

Detecting Actions of Fruit Flies



Eyrún Arna Eyjólfsdóttir
Computing + Mathematical Sciences
California Institute of Technology

A thesis submitted for the degree of
Master of Science

April 14, 2014

Acknowledgements

I would like to thank the biologists that provided me with videos and annotations, and explained to me behaviors of fruit flies, in particular Eric Hoopfer, Brian Duistermars, Kenta Ashina, Jonathan Shor, Rod Lim, and David Anderson. I would also like to thank Sarah Vernallis, Sarah Kim, Dylan Jack, and Caroline Arya, for learning how to annotate actions and providing me with additional annotations on a subset of the data. Thanks to Michael Maire for providing me with a tracking code base, and Steve Branson for getting me started with, and collaborating on, bout features and the structured SVM, and thanks to Kristin Branson who provided me with code for comparison with JAABA, and Xavier Burgos who gave me code for comparison with Auto-context. Last but not least I would like to thank my advisor, Pietro Perona, for guiding me through this project and brainstorming ideas, and for his support. This work was funded by the Leifur Eiriksson Foundation and by the Gordon and Betty Moore Foundation.

Abstract

In this thesis we describe a system that tracks fruit flies in video and automatically detects and classifies their actions. We introduce Caltech Fly-vs-Fly Interactions, a new dataset that contains hours of video showing pairs of fruit flies engaging in social interactions, and is published with complete expert annotations and articulated pose trajectory features. We compare experimentally the value of a frame-level feature representation with the more elaborate notion of ‘bout features’ that capture the structure within actions. Similarly, we compare a simple sliding window classifier architecture with a more sophisticated structured output architecture, and find that window based detectors outperform the much slower structured counterparts, and approach human performance. In addition we test the top performing detector on the CRIM13 mouse dataset, finding that it matches the performance of the best published method.

Contents

1	Introduction	1
1.1	Motivation	1
1.2	State of the art	2
1.3	Overview	4
2	Fly Data	6
2.1	Experimental setup	6
2.2	Annotations	7
3	Data Representation	11
3.1	Tracking	12
3.2	Feature extraction	13
4	Performance Measure	15
4.1	Types of measures	15
4.2	Bout vs. frame	18
5	Action Detection	19
5.1	Bout features	19
5.2	Sliding window framework	21
5.3	Structured output framework	22
6	Experiments and Analysis	25
6.1	Human vs. human	25
6.2	Method comparison	27
6.3	Performance on CRIM13	31
7	Discussion	37
	References	39

1

Introduction

1.1 Motivation

Machine understanding of human behavior is perhaps one of the most useful 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. The focus of this thesis is detection and recognition of actions, which in the case of humans is hampered by two difficulties. First, tracking and pose estimation of humans is very difficult, due to appearance variation, the amount of occlusion in natural environments, and the sheer complexity of human body motions. Second, it is difficult (both technically and legally) to film large numbers of humans behaving spontaneously in natural settings. As a result, human action datasets are small and unrepresentative, especially when social behavior is concerned.

In parallel, neurobiologists are interested in measuring and analyzing behavior of animals of different genotypes, in order to understand the link between genes, brains and behavior. One of their most popular model organism is the fruit fly; 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 a collaboration with biologists we have put together a large annotated dataset of fruit flies interacting spontaneously in controlled environments, which 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 thesis we describe an end-to-end approach for detecting the actions of fruit flies from video. The main contributions of our study are:

1. We introduce a new dataset, Caltech Fly-vs-Fly Interactions (Fly-vs-Fly for short), containing 22 hours of continuous video of fruit flies interacting, spontaneously and sporadically. The dataset was annotated by neurobiologists, with complete labeling of 10 social actions. It additionally comes with a second layer of annotations, obtained from trained novice annotators, which can be used as a reference point for action detection performance. Along with the videos, we publish pose (position, orientation, wing angles, etc.) trajectories, from which we have computed a number of time-varying features that may be used to detect, segment and classify actions between the flies.
2. 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.
3. We define bout features that are designed to extract statistical patterns from an interval of frame-level features 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%.
4. 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.

1.2 State of the art

Datasets: A large number of human action datasets have been published. Most of the earlier contributions, KTH [1], Weizmann [2], Hollywood 2 [3], Olympic Sports [4], HMDB51 [5], and UCF-101 [6], consist of short pre-segmented action clips, making them

suitable for action classification, but not for action detection and segmentation. UCF-motion capture [7], HumanEva [8], and CAD-60/120 [9, 10] come with fully tracked skeletons which makes them useful for analyzing a range of human motions; however, these datasets are very small, and their actions are acted. Finally, a recent dataset, VIRAT [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. Table 1.1 compares details of the mentioned datasets.

The publicly available datasets of animal behavior videos are Honeybee Dance [12], UCSD mice [13], Home-cage behaviors [14], and CRIM13 [15]. 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 spatio-temporal features. A large and well annotated dataset containing unsegmented spontaneous actions and including tracking of articulated body motion has not yet been published. Our dataset aims to fill this place.

Dataset	Year	#Citations	Duration*	#Actions	Natural	Social	Continuous	Articulated pose
KTH	2004	1397	3 hours	6	x	x	x	x
Weizmann	2005	858	1 hour	10	x	x	x	x
HumanEva	2006	501	22 minutes	6	x	x	✓	✓
UCF-mocap	2007	185	1 hour	5	x	x	x	✓
Hollywood 2(1)	2009	352(1073)	20 hours	12	✓	✓	x	x
Olympic Sports	2010	153	2 hours	16	✓	x	x	x
HMDB51	2011	83	2 hours	51	✓	✓	x	x
VIRAT	2011	59	9 hours	12	✓	x	✓	x
UCF-101(50,11)	2012	16(31,357)	30 hours	101	✓	x	x	x
CAD-60/120	2011/13	123/12	2 hours	10**	x	x	✓	✓
UCSD mice	2005	1211	2 hours	5	✓	x	x	x
Honeybee	2008	51	3 minutes	3	✓	x	✓	x
Home-cage behs	2010	44	12 hours	8	✓	x	✓	x
CRIM13	2012	15	37 hours	13	✓	✓	✓	x
Fly-vs-Fly	-	-	22 hours	11	✓	✓	✓	✓

* estimated upper limit

** 22 activities, 10 subactivities (actions)

Table 1.1: Summary of existing datasets and our new dataset, shown in chronological order and grouped by human vs. animal. Qualities that are desirable for the purpose of detecting realistic social actions from articulated pose are highlighted in green.

Action detection: The simplest approach to action detection and classification is frame-by-frame classification, where each frame is classified based a signal on the frame, or on a time window around it, using discriminative or generative classifiers. More sophisticated approaches globally optimize over possible temporal segmentations, outputting structured sequences of actions. Recent work in action detection falls into these two categories: Dankert et al. detected actions of fruit flies using manually set thresholds along with nearest neighbor comparison [16]; Burgos et al. used boosting and auto-context on sliding windows for detecting actions of mouse pairs [15]; and Kabra et al. also use window based boosting for detecting actions of fruit flies in their interactive behavior annotation tool JAABA [17]. Jhuang et al. used an SVMHMM, described in [18], for detecting actions of single housed mice [14]; Hoai et al. used a multi-class SVM with structured inference for segmenting the dance of the honeybee [19]; and Shi et al. used a semi-Markov model for segmenting human actions [20].

We implement variants of the above methods, specifically comparing sliding window SVM detectors against structured output SVM detectors, expecting the latter, which is similar to [14, 19, 20], to improve frame wise consistency and better capture structured actions. For reference, we also compare with the methods described in [17] and [15] and with the performance of trained novice annotators.

1.3 Overview

Figure 1.1 shows the architecture for an end-to-end action detection system, and demonstrates how data flows between user and software. The system takes as input a video of flies, extracts meaningful tracking features for each fly, runs them through an inference algorithm that uses an action model, learnt during a training stage on expert provided annotations, and outputs predicted annotations to the user. The key components in such a system, outlined in Figure 1.1, are: a) collecting a dataset of videos and annotations that can be used for training and testing a model (Chapter 2), b) tracking flies and extracting meaningful features at each frame (Chapter 3), and c) devising an algorithm for learning an action model, and an inference algorithm that utilizes the model to detect actions from new videos (Chapter 5). Chapter 4 describes performance measures for action detectors, Chapter 6 contains results and analysis of the different approaches, and in Chapter 7 we conclude and discuss future directions.

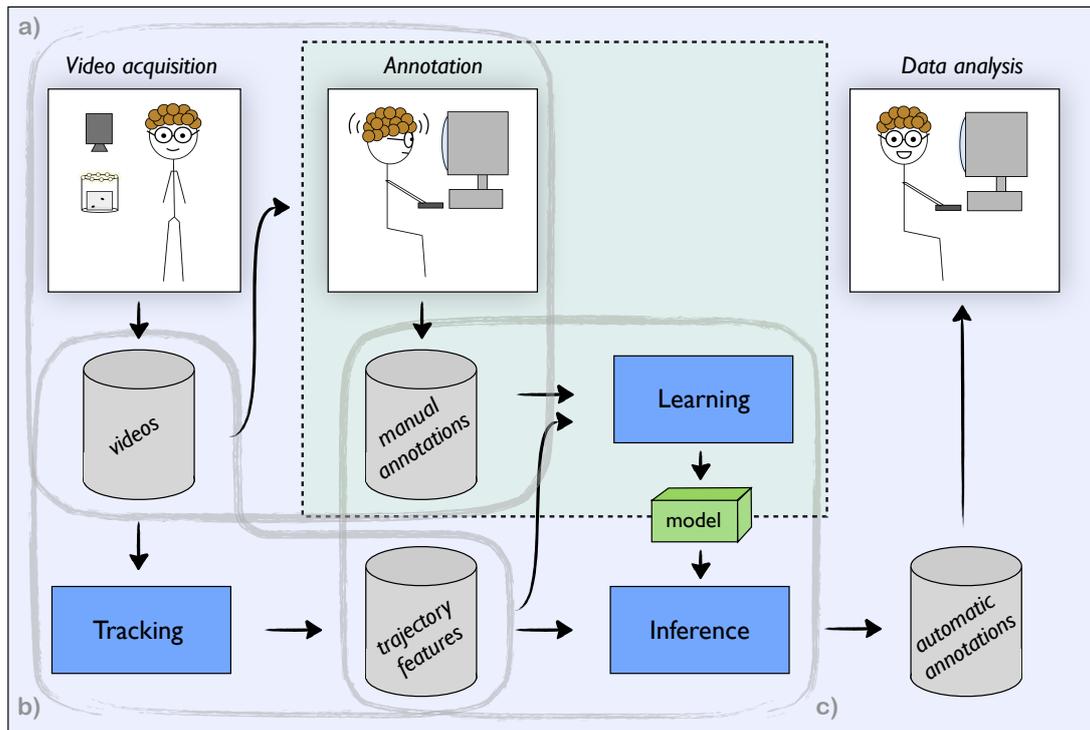


Figure 1.1: System overview: Videos are sent through a tracking module that extracts meaningful features, those features and manual annotations are used by a learning module to learn an action model, and an inference module uses that model to detect actions from features extracted from new videos. Gray cylinders represent data, blue boxes represent software, and the cut out section contains components that are used only during training. The outlined sections represent three major phases of the system: a) *Data acquisition*, which involves collecting videos and annotations of actions to be learnt (Chapter 2), b) *Feature extraction*, which involves tracking flies in videos and extracting features useful for action detection (Chapter 3), and c) *Training*, which involves training an action detection system from features of annotated actions (Chapter 5).

2

Fly Data

2.1 Experimental setup

In collaboration with biologists we have collected a new action dataset, Fly-vs-Fly, 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 may be organized into three sub datasets according to their preparations:

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 food located in the center and walls coated with Fluon such that flies are constrained to walking on the floor. It contains six 20 minute videos recorded at 200Hz with 24 pixels covering the fly body length (2mm).

Aggression contains two hyper aggressive males and is used to quantify the effect of genetic manipulation on their behavior. The flies are placed in a circular 16mm diameter chamber with no food. It consists of ten 30 minute videos recorded at 30Hz with only 16 pixels covering the fly body length.

Courtship has one female and one male fly, which in some of the videos is wild type and in the rest is a so-called hyper courter. 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, 10 of which contain hyper-courtiers and the remaining videos with wild type males.

The filming setups for the three different experiments are shown in Figure 2.1.

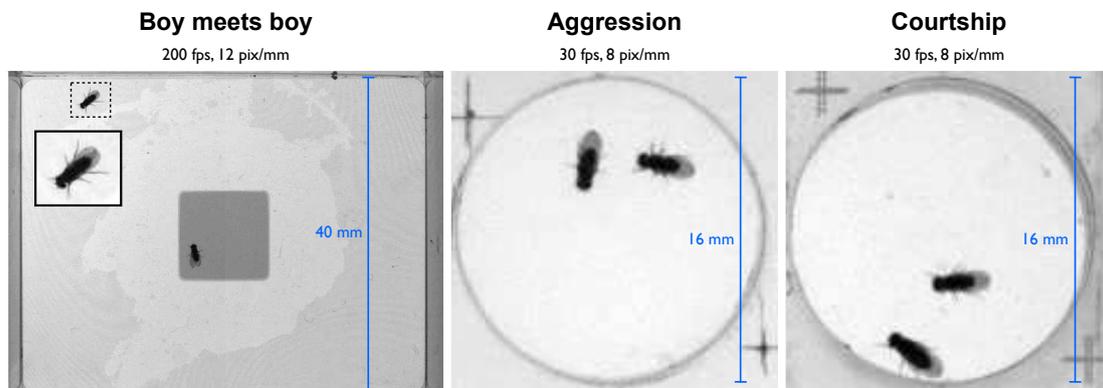


Figure 2.1: Experimental setup: *Boy meets boy* has high temporal and spatial resolution videos and a large chamber with food present, *Aggression* and *Courtship* have lower resolution, a much smaller chamber, and no food. The *Courtship* experiments contain one male and one female fly (the larger one), while the other two contain two male flies.

2.2 Annotations

The entire dataset was annotated by biologists, using an annotation tool that comes embedded with our tracking tool, with 10 different action classes that they have identified for the study of fruit fly interactions: the introductory behavior *touch*, the aggressive behaviors *wing threat*, *charge*, *lunge*, *hold*, and *tussle*, and the courtship behaviors *wing extension*, *circle*, *copulation attempt*, and *copulation*. Figures 2.2 and 2.3 show an example instance of each of these behaviors, with a sequence of 5 frames subsampled from an action bout, as well as a description of each behavior.

Annotating a video involves finding all intervals within the video that contain an action of interest, also referred to as *bouts* of actions, 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, a wing threat sometimes contains charge, wing extension and circling tend to overlap, and touch can overlap with many of the behaviors. The action classes can have substantial intraclass variation both in terms of duration and appearance, and some action instances are ambiguous. Each action takes up only 0.1 – 7% of the frames, except for copulation which takes up 57% of the Courtship videos. Figure 2.4 summarizes the dataset, showing the number of instances of each action, the fraction of time spent in each action, and a histogram of bout durations for each action.

2.2 Annotations

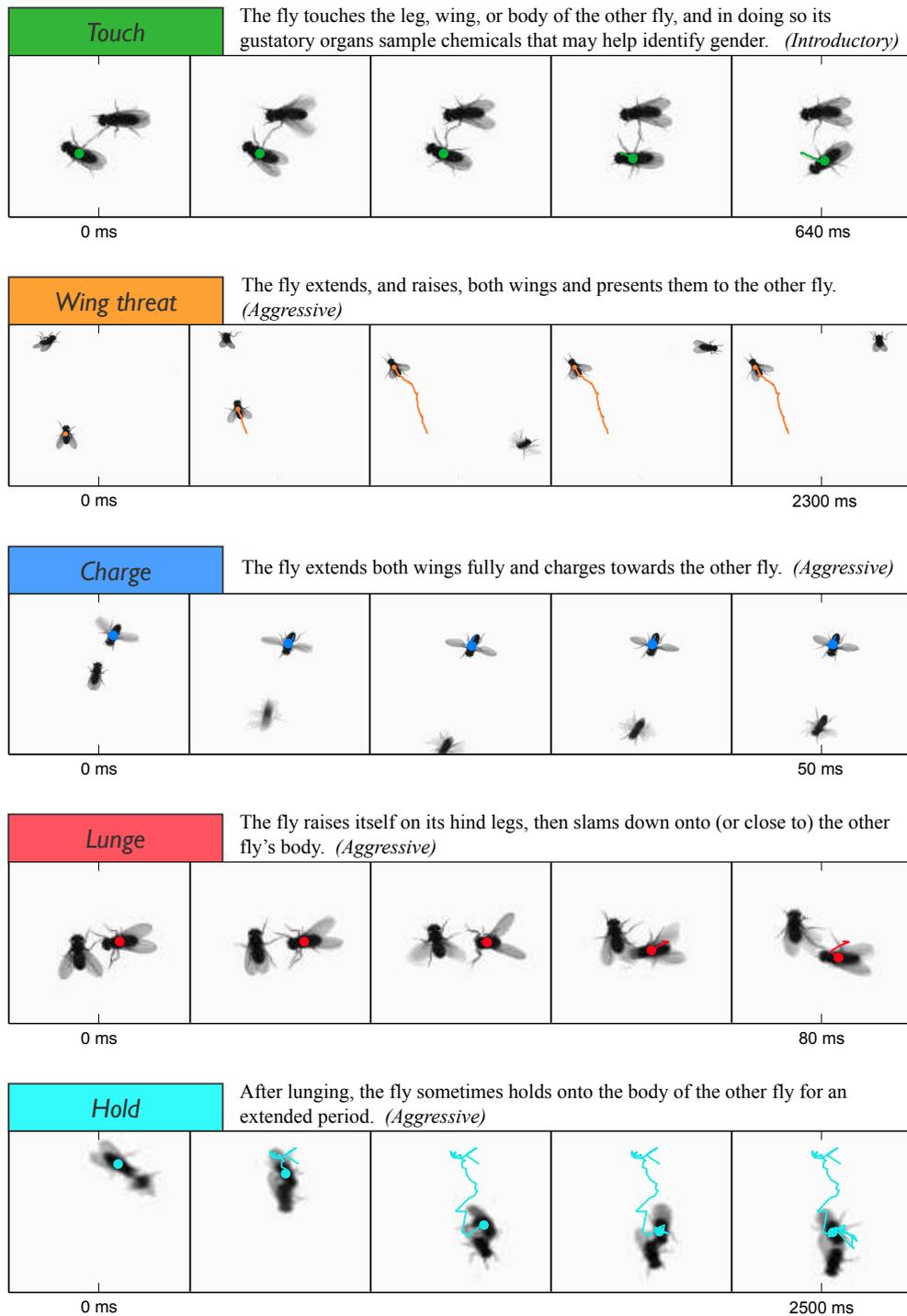


Figure 2.2: Descriptions and examples of all actions. (part 1 of 2)

2.2 Annotations

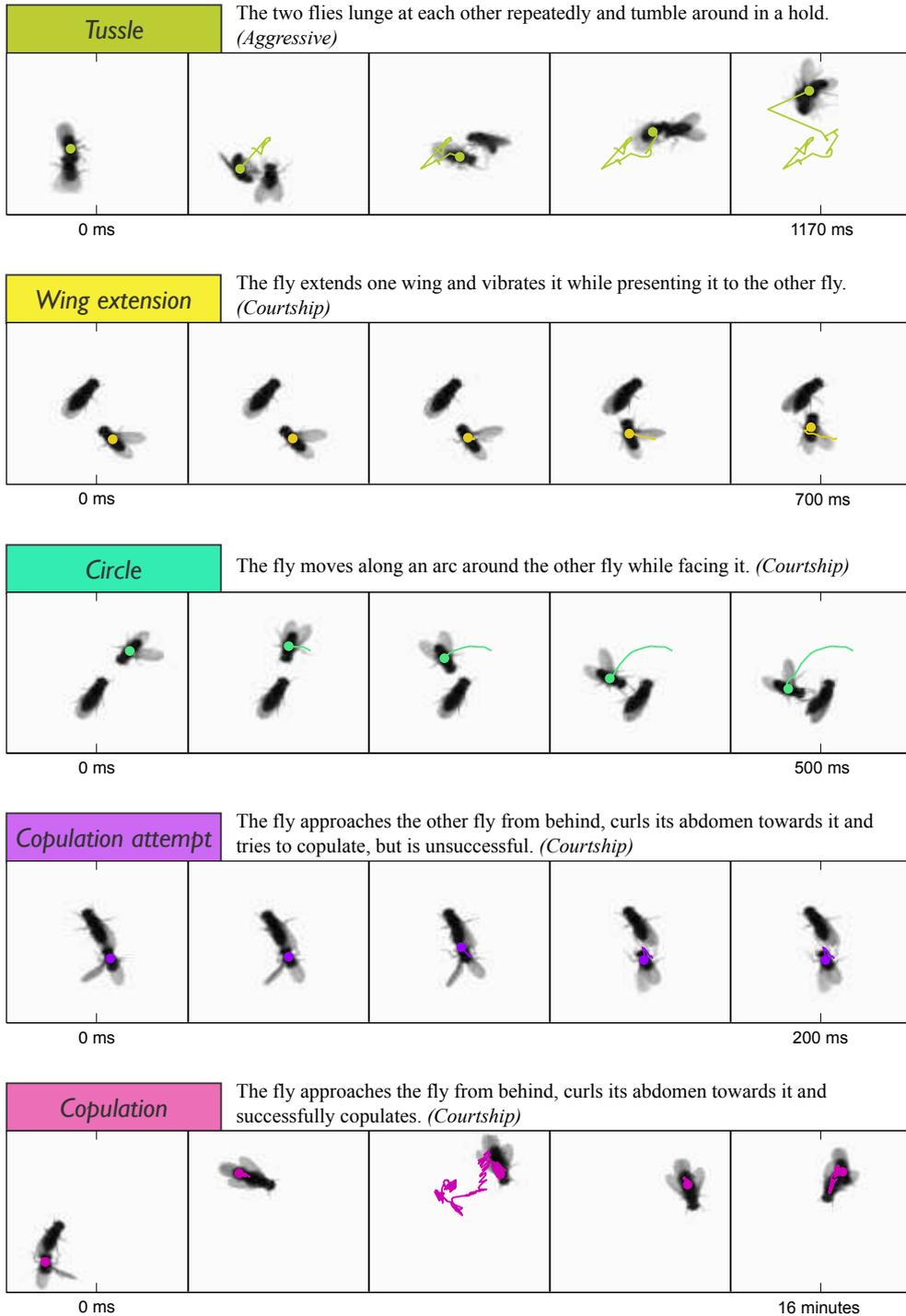


Figure 2.3: Descriptions and examples of all actions. (part 2 of 2)

2.2 Annotations

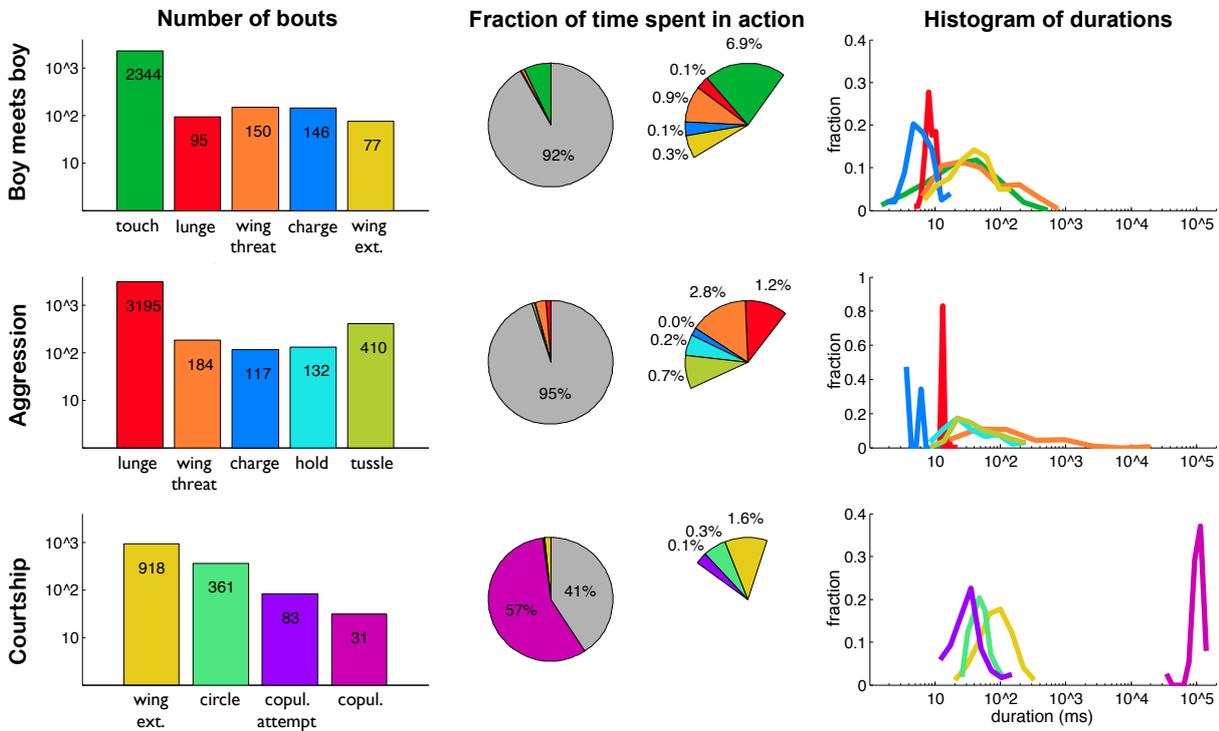


Figure 2.4: Action statistics: *Left:* Number of bouts for each action. *Center:* Fraction of time a fly spends in each action, where the gray area represents unlabeled frames and the right pie shows the unlabeled slices from the left pie expanded exponentially. *Right:* Distribution of bout durations for each action class.

In order to get a sense for the difficulty of the dataset, we hired novice annotators and trained them on expert annotations to re-label a subset of the data. Human performance provides a good reference point for evaluating automated action detection but it is not an upper bound; we should strive for performance as least as good as that of humans. Low human performance can result from difference in perception of an action, one annotator may be more or less conservative than another annotator, or some instances of an action may indeed be ambiguous. This, in fact, motivates the use of automated or semi-automated detection systems which enforces consistency between annotations. We discuss the human-human comparison in more detail in Chapter 6, and show that consistency varies substantially depending on the action and whether it is measured at a per frame level or a per bout level.

3

Data Representation

For action recognition, data representation is half the challenge. If we were to represent a 640x480 pixel resolution video by its grayscale image sequence, then its feature vector would be 307,200 dimensional at each frame, which would require very complex learning algorithms and a large number of training samples. An action classifier should recognize a wing extension, regardless of where within a chamber a fly is standing and which way it is facing, and whether it extends its left or its right wing. In other words, the classifier must be able to handle any variance within an action class, that does not define the action. To avoid having to work with massive amounts of data and/or requiring extremely complex algorithms, we bring some of our knowledge about the domain into the data representation. When we describe an action we find ourselves saying: it extends its *wings*, it raises its *body*, or it touches another fly with its *legs*. To capture this we have implemented a system that detects flies, segments them into body parts, and tracks them throughout a video.

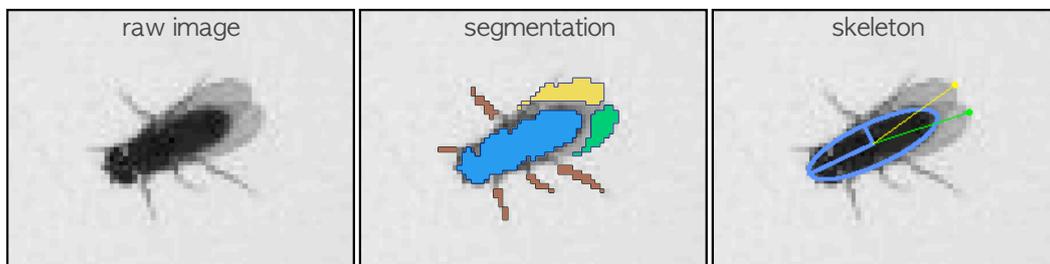


Figure 3.1: Three stages of the tracking process, starting from the raw image of a fly, its foreground segmented into body, wing, and legs, and the skeleton fit to those segments.

3.1 Tracking

We have implemented a tracking tool that extracts fly tracks from raw videos, by detecting the flies at each frame, segmenting them into body, wing, and leg components, connecting the detections across frames to form continuous trajectories, and finally parameterizing the body segments to further reduce dimensionality of the representation. A full description of the tool can be found in a separate technical report [manuscript in preparation], but here we describe in short how it works:

Detection: Our system assumes that videos are recorded with a fixed background, which allows for detection of foreground objects using background subtraction [21]. Given a grayscale video with T frames, $\{I(t)\}_{t=1\dots T}$, we compute a background image, Bg , by taking a weighted mean of a subset of equally spaced frames from the video. Using the fact that the flies are dark, we find the foreground at each frame by thresholding the background subtracted image. To account for variation in background intensity (for example due to food in the chamber) we normalize the background subtracted image with the background itself, such that the difference on top of darker areas are equally accentuated. The foreground image, Fg , at frame t and the foreground mask, fg , are computed as:

$$\begin{aligned} Fg &= |I(t) - Bg| ./ Bg \\ fg &= Fg > thresh_{fg} \end{aligned}$$

Segmentation: From the foreground mask, we compute masks for the body parts to be segmented, using the fact that a fly’s body is darker than the remaining part of the fly, its legs are slim, and when body and legs have been removed only wings should remain:

$$\begin{aligned} body &= Fg > thresh_{body} \\ legs &= fg - dilate(erode(fg)) \\ wings &= fg - legs - dilate(body) \end{aligned}$$

Masks are converted to a set of connected components, that are deemed as being a body part of a fly if they satisfy constraints involving the expected size and relative positioning of components, which can be determined from the video resolution. Before assigning wing components, pixels along the major axis of the body ellipse are subtracted, such that the wing component gets split into left and right components. In the case when flies are touching, their bodies form a single multi-fly component which we resolve by fitting its mask to a Gaussian mixture model with the appropriate number of components, using an Expectation Maximization algorithm [22].

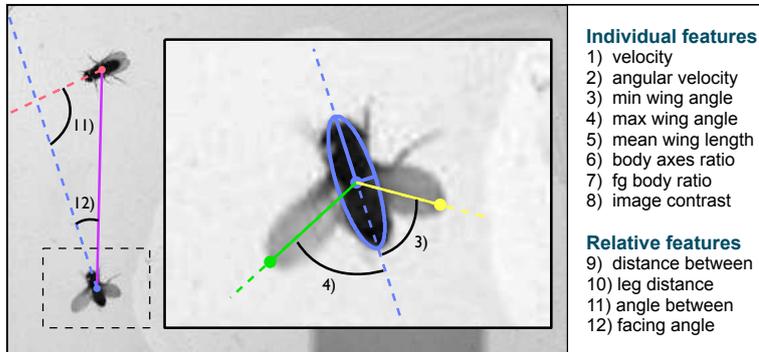


Figure 3.2: Illustration of features derived from the tracked fly skeletons, which are invariant of absolute position and orientation of the fly and relate the pose of the fly to that of the other fly.

Tracking: Detections between adjacent frames are connected by minimizing the cost of possible assignments. For n detections in frame $t-1$ and m detections in frame t , we construct an $n \times m$ cost matrix where entry (i, j) represents the cost between detection i in frame $t-1$ and detection j in frame t , measured in terms of the overlap of the body segments and the distance between its centroids. The optimal assignment is found by applying the Hungarian algorithm [23] to the cost matrix.

Parametrization: To narrow down the number of variables even further, we parameterize the segmented components of the flies. We fit an ellipse to the body components, and represent it as (x, y, a, b, θ) , where (x, y) is the centroid of the ellipse, (a, b) are the minor and major axes of the ellipse, and θ is its orientation. The orientation of an ellipse has a 180° ambiguity, so to determine what is front and what is back we use the following (in that order) for disambiguation: 1) wing position, when wings detected and not extended, 2) heading direction, if velocity is great enough, and 3) consistency with previous frames. Finally, we parameterize the wings by the position of their wing tips (the pixels farthest away from the body centroid), $(w_x^l, w_y^l, w_x^r, w_y^r)$.

The tracking tool extracts, for each fly, a sequence of parameters describing its pose: $(c, y, a, b, \theta, w_x^l, w_y^l, w_x^r, w_y^r)_{t \in \{1, \dots, T\}}$, and to further assist the learning algorithms we derive a set of features that encode our knowledge about the actions of flies.

3.2 Feature extraction

From the trajectories computed by the tracker we derive a set of features that are designed to be invariant of absolute position and orientation of a fly, and relate the pose of one fly to the pose of the other fly, similar to the approaches of [16, 17]. The features (illustrated in Figure 3.2) can be split into two categories: individual features

3.2 Feature extraction

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 and better separating actions. Figure 3.3 shows the distribution of each feature, for all actions in the Boy meets boy sub-dataset and the grab-bag action *other*, giving an idea of which features are important for which action. For each of the 12 features extracted we additionally take the first two time derivatives, resulting in a feature space of 36 *per frame* features. We compute these features for each fly, as annotation is done on a per fly basis, which yields double the amount of data for videos containing two flies - more formally, we represent each video by $\{x_1, x_2\}$, where $x_i(t)$ is the vector of per frame features for fly i at frame t .

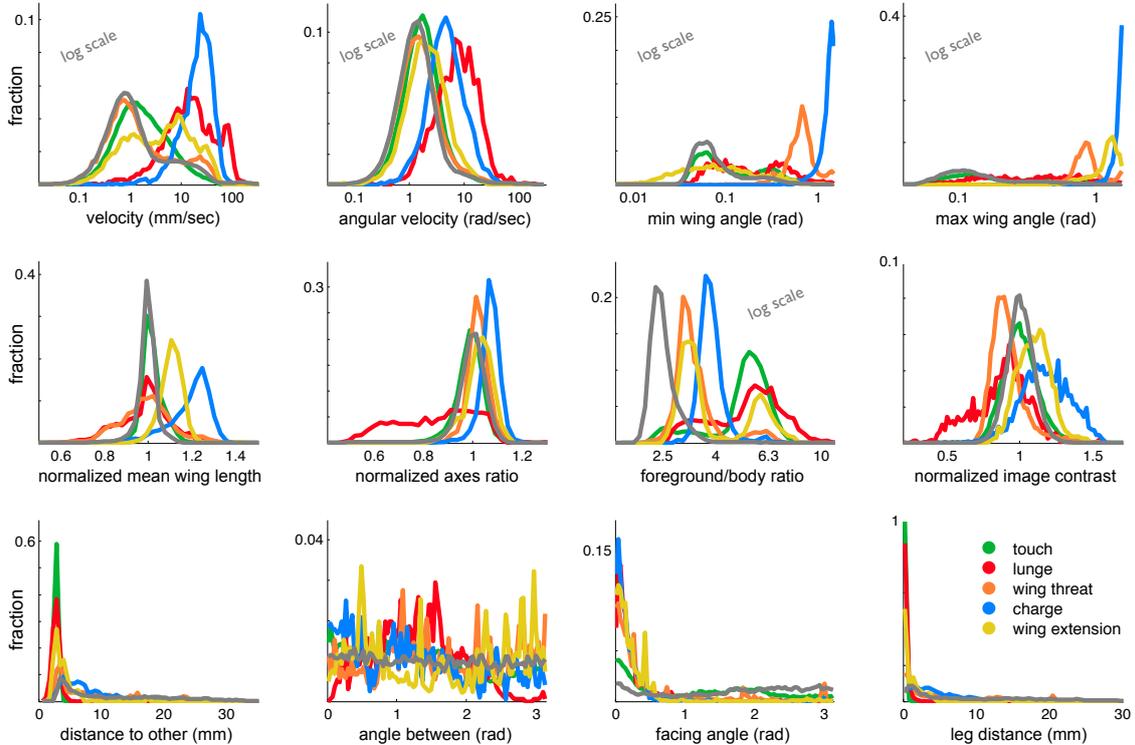


Figure 3.3: Feature distribution shown for the actions of the Boy meets boy sub-dataset, and for the grab-bag action *other* in gray.

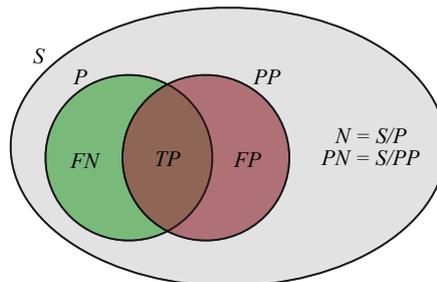
4

Performance Measure

When measuring the performance of an algorithm it is important to use a measure that is relevant to the problem at hand and favors desirable outcomes, which may depend on the objective of the user and the nature of the data (i.e. whether classes are balanced). Here we discuss a few different types of measures and their applicability to the problem of action detection, and describe the difference between a frame based and a bout based measure. For demonstration purposes we have generated synthetic data containing a ground truth sequence with 5 action classes that take up 0.3% – 5% of the frames each and less than 8% in total, and two different prediction sequences that are made to emphasize the differences between the various measures.

4.1 Types of measures

A performance measure involves comparison of ground truth labels and predicted labels and can generally be explained in terms of the following (depicted in diagram below): the set of all datapoints S , positives P , negatives N , predicted positives PP , predicted negatives PN , true positives TP , false positives FP , true negatives TN , and false negatives FN .



Confusion matrix: Commonly used for multi-class classification, a confusion matrix is a square matrix whose rows contain the normalized distribution of predicted classes for all ground truth instances of a single class, i.e. each entry (i,j) represents the fraction of ground truth instances of class i that are predicted as class j . An identity matrix represents a perfect result where all instances are correctly classified and a value lower than 1 on the diagonal means that some instances of the class are misclassified, it is therefore common to use the average of the diagonal as a single number to compare the quality of results. This works when classes are balanced within a datasets, but fails when classes are very imbalanced, as is the case with detection problems. In this case the diagonal of a confusion matrix captures well false negatives, but false positives are absorbed into the grab-bag class *other* which contains a large majority of datapoints. To account for this one must also compare the *transpose* confusion matrix, where entry (i,j) represents the fraction of predicted instances of class i that belong to class j according to ground truth. This is demonstrated in figure 4.1 where we show the results for two synthetic prediction examples, both with the same mean on the confusion matrix diagonal but vastly different on the transposed confusion matrix. Effectively, the two matrices represent *recall* and *precision*, which we define below.

ROC: A receiver operating characteristic curve is a common tool to measure quality of detection results for a single class. It plots two values, the *true positive rate* $= TP/P$ and the *false positive rate* $= FP/N$. A single prediction output can be represented as a single point on the plot, however, most detection algorithms output scores (or probabilities) and all data points whose score is above a threshold are said to belong to the class under detection, so by varying the threshold one can obtain the full ROC curve. This again places little emphasis on false positives; in a dataset with millions of frames, a false positive rate of 1% means there are thousands of false positives, which may be considerably higher than the number of positives. For the purpose of quantifying actions between fruit flies, having more false positives than true positives of a behavior is unacceptable, hence the false positive rate is insufficient to measure the quality of an action detector.

Precision, Recall and F-score: A precision-recall curve is similar to an ROC curve; on one axis it plots *recall* $= TP/P$, which is the same as the true positive rate, but on the other axis it plots *precision* $= TP/PP = 1 - FP/PP$, which compares the false positives

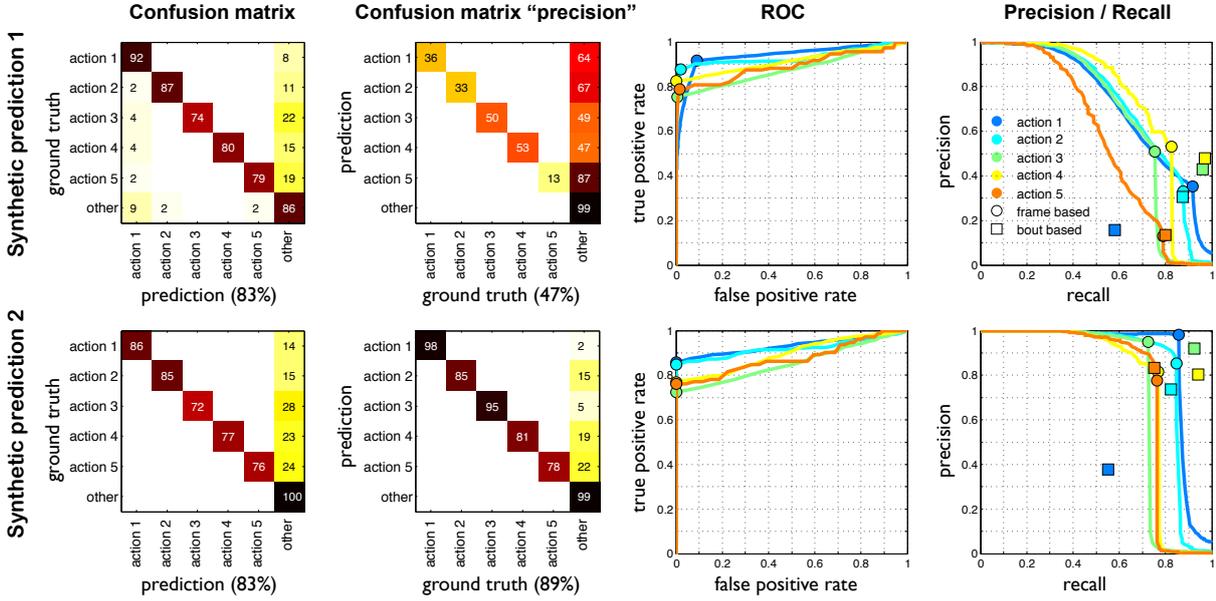


Figure 4.1: Results shown for two synthetic predictions compared to a single synthetic ground truth. The first two columns show how similar the confusion matrices are for the two examples, but how different their transposed confusion matrices are. The third column shows ROC curves for each class, and the fourth column demonstrates how precision recall curves are able to better reveal the discrepancy between the two examples (points on the curves represent the 0-threshold). The last column also shows how bout wise and frame wise measurements can differ.

to the total number of predicted positives rather than the negatives. As shown in figure 4.1, this measure emphasizes the difference between the two synthetic predictions better than the ROC curves. For ranking different methods we combine precision and recall into a single value using the F -score, defined as $F_\beta = (1 + \beta^2) \cdot \frac{\text{precision} \cdot \text{recall}}{\beta^2 \cdot \text{precision} + \text{recall}}$, which for $\beta = 1$ represents the harmonic mean. We prefer the harmonic mean over the standard mean as it favors balanced precision-recall combinations.

We conclude that a confusion matrix is a good choice for multi-class detection problems where classes are mutually exclusive, due to its ability to compare classes against each other, but one must also look at the transpose confusion matrix. However, for experiments such as ours, where classes may overlap and false positives are expensive, precision-recall curves are the best performance measurement tools.

4.2 Bout vs. frame

For behavior analysis, correctly counting the number of action instances is equally as important as correctly measuring the duration spent in an action, hence we must also measure performance at a bout level. To do that, we use an overlap criteria that deems a ground truth bout at interval $[s_g, e_g]$ and predicted bout at $[s_p, e_p]$ to match only if

$$\frac{\min(e_g, e_p) - \max(s_g, s_p)}{\max(e_g, e_p) - \min(s_g, s_p)} > \text{threshold}.$$

When multiple bouts fit that criteria, we match only the one with the highest ratio. A predicted output can have high precision-recall when measured on a per frame basis but low when measured on a per bout basis, and vice versa. This can be seen on the precision recall plots in Figure 4.1 and is explained by scenarios listed in Figure 4.2. We define F^* -score, a β -weighted mean of F1-frame and F1-bout, as a single metric for comparing different algorithms, where β can be used to control whether emphasis should be placed on bout- or frame-wise performance.

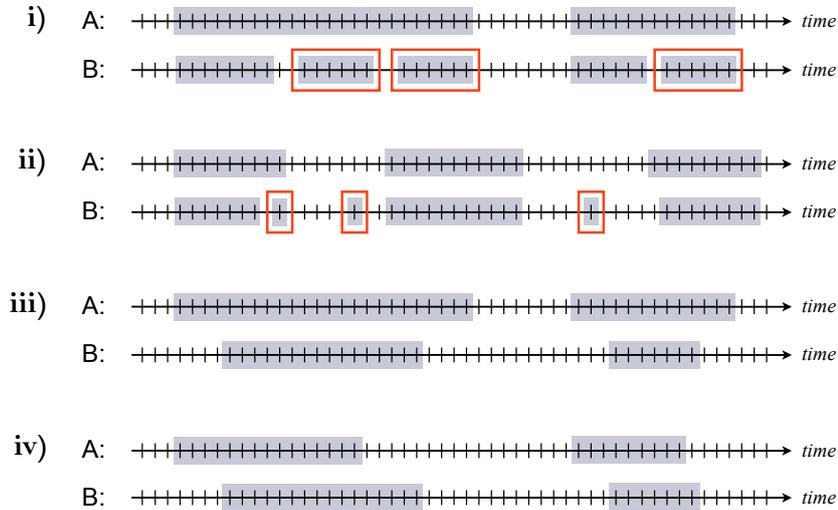


Figure 4.2: Scenarios explaining discrepancies in bout wise and frame wise performance:

- i) Under/over segmented bouts → lower bout than frame wise performance
- ii) Short missed/false detections → lower bout than frame wise performance
- iii) Under/over estimated duration → lower frame than bout wise performance
- iv) Offset bout boundaries → lower frame than bout wise performance

5

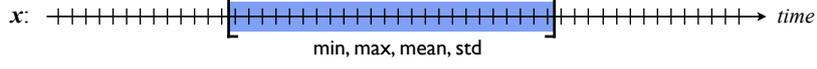
Action Detection

In this thesis we focus on action detection by exhaustive classification, in particular we compare two different architectures: *Sliding window detection* which refers to classifying fixed size windows that move frame-by-frame over a video sequence, and smoothing the predicted labels with post processing, and *structured output detection* 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, and because it is more general than a multi-class approach with respect to adding new classes of actions. Before describing the detection architectures in detail we define *bout features* that aggregate per frame features over an interval of frames, and are used in our implementation of both detection schemes.

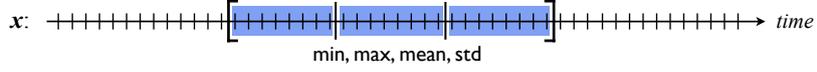
5.1 Bout features

We define a number of bout-level features that are designed to extract statistical patterns from an interval of per frame features, and emphasize the similarities of bouts within an action class, invariant of their durations. A bout feature $\psi_k(x, t_{start}, t_{end})$ is a function of sequence x and interval $[t_{start} \ t_{end}]$; we consider the following:

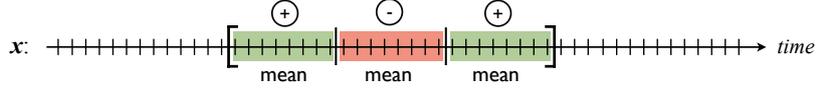
Bout statistics features capture statistics of frame-features over an interval and can be expressed as: $\text{op}(x(t_{start} : t_{end}))$, where $\text{op} \in \{\min, \max, \text{mean}, \text{std}\}$.



Temporal region features capture statistics over r equal subintervals, and are meant to handle the differences within an action composed of r sub actions. They can collectively be expressed as: $\{\text{op}(x(t_{start} + (i-1)\text{dt} : t_{start} + i\text{dt}-1))\}_{i \in \{1, \dots, r\}}$, where $\text{dt} = (t_{end} - t_{start} + 1)/(r-1)$ and $\text{op} \in \{\min, \max, \text{mean}, \text{std}\}$.

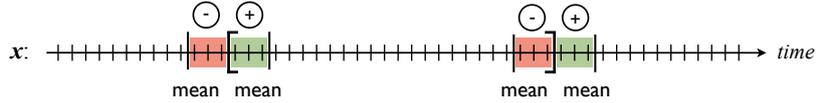


Harmonic features are meant to capture harmonic actions and can be expressed as: $\sum_{i=1}^r (-1)^i \text{mean}(x(t_{start} + (i-1)\text{dt} : t_{start} + i\text{dt}-1))$, where $\text{dt} = (t_{end} - t_{start} + 1)/(r-1)$.



Boundary difference features emphasize the change in features at the start and end of a bout, and help with locating boundaries. They are expressed as:

$$\text{mean}(x(t_{start/end} : t_{start/end} + \text{dt})) - \text{mean}(x(t_{start/end} - \text{dt} : t_{start/end})).$$



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 $\text{op} \in \{\min, \max, \text{mean}\}$.

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.

With K total 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 . In our experiments we use two temporal region splits, $n \in \{2, 3\}$, and set the number of histogram bins to be 2^3 , resulting in a total of $K = 48$ bout features.

5.2 Sliding window framework

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 an iterative 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,

$$P(x(t)|z(t) = 1) := 1/(1+\exp(-\text{score}(t))), P(x(t)|z(t) = 0) := 1 - P((t)|z(t) = 1),$$

compute priors, $P(z(1) = c)$, and transition matrix, $P(z(t+1) = c_i|z(t) = c_j)$, from the training data, and run the Viterbi algorithm [24] to find the most probable frame-wise sequence of actions. For comparison with previous work [15, 17] we also consider box filtering and Auto-context as a way to smoothen the output.

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 [25]. 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.

5.3 Structured output framework

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 [26] 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 is used both by the *training* algorithm and the *inference* algorithm, and a *loss function*, $\mathcal{L}(y, \hat{y})$, which penalizes wrong segmentations and steers the training algorithm.

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 [26] 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 [27] 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:

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

The 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 , which is 0 if the duration is within that action's standard range (observed in training) and grows exponentially with its distance from the range.

Loss function: The loss function compares a ground truth segmentation y with a predicted segmentation \hat{y} and penalizes intervals where the two disagree. It should be constructed such that when the loss is small, then the results from the inference are satisfactory. Since our experiments focus on detecting actions of flies, our objective is to maximize the precision and recall of the actions on a bout-level, while maintaining good per frame accuracy. We define the function 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 effectively 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.

Training: We extract bout features from the positive bouts, $\{(s^{j_i}, e^{j_i})\}_i$, 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 an iterative training process that gradually adds useful negative samples. At each iteration we train a classifier on the current training data, run inference with 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.

6

Experiments and Analysis

In this chapter we present results for each described method on the Fly-vs-Fly dataset. First, we show how detections of trained human annotators compare to those of experts, to give an idea of the difficulty of each action, then we discuss model selection and compare and analyze the performance of each method, and finally we show how the best method performs on CRIM13 and compare it to published results.

6.1 Human vs. human

We trained novice annotators to learn to detect each action 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 portion of the test dataset, completing all movies for Boy meets boy, 1/10 of movies for Aggression, and 4/15 of Courtship movies, enough to give an idea about the difficulty of each action. Overall, the trained annotators achieved best performance on the Courtship sub-dataset, with bout wise precision-recall over 90% for each action, on Boy meets boy their performance was in the 70 – 100% range, and on Aggression recall was generally above 80% but precision in the 60 – 80% range. The annotators described Courtship as being easier to annotate than the other two sub-datasets, with actions seemingly less ambiguous, and the lower precision on Aggression than Boy meets boy can partly be explained by the high commotion between flies as well as lower spatial and temporal video resolution, and partly due to potentially missed detections by the original annotator. Figure 6.1 shows the bout- and frame wise

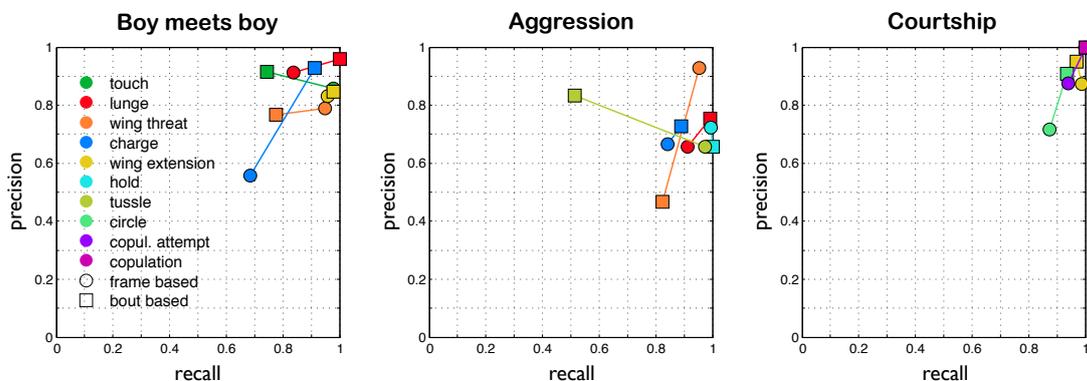


Figure 6.1: Human performance measured in terms of frame based (circles) and bout based (squares) precision-recall. Note the large discrepancies between bout based and frame based performance for a few of the actions, which is generally caused by over- or under-segmentation of bouts, or due to the short duration of misclassified bouts.

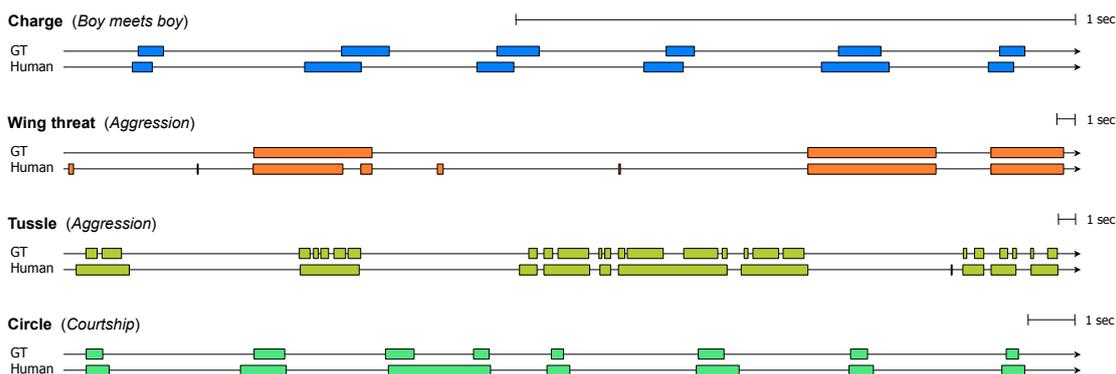


Figure 6.2: Segmentation samples from video sequences, explaining the high bout vs. frame performance discrepancy of selected actions. The upper sequence represents the ground truth (expert) annotations and the lower shows those of our trained annotators.

precision-recall for each action in the Fly-vs-Fly dataset. Actions that stand out from the average are lunge (BMB) and charge (BMB), with over 90% bout wise performance, charge (BMB) and circle, with much lower frame- than bout wise performance, and tussle and wing threat with lower bout- than frame wise performance. Figure 6.2 shows action segmentation samples that explain this bout- vs. frame wise performance discrepancy. 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 performance as humans.

6.2 Method comparison

We compare five action detection methods: frame based SVM, window based SVM, and JAABA backend [17], which all fall under the window based detector framework described in Chapter 5.2, structured SVM (5.3) and semi-structured SVM (5.3).

Model selection: All of the implemented methods have free parameters that affect their outcome, such as the weight of regularization term in the optimization function, the error tolerance for optimization termination, and threshold on output scores. These parameters were set using a parameter sweep on the training dataset, such that optimal bout- and frame wise precision and recall combination was achieved.

In order to account for over-segmentation of the window based SVM, we tested three post-processing methods: Auto-context as described in [15], box-filtering which is proposed as a post-processing step for JAABA [28], and HMM fitting. Table 6.1 shows the test performance of each approach, averaged over all actions, and from the results we can infer that: HMM outperforms Auto-context and box-filtering; post-processing improves bout wise performance for each method; and bout features significantly improve performance upon per-frame features. For the remainder of this chapter, all results are reported with post-processing applied.

For comparison with JAABA we trained detectors on their bout features, plugging their Boosting implementation into the learning and inference modules of our window based framework. JAABA as presented in [17] does not include post processing, but we include it here for a fair bout wise performance comparison.

	<i>Bout wise</i>			<i>Frame wise</i>			F*-score
	Recall	Precision	F1-score	Recall	Precision	F1-score	
Window based SVM	0.90	0.46	0.61	0.73	0.74	0.74	0.67
+ Autocontext	0.88	0.50	0.64	0.74	0.76	0.75	0.69
+ filtering	0.84	0.63	0.72	0.71	0.74	0.72	0.72
+ HMM	0.85	0.73	0.79	0.76	0.78	0.77	0.78
Frame based SVM	0.80	0.32	0.46	0.55	0.55	0.55	0.50
+ HMM	0.54	0.56	0.55	0.77	0.38	0.51	0.53
JAABA	0.90	0.50	0.65	0.79	0.78	0.79	0.71
+ filtering	0.86	0.67	0.75	0.79	0.78	0.78	0.77

Table 6.1: Comparison of post-processing schemes for the window based framework. Results show that post-processing of any kind improves performance for all of the methods, and that bout features significantly improve performance over per-frame features.

6.2 Method comparison

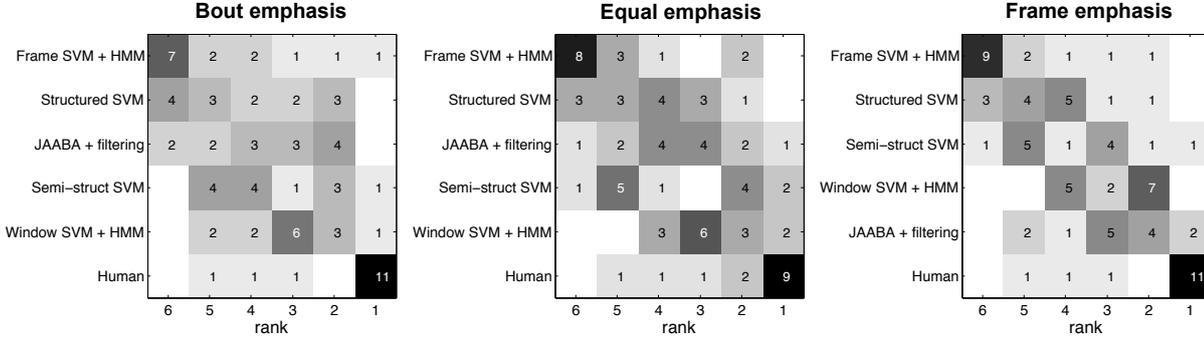


Figure 6.3: Histogram of method ranks over all actions, from F^* -score computed with varying emphasis (bout-equal-frame). Methods are ordered based on mean rank.

Performance: To measure the performance of our action detectors, we compute precision, recall, and F1-score, on a frame- and bout-wise level, and the F^* -score with equal emphasis on bout and frame, which can be used to rank the different methods. Figures 6.9 - 6.11 (at the end of this chapter) show how the different methods compare on each action, showing these measures as well as count- and duration consistency which are representative of what biologists use for quantifying actions in their experiments.

The results show considerable variation in method rank depending on the action, so to get a holistic view we consolidate the results in detector rank histograms, which show the number of times each detector achieved each rank and order methods according to their mean rank. Figure 6.3 shows detector rank histograms for ranking based on F^* -scores with varying β , emphasizing bout performance, frame performance, and the two equally. Through varying emphases ranking stays consistent, placing the window based SVM at the top (after humans), followed by semi-structured SVM, JAABA, structured SVM, and frame based SVM at the bottom, with the exception being that JAABA moves up to first place when emphasis is placed on frame wise performance.

For a finer resolution view of how the methods line up we show the mean F-scores, averaged over all actions, and the performance as a function of time it takes to run the detector on 1 million frames, in Figure 6.4. This view preserves the rank observed in Figure 6.3, but it also shows that most methods cluster around the 70% performance, apart from humans at around 85% and frame based SVM at around 50%. In addition, it shows that the window based methods, performing slightly better than the structured output counterparts, are orders of magnitude faster.

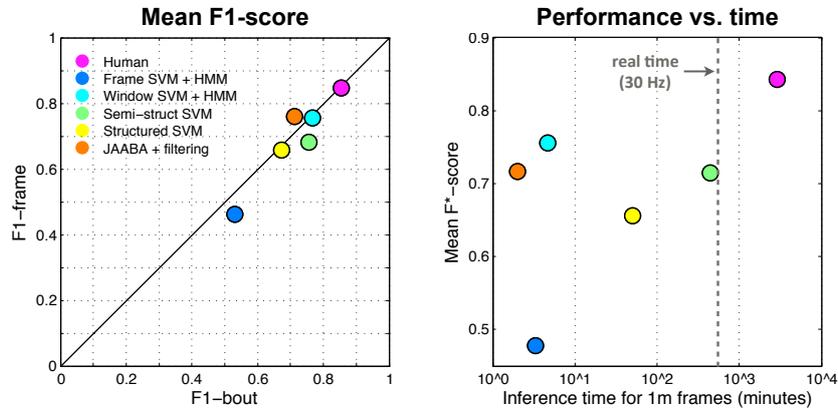


Figure 6.4: Left: Comparison of F1 scores of each method, averaged over all actions. Right: F*-score of each method as a function of approximate inference time.

Analysis: The summary measures described above abstract away information about performance patterns that may exist between methods and actions. To explore that, we cluster methods based on their F*-score on each action, by first applying principal component analysis to the $\#methods \times \#actions$ F*-matrix, eliminating correlations, and then applying k-means to the dimension-reduced matrix, splitting methods into 4 groups. Similarly to the detector rank histograms we perform this analysis with emphasis on bout, frame, or equal emphasis, to see whether clusters remain consistent (see Figure 6.5). With equal- and frame wise emphasis, the window based SVM and JAABA form a group, the structured output methods form another one, and humans and frame based SVM are singletons. With emphasis on bout performance, the window based SVM, semi-structured SVM and humans form one group while the remaining methods are singletons. A perhaps more interesting grouping is the converse, where we cluster actions into 4 clusters based on the transposed F*-matrix. A group that is consistent across emphases contains lunge, charge, and copulation attempt, which all share the characteristic of being short and concise but poorly captured by the frame based detector. Wing threat and tussle are also consistently grouped together, both sharing the property of high appearance- and duration variation. Wing extension and touch are almost consistently grouped together, they can both be somewhat defined by a feature (wing angle, leg distance) exceeding a threshold for an extended period of time, and on both actions the frame based detector performs competitively. The final notable observation is that copulation is a singleton in the bout wise grouping, being the only action with near-perfect performance by all methods.

6.2 Method comparison

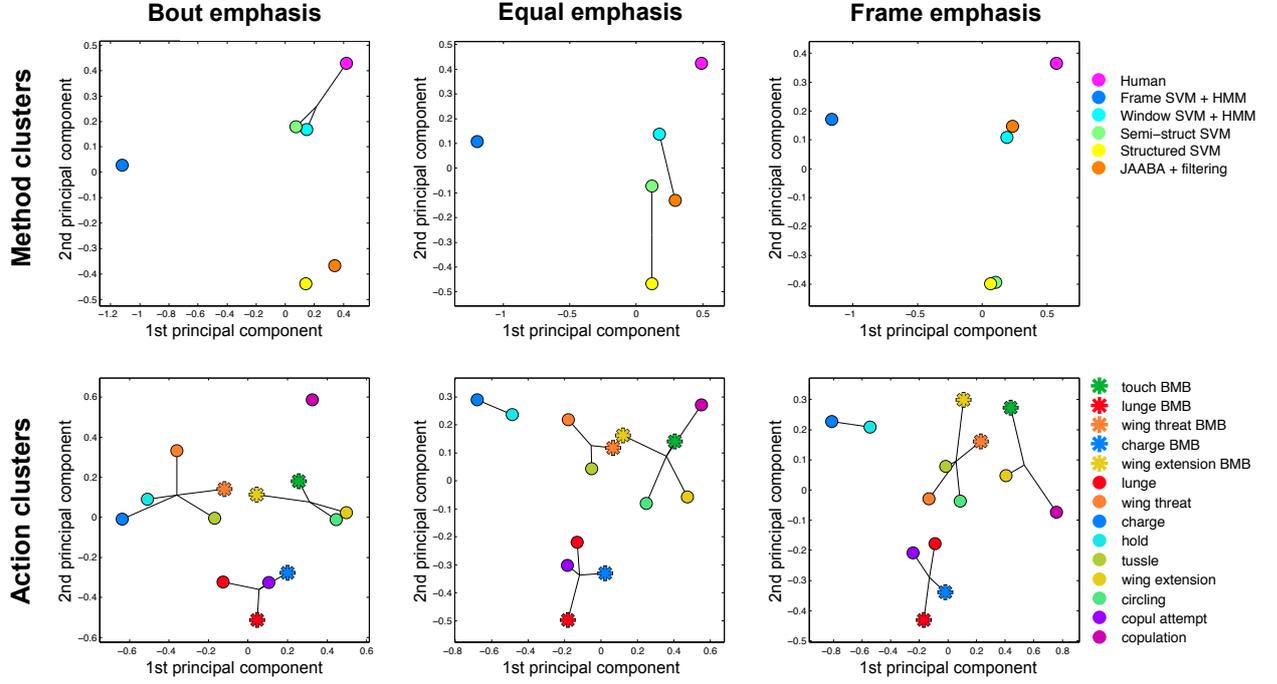


Figure 6.5: Upper row: clustering of methods based on their F^* -score on all actions, showing that the window based methods form a group and the structured ones form another, except when emphasis is on bout performance, in which case window based and semi structured SVM group with humans. Lower row: clustering of actions based on their F^* -score by each method, showing that short and concise actions are grouped together and actions across datasets generally belong to the same cluster.

For a closer look at the learnt detector models, we show the highest weighing features for each of the SVM based classifiers, ordering features based on the sum of the absolute of all bout features derived from each frame wise feature. Figure 6.6 shows the top 5 features for each action, grouping per frame features with their derivatives, and the 3 methods together for each action, and it can be seen that similar features tend to be selected by different methods, and across sub-datasets. The cumulative vote over methods and derivative orders deems the following features as most important:

touch:	<i>leg distance, fg body ratio</i>	lunge:	<i>axis ratio</i>
wing threat:	<i>wing angles, wing length</i>	charge:	<i>min wing angle, velocity</i>
wing extension:	<i>wing angles, wing length</i>	hold:	<i>dist. to other, facing angle</i>
tussle:	<i>dist. to other, angle between</i>	circle:	<i>velocity, angular velocity</i>
copul. attempt:	<i>dist. to other, axis ratio</i>	copul.:	<i>contrast, angle between</i>

6.3 Performance on CRIM13

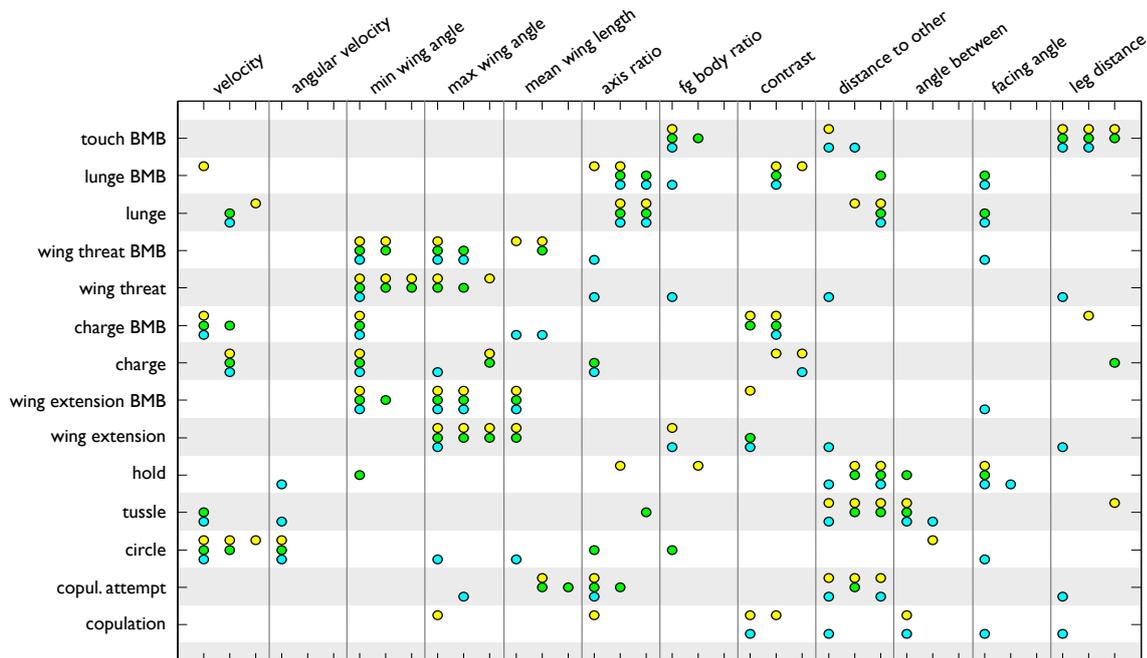


Figure 6.6: Top 5 contributing features for the detectors of each action shown. Perframe features are grouped together with their 1st and 2nd derivatives (in that order), and the different rows per action show the selection for window based (yellow), semi-structured (green), and structured (cyan) SVMs.

In addition we analyzed which type of bout features were most used, and found that the window based detector made most use of the bout statistics and histogram features, while the others used boundary dependent features to a similar extent.

6.3 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 [15].

The dataset is labeled such that the 13 actions classes are non-overlapping, and the detection problem is treated as a multi-class classification problem. In order to match that we convert our binary action detectors to a single multi-class detector by fitting them to a HMM with 13 states: First, we train a separate detector for each action, including the null action. Then, we convert the scores from each detector to posterior probabilities, the same way as we did before, and normalize them at each frame such

that the probabilities of all actions sum to 1. Finally, we apply the Viterbi algorithm on the frame-wise posteriors, and the transition matrix and class priors computed from the training labels, obtaining a unique class label for each frame. Results on CRIM13 are presented in terms of a frame wise confusion matrix, and its diagonal mean, so in order to compete with their performance we optimize our multi-class detector accordingly. We do that by shifting the scores output from each individual class detector, before converting the scores to posterior probabilities, by subtracting scalars that are found by greedy optimization on the training set, maximizing the diagonal mean of the confusion matrix. Since the diagonal of the confusion matrix effectively measures only the recall of each action, but we are also interested in precision, we do the same type of shift optimization with respect to the mean F1-score, computed from the diagonals of the confusion- and transposed confusion matrices.

Figure 6.7 shows the performance of our method using both optimization schemes, and compares it to the method used in [15], and Figure 6.8 shows the confusion matrices, and the transposed confusion matrices, for both optimization schemes using our method. It can be seen that the recall-optimized shift results in a higher diagonal mean on the confusion matrix, but a more imbalanced precision-recall. In our experiment we use only the tracking features provided for CRIM13, but the best results reported on CRIM13 additionally used spatio-temporal features (STF). Figure 6.7 shows that our results are competitive with their full method, and considering only the tracking features we outperform their reported results by 3.4%.

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

Figure 6.7: Comparison of the window based SVM to the methods used in [15], showing performance on the CRIM13 test dataset.

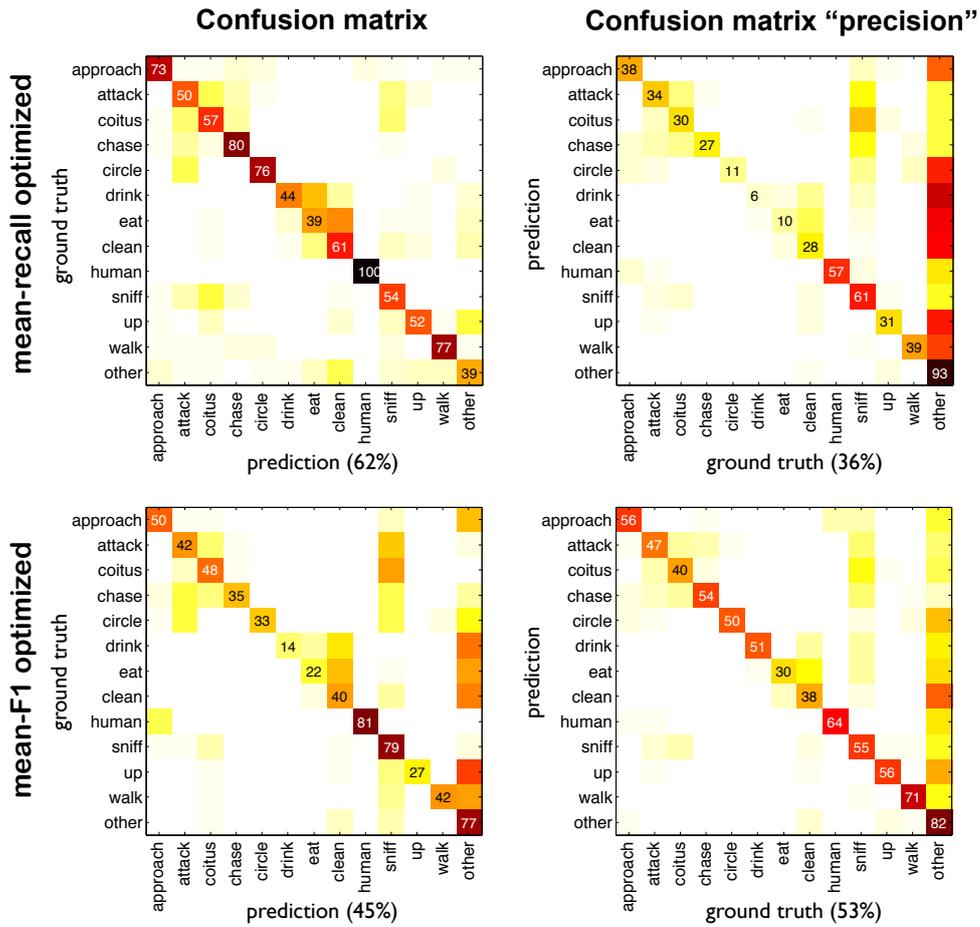


Figure 6.8: Confusion matrices for Window SVM + HMM on the CRIM13 test dataset. The upper row shows performance obtained by optimizing the diagonal mean on the confusion matrix and the lower row performance obtained by optimizing the mean diagonal F1-score, which results in a more balanced precision and recall.

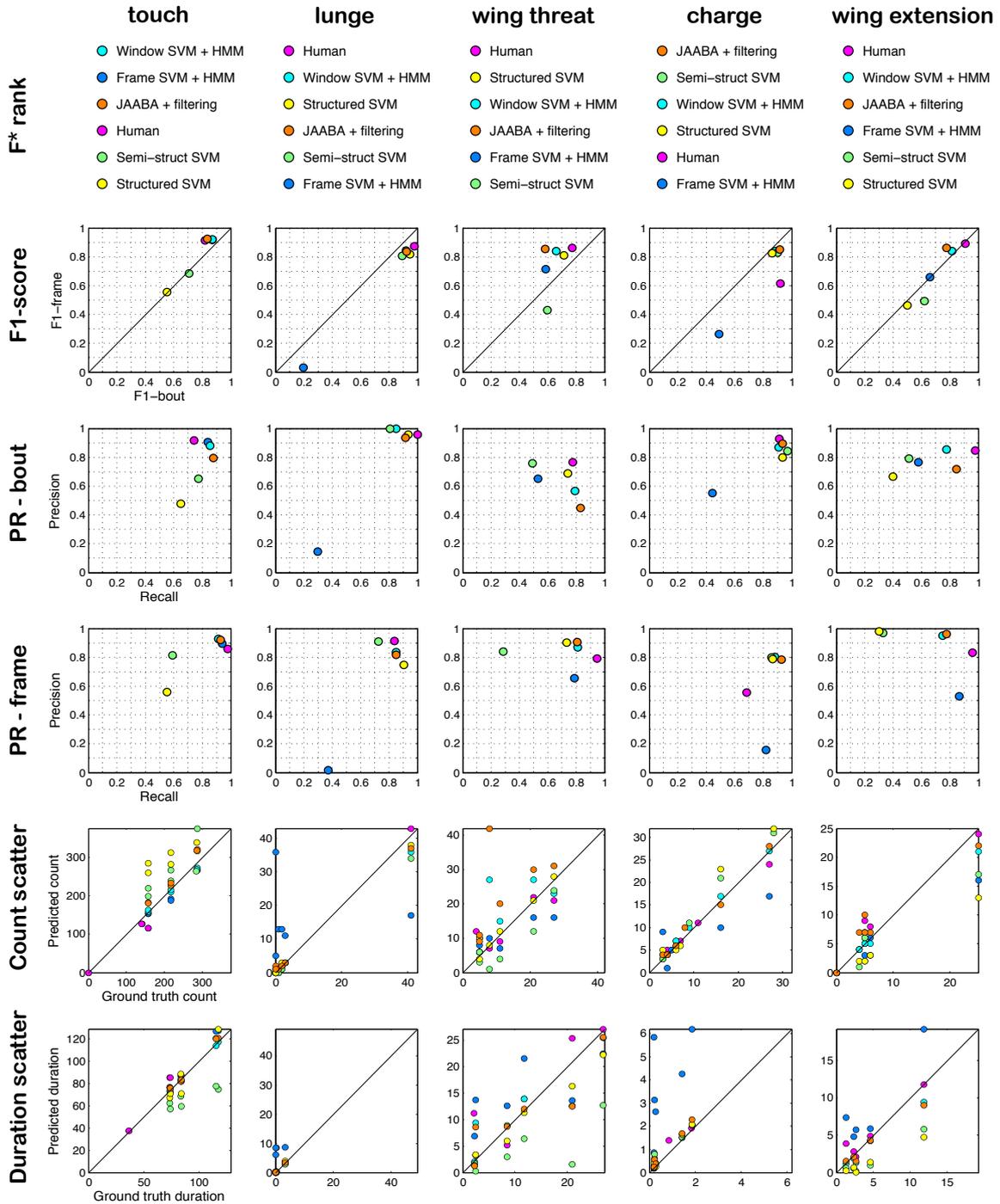


Figure 6.9: Results for **Boy meets boy**, showing F*-based method rank, F1-scores, bout- and frame wise precision-recall, and count- and duration scatter, for each action.

6.3 Performance on CRIM13

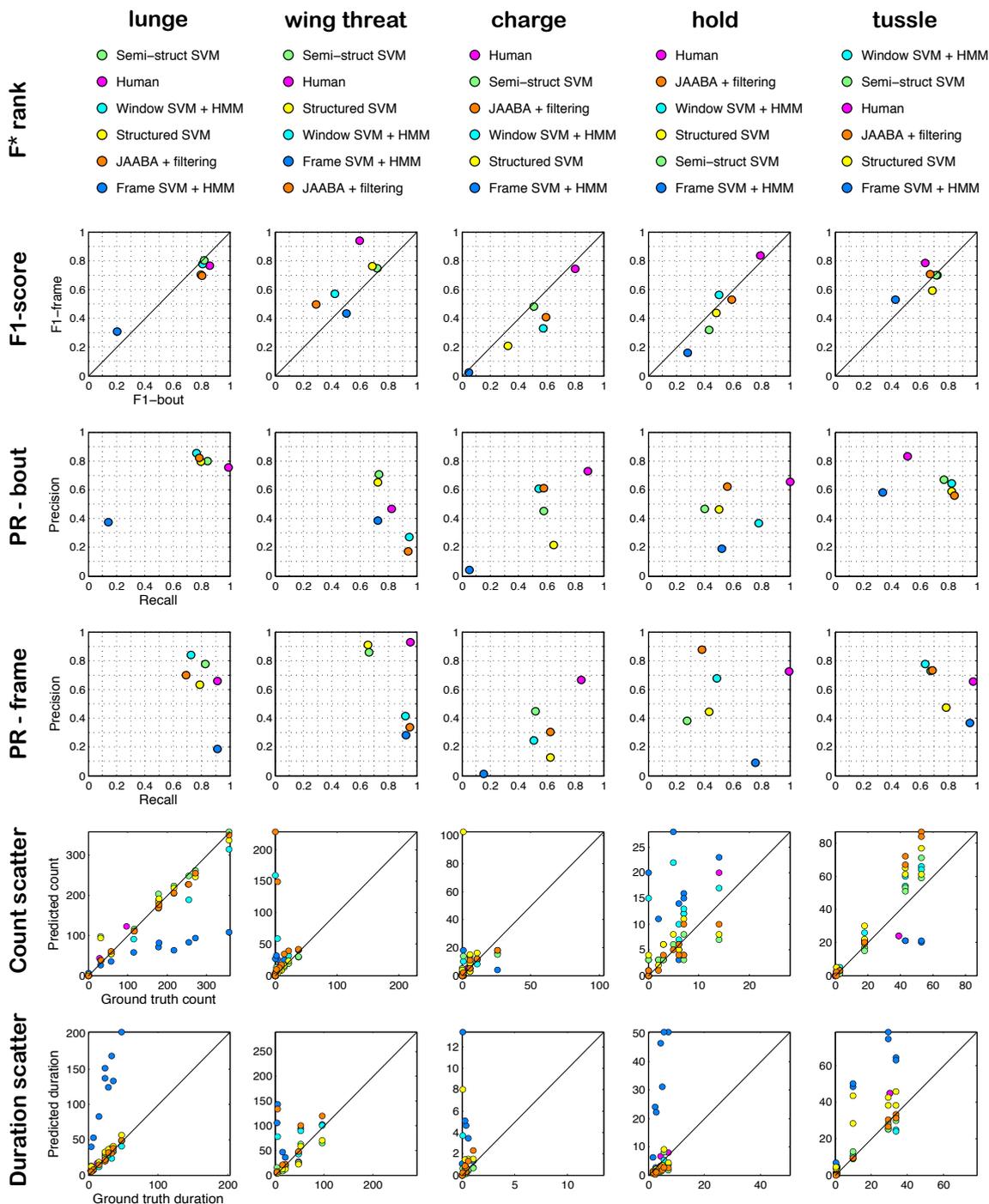


Figure 6.10: Results for **Aggression**, showing F*-based method rank, F1-scores, bout- and frame wise precision-recall, and count- and duration scatter, for each action.

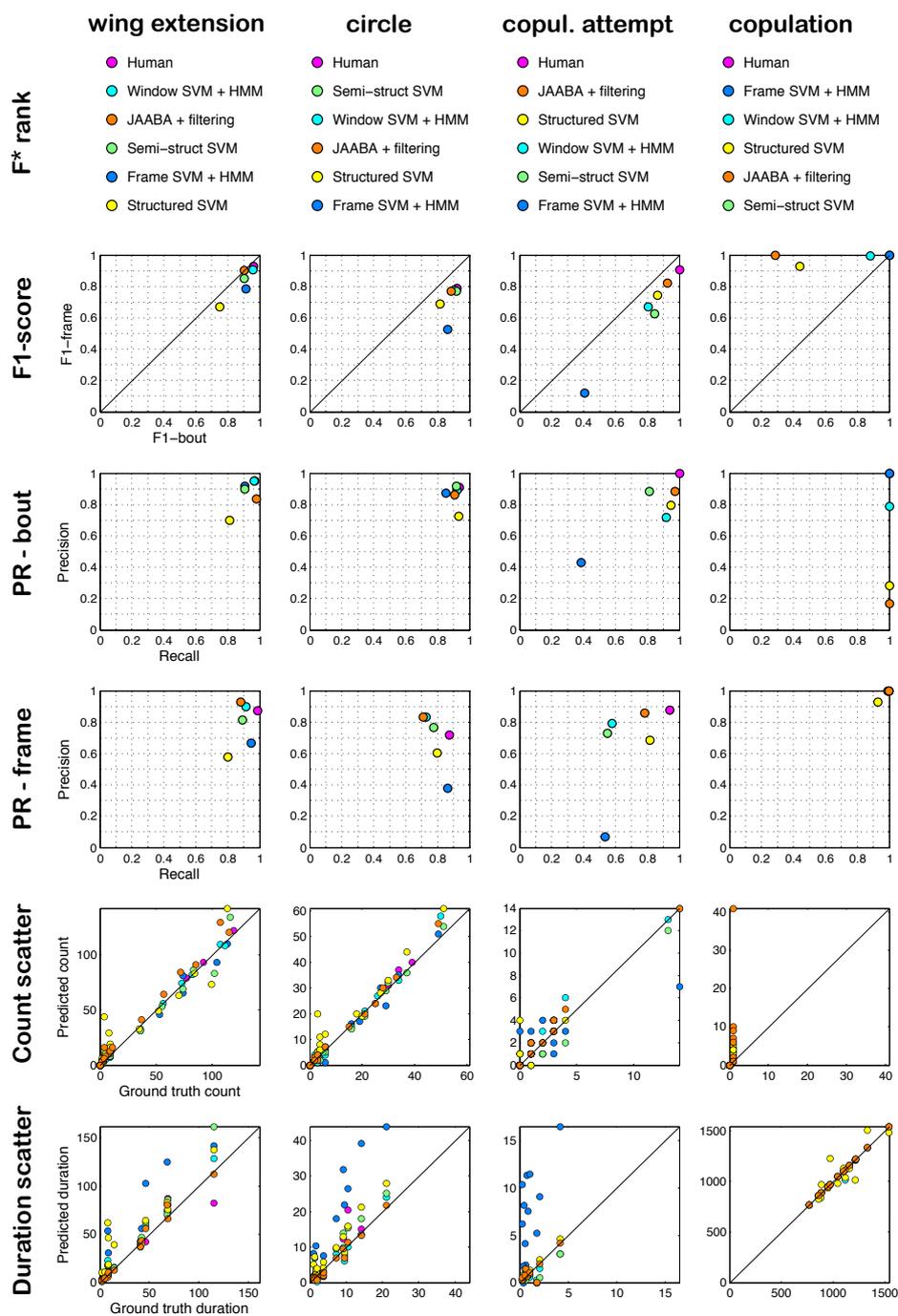


Figure 6.11: Results for **Courtship**, showing F*-based method rank, F1-scores, bout- and frame wise precision-recall, and count- and duration scatter, for each action.

7

Discussion

In this thesis, we have described an end-to-end system for detecting actions of fruit flies: We introduced a new dataset, Fly-vs-Fly, which we used for training and testing the system, described a tracking tool used for parameterizing the videos, and compared and analyzed the performance of several action detection algorithms. We tested our approach on the most recently published animal dataset, and showed that our action detection approach achieved performance at least as good as that of reported results.

What we have learnt from our experiments is that for most actions, bout features provide a great performance gain over per frame features, and for bout wise performance, post processing is essential for window based detectors, HMM fitting being the most promising. Our experiments showed that overall, window based detectors work as well or better than structured ones, but are orders of magnitude faster. This is contrary to what we would have expected, which is that, especially for composite actions, the structured output methods would perform better as they allow for elastic sized windows which should in theory better capture the structure within bouts. However, this can potentially be explained by the fact that the actions with most structure in the dataset, such as lunge and copulation attempt, have low duration variation, so with bout features that capture well the structure within a window, fixed sized windows may suffice. Structured output methods also suffer from the problem of over-segmenting long bouts of actions that do not have much structure, which leads to a lower bout wise performance. The highest performing method, window based SVM+HMM, had an overall bout- and frame precision-recall combination of 76%, averaged over all actions, while trained human annotators achieved 84%.

Something to consider is that human annotations are never perfect. There is always the potential that an annotator overlooks action instances, and in our case, since annotations are binary, ambiguous bouts are included based on the flip of a coin. This causes a problem both during training of classifiers, and when estimating the performance of the trained detectors. An ideal solution would be to have multiple annotators label each video such that labeled bouts would have certainty associated with them, which could be used in training and in performance computations - placing less emphasis on low certainty bouts. However, obtaining such thorough annotations is expensive.

In future work, we would like to abandon the approach of detection by exhaustive classification, and focus on developing more hierarchical methods that find interest points within the video and center classification around those. We would like to apply unsupervised learning on the domain, such that fewer training samples will be needed to train good classifiers and unlabeled bouts will have less of an impact. To capture the intraclass variability of actions, a considerable number of labeled action samples will always be needed, so in order to bridge the gap between few training samples and good classifiers, we see training with humans in the loop as a viable option.

References

- [1] 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*, **3**, pages 32–36. IEEE, 2004. 2
- [2] LENA GORELICK, MOSHE BLANK, ELI SHECHTMAN, MICHAL IRANI, AND RONEN BASRI. **Actions as Space-Time Shapes.** *Transactions on Pattern Analysis and Machine Intelligence*, **29**(12):2247–2253, December 2007. 2
- [3] MARCIN MARSZALEK, IVAN LAPTEV, AND CORDELIA SCHMID. **Actions in Context.** In *IEEE Conference on Computer Vision & Pattern Recognition*, 2009. 2
- [4] 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*, pages 392–405. Springer, 2010. 2
- [5] 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. 2
- [6] KHURRAM SOOMRO, AMIR ROSHAN ZAMIR, AND MUBARAK SHAH. **UCF101: A Dataset of 101 Human Actions Classes From Videos in The Wild.** *arXiv preprint arXiv:1212.0402*, 2012. 2
- [7] SAAD ALI, ARSLAN BASHARAT, AND MUBARAK SHAH. **Chaotic invariants for human action recognition.** In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8. IEEE, 2007. 3
- [8] LEONID SIGAL AND MICHAEL J BLACK. **Humaneva: Synchronized video and motion capture dataset for evaluation of articulated human motion.** *Brown University TR*, **120**, 2006. 3
- [9] 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*, pages 842–849. IEEE, 2012. 3
- [10] HEMA SWETHA KOPPULA, RUDHIR GUPTA, AND ASHUTOSH SAXENA. **Learning human activities and object affordances from rgb-d videos.** *arXiv preprint arXiv:1210.1207*, 2012. 3
- [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*, pages 3153–3160. IEEE, 2011. 3
- [12] SANG MIN OH, JAMES M REHG, TUCKER BALCH, AND FRANK DELLAERT. **Learning and inferring motion patterns using parametric segmental switching linear dynamic systems.** *International Journal of Computer Vision*, **77**(1-3):103–124, 2008. 3
- [13] P. DOLLÁR, V. RABAUD, G. COTTRELL, AND S. BELONGIE. **Behavior Recognition via Sparse Spatio-Temporal Features.** In *VS-PETS*, October 2005. 3
- [14] HUEIHAN JHUANG, ESTIBALIZ GARROTE, XINLIN YU, VINITA KHLNANI, TOMASO POGGIO, ANDREW D STEELE, AND THOMAS SERRE. **Automated home-cage behavioural phenotyping of mice.** *Nature communications*, **1**:68, 2010. 3, 4
- [15] XAVIER P BURGOS-ARTIZU, PIOTR DOLLÁR, 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*, pages 1322–1329. IEEE, 2012. 3, 4, 21, 27, 31, 32
- [16] HEIKO DANKERT, LIMING WANG, ERIC D HOOPFER, DAVID J ANDERSON, AND PIETRO PERONA. **Automated monitoring and analysis of social behavior in Drosophila.** *Nature methods*, **6**(4):297–303, 2009. 4, 13
- [17] MAYANK KABRA, ALICE A ROBIE, MARTA RIVERA-ALBA, STEVEN BRANSON, AND KRISTIN BRANSON. **JAABA: interactive machine learning for automatic annotation of animal behavior.** *nature methods*, 2012. 4, 13, 21, 27
- [18] YASEMIN ALTUN, IOANNIS TSOCHANTARIDIS, THOMAS HOFMANN, ET AL. **Hidden markov support vector machines.** In *ICML*, **3**, pages 3–10, 2003. 4
- [19] 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*, pages 3265–3272. IEEE, 2011. 4
- [20] QINFENG SHI, LI CHENG, LI WANG, AND ALEX SMOLA. **Human action segmentation and recognition using discriminative semi-Markov models.** *International journal of computer vision*, **93**(1):22–32, 2011. 4
- [21] MASSIMO PICCARDI. **Background subtraction techniques: a review.** In *Systems, man and cybernetics, 2004 IEEE international conference on*, **4**, pages 3099–3104. IEEE, 2004. 12
- [22] ARTHUR P DEMPSTER, NAN M LAIRD, AND DONALD B RUBIN. **Maximum likelihood from incomplete data via the EM algorithm.** *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 1–38, 1977. 12
- [23] HAROLD W KUHN. **The Hungarian method for the assignment problem.** *Naval research logistics quarterly*, **2**(1-2):83–97, 1955. 13

REFERENCES

- [24] ANDREW VITERBI. **Error bounds for convolutional codes and an asymptotically optimum decoding algorithm.** *Information Theory, IEEE Transactions on*, **13**(2):260–269, 1967. 21
- [25] RONG-EN FAN, KAI-WEI CHANG, CHO-JUI HSIEH, XIANG-RUI WANG, AND CHIH-JEN LIN. **LIBLINEAR: A library for large linear classification.** *The Journal of Machine Learning Research*, **9**:1871–1874, 2008. 22
- [26] IOANNIS TSOCHANTARIDIS, THORSTEN JOACHIMS, THOMAS HOFMANN, AND YASEMIN ALTUN. **Large margin methods for structured and interdependent output variables.** In *Journal of Machine Learning Research*, pages 1453–1484, 2005. 22
- [27] RICHARD BELLMAN. **Dynamic programming and Lagrange multipliers.** *Proceedings of the National Academy of Sciences of the United States of America*, **42**(10):767, 1956. 22
- [28] <http://jaaba.sourceforge.net> [online]. 27