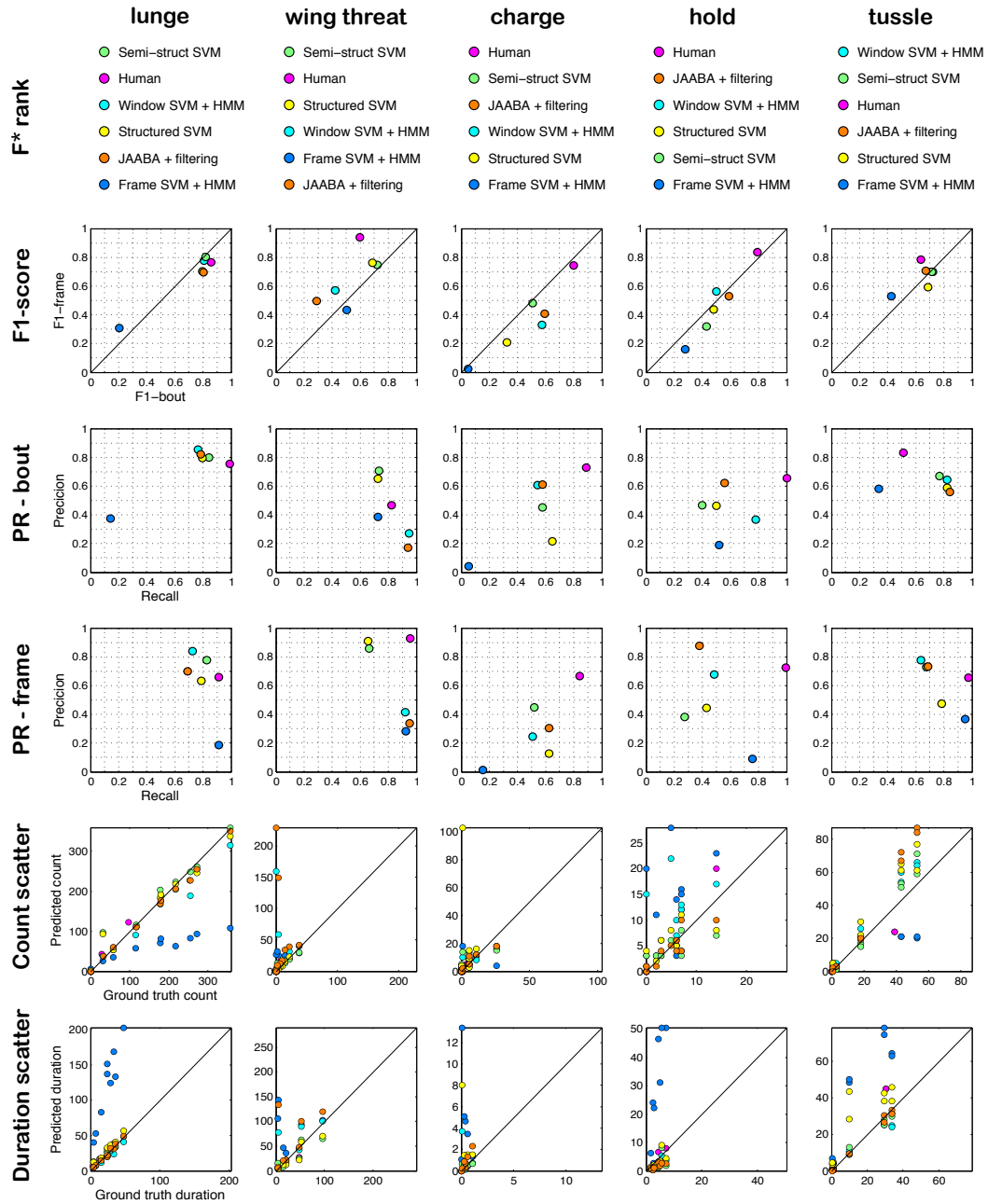


Figure 3.16: Results on Aggression

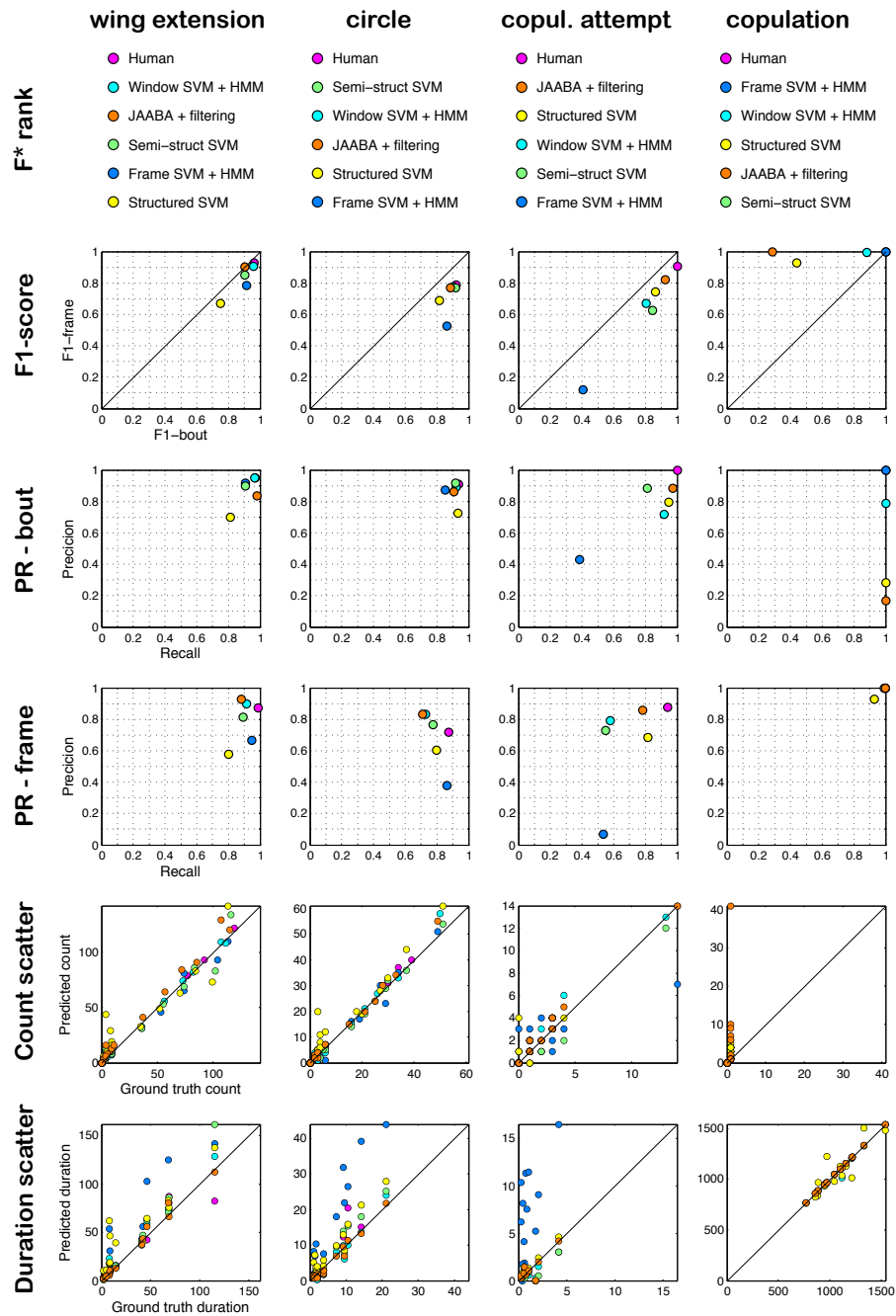


Figure 3.17: Results on Courtship

Chapter 4

BEHAVIOR MODELING

In this chapter we address the problem of detecting and classifying actions in sequential data, given an abundance of unlabeled sequences and a small set of labeled actions. Supervised learning is a powerful tool for learning action classifiers from expert-labeled examples [Jhu+10b; Bur+12b; Kab+13; Eyj+14]. However, it has two drawbacks. First, it requires a lot of training labels which involves time consuming and painstaking annotation. Second, behavior measurement is limited to actions that a human can perceive and believes to be important. We propose a framework that takes advantage of both labeled and unlabeled sequences; it simultaneously learns to predict future motion and detect actions, allowing the system to learn from fewer expert labels and discover unbiased behavior representations.

The framework models the sensory-motor relationship of an agent, predicting motion based on its sensory input and motion history. It can be used to simulate an agent by iteratively feeding motion predictions as input to the network and updating sensory inputs accordingly. A model that can simulate realistic behavior has learnt to emulate the generative control laws underlying behavior, which could be a useful tool for behavior analysis [Sim96; Bra84].

Our model is constructed with the goal that it will learn to represent and discover behaviors at different semantic scales, offering an unbiased way of measuring behavior with minimal human input. Recent work by [Ber+14] and [Wil+15] shows promising results towards unsupervised behavior representation. Compared to their work our framework offers three advantages. Our model learns a **hierarchical** embedding of behavior, can be trained **semi-supervised** to learn specific behaviors of interest, and our sensory-motor representation enables the model to learn **interactive** behavior of an agent with other agents and with its environment.

Our experiments focus mainly on the behavior of fruit flies, *Drosophila Melanogaster*, a popular model organism for the study of behavior [SK09]. To explore the generality of our approach we also test our model on online handwriting data, an interesting human behavior that produces two dimensional trajectories.

To summarize our contributions:

- 1) We propose a framework that simultaneously models the sensory-motor relationship of an agent and classifies its actions, and can be trained with partially labeled sequences.
- 2) We show that motion prediction is a good auxiliary task for action classification, especially when training labels are scarce.
- 3) We show that simulated motion trajectories resemble trajectories from the data domain and can be manipulated by activating discriminative cell units.
- 4) We show that the network learns to represent high level information, such as gender or identity, at higher levels of the network and low level information, such as velocity, at lower levels.
- 5) We test our framework on the spontaneous and sporadic behavior of fruit flies, and the intentional and structured behavior of handwriting.

4.1 Background

Hidden Markov models (HMMs) have been extensively used for sequence classification. The motivating assumption for HMMs is that there exists a process that transitions with some probability between discrete states, each of which emits observations according to some distribution, and the objective is to learn these functions given a sequence of observations and states. This model is limited in that its transition functions are linear, state space is discrete, and emission distribution is generally assumed to be Gaussian, although generalizations of the model that fall under the category of dynamic Bayesian networks are more expressive [Mur02].

Recurrent neural networks (RNNs) have recently been shown to be extremely successful in classifying time series data, especially with the popularization of long short term memory cells [HS97], in applications such as speech recognition [GMH13]. RNNs have also been used for generative sequence prediction of handwriting [Gra13] as well as speech synthesis [Chu+15].

Imitation learning involves learning to map a state to an action, from demonstrated sequences of actions. This is a supervised learning technique which, when implemented as an RNN, can be trained via backpropagation using action-error computed at every time step. The problem with this approach is that the domain of states that an agent is trained on consists only of states that the demonstrators encounter, and when an agent makes a mistake it finds itself in a situation never experienced during training. Reinforcement learning handles this by letting an agent

explore the domain using an action policy, and updating the policy based on a goal-specific penalty or reward which may be obtained after taking several actions. This exploration can be extremely expensive, and therefore it is common to precede reinforcement learning with imitation learning to start the agent off with a reasonable policy. This strategy is used in [Mni+15] where an agent is trained to play Atari games, and in [Sil+16] for mastering the game of GO.

Autoencoders [RHW86] have been used in semi-supervised classification to pre-train a network on an auxiliary task, such as denoising, to prevent overfitting on a small number of labeled data [Bal12]. Recent work in this area [Ras+15] proposes to train on the primary and auxiliary task concurrently and using lateral connections [Val15] between encoding and decoding layers to allow higher layers of the network to focus on high level features.

Our framework takes inspiration from each of the works described here.

4.2 Model

Our model is a recurrent neural network, with long short term memory, that simultaneously classifies actions and predicts future motion of *agents* (insects, animals, and humans). Rather than actions being a function of the recurrent state, as is common practice, our model embeds actions in recurrent state units. This way the recurrent function encodes action transition probabilities and motion prediction is a direct function of actions, similar to an HMM. The network takes as input an agent's motion and sensory input at every time step, and outputs the agent's next move according to a policy, which is effectively learnt via imitation learning. Similar to autoencoders, our model has a discriminative path, used to embed high level information, and a generative path used to reconstruct the input domain, in our case filling in the future motion. Each discriminative recurrent cell is fully connected with its corresponding generative cell, allowing higher level states to represent higher level information, similar to the idea of Ladder networks [Val15].

Architecture

The model can be thought of as two parallel recurrent networks: The *discriminative* network takes as input an agent's motion, x , and environmental sensory input, v , and propagates them up through its hidden states which encode high level information, including action labels, y . The *generative* network decodes the states of the discriminative network, propagating information down to predict the agent's motion at the next time step, \hat{x} . The two networks have the same number of lay-

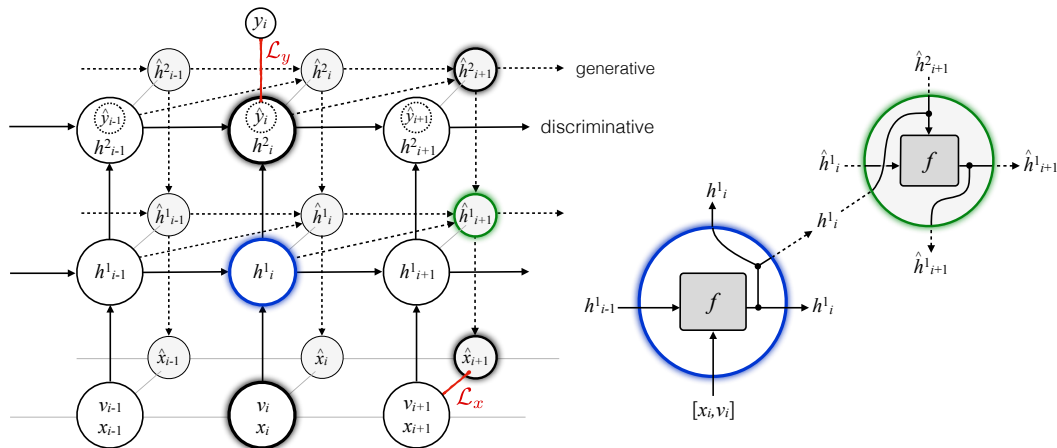


Figure 4.1: **Left:** A 3D depiction of our network unrolled for 3 timesteps. The highlighted cells show the path from an input through a classification cell to a motion prediction output. During training, motion prediction loss \mathcal{L}_x is computed at every timestep, and classification loss \mathcal{L}_y is computed only at frames for which labels are provided. The diagonal connections between discriminative and generative cells enable higher levels of the network to represent high level information. Vector v represents agent’s sensory input, x its motion, h its internal state, and y labeled actions. **Right:** A zoom in on the blue and green cells showing the recurrent state (horizontal arrows) and inputs to the recurrent cell function f . Merging of arrows represents vector concatenation, and branching vector duplication.

ers and are connected diagonally at each layer such that the information encoded in the hidden units of the discriminative network is propagated to the corresponding layer of the generative network at the next time step. Intuitively, these can be thought of as skip connections or “shortcuts” which let low level motion information propagate directly through lower levels of the network, leaving higher levels of the network free to represent high level phenomena, such as goals or individual characteristics. Our experiments confirm this intuition. The model can be trained without any action labels, in which case the hidden state may be used to discover high level information about the data, or with action labels for a subset of the data, in which case each action is assigned to a hidden state unit and will thus contribute to subsequent motion prediction and action classification.

The flow of information through the network and the cost associated with its classification and prediction is expressed by the following equations:

Discriminative	Generative	Cost
$h_i^1 = f([x_i, v_i], h_{i-1}^1)$	$\hat{h}_{i+1}^L = f(h_i^L, \hat{h}_i^L)$	$C_y = \sum_{i=1}^T \mathcal{L}_y(y_i, \hat{y}_i)$
$h_i^l = f(h_i^{l-1}, h_{i-1}^l)$	$\hat{h}_{i+1}^l = f([\hat{h}_{i+1}^{l+1}, h_i^l], \hat{h}_i^l)$	$C_x = \sum_{i=1}^T \mathcal{L}_x(x_{i+1}, \hat{x}_{i+1})$
$\hat{y}_i = (h_i^L(1 : N) + 1)/2$	$\hat{x}_{i+1} = g(\hat{h}_{i+1}^1)$	$C = \lambda C_y + (1 - \lambda)C_x$

where f is a recurrent cell function, g is a transformation, \mathcal{L}_y computes classification loss (on frames for which labels are provided) and \mathcal{L}_x computes motion prediction loss. The total cost, C , combines the misclassification cost, C_y , and misprediction cost, C_x , using λ to trade off the two. N is the number of labeled action classes, L the number of levels, T the number of frames, l is the layer index and i the frame index. The first N units of state h^L are forced to be classification units, they are scaled from $[-1 \ 1]$ to $[0 \ 1]$ (assuming f 's activation function is tanh) and assigned to \hat{y} .

The model is presented as part of a general framework where f , g , and number of levels/units are architectural choices to be optimized for each dataset. For our experiments we found that 2-3 levels of recurrent cells with 100-200 units worked well, with f as a gated recurrent unit (GRU) cell [Cho+14] and g as linear transformation. The choice of loss functions depends on the target type; sigmoid cross entropy for multitask classification (where actions can co-occur), softmax cross entropy for multiclass classification (where actions are mutually exclusive), and sum of squared differences for regression (where outputs are real valued). The optimal value for λ depends both on the output domain of \mathcal{L}_y and \mathcal{L}_x and whether the primary goal is classification or simulation.

Multimodal prediction

Evidence suggests that animal behavior is nondeterministic [Rob+16]; thus, motion prediction may be better represented as a probability distribution than a function. When future motion is multimodal, the best regression model will pick the average motion of the different modes which may not lie within any of the actual modes (visualized in supplementary material)¹. This observation has been made by others in the context of modeling real-valued sequences with RNNs, [Gra13] model the output of an RNN as a Gaussian mixture model and [Chu+15] additionally model the hidden recurrent states as random variables. We take a nonparametric approach,

¹www.vision.caltech.edu/~eeyjof/fs/behavior_modeling

making no assumption about the shape of the distribution. We discretize motion into bins and treat the task of predicting future motion as independent multiclass classification problems for each motion feature, which results in a probability distribution over all bins for each dimension. More concretely, each dimension of x is assigned n bins and the target for \hat{x}_{i+1} becomes the binned version of x_{i+1} , denoted as \tilde{x}_{i+1} , which has exactly one nonzero entry for each dimension of x . The prediction \hat{x}_{i+1} then becomes a discrete distribution over the bins for each feature dimension and the motion prediction loss becomes $\mathcal{L}_x(x_{i+1}, \hat{x}_{i+1}) = \sum_d(\text{crossentropy}(\tilde{x}_{i+1}, \hat{x}_{i+1}))$, as opposed to the Euclidean distance in the case when x is a real valued vector. The number of bins determines the granularity of the motor control; a greater number of bins means more precise motion control but is also more expensive to train.

Training details

In order to efficiently train on long temporal sequences that contain sporadic actions we introduce the two following ideas to the training routine:

Maintaining temporal dependency Recurrent neural networks are in theory able to pass information from the beginning of a sequence to any subsequent time step, but in practice, propagating a loss term several frames backwards is expensive and results in vanishing gradients. This is generally handled by splitting longer sequences into sub-sequences of length l , such that gradients are propagated back by at most l time steps. When processed sequentially, this approach can maintain temporal dependency by initializing hidden states of subsequences with the final state of preceding sub-sequences, however, for computational efficiency it is common practice to process samples in batches and in stochastic order which makes that impossible. One way around this is to set the initial state of each sub sequences to 0, however, doing that breaks the temporal dependency beyond l time steps. Instead, we approximate the true initial state of each sub-sequence by setting it to be the latest computed state for that time step (computed at the previous epoch). The approximate state therefore corresponds to a previous version of the model, but as the model converges so does this approximation. This means that although the loss is only propagated back by l frames, the model can make classifications and predictions based on events that occurred more than l frames ago.

Importance sampling When analyzing behavior of animals over long durations of time it is often the case that behaviors of interest are temporally sparse, so the datasets tend to be extremely imbalanced with the negative class taking up majority

of the frames. To prevent the negative class from dominating the cost, one may choose to train on a subset of the negative samples or on repeated samples from the rare classes, to artificially balance the data. Our approach is similar to this idea, but rather than setting a fixed sample size for each class and randomly selecting samples from it, we dynamically update the class- and sample probabilities after each epoch, such that classes with low recall are sampled more often, and misclassified frames are more likely to be sampled for each class. This allows us to effectively optimize the precision and recall of each class, which cannot be directly done using the cost function as precision depends on the total number of predicted positives.

4.3 Applications

Classification

When the goal is to classify actions of an agent, the generative part of the network can be discarded and inference done by sequentially applying the discriminative equations from Section 4.2 to the input sequence. The output is a soft classification for every class at every frame, with values in the range $[0, 1]$. To obtain hard classification the output is thresholded (in the case of multitask classification) or the argmax over all classes selected (in the case of multiclass classification).

In our experiments we make two adjustments to improve classification performance.

1) The recurrent function between classification units should ensure smoothness in the labels, but for further smoothness we filter the raw output with a linear kernel before making the hard classification. 2) The network can in theory keep track of inputs from previous frames, but has no information about the future. We trained a window variant of the our model, where $h_i^1 = f_{h^1}([x_{i-r}, \dots, x_{i+r}, v_{i-r}, \dots, v_{i+r}], h_{i-1}^1)$ and the predicted future motion is $\hat{x}_{i+r+1} = g_{\hat{x}}(\hat{h}_{i+1}^1)$, for a window with radius r .

Simulation

Given a model that can predict an agent’s future motion from its current state, a virtual agent can be simulated by iteratively feeding predicted motion \hat{x}_{i+1} as input x_{i+1} to the network. We pick a bin by sampling from the distribution given by \hat{x}_{i+1} and assign a real value to x_{i+1} by sampling uniformly from the selected bin. An agent’s perception of the environment depends on the agent’s location, and therefore sensory features v_{i+1} must be updated for each forward simulation step to correspond to the agent’s perspective at time $i + 1$. When simulating multiple agents that interact with one another, each agent is moved according to its x_{i+1} and then v_{i+1} is computed for each agent based on the new configuration of all agents.

Discovery

Whether the model is trained unsupervised or supervised, its hidden units may be used to discover phenomena not explicitly encoded in the model. One way to do this is to visualize the data responsible for the activation of individual units, or by activating individual units during simulation and observing its effect on the agent’s behavior. Using synthetically generated data, our experiments revealed individual neurons encoding generative control laws (see Section 4.5).

High level phenomena may not be represented by a single neuron but encoded by a group of neurons. To look for such patterns we map the output of all units from a single recurrent cell to low dimensional representation, using t-distributed stochastic neighbor embedding (tSNE) [MH08] and examine data points within a cluster. We tested this by training a network fully unsupervised, and visualizing high level phenomena, such as gender, individuality, and actions, in the low dimensional representation of individual recurrent cells, and found that different phenomena were grouped together at different levels of the network (see Section 4.5).

4.4 Data

Our framework is *agent centric*, it models the behavior of each agent individually based on how it moves and senses its surroundings, including other agents. It is applicable to any data that can be represented in terms of motor control (e.g. joystick controller) and sensory input that captures context from the environment (e.g. 1st person camera). We test our model on two types of data, fly behavior and online handwriting. Both can be thought of as a type of behavior represented in the form of trajectories, but the two are complementary. First, flies behave spontaneously, performing actions of interest sporadically and in response to their environment, while handwritten text is intentional and highly structured. Second, handwriting varies significantly between different writers in terms of size, speed, slant, and proportions, while inter-fly variation is relatively small. We use 4 datasets for our experiments (listed below) with the aim to answer the following questions: 1) does motion prediction improve action classification, 2) can the model generate realistic simulations (does it learn the sensory-motor control), and 3) can the model discover novel behavioral phenomena?

Fly-vs-fly [Eyj+14] contains pairs of fruit flies engaging in 10 labeled courtship- and aggressive behaviors. We include this dataset in our experiments to see how our model compares with our work described in the previous chapter.

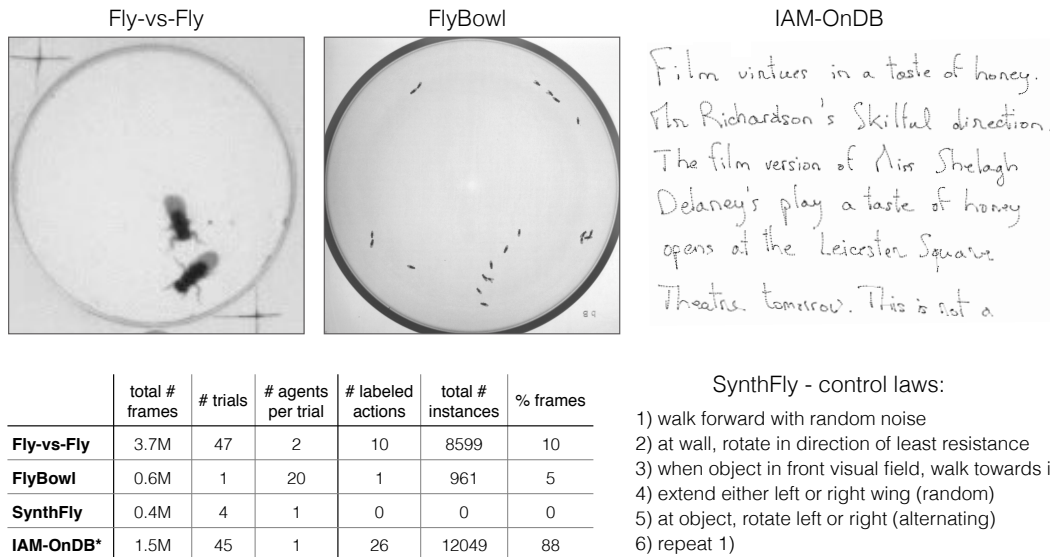


Figure 4.2: Snapshots from the three labeled datasets used for our evaluation and a list of control laws used to generate synthetic fly trajectories. The table summarizes the statistics of each experimental dataset, where **total # frames** sums over all **trials** (videos / text documents) within an experiment and **agents** within a trial, **total # instances** sums over all action classes, and **% frames** is the percent of frames in labeled sequences containing actions of interest. IAM-OnDB* is a subset of IAM-OnDB with additional annotations for 10 of its trials.

FlyBowl is a video of 10 male and 10 female fruit flies interacting and is labeled with male wing extensions which is part of their courtship behavior. With this dataset we were particularly interested in whether our model could simulate a virtual fly in a complex, dynamic environment.

SynthFly is a synthetic dataset containing a single fly moving inside of a rectangular chamber with a stationary object located in the center. The fly is synthesized to move according to the control laws listed in Figure 4.2. The purpose of this dataset is to test whether our model could learn generative control rules, particularly ones that enforce non-deterministic behavior (see laws 4 and 5).

IAM-OnDB [LB05] contains handwritten text from 195 different writers, acquired using a smart whiteboard that records a list of (x, y) coordinates for each pen stroke. The data is weakly labeled, with each sequence separated into short lines of transcribed text. For consistency with our framework we hand annotated strokes of 10 writers, marking the start and end of the 26 lower case characters, which we use along with data from 35 unlabeled writers for our experiments.

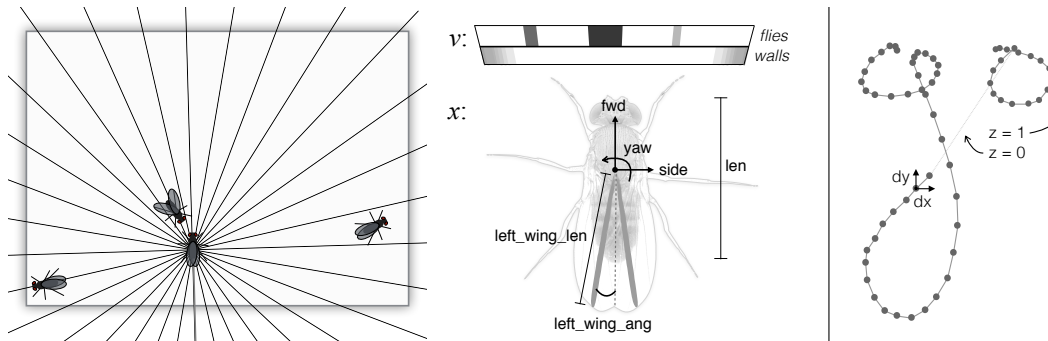


Figure 4.3: **Left:** Sensory input v for fruit flies represents how a fly sees other flies and chamber walls, their motor control x lets them move their body along 8 dimensions (incl. `right_wing_ang/len`). **Right:** Motor control x for handwriting is represented as vector (dx, dy) along with binary stroke visibility z (pen on/off whiteboard).

Fly representation: Motor control features, x , describe the locomotion of a fly. The flies are tracked from video using FlyTracker² and from the tracked fly poses we extract motion features represented in the fly’s frame of reference. The 8 motion features, displayed on top of the close-up fly³ in Figure 4.3, are designed such that they can animate virtual fly agents. Sensory input features, v , are inspired by a fly’s compound eye which consist of 750 compactly aligned ommatidia. Approximating its vision as a one dimensional 360° view, we place 72 5° circular sectors around a fly agent, aligned with its orientation, and project flies that overlap with a sector onto its artificial retina with intensity inversely proportional to their distance to the agent. Thus, flies close to the agent yield high intensity in several pixels and flies that are far away take up few pixels with low intensity (compare scene in Figure 4.3 with v sensed by the agent). We represent chamber walls similarly, projecting them onto a separate channel decreasing intensity exponentially with distance to the agent. This representation is invariant of the shape of the chamber and the number of flies present in the chamber.

In order to compare our model with methods presented in [Eyj+14], independently of feature representation, we use the 36 features provided with the Fly-vs-Fly dataset. We assign the first 8 dimensions (describing fly’s motion) as motor control x , and the remaining features (describing fly’s position relative to the other fly, and feature derivatives) as sensory input v .

²www.vision.caltech.edu/Tools/FlyTracker

³Original photograph from gompel.org/drosophilidae

Handwriting representation: We represent the motor control, x , as (dx, dy, z) where dx and dy are the x and y displacements from the previous pen recording and z is a binary variable denoting segment visibility. We normalize dx and dy for each writer, providing invariance to writing speed, but character size (number of points per character), slant, and other variations are not explicitly accounted for. As handwriting is not influenced by a changing environment, but rather a function of the internal state and current motion of the writer, we leave the sensory input, v , empty.

4.5 Experiments and analysis

We evaluate our framework on three objectives: classification, simulation, and discovery. For classification we show the benefit of motion prediction as an auxiliary task, compare our performance on Fly-vs-Fly with previous work, and analyze the performance on IAM-OnDB. We qualitatively show that simulation results for fly behavior and handwriting look convincing, and that the model is able to learn control laws used to generate the SynthFly dataset. For discovery we show that hidden states of the model, trained only to predict motion (without any action labels), cleanly capture high level phenomena that affect behavior, such as fly gender and writer identity.

Model details: We trained a separate model for each dataset, using a sequence length of 50, a batch size of 20, and 51 bins per dimension for motion prediction. For fly behavior data we used 2 levels of GRU cells (4 cells total) of 100 units each, and for handwriting we used 3 levels of GRU cells (6 cells total) of 200 units each. Parameters were determined using a rough parameter sweep on a subset of the training data. Our model is implemented in Tensorflow [Mar+15].

Classification

Action labeling involves recording the start frame, end frame, and class label, of each action interval, which we refer to as a *bout*. From a sequence of frame-wise classifications, consecutive frames of the same class prediction are consolidated into a single bout. To measure both duration and counting accuracy we use the performance measures described in [Eyj+14], namely the F1 score (harmonic mean of precision and recall), on a per-frame and per-bout level. Bout-wise precision and recall is computed by assigning predicted bouts to ground truth bouts one-to-one, maximizing intersection over overlap. F^* is the harmonic mean of the F1-frame and F1-bout scores.

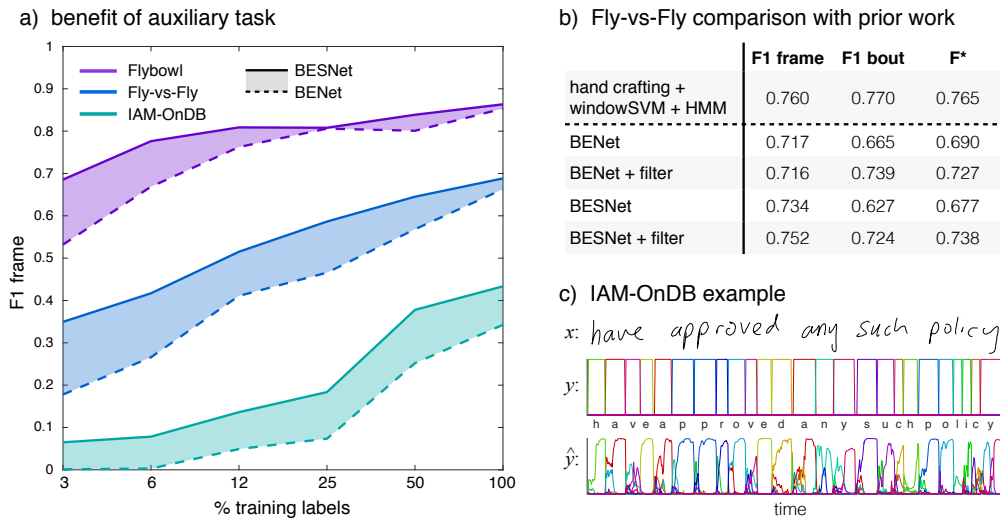


Figure 4.4: **a)** Performance of model trained with (solid, BESNet) and without (dashed, BENet) motion prediction, showing that BESNet requires significantly fewer labels to match the performance of BENet. **b)** Our model reaches performance competitive with [Eyj+14], without handcrafting or context from future frames. **c)** Input x , label y , and classification score \hat{y} , colored according to character label, showing high confusion at the beginning of characters, partly explaining the lower F1-frame performance on IAM-OnDB.

Our goal for classification is to reduce the number of training labels without loss in performance. To measure the benefit of motion prediction as an auxiliary task we compare our model, which we will refer to as Behavior Embedding Sensory-motor Network (BESNet), with our model without its generative part (similar to a standard RNN but with action labels embedded in hidden states, shown in Figure 4.6), referred to as Behavior Embedding Network (BENet). We trained both models on each dataset using 3-100% of available labels. As BESNet is trained to predict future motion it makes use of unlabeled sequences during training whereas BENet does not. Figure 4.4 a) shows the frame-wise F1 score for each of the 36 trained models (3 datasets, 6 label fractions, 2 model types), averaged over all action classes in a dataset. This experiment shows that motion prediction as an auxiliary task significantly improves classification performance, especially when labels are scarce.

In Figure 4.4 b) we compare the performance of our network with the best performing method on **Fly-vs-Fly**, a window based support vector machine (SVM) that uses hand crafted window features and fits an HMM to the output for smoother classification – outperforming sophisticated methods such as structured SVM. For this comparison we used the features published with the dataset as described in

Section 4.4. Although recurrent networks implicitly enable smooth classification, different actions require different levels of smoothness. To avoid over segmentation of action intervals, we smooth the output of our network by applying a flat filter, of size equal to 10% of the mean duration of each class. Our results show that filtering significantly improves the bout-wise performance and that our performance on the Fly-vs-Fly test set is comparable with that of [Eyj+14], using no handcrafting and no context of future frames (apart from smoothing).

We applied the same type of filtering to the classification output of **IAM-OnDB** as we did for Fly-vs-Fly and obtained an F1-(frame, bout) of (0.445, 0.585) averaged over all classes, and (0.567, 0.690) averaged over all instances (weighted average of classes). Figure 4.4 c) demonstrates that at the beginning of some characters there tends to be more confusion in \hat{y} than towards the end, which is unsurprising as the beginning of these characters looks approximately the same.

Architecture exploration: To explore the effect of the diagonal connections on the classification performance, we trained our model on IAM-OnDB, with and without diagonal connections, for 2, 3, and 4 levels of GRU cells, with 10, 20, 50, 100, and 200 units. Results on the validation set, shown in Figure 4.5 showed that the objective cost was lower for diagonal models, especially for models of higher levels, and that convergence was obtained in fewer iterations, especially for models with a lower number of units, supporting the use of diagonal connections between layers.

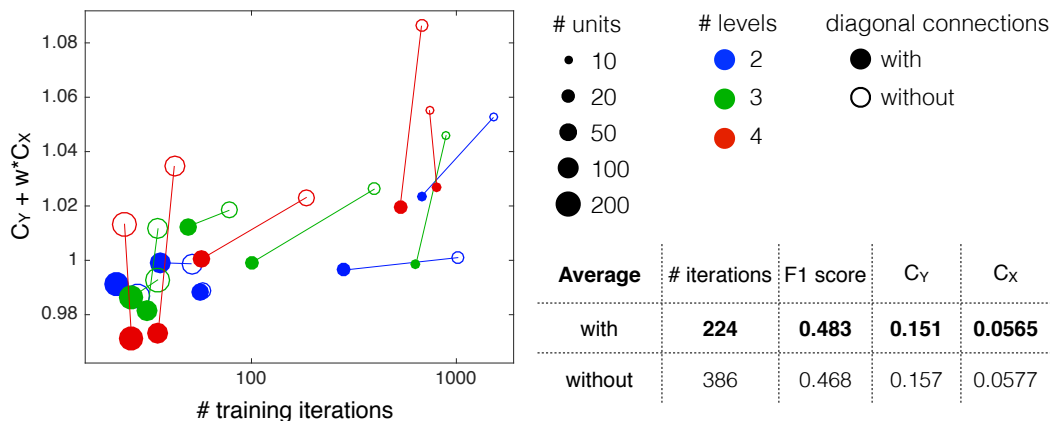


Figure 4.5: Effect of diagonal connections on validation cost, with/without diagonal connections, for varying number of layers and units per layer.

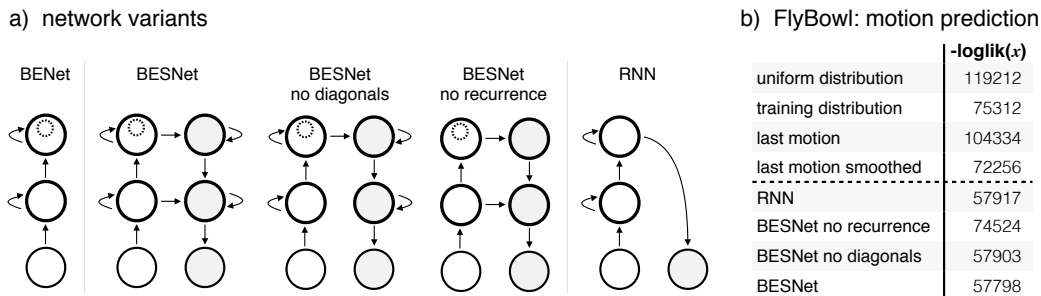


Figure 4.6: **a)** Network variants used in experiments (compare BESNet to highlighted cells in its unrolled visualization in Figure 4.1). **b)** 1-step motion prediction performance on FlyBowl testset, see text above for explanation.

Motion prediction

Before we look at simulation results, we quantitatively measure the accuracy of one-step predictions. We compute the log-likelihood of FlyBowl test sequences under the motion prediction model: $\log\text{lik}(x) = \sum_{i=1}^{T-1} \sum_{d=1}^8 \log(\tilde{x}_{i+1}^d \cdot \hat{x}_{i+1}^d)$, where \tilde{x}_{i+1}^d is the ground truth indicator vector for bins of motion dimension d , and \hat{x}_{i+1}^d is a probability distribution over the bins predicted by the model.

We compare our model with the following motion prediction policies: 1) uniform distribution over bins, 2) distribution over bins computed from training set, 3) constant motion policy that copies previous indicator vector as motion prediction, and 4) a smooth version of 3) filtered using an optimized Gaussian kernel. The results, shown in Figure 4.6, demonstrate that the recurrent models learn a significantly better policy. In addition, we compare variants of our model and a standard RNN within our framework (with the same sensory-motor representation, multimodal output, and GRU cells) which shows that recurrence is essential for good motion prediction and that diagonal connections provide a slight performance gain. In Section 4.5 we show the main benefit of the diagonals.

Simulation

One-step prediction performance does not clearly reveal whether a model has learnt the generative process underlying the training data. In order to get a better notion of that we look at simulations produced by the learnt models, which can be thought of as very long term predictions. As motion prediction is probabilistic, comparing long term predictions with ground truth becomes difficult as the domain of probable positions becomes exponentially large. Qualitative inspection, however, gives a good intuition about whether the simulated agent has learnt reasonable control laws.

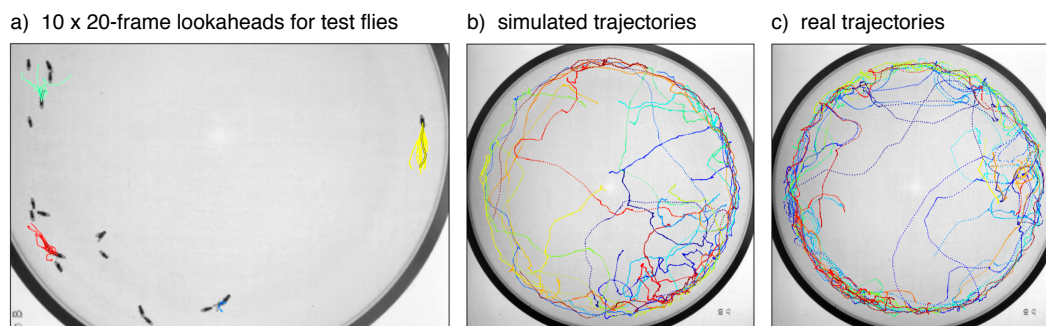


Figure 4.7: **a)** 10 x 20-frame lookaheads (simulations) for each test fly from its current location, demonstrating the non-deterministic nature of the motion prediction. Comparison of 1000-frame trajectories for simulated flies (**b)** and real flies (**c)** shows that the model has learnt a preference for staying near the boundary and to avoid walking through it.

While the underlying generative process for the motion of real flies is unknown, simulations from the model trained to imitate them suggest that the model has learnt a reasonable policy. During simulation we place no physical constraints on how the flies can move but our results show that simulated **FlyBowl** agents avoid collisions with the chamber walls and with other flies, and that agents are attracted to other flies and occasionally engage in courtship-like behavior. This is shown in Figure 4.7 and better visualized as video in supplementary material.

The Courtship subdataset of Fly-vs-Fly contains male and female fruit flies, and our model is trained only on the males. In the beginning of each video flies tend to be agitated, flying around the chamber and walking along the boundary, and eventually the male will court the female and copulate. The simulated agent seemed to also fly around the chamber, walk along the boundary, and extend its wing, as shown in Figure 4.8. When flying around the chamber the agent did sometimes exit the chamber, but interestingly, as it approached the walls its body length would decrease, as is the case when real flies rear up against the wall.

Simulated handwriting is easier to visualize in an image and we are used to recognizing the structure it should produce. Figure 4.9 shows that the model trained on **IAM-OnDB** produces character-like trajectories in word-like combinations. Note that handwriting is generated one (dx, dy, z) vector at a time, and each character is composed of roughly 20 such points on average. On the right hand side of Figure 4.9 we show that we can increase the generation of specific characters by activating their classification units (forcing their values to 1 and others to 0) during simulation.

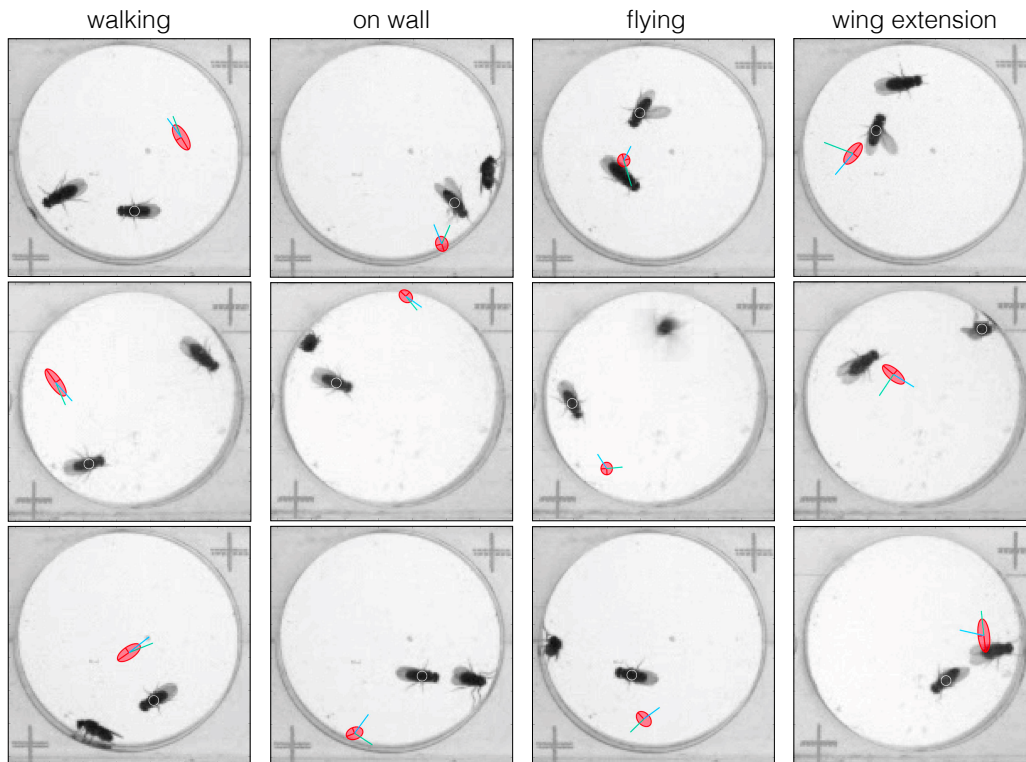


Figure 4.8: Frames from simulation using model trained unsupervised on Fly-vs-Fly (Courtship), showing examples of learnt behaviors.

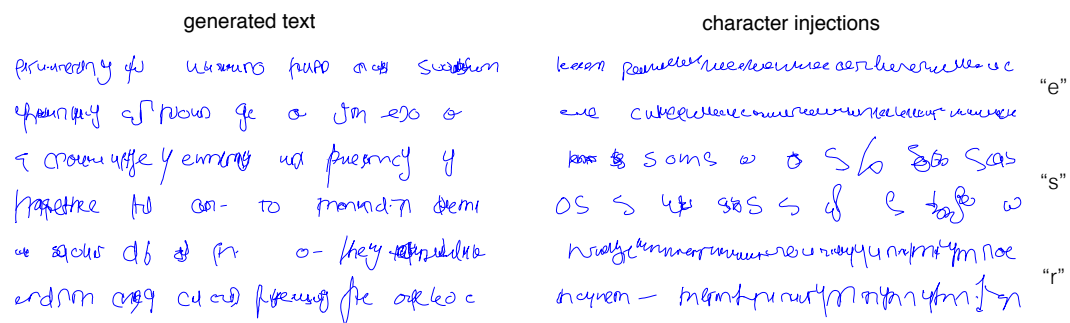


Figure 4.9: **Left:** Text generated by our model, one vector at a time (approximately 20 vectors per character). **Right:** Text generated by the same model while "activating" character classification units of the model during simulation, shown in two lines per character.

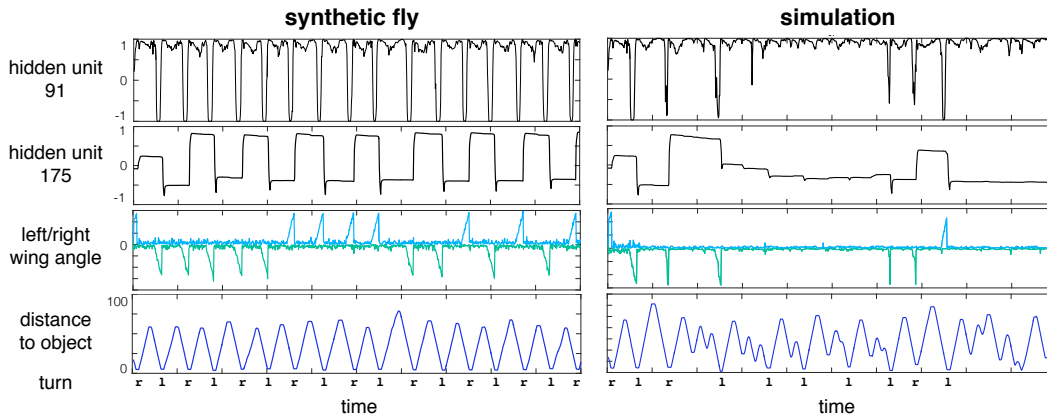


Figure 4.10: Comparison between synthetic fly (ground truth) and simulation by our model. The wing angles, distance to object, and left/right turn show the agent’s motion over time, and the two hidden units indicate that the model has learnt to represent control laws 4 (to extend left or right wing and random) and 5 (to alternate between left and right/right avoidance) used to generate the synthetic trajectories.

Figure 4.10 shows the output of two recurrent units of the **SynthFly** model that indicate that the model was able to learn control rules that were designed to ensure a multimodal motion prediction target. One unit fires in correlation with either left or right wing extension, and the other toggles between a negative and positive state as the agent turns left or right to avoid the object. In supplementary material we show a video of this simulation and compare it to a simulation from the model trained with deterministic motion prediction. This comparison clearly demonstrates the benefit of treating motion prediction as a distribution over bins, as the deterministic agent quickly becomes degenerate.

Discovery

We motivated the structure of our network, specifically the diagonal connections between discriminative and generative cells, with the intuition that it would allow higher levels of the network to better represent high level phenomena. To verify this we train models to only predict future motion, with **no classification target**, and visualize what the hidden states capture. We apply the model to $[x, v]$, obtaining hidden state vectors h^l and \hat{h}^l , $l \in \{1, \dots, L\}$, and prediction \hat{x} , map the data points (time steps of each fly/writer) from each state to 2 dimensions using t-distributed stochastic neighbor embedding (tSNE, [MH08]), and plot them in colors based on known phenomena.

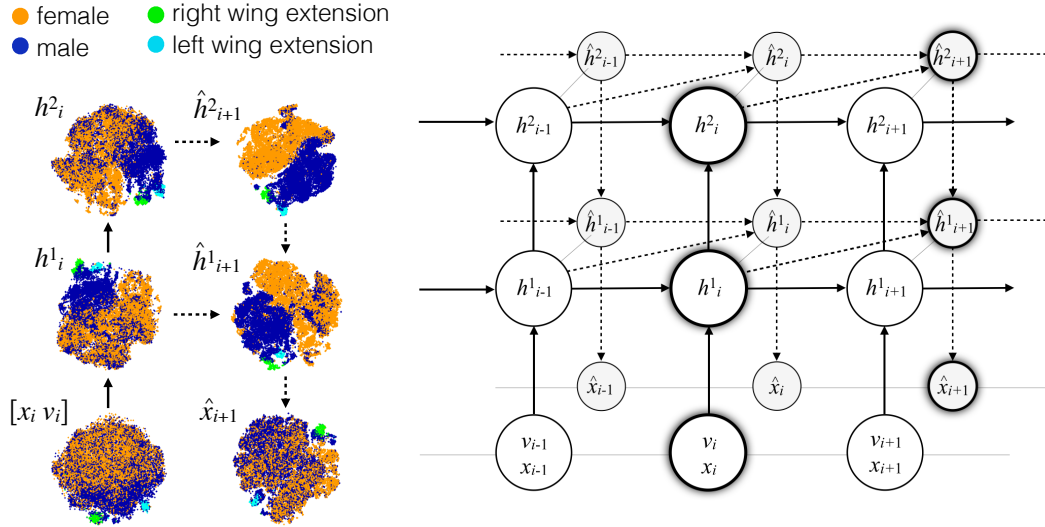


Figure 4.11: tSNE mapped input, output, and hidden state values of FlyBowl model (trained without any labels), colored by gender and male wing extension.

In Figure 4.12 we plot the data points of a 2 level ($L=2$) model trained on **IAM-OnDB** in this low dimensional embedding, color coded by to gender and left/right wing extension. This visualization shows that gender is very mixed in the input and output states but well separated in the top generative state, while lower level information such as wing extension is well represented at lower levels of the network.

We similarly visualize a 3 level model trained on **IAM-OnDB**, coloring data points according to three criteria: stroke length, character class, and writer identity. The results show that stroke length is well clustered at low levels but not at high levels, characters are best clustered at mid to top discriminative levels, and writer identity is extremely well clustered at the top generative level but not at low levels. We ran the same experiment for the model trained without diagonal connections (which without a classification target is effectively a standard RNN with 6 levels of GRU cells), which did not learn to represent writer identity in any of its hidden states. Intuitively this is because the network has to carry low level information through every state to predict low level information at the other end, whereas BESNet carries it directly through the low level diagonal connections leaving higher hidden states free to capture high level information.

We quantify this by computing the probability of a class a given the distribution of data points in state S , as $P(a|S) = P(S|a) * P(a)/P(S)$, where $P(S|a)$ is the 2D histogram of data points belonging to class a in state S , $P(a)$ is the probability of class a , $P(S)$ is the 2D histogram of all data points in state S . Figure 4.13 shows

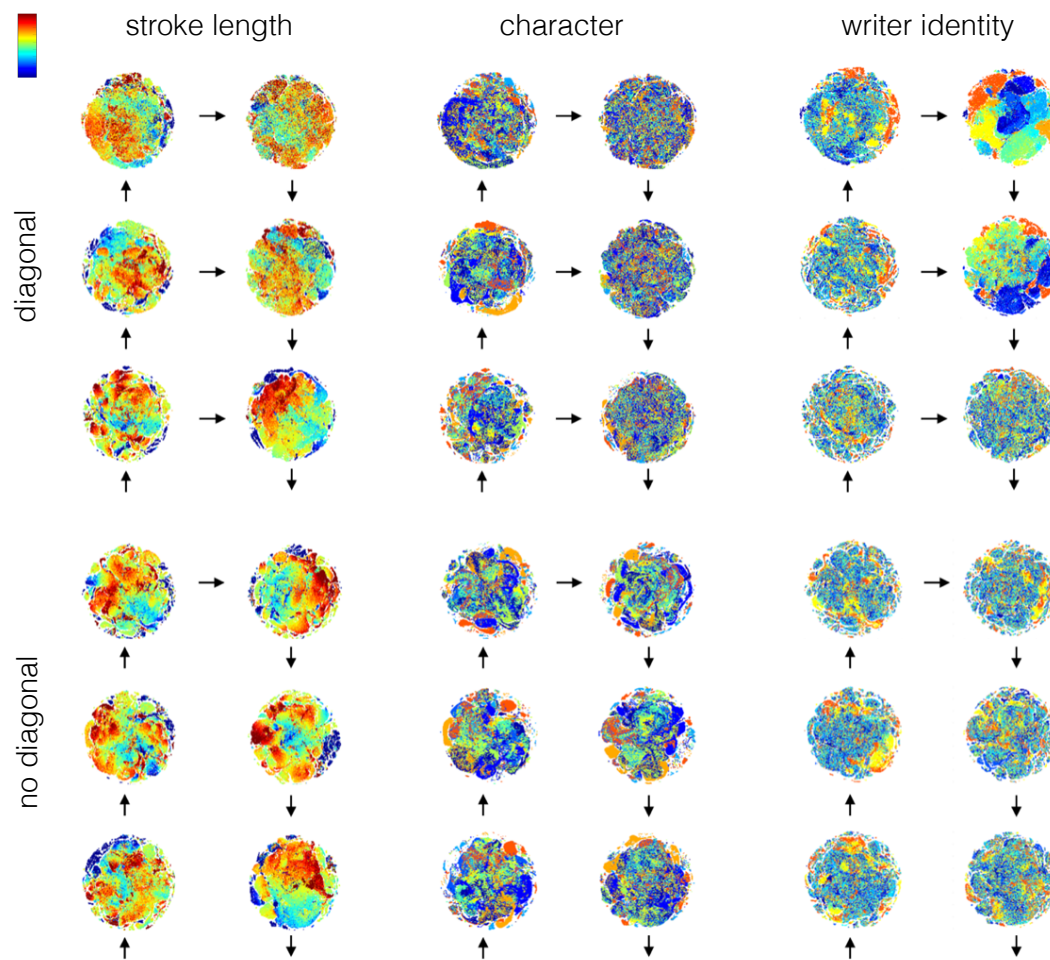


Figure 4.12: Hidden state values of a 3 level model trained without any labels on IAM-OnDB, reduced to 2 dimensions using tSNE mapping. The network discovers writer identity at the highest level, while lower level phenomena such as stroke length are represented at lower levels.

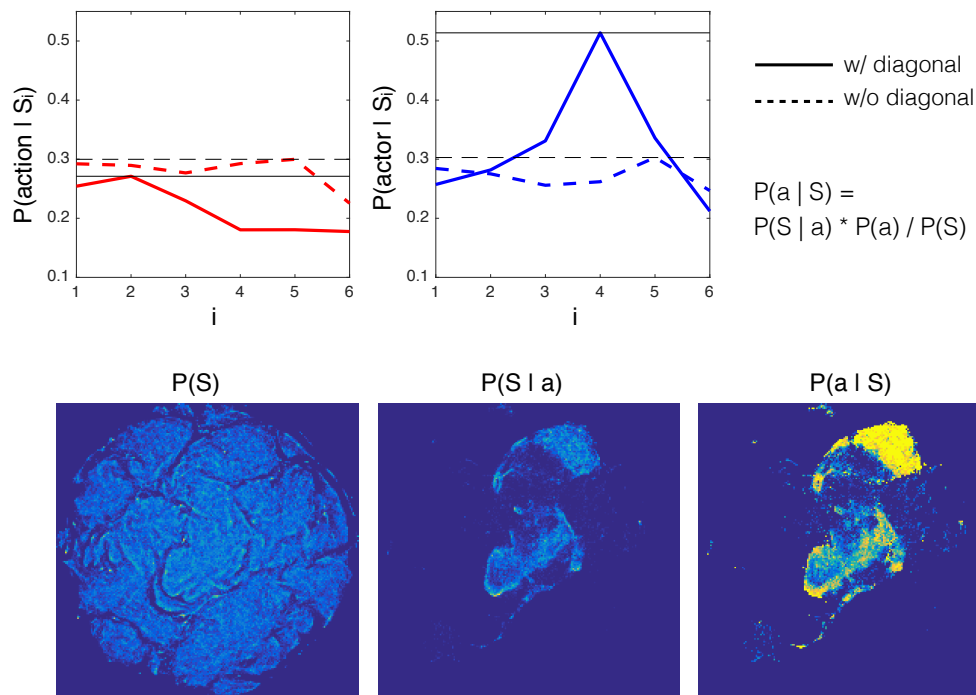


Figure 4.13: Quantitative measure of the separability of actions and actors from hidden states of model with/without diagonal connections.

the mean probability of an action and an actor at every hidden state of the network, for the full architecture (solid) and the architecture without diagonal connections (dashed). It clearly shows that our architecture can much better represent actors at its top level, while action is represented only at the second cell state.

4.6 Discussion

We have proposed a framework for modeling the behavior of animals, that simultaneously classifies their actions and predicts their motion. We showed empirically that motion prediction (a target that requires no labeling) is a good auxiliary task for training action classifiers, especially when labels are scarce. We also showed that the generative task can be used to simulate trajectories that look natural to the human eye, and that activating classification units increases the frequency of that action in the simulation. Finally, we showed that our model lends itself well to discovery of high level information from the data, by visualizing what is captured in its hidden states.

We tested the framework on two types of data, fly behavior and online handwriting, and we anticipate that it will scale to more complex data with appropriate tuning of

hyperparameters and abstraction of visual input. For example, application to human motion capture with 1st person video as sensory input might require greater model complexity to account for the higher dimensional motor control and pre-processing of the sensory input, e.g. with a convolutional neural network, to extract a higher level sensory representation before feeding it to the dynamical system.

Moving forward, we are interested in working on hierarchical label embedding in the states, assigning higher order activities to units higher in the network. Along those lines, a discrete recurrent network could be trained separately on the wealth of available text, and be placed on top of a real-valued handwriting network. We also aim to explore how this framework can be used to understand the neural mechanisms underlying the generation of behavior in flies.

Chapter 5

CONCLUSION

In this thesis we set out to develop computational tools for automating behavior analysis. We introduced a fly tracking tool that combines capabilities of state of the art trackers currently available, and showed how its raw output can be used to quantify behavior. On top of those features we trained supervised learning algorithms to detect and classify social actions between flies, reaching 90% of human performance. In order to alleviate the need for fully labeling the training videos, and to make use of the abundance of unlabeled videos available, we presented a semi-supervised learning algorithm that requires only half of the training labels to reach the same performance as its supervised counterpart. This was achieved by using motion prediction as an auxiliary task, which we chose based on our intuition that a model that can predict how a fly will move must have some notion about high level information such as the fly's goals or current activity. Further confirming our intuition we found that, when trained unsupervised, the model learned to represent high level information such as actions and gender in its hidden states. Furthermore, by representing the environment as seen from an agent, and its motion features as motor controls, we showed that predicted motion can be used to simulate a fly that interacts with a dynamically changing environment.

We see promising future research directions in the following areas: **Tracking:** While FlyTracker might currently be the most feasible fly tracking software freely available, we showed that idTracker has higher identity accuracy than FlyTracker under the right conditions at a significant time cost. Combining the strengths of the two algorithms could be worthwhile. **Learning from few labels:** Although our semi-supervised model significantly reduces the need for training labels, it still requires more labels than we would like it to. We believe that active learning is the next step towards further reducing the number of labels needed, helping the expert find the most useful samples to label. This could perhaps be done by training the model unsupervised and selecting samples that are dissimilar in the hidden space. **Performance:** We have proposed a framework that is a good starting point for sensory-motor modeling; our simulation and classification results look promising but there is ample room for improvement. Possible future directions are 1) further exploration of the architecture, 2) defining objective functions that enforce label

smoothness, 3) defining what constitutes a good/bad simulation and further train the model with reinforcement learning, 4) predicting not only the future motion of an agent but also how the environment evolves, and 5) richer sensory input representation, for instance adding a channel for the gender of the other flies or for the action classification output of the other flies. **Discovery:** We demonstrated that our model can be used to discover higher level phenomena and we think this could be further explored, specifically, finding out what type of new information can be discovered and how to extract it from the hidden states.

Behavioral scientists are trying to understand intelligence and AI researchers are trying to create intelligence, in our view the two are mutually beneficial. If we knew how the brain worked we might create an artificial one, and if we could imitate intelligent behavior we might better understand its true underlying mechanism.

BIBLIOGRAPHY

- [A+03] Yasemin Altun, Ioannis Tsochantaridis, Thomas Hofmann, et al. “Hidden markov support vector machines”. In: *ICML*. Vol. 3. 2003, pp. 3–10.
- [AP14] David J Anderson and Pietro Perona. “Toward a Science of Computational Ethology”. In: *Neuron* 84.1 (2014), pp. 18–31.
- [Asa+14] Kenta Asahina, Kiichi Watanabe, Brian J Duistermars, Eric Hoopfer, Carlos Roberto González, Eyrún Arna Eyjólfsdóttir, Pietro Perona, and David J Anderson. “Tachykinin-Expressing Neurons Control Male-Specific Aggressive Arousal in *Drosophila*”. In: *Cell* 156.1 (2014), pp. 221–235.
- [Bal12] Pierre Baldi. “Autoencoders, unsupervised learning, and deep architectures.” In: *ICML unsupervised and transfer learning 27.37-50* (2012), p. 1.
- [Bel56] Richard Bellman. “Dynamic programming and Lagrange multipliers”. In: *Proceedings of the National Academy of Sciences of the United States of America* 42.10 (1956), p. 767.
- [Ben+08] Yannick Benezeth, Pierre-Marc Jodoin, Bruno Emile, H el ene Laurent, and Christophe Rosenberger. “Review and evaluation of commonly-implemented background subtraction algorithms”. In: *Pattern Recognition, 2008. ICPR 2008. 19th International Conference on*. IEEE. 2008, pp. 1–4.
- [Ber+14] Gordon J Berman, Daniel M Choi, William Bialek, and Joshua W Shaevitz. “Mapping the stereotyped behaviour of freely moving fruit flies”. In: *Journal of The Royal Society Interface* 11.99 (2014), p. 20140672.
- [Bra+09] Kristin Branson, Alice A Robie, John Bender, Pietro Perona, and Michael H Dickinson. “High-throughput ethomics in large groups of *Drosophila*”. In: *Nature methods* 6.6 (2009), pp. 451–457.
- [Bra84] Valentino Braitenberg. *Vehicles Experiments in Synthetic Psychology*. MIT Press, 1984.
- [Bur+12a] Xavier P Burgos-Artizzu, Piotr Doll ar, Dayu Lin, David J Anderson, and Pietro Perona. “Social behavior recognition in continuous video”. In: *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*. IEEE. 2012, pp. 1322–1329.
- [Bur+12b] Xavier P Burgos-Artizzu, Piotr Doll ar, Dayu Lin, David J Anderson, and Pietro Perona. “Social behavior recognition in continuous video”. In: *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*. IEEE. 2012, pp. 1322–1329.

- [Che+02] Selby Chen, Ann Yeelin Lee, Nina M Bowens, Robert Huber, and Edward A Kravitz. “Fighting fruit flies: a model system for the study of aggression”. In: *Proceedings of the National Academy of Sciences* 99.8 (2002), pp. 5664–5668.
- [Cho+14] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. “Learning phrase representations using RNN encoder-decoder for statistical machine translation”. In: *arXiv preprint arXiv:1406.1078* (2014).
- [Chu+15] Junyoung Chung, Kyle Kastner, Laurent Dinh, Kratarth Goel, Aaron C Courville, and Yoshua Bengio. “A recurrent latent variable model for sequential data”. In: *Advances in neural information processing systems*. 2015, pp. 2962–2970.
- [Dan+09] Heiko Dankert, Liming Wang, Eric D Hoopfer, David J Anderson, and Pietro Perona. “Automated monitoring and analysis of social behavior in *Drosophila*”. In: *Nature methods* 6.4 (2009), pp. 297–303.
- [De +09] Fernando De la Torre, Jessica Hodgins, J Montano, S Valcarcel, R Forcada, and J Macey. *Guide to the carnegie mellon university multi-modal activity (cmu-mmact) database*. Tech. rep. Citeseer, 2009.
- [Del+14] Anthony I Dell, John A Bender, Kristin Branson, Iain D Couzin, Gonzalo G de Polavieja, Lucas PJJ Noldus, Alfonso Pérez-Escudero, Pietro Perona, Andrew D Straw, Martin Wikelski, et al. “Automated image-based tracking and its application in ecology”. In: *Trends in ecology & evolution* 29.7 (2014), pp. 417–428.
- [DLR77] Arthur P Dempster, Nan M Laird, and Donald B Rubin. “Maximum likelihood from incomplete data via the EM algorithm”. In: *Journal of the royal statistical society. Series B (methodological)* (1977), pp. 1–38.
- [Dol+05] P. Dollár, V. Rabaud, G. Cottrell, and S. Belongie. “Behavior Recognition via Sparse Spatio-Temporal Features”. In: *VS-PETS*. Oct. 2005.
- [Eyj+14] Eyrun Eyjolfsdottir, Steve Branson, Xavier P Burgos-Artizzu, Eric D Hoopfer, Jonathan Schor, David J Anderson, and Pietro Perona. “Detecting social actions of fruit flies”. In: *Computer Vision–ECCV 2014*. Springer, 2014, pp. 772–787.
- [Fan+08] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. “LIBLINEAR: A library for large linear classification”. In: *The Journal of Machine Learning Research* 9 (2008), pp. 1871–1874.
- [GMH13] Alan Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. “Speech recognition with deep recurrent neural networks”. In: *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*. IEEE. 2013, pp. 6645–6649.

- [Gom+14] Alex Gomez-Marin, Joseph J Paton, Adam R Kampff, Rui M Costa, and Zachary F Mainen. “Big behavioral data: psychology, ethology and the foundations of neuroscience”. In: *Nature neuroscience* 17.11 (2014), pp. 1455–1462.
- [Gor+07] Lena Gorelick, Moshe Blank, Eli Shechtman, Michal Irani, and Ronen Basri. “Actions as Space-Time Shapes”. In: *Transactions on Pattern Analysis and Machine Intelligence* 29.12 (Dec. 2007), pp. 2247–2253.
- [Gra13] Alex Graves. “Generating sequences with recurrent neural networks”. In: *arXiv preprint arXiv:1308.0850* (2013).
- [Hal94] Jeffrey C Hall. “The mating of a fly”. In: *Science* 264.5166 (1994), pp. 1702–1714.
- [HLD11] Minh Hoai, Zhen-Zhong Lan, and Fernando De la Torre. “Joint segmentation and classification of human actions in video”. In: *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*. IEEE. 2011, pp. 3265–3272.
- [Hoo+15] Eric D Hoopfer, Yonil Jung, Hidehiko K Inagaki, Gerald M Rubin, and David J Anderson. “P1 interneurons promote a persistent internal state that enhances inter-male aggression in *Drosophila*”. In: *Elife* 4 (2015), e11346.
- [Hoy+08] Susanne C Hoyer, Andreas Eckart, Anthony Herrel, Troy Zars, Susanne A Fischer, Shannon L Hardie, and Martin Heisenberg. “Octopamine in Male Aggression of *Drosophila*”. In: *Current Biology* 18.3 (2008), pp. 159–167.
- [HS97] Sepp Hochreiter and Jürgen Schmidhuber. “Long short-term memory”. In: *Neural computation* 9.8 (1997), pp. 1735–1780.
- [Jhu+10a] Hueihan Jhuang, Estibaliz Garrote, Xinlin Yu, Vinita Khilnani, Tomaso Poggio, Andrew D Steele, and Thomas Serre. “Automated home-cage behavioural phenotyping of mice”. In: *Nature communications* 1 (2010), p. 68.
- [Jhu+10b] Hueihan Jhuang, Estibaliz Garrote, Xinlin Yu, Vinita Khilnani, Tomaso Poggio, Andrew D Steele, and Thomas Serre. “Automated home-cage behavioural phenotyping of mice”. In: *Nature communications* 1 (2010), p. 68.
- [Kab+12] Mayank Kabra, Alice A Robie, Marta Rivera-Alba, Steven Branson, and Kristin Branson. “JAABA: interactive machine learning for automatic annotation of animal behavior”. In: *nature methods* (2012).
- [Kab+13] Mayank Kabra, Alice A Robie, Marta Rivera-Alba, Steven Branson, and Kristin Branson. “JAABA: interactive machine learning for automatic annotation of animal behavior”. In: *nature methods* 10.1 (2013), pp. 64–67.

- [KGS12] Hema Swetha Koppula, Rudhir Gupta, and Ashutosh Saxena. “Learning human activities and object affordances from rgb-d videos”. In: *arXiv preprint arXiv:1210.1207* (2012).
- [Kue+11] H. Kuehne, H. Jhuang, E. Garrote, T. Poggio, and T. Serre. “HMDB: a large video database for human motion recognition”. In: *Proceedings of the International Conference on Computer Vision (ICCV)*. 2011.
- [Kuh55] Harold W Kuhn. “The Hungarian method for the assignment problem”. In: *Naval research logistics quarterly* 2.1-2 (1955), pp. 83–97.
- [LB05] Marcus Liwicki and Horst Bunke. “IAM-OnDB-an on-line English sentence database acquired from handwritten text on a whiteboard”. In: *Document Analysis and Recognition, 2005. Proceedings. Eighth International Conference on*. IEEE. 2005, pp. 956–961.
- [Lim+14] Rod S Lim, Eyrún Eyjólfsdóttir, Euncheol Shin, Pietro Perona, and David J Anderson. “How food controls aggression in *Drosophila*”. In: *PloS one* 9.8 (2014), e105626.
- [LLF09] Daniel A Levitis, William Z Lidicker, and Glenn Freund. “Behavioural biologists do not agree on what constitutes behaviour”. In: *Animal behaviour* 78.1 (2009), pp. 103–110.
- [Mar+15] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: <http://tensorflow.org/>.
- [MH08] Laurens van der Maaten and Geoffrey Hinton. “Visualizing data using t-SNE”. In: *Journal of Machine Learning Research* 9.Nov (2008), pp. 2579–2605.
- [MLS09] Marcin Marszałek, Ivan Laptev, and Cordelia Schmid. “Actions in Context”. In: *IEEE Conference on Computer Vision & Pattern Recognition*. 2009.
- [Mni+15] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. “Human-level control through deep reinforcement learning”. In: *Nature* 518.7540 (2015), pp. 529–533.
- [Moo02] J Moore. “Some thoughts on the relation between behavior analysis and behavioral neuroscience”. In: *The Psychological Record* 52.3 (2002), p. 261.
- [Mül+07] Meinard Müller, Tido Röder, Michael Clausen, Bernhard Eberhardt, Björn Krüger, and Andreas Weber. “Documentation mocap database hdm05”. In: (2007).

- [Mur02] Kevin Patrick Murphy. “Dynamic bayesian networks: representation, inference and learning”. PhD thesis. University of California, Berkeley, 2002.
- [NCF10] Juan Carlos Niebles, Chih-Wei Chen, and Li Fei-Fei. “Modeling temporal structure of decomposable motion segments for activity classification”. In: *Computer Vision–ECCV 2010*. Springer, 2010, pp. 392–405.
- [Oh+08] Sang Min Oh, James M Rehg, Tucker Balch, and Frank Dellaert. “Learning and inferring motion patterns using parametric segmental switching linear dynamic systems”. In: *International Journal of Computer Vision* 77.1-3 (2008), pp. 103–124.
- [Oh+11] Sangmin Oh, Anthony Hoogs, Amitha Perera, Naresh Cuntoor, Chia-Chih Chen, Jong Taek Lee, Saurajit Mukherjee, JK Aggarwal, Hyungtae Lee, Larry Davis, et al. “A large-scale benchmark dataset for event recognition in surveillance video”. In: *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*. IEEE. 2011, pp. 3153–3160.
- [OZR11] Tyler A Ofstad, Charles S Zuker, and Michael B Reiser. “Visual place learning in *Drosophila melanogaster*”. In: *Nature* 474.7350 (2011), pp. 204–207.
- [Pér+14] Alfonso Pérez-Escudero, Julián Vicente-Page, Robert C Hinz, Sara Arganda, and Gonzalo G de Polavieja. “idTracker: tracking individuals in a group by automatic identification of unmarked animals”. In: *Nature methods* 11.7 (2014), pp. 743–748.
- [RA10] MS Ryoo and JK Aggarwal. *UT-Interaction Dataset, ICPR contest on Semantic Description of Human Activities (SDHA)*. 2010.
- [Ram+15] Pavan Ramdya, Pawel Lichocki, Steeve Cruchet, Lukas Frisch, Winnie Tse, Dario Floreano, and Richard Benton. “Mechanosensory interactions drive collective behaviour in *Drosophila*”. In: *Nature* 519.7542 (2015), pp. 233–236.
- [Ras+15] Antti Rasmus, Mathias Berglund, Mikko Honkala, Harri Valpola, and Tapani Raiko. “Semi-Supervised Learning with Ladder Networks”. In: *Advances in Neural Information Processing Systems*. 2015, pp. 3532–3540.
- [RHW86] DE Rumelhart, Geoffrey E Hinton, and Ronald J Williams. “Learning Internal Representation by Error Propagation, Parallel Distributed Processing”. In: *Explor. Microstruct. Cognition* 1 (1986), pp. 318–362.

- [Rob+16] William M Roberts, Steven B Augustine, Kristy J Lawton, Theodore H Lindsay, Tod R Thiele, Eduardo J Izquierdo, Serge Faumont, Rebecca A Lindsay, Matthew Cale Britton, Navin Pokala, et al. “A stochastic neuronal model predicts random search behaviors at multiple spatial scales in *C. elegans*”. In: *eLife* 5 (2016), e12572.
- [SB06] Leonid Sigal and Michael J Black. “Humaneva: Synchronized video and motion capture dataset for evaluation of articulated human motion”. In: *Brown University TR* 120 (2006).
- [Shi+11] Qinfeng Shi, Li Cheng, Li Wang, and Alex Smola. “Human action segmentation and recognition using discriminative semi-Markov models”. In: *International journal of computer vision* 93.1 (2011), pp. 22–32.
- [Sil+16] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. “Mastering the game of Go with deep neural networks and tree search”. In: *Nature* 529.7587 (2016), pp. 484–489.
- [Sim96] Herbert A Simon. *The sciences of the artificial*. MIT press, 1996.
- [SK09] Kathleen K Siwicki and Edward A Kravitz. “Fruitless, doublesex and the genetics of social behavior in *Drosophila melanogaster*”. In: *Current opinion in neurobiology* 19.2 (2009), pp. 200–206.
- [SLC04] Christian Schuldt, Ivan Laptev, and Barbara Caputo. “Recognizing human actions: a local SVM approach”. In: *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*. Vol. 3. IEEE. 2004, pp. 32–36.
- [Sun+12] Jaeyong Sung, Colin Ponce, Bart Selman, and Ashutosh Saxena. “Unstructured human activity detection from rgb-d images”. In: *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE. 2012, pp. 842–849.
- [SZS12] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. “UCF101: A Dataset of 101 Human Actions Classes From Videos in The Wild”. In: *arXiv preprint arXiv:1212.0402* (2012).
- [TBB09] Moritz Tenorth, Jan Bandouch, and Michael Beetz. “The TUM kitchen data set of everyday manipulation activities for motion tracking and action recognition”. In: *Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on*. IEEE. 2009, pp. 1089–1096.
- [Tso+05] Ioannis Tsochantaridis, Thorsten Joachims, Thomas Hofmann, and Yasemin Altun. “Large margin methods for structured and interdependent output variables”. In: *Journal of Machine Learning Research*. 2005, pp. 1453–1484.

- [Val15] Harri Valpola. “From neural PCA to deep unsupervised learning”. In: *Adv. in Independent Component Analysis and Learning Machines* (2015), pp. 143–171.
- [Vit67] Andrew Viterbi. “Error bounds for convolutional codes and an asymptotically optimum decoding algorithm”. In: *Information Theory, IEEE Transactions on* 13.2 (1967), pp. 260–269.
- [WA10] Liming Wang and David J Anderson. “Identification of an aggression-promoting pheromone and its receptor neurons in *Drosophila*”. In: *Nature* 463.7278 (2010), pp. 227–231.
- [Wan+11] Liming Wang, Xiaoqing Han, Jennifer Mehren, Makoto Hiroi, Jean-Christophe Billeter, Tetsuya Miyamoto, Hubert Amrein, Joel D Levine, and David J Anderson. “Hierarchical chemosensory regulation of male-male social interactions in *Drosophila*”. In: *Nature neuroscience* 14.6 (2011), pp. 757–762.
- [Wil+15] Alexander B Wiltschko, Matthew J Johnson, Giuliano Iurilli, Ralph E Peterson, Jesse M Katon, Stan L Pashkovski, Victoria E Abaira, Ryan P Adams, and Sandeep Robert Datta. “Mapping sub-second structure in mouse behavior”. In: *Neuron* 88.6 (2015), pp. 1121–1135.