2016

3D pose estimation of flying animals in multi-view video datasets

BOSTON UNIVERSITY

GRADUATE SCHOOL OF ARTS AND SCIENCES

Dissertation

3D POSE ESTIMATION OF FLYING ANIMALS IN MULTI-VIEW VIDEO DATASETS

by

MIKHAIL BRESLAV

B.S., The Pennsylvania State University, 2008
M.S., The Pennsylvania State University, 2010

Submitted in partial fulfillment of the

requirements for the degree of

Doctor of Philosophy

2016

Approved by

First Reader

Margrit Betke, PhD
Professor of Computer Science

Second Reader

Stan Sclaroff, PhD
Professor of Computer Science

Third Reader

Tyson L. Hedrick, PhD
Associate Professor of Biology
University of North Carolina at Chapel Hill

“The mountains are calling and I must go.”
-*John Muir*

Acknowledgments

First and foremost, I would like to thank my family, especially my parents. Without their hard work, sacrifices, and support, none of this would be possible.

I would like to thank my advisor Margrit Betke for her support and encouragement over the years. I am grateful to her for giving me the amazing opportunity to study at Boston University. I have enjoyed a great amount of intellectual freedom during my PhD and I am thankful for that. I would like to thank Stan Sclaroff for his time and detailed feedback. I would like to thank Ty Hedrick for not only his willingness to collaborate, but his responsiveness in the process. Through the act of sharing data, Ty has allowed me to explore new kinds of problems, for which I am very grateful. I also thank Victor Ortega-Jimenez for his involvement in the creation of hawkmoth video data. I thank Nathan Fuller for his help with my research on bats, as well as everyone else who helped with data collection endeavors.

I thank Kate Saenko for serving on my defense committee. I would like to thank Wayne Snyder for serving as the chair of my defense. I have also had the pleasure to serve as a teaching fellow for Wayne on several occasions, and it has been eye opening to work with someone who so strongly cares about his students.

Over the six years I've been a PhD student I have spent the day to day surrounded by great colleagues and friends. I am grateful to them for their help and support over the years.

I would like to thank current and former members of the Image and Video Computing group for their valuable feedback. In particular, thank you Qinxun Bai, Sarah Bargal, Fatih Cakir, Sam Epstein, Wenxin Feng, Danna Gurari, Kun He, Toni Hernandez, Ajjen Joshi, Chris Kwan, Liliana Lo Presti, Shugao Ma, John Magee, Eric Missimer, Mehrnoosh Sameki, Ashwin Thangali, Diane Theriault, Zheng Wu, and Jianming Zhang. I would also like to thank Matthew Danish, Ye Li, Zhiqiang Ren,

Larissa Spinelli, and many other students, staff, and faculty at Boston University that have helped me over the years.

I have made some amazing friendships during my time in Boston and I owe much of my happiness and who I have become to them. Thank you for all the adventures and making me a better person: Andrew Berman, Matthew Danish, Hsiang-Ting Huang, Carsten Maedler, Javad Noorbakhsh, and Larissa Spinelli. Thank you Boston for being my home for the last six years, I could not have asked for a better place to spend this period of my life. Thank you to all my other friends both in Boston and elsewhere.

3D POSE ESTIMATION OF FLYING ANIMALS IN MULTI-VIEW VIDEO DATASETS

MIKHAIL BRESLAV

Boston University, Graduate School of Arts and Sciences, 2016

Major Professor: Margrit Betke, Professor of Computer Science

ABSTRACT

Flying animals such as bats, birds, and moths are actively studied by researchers wanting to better understand these animals' behavior and flight characteristics. Towards this goal, multi-view videos of flying animals have been recorded both in laboratory conditions and natural habitats. The analysis of these videos has shifted over time from manual inspection by scientists to more automated and quantitative approaches based on computer vision algorithms.

This thesis describes a study on the largely unexplored problem of 3D pose estimation of flying animals in multi-view video data. This problem has received little attention in the computer vision community where few flying animal datasets exist. Additionally, published solutions from researchers in the natural sciences have not taken full advantage of advancements in computer vision research. This thesis addresses this gap by proposing three different approaches for 3D pose estimation of flying animals in multi-view video datasets, which evolve from successful pose estimation paradigms used in computer vision. The first approach models the appearance of a flying animal with a synthetic 3D graphics model and then uses a Markov Random Field to model 3D pose estimation over time as a single optimization problem. The second approach builds on the success of Pictorial Structures models and further improves them for the case where only a sparse set of landmarks are annotated in training data. The proposed approach first discovers parts from regions of the training images that are not annotated. The discovered parts are then used to generate more accurate appearance likelihood terms which in turn produce more accurate landmark localizations. The third approach takes advantage of the success of deep learning models and adapts existing deep architectures to perform landmark localization. Both the second and third approaches perform 3D pose estimation by first obtaining accurate localization of key landmarks in individual views, and then using calibrated cameras and camera geometry to reconstruct the 3D position of key landmarks.

This thesis shows that the proposed algorithms generate first-of-a-kind and leading results on real world datasets of bats and moths, respectively. Furthermore, a variety of resources are made freely available to the public to further strengthen the connection between research communities.

Contents

List of Tables

List of Figures

List of Abbreviations

2D	Two Dimensional
3D	Three Dimensional
3D-PEB	3D Pose Estimation of Bats
BOW	Bag Of Words
CNN	Convolutional Neural Network
DLT	Direct Linear Transformation
DNN	Deep Neural Network
DOF	Degrees of Freedom
DPM	Deformable Part Models
FC	Fully Connected
FFT	Fast Fourier Transform
FPS	Frames Per Second
GPU	Graphics Processing Unit
HOG	Histogram of Oriented Gradients
HRMF	High-Resolution Moth Flight
LDA	Linear Discriminant Analysis
MAE	Mean Absolute Error
MAP	Maximum A Posteriori
MPS	Mixture of Pictorial Structures
MRF	Markov Random Field
MSE	Mean Squared Error
PS	Pictorial Structures
RVM	Relevance Vector Machine
SGD	Stochastic Gradient Descent
SIFT	Scale-Invariant Feature Transform
VGG	Visual Geometry Group
WHOG	Whitened Histogram of Oriented Gradients

Chapter 1

Introduction

Flying animals are actively studied by researchers working to better understand these animals' behaviors and flight characteristics. For example, researchers study the individual and group behavior of bats as well as their obstacle avoidance abilities [9, 30, 48, 77, 83], the maneuverability of cliff swallows involved in a chase [70], and the flight performance of hawkmoths under varying wind conditions [59, 60]. Towards this goal, multi-view camera systems have been used to record video datasets of flying animals both in lab conditions and in natural habitats [8, 43, 59, 60, 65, 70, 77, 78]. Analyses of these datasets, which are crucial to making scientific progress, are increasingly performed with the help of algorithms. These algorithms, which are semi or fully automatic, allow scientists to save lots of time and manual labor, while also permitting the analysis of larger datasets. In this thesis, we focus on the largely unexplored problem of 3D pose estimation of flying animals in multi-view video datasets. An ideal solution to this problem is one that can automatically output an accurate estimate of the 3D configuration (3D pose) of the flying animal over time. Developing such a solution would make it easier and faster for researchers to obtain measurements of what flying animals are doing with their bodies in 3D, allowing scientists to spend more time researching the how and why. To date, this problem has received little direct attention in the computer vision community where few flying animal datasets exist and 3D pose estimation has mostly been applied to datasets of people. Additionally, published approaches [8, 43, 59, 60, 64, 65, 70, 78] from researchers in the natural sciences have not taken full advantage of advancements in computer vision research, hindering both their performance and their broader applicability.

Our thesis addresses these issues by introducing three different novel approaches for estimating 3D pose of flying animals in multi-view video datasets. The choice to focus on this problem was also partly inspired by our formative experience working with low-resolution video of bats from which 2D shape information was leveraged to estimate wing beat frequencies of individual bats [17]. Our three approaches are inspired by three successful pose estimation paradigms developed in computer vision. The first paradigm is the use of 3D graphics models for pose estimation. The second paradigm is the use of part-based models for pose estimation. The third paradigm is the use of deep neural networks for pose estimation. With the overall problem statement and motivation of our work established, we now further detail the specific motivations, problem statements, and challenges that led to our three approaches.

1.1 3D Pose Estimation of Bats in the Wild

1.1.1 Motivation and Problem Statement

Our first approach is motivated by the desire biologists and aerospace engineers have to understand how bats fly and why they behave the way they do. New questions are being asked about bat behavior: How do large numbers of bats behave as a group? What is their behavior when they forage? How do bats maneuver through dense vegetation while avoiding obstacles? Bats flying in the wild are shown in Figure 1·1.

Vision-based methods have been used to detect bats in visible and thermal-infrared video [8, 77, 83], track them in 3D [8, 77, 83], and analyze their kinematics, behaviors, and flight trajectories [8, 77]. An important distinction among these works is whether they deal with data captured in a laboratory or in the wild, and whether they model a bat as a point or a 3D articulated body.

Works that deal with video data of bats in the wild have modeled bats as points instead of articulated bodies [77, 83]. To uncover detailed flight behavior of bats in the wild, however, scientists would like to retrieve their 3D articulated motion.

Works that do model bats as 3D articulated bodies have relied on data of bats in confined laboratory spaces, where the motion of a bat can be captured up close in great detail [8, 43, 64]. The work of Bergou et al. [8] modeled a bat with a 52 degree of freedom (DOF) articulated model, whose parameters are estimated from real data. Their data was obtained by first attaching tape markers to various landmarks on the bat, then placing the bat in a confined flight corridor where cameras can be positioned as close as necessary, and finally capturing flight data using high frame rate (1000 fps) cameras. This high quality data, from multiple cameras, served as input to a 3D tracking algorithm whose output led to a set of 3D pose estimates. While this method and similar approaches may offer relatively accurate 3D pose estimates, they have several limitations. First, these methods are not suited for bats in the wild where the use of tape markers is especially impractical and the ability to capture high resolution image data is limited. Second, there is no guarantee that bat behavior observed in confined laboratory spaces is representative of behaviors exhibited in the wild. As a result, analyses performed on data obtained in a laboratory may be misleading or limited in scope. Lastly, laboratory experiments tend to be performed on a small number of bats, one at a time, which is of limited use for studies on group behavior.

We aim to bridge the gap between methods that use 3D articulated bat models and methods that work on data of bats in the wild. This leads to our problem statement: given three low resolution camera views capturing *Tadarida brasiliensis* bats in flight, estimate the 3D pose of an individual bat for the duration of its observed flight.

Figure 1·1: Bats emerging from a cave in Texas at sunset.

1.1.2 Challenges

In addition to the inherent challenge of estimating 3D pose from 2D imagery, where multiple 3D poses can map to an identical 2D projection, many of the challenges faced are domain and dataset specific. Our choice to study *Tadarida brasiliensis* bats in particular, means that off the shelf resources are not readily available for use in modeling the appearance of the bat. The "in the wild" environment in which the bats are filmed, necessitates that cameras be placed far enough away from the bats to observe a large enough interval of their flight. These filming conditions result in a significant loss of resolution, with an average bat being covered by only an area of 30 x 30 pixels. Evaluation of 3D pose estimates on our dataset also poses a challenge as no ground truth is supplied.

To overcome these challenges, we propose a model-based framework called 3D-

PEB, which uses a synthetic 3D graphics model, which we develop, to capture the appearance, geometry, and articulations of the bat. The synthetic model is posed in 3D as to densely sample 3D pose space, and for each pose a view is generated by projection onto a virtual camera. The collection of views generated represent the mapping between 2D appearance and 3D pose parameters. Since this mapping from 2D to 3D is one-to-many we use a Markov Random Field (MRF) model to add across camera and temporal constraints into our formulation. The across camera constraints encourage agreement amongst cameras on the 3D pose estimate. The temporal constraints encourage solutions where 3D poses vary smoothly across time. Approximate inference is performed on the MRF to yield 3D pose estimates for a window of time. Validation of our system was performed by comparing manually annotated 3D poses, obtained from a bat biologist who used our 3D bat pose annotation tool, with automatic estimates produced by our system.

1.2 Discovering Useful Parts for Pose Estimation in Sparsely Annotated Datasets

1.2.1 Motivation and Problem Statement

Our second approach is inspired by the question of how hawkmoths (*Manduca sexta*) fly in varying wind conditions, a problem studied by Ortega-Jimenez et al. [59]. In their work, hawkmoths were placed into a chamber with varying wind conditions where their flight was captured using multiple high-resolution, high frame-rate cameras. To analyze the flight sequences of hawkmoths, we used computer vision. First key body landmarks were localized across multiple camera views and time. Secondly, 3D positions of these landmarks were reconstructed across time. While Ortega-Jimenez et al. [59] obtained interesting results, their approach to landmark localization only works on datasets where the hawkmoth is observed from a particular

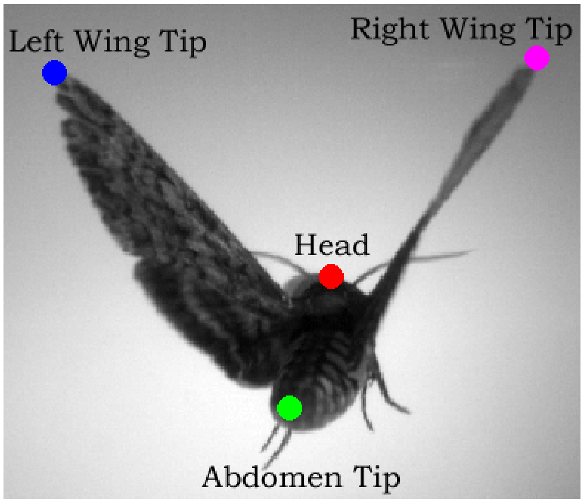


Figure 1·2: A hawkmoth (*Manduca sexta*) is viewed from behind. Four key body landmarks sufficient for describing the pose of the moth are labeled with text alongside a colored circle at the location of the landmark.

view point, thus limiting the general applicability of their approach.

Among the large body of works that exist for pose estimation in computer vision, part-based models in particular have shown great success in both 2D and 3D human pose estimation [2, 3, 13, 14, 20, 28, 29, 34, 41, 47, 62, 75, 80, 84, 86]. Part-based models have a common approach of modeling an object by a collection of parts. The definition of what a part is varies, but common to all of the mentioned approaches, the representation of a part is learned from annotations provided with training images.

We argue that the complete dependence of part-based models on annotations is a weakness, especially limiting in applications where training data is sparsely annotated. Consider the problem of localizing the positions of four landmarks of interest, shown in Figure 1·2, on a hawkmoth test image. Assume training images are given with those same landmarks annotated. Part-based models would do their job of modeling parts based on annotations, while regions of the training images without annotations, including most of the wings, abdomen, and antennae, risk being thrown away. Thrown away regions may contain parts which are helpful for localizing the landmarks of interest. We hypothesize that augmenting traditional part-based models with parts discovered from the unannotated regions of training images can improve the localization accuracy of landmarks of interest, especially in sparsely annotated datasets.

This leads to our problem statement: given a high-resolution image of a hawkmoth, how do you find and leverage parts that have not been explicitly annotated in a training set, in order to improve the accuracy of the appearance likelihood terms used in Pictorial Structures (PS) [28, 29] based models.

1.2.2 Challenges

Though we follow a common pipeline to unsupervised part discovery [46], where patches from training images are generated and then clustered, the specific implementation details depend on our domain and dataset. The first challenge is in choosing a feature representation that is sensitive enough to the texture and appearance of a moth to be able to discriminate one patch taken from an image of a moth from another. The second challenge is in finding or developing a clustering algorithm that can yield clusters that corresponded to meaningful parts of the moth. The dependency of the clustering output on both the clustering approach and the feature representation makes diagnosis of poor clustering output challenging. The third challenge involves deciding how to evaluate the quality and usefulness of discovered parts, represented as clusters of patches. The last challenge is in deciding how to leverage the discovered parts to obtain more accurate appearance likelihood terms.

To overcome these challenges, we propose a carefully engineered system that leverages SIFT based features with a custom clustering algorithm to obtain parts that are visually similar. An additional step is taken to ensure more semantic consistency within clusters by looking for agreement among patches on the relative location of one or more semantic parts. Consistent clusters of patches are then modeled with learned classifiers and are available for use on a test image. Parts that are detected on a given test image are allowed to vote for the location of one or more semantic landmarks. The votes are aggregated across part detectors and subsequently used as appearance likelihood terms in a MPS model which produces the final output.

1.3 Automatic Image Analysis by Annotating Landmarks with Deep Neural Networks

1.3.1 Motivation and Problem Statement

Our third approach is inspired by the recent successes of deep neural networks (DNNs) which have produced state of the art results on a variety of tasks including: object recognition and image classification [49, 73**?**], landmark localization [85], and 2D pose estimation [79]. Our work aims to evaluate how suitable DNNs are for accurate and automatic localization of landmarks in the challenging case where the amount of labeled training data is small, as is typical in video data collected for analysis of animals. To facilitate our study we performed experiments on high speed hawkmoth (*Manduca sexta*) video data [18, 60]. Our work aims to answer several questions including:

- What kind of DNN design can be used to automatically annotate landmarks?
- What decisions and considerations must take place before training a DNN?
- What level of performance can be obtained using DNNs for automatic landmark localization in hawkmoth videos?
- How is the level of performance impacted by various factors including: dataset

size, dataset augmentation, network architecture, training parameter values, and more?

Additionally, with this work we are motivated to make DNNs more understandable and accessible to scientists and research communities outside of computer vision and machine learning. Our problem statement is: given high-resolution video of a hawkmoth, can a deep neural network be designed and trained to produce accurate localization of landmarks despite the training set being relatively small.

1.3.2 Challenges

Despite DNNs having been shown to be successful on a variety of problems, designing and training a DNN for a domain and dataset that is qualitatively different than what is commonly published in computer vision, is a significant challenge. The first challenge our work needs to overcome is the small size of our training set which is on the order of hundreds of images. This is in contrast to the relatively large datasets that are commonly used in computer vision, including ImageNet [23], which contains more than 1 million images, each annotated with bounding boxes for all objects in the image. The second challenge involves choosing and designing a suitable network architecture for the task of regression that also leverages already trained networks, which are largely designed to classify images containing many different everyday objects. The third challenge involves choosing reasonable settings for a large number of parameters that impact training and the overall performance of the trained network. The last challenge is in exploring this large parameter space as training is costly to perform, taking on the order of tens of hours to train a network with a single set of parameter values.

We overcome these challenges by performing a comprehensive set of experiments. We trained a variety of networks adapted from the VGG-16 architecture of Simonyan et al. [73], and examined the influence of many parameters.

1.4 Contributions

The main contributions of this thesis are:

- A method for wing beat analysis of flying animals using 2D shape analysis, realized on low resolution bat datasets [17].
- A novel framework for 3D pose estimation of flying animals in low resolution multi-view datasets, realized by our system 3D-PEB [16]. Our work represents the first published results on automatically generated 3D pose estimates of bats in the wild. We also make a multi-view low resolution bat dataset freely available, along with a 3D graphics model of a *Tadarida Braziliensis* bat[1].
- A novel framework for increasing pose estimation accuracy by discovering parts from unannotated regions of training images. Discovered parts are used to generate more accurate appearance likelihoods for traditional part-based models like PS [28, 29] and its derivatives. Our experiments on images of a hawkmoth in flight show that our proposed approach significantly improves over existing work [60] for this application, while also being more generally applicable. We also make a single-view high resolution hawkmoth dataset freely available along with landmark annotations and segmentations[2].
- A work that shows, through extensive experimentation on videos of hawkmoths, that DNNs are suitable for automatic and accurate landmark localization. In particular, we show that one of our proposed DNNs is more accurate than the current best algorithm for automatic localization of landmarks on hawkmoth

[1] http://www.cs.bu.edu/~betke/research/3dpeb/

[2] http://www.cs.bu.edu/~betke/research/HRMF/

videos [18]. Moreover, we demonstrate how these annotations can be used to quantitatively analyze the 3D pose of a hawkmoth. To facilitate the use of DNNs by scientists from many different fields we provide a self contained explanation of what DNNs are, how they work, and how to apply them to other datasets using the freely available library Caffe [44]. We also provide free code[3] to lower the barrier to training and testing DNNs in Caffe. Furthermore, we make freely available an additional high resolution hawkmoth video along with landmark annotations and segmentations to provide the first published high resolution multi-view dataset of a flying animal, which we call HRMF2.0[4].

1.5 List of Related Papers

This thesis is based on the following papers:

- Breslav, M., Hedrick, T., Sclaroff, S., and Betke, M. "Automating Image Analysis by Annotating Landmarks with Deep Neural Networks." To be submitted.

- Breslav, M., Hedrick, T., Sclaroff, S., and Betke, M. "Discovering useful parts for pose estimation in sparsely annotated datasets." In *Winter Conference on Applications of Computer Vision* (2016), IEEE.

- Breslav, M., Fuller, N., Sclaroff, S., and Betke, M. "3D pose estimation of bats in the wild." In *Winter Conference on Applications of Computer Vision* (2014), pp. 9198.

- Breslav, M., Fuller, N. W., and Betke, M. "Vision system for wing beat analysis of bats in the wild." In *Workshop on Visual Observation and Analysis of*

[3] `http://www.cs.bu.edu/~betke/research/ALADNN/`

[4] `http://www.cs.bu.edu/~betke/research/HRMF2/`

Animals and Insect behavior held in conjunction with International Conference of Pattern Recognition (Tsukuba, Japan, 2012).

Chapter 2

Related Work

The related work for this thesis can be organized into two categories: works which estimate the 3D pose of a flying animal from multi-view video data, and works which estimate the 2D or 3D pose of humans.

2.1 3D Pose Estimation of Flying Animals in Multi-View Data

3D pose estimation of flying animals in multi-view datasets has been a problem typically worked on by natural science researchers primarily interested in analyzing the behavior and flight characteristics of these animals [8, 30, 32, 43, 59, 60, 64, 65, 70, 78]. One common approach to capturing the 3D pose of flying animals in laboratory spaces has been with the aid of physical markers or paint placed on landmarks across the animal's body [8, 43, 64, 78]. These landmarks are subsequently annotated in the image data and then reconstructed in 3D to obtain a final 3D pose. Tobalske et al. [78] applied this approach to videos of hummingbirds, and Bergou et al. [8], Hubel et al. [43], and Riskin et al. [64] applied the approach to videos of bats. To reduce the amount of annotation necessary, Bergou et al. [8] annotated landmarks in an initial set of frames and then used tracking algorithms, which can be corrected with user input, to follow and annotate the landmarks over time. Fisher et al. [30] used a similar approach to track landmarks on a bat, but did not place markers or paint on the bat. In Shelton et al. [70], cliff swallows were observed in a natural environment precluding the use of markers. Manual annotations were used to localize landmarks in the image data. While these works tend to produce accurate results, they have one or more weaknesses that hinder their general applicability. Marker or paint based approaches require the animal of interest to be captured and carefully fitted with markers or painted. This approach is not only impractical when observing animals in their natural habitats, but it also cannot scale to a large number of individuals. Additionally, each of these methods require some amount of user interaction at the stage where landmarks, markers, or paint, are localized in image space; this user interaction can be laborious and time consuming. In contrast, our proposed approaches are marker-less and do not require any user interaction on testing datasets.

More automated approaches have also been proposed [32, 59, 60, 65]. Fontaine et al. [32] built a 3D graphics model of a *Drosophila* fly and estimated its 3D motion by aligning the model to 2D image features. Ristroph et al. [65] segmented multiple views of a fruit fly and then back projected them for visual hull reconstruction. Reconstructed voxels were clustered into groups corresponding to different body parts. A final 3D pose estimate was obtained by computing the position and orientation of part clusters. This approach inherits the weaknesses of visual hull methods including sensitivity to segmentation and camera placement, and inherent coarseness in the reconstructed volume. Ortega-Jimenez et al. [59, 60] segmented hawkmoths in a multi-view dataset and body parts were localized in each view using view specific rules. Corresponding part positions were reconstructed in 3D yielding 3D pose estimates. This approach is sensitive to segmentation results, and heavily relies on seeing a specific viewpoint of the hawkmoth. In contrast, two of our approaches localize body parts using discriminatively trained models and as a result do not depend on segmentation results.

2.2 2D and 3D Pose Estimation of Humans

In the field of computer vision, a substantial amount of work exists on both 2D and 3D pose estimation, largely applied to images and videos of people. At the heart of this thesis, we take inspiration from these works and explore how they can be adapted to the problem of 3D pose estimation of flying animals in multi-view datasets. Several survey papers detail the large body of work that exists for human hand pose estimation, head pose estimation, and full body pose estimation [25, 56, 57, 58].

2.2.1 3D Graphics Model Based

One popular class of approaches is the model-based approach. These works leverage a 3D graphics model by generating a large number of synthetic views labeled with the 3D pose that generated them. These labeled views can then be used directly to compare with an input view. One example of such an approach is the 3D hand pose estimation work by Athitsos & Sclaroff [4]. In their work, a synthetic articulating hand model was used to generate a large set of views. Each view of the hand, encoded by edge based features, represented the appearance of a posed hand from the viewpoint of an orthographic camera. 3D hand pose was estimated for a novel input image by finding the view whose edges match best to the input.

Some 3D pose estimation methods take advantage of multiple cameras. The work on 3D human upper body pose estimation by Hofmann & Gavrila [42] used the view in each camera to hypothesize a set of 3D pose candidates. The 3D pose candidates were reprojected into other camera views to evaluate a likelihood. Additionally, temporal constraints were used.

2.2.2 Motion Capture

Another class of 3D pose estimation methods are those that learn a regressor or classifier from a motion capture dataset. An example of one of these methods is the work by Agarwal & Triggs [1] for 3D body pose estimation. Their training data consisted of images of different full body poses, along with the joint angles that parameterize the 3D body pose. Then, a Relevance Vector Machine (RVM) was trained to learn a mapping from shape context based features to 3D body pose. Motion capture data for flying animals is not readily available limiting the applicability of this class of methods.

Depth cameras have also been used for 3D pose estimation. One of the earliest successful demonstrations of 3D pose estimation from depth images is the work of Shotton et al. [71] where decision forests were trained to classify individual pixels by the body part type they belong to. Currently, depth cameras are not viable for recording flying animals in the wild, however they could potentially be of use for laboratory based experiments.

2.2.3 Part-based Models

Part-based models, which represent objects by a collection of parts, have been popular and successful for the problems of object detection, 2D pose estimation, and more recently 3D pose estimation [2, 3, 14, 20, 27, 28, 29, 47, 75, 84, 86].

One large class of part-based models are those based on the influential Pictorial Structures framework [2, 3, 20, 27, 47, 61, 84, 86]. The Pictorial Structures (PS) framework [28, 29] models objects by a collection of parts having spatial relationships. In the model proposed by Felzenszwalb & Huttenlocher [28], parts are organized in a tree structure where pairs of parts connected with an edge are modeled spatially by a Gaussian distribution. The parameters of the Gaussian, based on part positions in a transformed space, are estimated from images with annotated part positions. Each part is also associated with an appearance model which is typically learned in a discriminative way from supervised data. Inference of the tree model is performed with dynamic programming, implemented using the generalized distance transform [26] in the case of Gaussian spatial relationships.

Many works have built upon the PS framework and extended it in numerous directions. The Deformable Part Model (DPM) by Felzenszwalb et al. [27] improves over PS by only requiring bounding box annotations and foregoing more expensive part annotations. Instead, parts are considered as latent variables which are learned through a latent SVM formulation. Additionally, DPM uses a mixture model to represent significant variations in the appearance of an object, a strategy which is also used by others [61, 84, 86]. Ott and Everingham [61] reformulated the DPM to allow part appearances to be shared across different mixtures and also across different object detectors. Yang and Ramanan [84] introduced part-types which are mixtures at the part level. Their formulation also includes unary terms representing preferences over part-types and a pairwise term which models the co-occurrence of part-types. Zhu and Ramanan [86] used a mixture of trees and also allowed parts to be shared amongst the mixtures. Sun and Savarese [75] defined an object by recursive parts, in a coarse to fine manner, where a body part is represented by multiple part-types. These part-types capture how a body part changes in appearance and can represent different poses.

Another popular part-based model is the work of Bourdev and Malik [14] who introduced parts tightly clustered in appearance and 3D configuration, which they call *poselets*. The process of generating poselets produces a large number of them, so the authors used heuristics to reduce the number of them to a smaller subset that is effective and complementary. For these poselets, detectors were trained in a discriminative way and at test time all poselet detectors were evaluated on the input. The final location of parts are decided by a Max Margin Hough Transform which allows each poselet to vote, with a learned weight, for a part being located at some position x.

Recently, several works have extended the usefulness of poselets [13, 34, 41, 62, 80]. Bourdev et al. [13] proposed a method where poselets are learned from 2D annotations instead of 3D annotations making the approach more general. Additionally, poselet activations are reweighted based on the spatial distribution of other poselet activations. Finally, poselets are clustered into consistent hypotheses and bounding boxes are predicted from these. Wang et al. [80] modeled objects hierarchically using poselets. Pishchulin et al. [62] used PS models conditioned on poselet activations in the input image. In the work by Gkioxari et al. [34] a deformable part model is built on top of poselets which are then used for detection and keypoint localization of people. Hernandez et al. [41] found a set of mid-level parts based on poselets which are used to improve the accuracy of appearance likelihood terms used in PS models.

There are other works which also look to discover useful parts but with less supervision [74, 46]. The method by Singh et al. [74] discovers mid-level parts by an iterative procedure which alternates between clustering parts and training discriminative classifiers. Juneja et al. [46] also used an iterative approach to discover parts by starting with a single example and gradually adding parts while updating the part model.

Recently several works have used part-based models for the problem of 3D pose estimation [2, 20, 47]. Amin et al. [2] extended the pictorial structures formulation to the multi-view case by adding correspondence constraints and a multi-view appearance term. Once 2D pose estimates are obtained for all camera views linear triangulation is used to reconstruct 3D pose. In the work of Kazemi et al. [47] random forests classifiers are used to generate appearance likelihoods in each camera view, which are then back projected to obtain a 3D appearance likelihood. A final 3D pose is obtained by combining the 3D appearance likelihood with a 3D pose prior using the 3D pictorial structures work of Burenius et al. [20]. Burenius et al. [20] formulated pictorial structures in 3D where parts are described by a 3D position and 3D rotation, and the prior on parts is as in the 2D case a tree.

2.2.4 Deep Neural Networks

Most recently, deep neural networks (DNNs) have become widely adopted by the computer vision community after results by Krizhevsky et al. [49] showed convolutional neural networks (CNNs) could achieve state of the art performance on the task of image classification. CNNs, a type of DNN which makes use of convolutions in the architecture design, have achieved great results on computer vision problems dating to at least as far back as 1998, when the work of LeCun et al. [52] showed state of the art performance on the problem of handwritten character recognition.

In addition to image classification, where CNNs have continued to push the state of the art [73**?**], CNNs are rapidly being applied to many other problems including human pose estimation [54**?** **?** , 79, 85]. Toshev et al. [79] proposed to use a DNN based on the architecture of Krizhevsky et al. [49], with a squared loss for regressing human joint locations. A cascade of DNNs, trained on different images of different resolutions, are used to sequentially refine a coarse pose estimate to a final more accurate one. Tompson et al. [**?**] trained a CNN to predict pixel wise distributions over joint types and train an additional network to learn spatial constraints among joints. Pfister et al. [**?**] trained CNNs to take a temporal window of input images and regress a set of heatmaps, each representing the location of a particular joint for a particular input frame. Additional spatial fusion layers are learned to incorporate spatial constraints between joints. The network output consists of heatmaps for each frame in a window of time, which are warped to a reference heatmap. A final layer in the network learns how to combine the warped heatmaps into a final output. Lifshitz et al. [54] proposed learning CNNs to predict the location of body keypoints given an image patch, similar in spirit to our second proposed approach [18]. Given keypoint distributions, consensus is achieved by having the center of each patch vote for a combination of keypoints.

Chapter 3

Wingbeat Analysis of Flying Animals

In this thesis chapter, we propose a method to extract the wing beat of individual bats by analyzing their shape. Having an estimate of the wing beat frequency of a bat species may improve our ability to design algorithms for detection and tracking of bats in video data, much like periodicity estimation has done for video analysis of pedestrians [19, 21].

Existing vision-based approaches that estimate the wing beat of individual bats typically do so by placing the bats in laboratory spaces and setting up visible-light cameras with high frame rates to record the flying bats in close proximity. Our system can extract the wing beat of individual bats as they fly in their natural environments. Field conditions are more challenging than laboratory conditions because we cannot guide the direction of flight of the bats, our infrared cameras are relatively far from the bats yielding less pixels per bat, and bats stay in the field of view for a short amount of time (< 1.5 s).

The video data used in this work show Brazilian free-tailed bats (*Tadarida brasiliensis*) as they emerge from a cave in Texas, recorded with a thermal infrared camera. Our FLIR SC8000 camera records 14-bit video at a resolution of 1024×1024 pixels with a frame rate of 131.5 frames per second. A representative frame from our infrared data is seen in Figure 3·1 (top) and segmented bats are shown underneath (Figure 3·1 (bottom)).

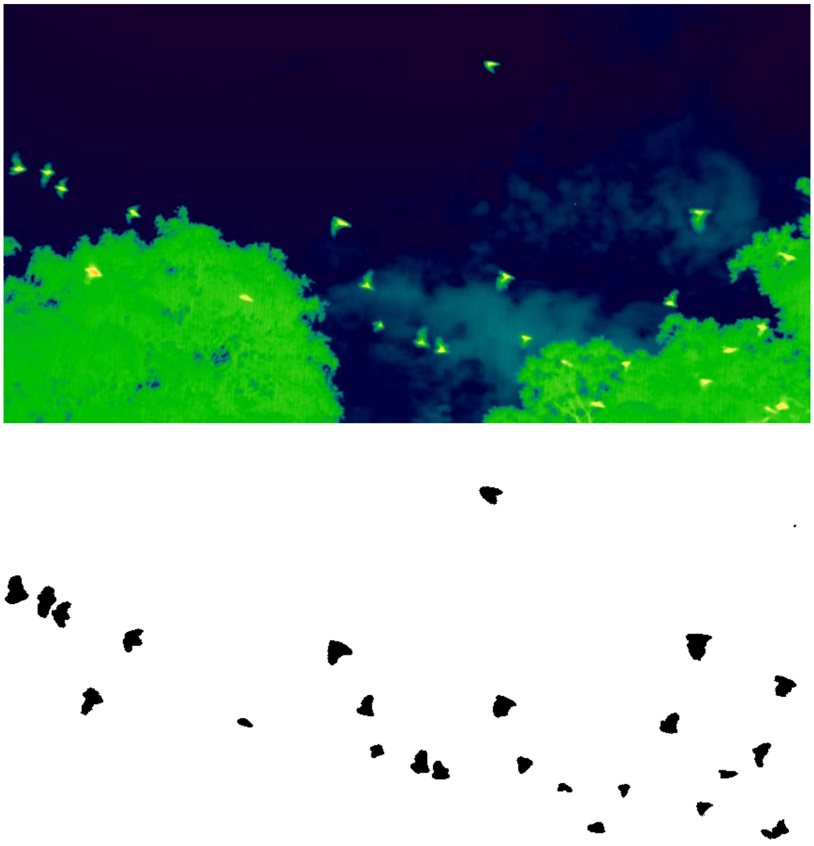


Figure 3·1: Frame of infrared video showing Brazilian free-tailed bats emerging from a cave (top) and the same frame segmented (bottom). An average bat shown here has a projected area of 30×30 pixels.

3.1 Approach

An overview of our system, which is made up of three stages, is shown in Figure 4·1. The main goal of the first stage is to extract the sequence of shapes generated by an individual bat as it is observed flying through the field of view of the camera in our video data. We use the word "shape" to mean the binary connected component associated with a bat as produced by our segmentation algorithm. The sequence of shapes combined with the 2D trajectory of the bat (obtained by a 2D tracker) forms a signal which we refer to as a "shape-time signal." Example shape-time signals are

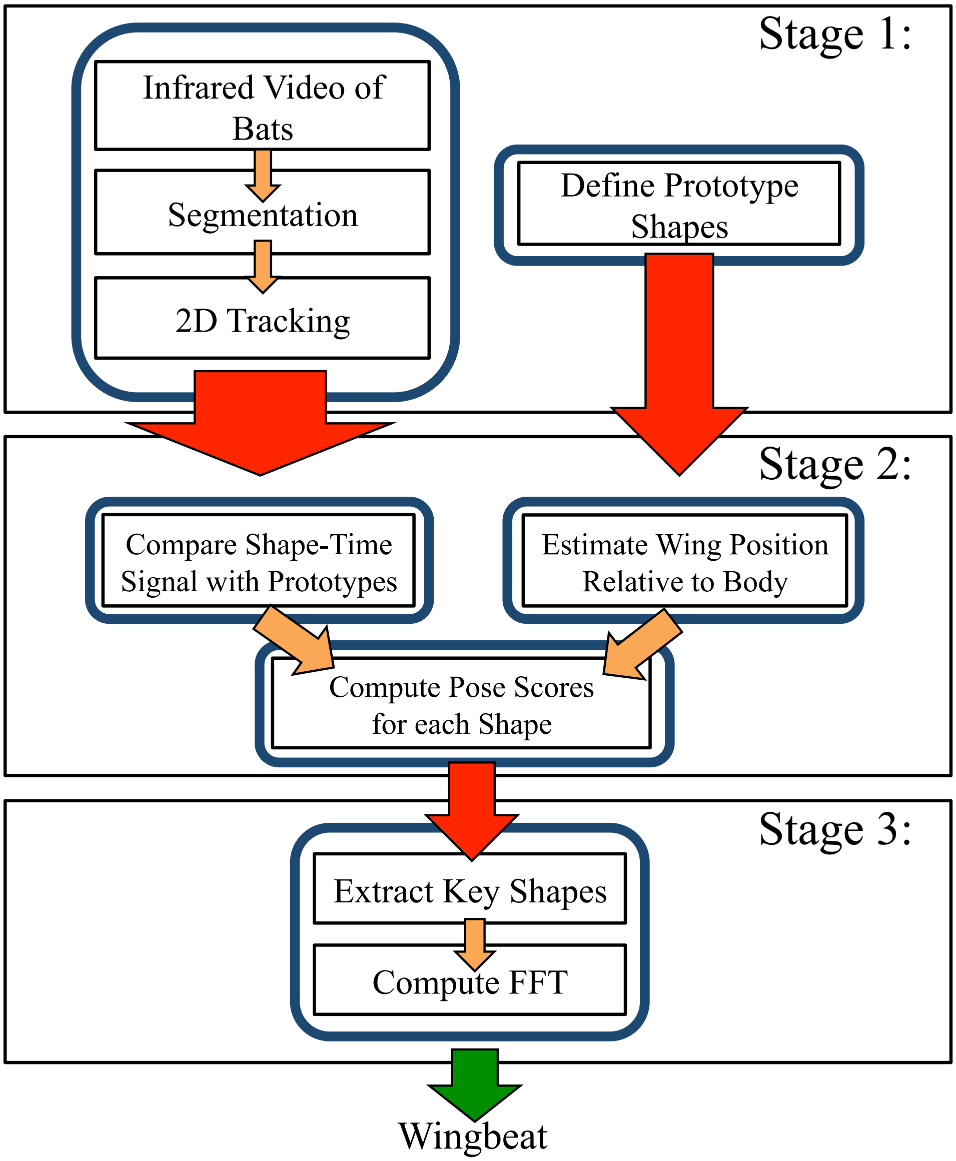
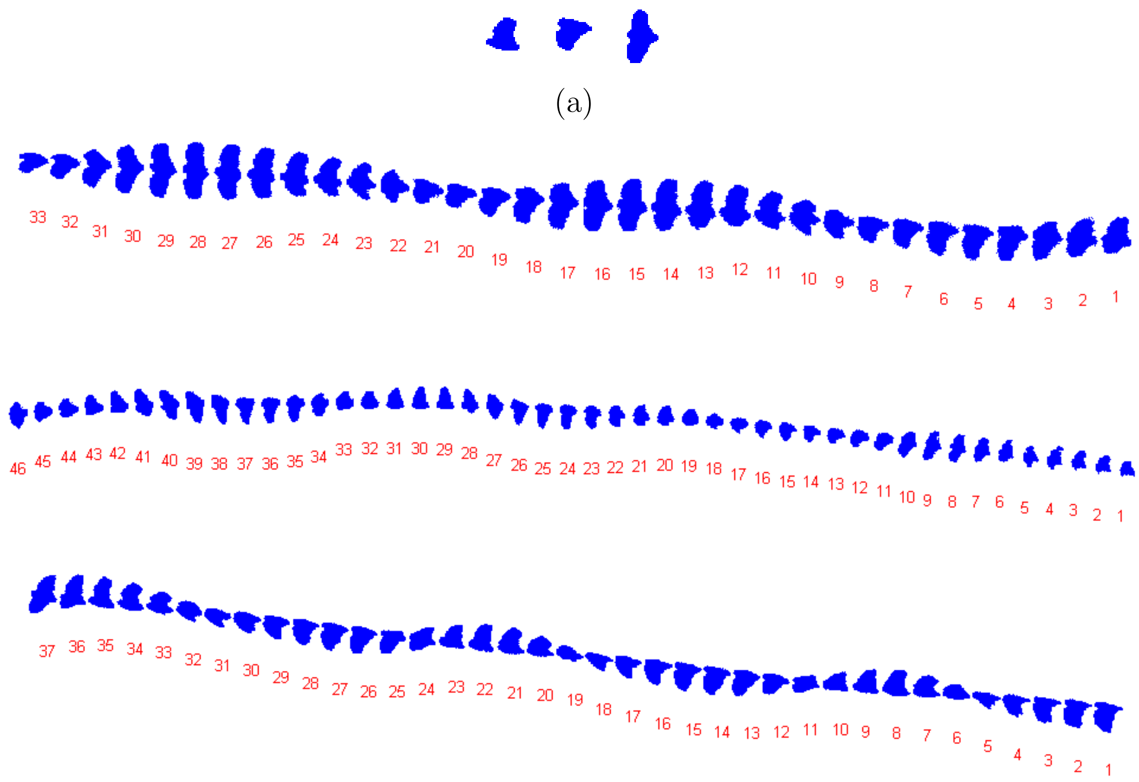


Figure 3·2: Overview of our 3 stage system. The goal of stage 1 is to generate shape-time signals and prototype shapes. Stage 2 takes as input a shape-time signal and uses shape analysis to assign scores to each shape indicating how well they match different poses. Stage 3 processes these scores and determines which shapes play a key role in estimating the wing beat. Then, time intervals, associated with the repetition of key shapes, are used by the FFT implementation to estimate wing beat.

shown in Figure 3·3b. The other component of the first stage of our system involves manually choosing shapes that serve as prototypes for unique 3D bat poses.



(a)

(b)

Figure 3·3: (a) From left to right, three prototype shapes that represent three unique 3D poses: wings above the body nearly completing the upstroke (‘up’), wings below the body nearly completing the downstroke (‘down’), and wings in a neutral position, level with the body, and spread out (‘neutral’). All three shapes are of bats headed in a direction perpendicular to the optical axis of the camera. (b) Three examples of shape-time signals. For visualization purposes, the projections of each bat are spaced out horizontally so the full shape is seen. Each bat is flying from right to left in the field of view of the camera and the numbers underneath a bat indicate the frame number in which it was imaged.

The second stage is the main part of our system which operates on individual shape-time signals. The goal is to assign a label corresponding to a discrete pose to each shape in the shape-time signal. To assign the best label to a given shape, the algorithm computes a distance between the shape and each prototype shape. To make these assignments more robust to noise, our system estimates where the wings of the bat are in relation to the body. The end result is that each shape in the shape-time signal is assigned a set of scores indicating how well the shape matches each discrete pose.

The third stage processes the scores and defines which shapes in the shape-time signal are key for the estimation of wing beat. Then, time intervals, which are associated with the repetition of key shapes, are passed along to a Fast Fourier Transform (FFT) implementation that yields the final wing beat estimate. Stages 2 and 3 must be executed once for each bat (shape-time signal).

3.2 Segmentation and Tracking

Segmentation of bats from infrared video is performed by modeling the distribution of background intensities at each pixel as a Gaussian. The mean and variance of the Gaussian are updated over time and any intensity value outside some fixed number of standard deviations from the mean (15) is considered a bat. Morphological operations are used to fill holes and delete single pixel components. A sample segmentation is shown in Figure 3·1 (bottom). Using segmented video frames, a Kalman filter tracks the 2D image coordinates of bats and at the same time saves their shapes for further analysis. The collection of shapes stored for an individual bat is the shape-time signal we defined earlier. Three example shape-time signals are shown in Figure 3·3b. In this work, we do not deal with occlusions, so only bats that stay unoccluded are used in our analysis.

3.3 Prototype Shapes and Intuition

We motivate defining prototype shapes by observing that some 2D projected shapes of bats relay more information about the 3D pose of a bat than others. We make the assumption that, for a fixed camera setup, certain 2D projected shapes map to unique 3D poses. Shapes that satisfy this property will be defined as the prototype shapes because they model unique 3D poses. Now it follows that the repeated occurrence of a prototype shape in a shape-time signal is equivalent to the repeated occurrence of a unique 3D pose. Since each shape has a time stamp, based on the frame it came from, the periodicity of a 3D pose can be estimated. Using this central idea, our system can compute the wing beat for individual bats. In our work, we manually selected prototype shapes from already acquired automatic segmentations of bats. As an example, three prototype shapes are shown in Figure 3·3a.

3.3.1 Shape Comparison

Our system compares every shape in the input shape-time signal to every prototype shape using the shape context descriptor [5] and the Hungarian algorithm [51]. The shape context descriptor is a log-polar histogram that is computed for points along the contour of a shape. The histogram at a point is computed by finding the relative distance and angle to other contour points. All the histograms taken together yield a shape descriptor that is invariant to translation, scale, and rotation (with some added work). Next, the Hungarian algorithm produces a correspondence between points on the first shape with those on the second. If one shape has more points than the other, "dummy points" are added to the smaller shape. The cost of matching point i in shape 1 to point j in shape 2 is the χ^2 distance between their histograms. If one of the points being matched is a dummy point, then a dummy cost can be used (e.g. 0.5). The final distance is obtained by summing the costs of all corresponding points.

3.3.2 Wing Position Relative to Body

The choice of using a rotationally invariant shape descriptor makes it difficult to disambiguate shapes that are similar under rotation such as 'up' and 'down' (Figure 3·3a). To resolve this, our system estimates where the wings of a bat are in relation to the position of its body. We observed that the 2D position of the bat changes smoothly across time, which allows us to approximate where the body of the bat is by fitting a polynomial to the shape-time signal. Figure 3·4 shows part of a shape-time signal with the fitted polynomial. To help differentiate 'up' from 'down,' four features are extracted from the shape: area A_a above the polynomial, area A_b below the polynomial, the furthest point D_a on the shape above the polynomial, and the furthest point D_b below the polynomial. Then the ratio $W = (A_a D_a)/(A_b D_b)$ indicates whether the shape is more likely to be 'up' (large ratio), 'down' (small ratio), or 'neutral' (ratio ≈ 1).

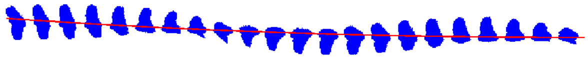


Figure 3·4: A fifth-order polynomial (red) is fit to the shape-time signal to approximate the location of the body of the bat.

3.3.3 Pose Scores

Our system classifies each shape in a shape-time signal as one of the prototype shapes (representing unique 3D poses) or 'neither' (the label given when no prototype shape is a good match). For this classification shape distances (Section 3.3.1) are combined with the ratio W (Section 3.3.2) to assign a set of scores indicating how similar the shape is to each class. Shape distances are converted to scores by normalizing the distance to each prototype and subtracting the scores from 1 to reflect similarity. A fixed score is given for 'neither,' and the scores are renormalized. The ratio W is converted to a set of scores by observing that a large ratio indicates a larger score for 'up,' and a smaller score for everything else. Similar reasoning is applied for small ratios, and ratios close to 1. The final set of scores is an average of these two component scores, and they sum to 1.

3.3.4 Wing Beat Estimation

The last stage of our system uses the previously computed scores to classify each shape in the shape-time signal, and subsequently produce a wing beat estimate. Each shape is assigned the class for which it has the highest score, and only shapes scoring high enough (≥ 0.3 is a 'confident' score) are used for the wing beat estimate. Next, our system finds repetition of poses that are close together in time, which occurs because bats maintain the same general pose for several consecutive frames. To remove these redundant measurements, our system selects a key shape to represent the group. Once key shapes are extracted for the whole shape-time signal, they can be organized based on the pose they represent. Key shapes of the same pose, along with their time stamps, form a time signal which exactly specifies how frequently a pose repeats. All time signals, one for each pose, are summed and sent to an FFT implementation. The result is a frequency spectrum where the fundamental frequency is the wing beat estimate.

3.4 Preliminary Experimental Results

In our experiments, we tracked and analyzed the movement of 20 different bats from an infrared video sequence consisting of 1,000 frames (7.6 s). The shortest track lasted 26 frames and the longest 146 frames (0.19 s and 1.1 s respectively). The prototype shapes used are the same three from Figure 3·3a ('up', 'down', and 'neutral'). For shape comparison we subsample the contour of the larger shape until the number of contour points is roughly equal to that of the smaller shape. Any remaining difference in the number of points is filled with dummy points. In our experiments the number of contour points used was in the range of 50 to 150 points. The shape context histogram contained 12 angle bins (uniformly across 360^o) and 5 distance bins (log-uniform from 0 to 2). The FFT is computed at 1,024 points, where the sampling rate is the frame rate of our cameras (131.5 frames per second). The fundamental frequency extracted from the frequency spectrum was taken to be the highest peak with a positive frequency immediately following 0 Hz. Using our system, we obtained wing beat estimates for 20 bats. We compared them with “ground truth” wing beat estimates obtained from the results of manually performed shape matching (Table 3.1).

Table 3.1: Automatic wing beat estimates for 20 bats and manually obtained wing beat estimates. Their mean μ and standard deviation σ are also provided.

Id:	Wing Beat (Hz)		Id:	Wing Beat (Hz)	
	Autom.	Manual		Autom.	Manual
1	10.6	10.8	11	9.3	9.4
2	10.0	9.8	12	9.5	9.8
3	10.4	10.3	13	10.0	10.9
4	10.9	9.8	14	9.2	9.9
5	9.7	9.9	15	9.7	9.4
6	10.5	11.7	16	11.3	11.2
7	13.1	10.9	17	8.7	9.4
8	10.0	9.0	18	6.0	11.4
9	18.7	10.2	19	10.0	10.5
10	13.1	9.9	20	10.2	9.6
Autom.: μ = 10.5, σ = 2.4			Man.: μ = 10.2, σ = 0.7		

3.5 Discussion

Our proposed method for wing beat frequency estimation relies on using the shape context descriptor along with the Hungarian algorithm to compare shapes. This choice necessitated that our system estimates the position of the wings of the bats relative to their bodies. Sources of error in the current system include noisy segmentations, noisy correspondences from the Hungarian algorithm, and incorrect localization of the body of the bats. Wing beat estimates of bats 7, 9, 10, and 18 may be inaccurate due to too few observed wing beat cycles (≤ 2). Our wing beat estimates differ from the manually obtained ones by 1.4 Hz on average and they fall within or are near the range of 10-15 Hz which is the wing beat frequency range reported in the biology literature for Brazilian free-tailed bats [31]. One limitation of our approach is that it was designed for cases where an animal is predominantly viewed laterally. This limitation is addressed by our proposed approach for 3D pose estimation of bats, detailed in Chapter 4.

3.6 Summary

In this chapter, we have proposed a method for isolating key shapes of bats in flight along with a way to use these key shapes to estimate wing beat frequencies automatically. To the best of our knowledge, our system is the first to estimate the wing beat of bats in the wild automatically using shaped-based computer-vision methods on thermal infrared video data. Potential future directions for this work include: evaluation of system performance using different shape distance measures, learning which prototype shapes are best suited for a particular camera view and bat flight trajectory, and performing a quantitative analysis on the mapping between 2D shapes and 3D poses.

Chapter 4

3D Pose Estimation of Bats in the Wild

In this thesis chapter we describe a method to estimate the 3D pose of bats in the wild. Vision-based methods have been used to detect bats in visible and thermal-infrared video [8, 77, 83], track them in 3D [8, 77, 83], and analyze their kinematics, behaviors, and flight trajectories [8, 77]. An important distinction among these works is whether they deal with data captured in a laboratory or in the wild, and whether they model a bat as a point or a 3D articulated body.

Works that deal with video data of bats in the wild have modeled bats as points instead of articulated bodies [77, 83]. To uncover detailed flight behavior of bats in the wild, however, scientists would like to retrieve their 3D articulated motion. Works that do model bats as 3D articulated bodies have relied on data of bats in confined laboratory spaces, where the motion of a bat can be captured up close in great detail [8, 43, 64]. The work of Bergou et al. [8] modeled a bat with a 52 degree of freedom (DOF) articulated model, whose parameters are estimated from real data. Their data was obtained by first attaching tape markers to various landmarks on the bat, then placing the bat in a confined flight corridor where cameras can be positioned as close as necessary, and finally capturing flight data using high frame rate (1000 fps) cameras. This high quality data, from multiple cameras, served as input to a 3D tracking algorithm whose output led to a set of 3D pose estimates. While this method and similar approaches may offer relatively accurate 3D pose estimates, they have several limitations. First, these methods are not suited for bats in the wild, where the use of tape markers is especially impractical. Second, there is no guarantee that bat behavior observed in confined laboratory spaces is representative of behaviors exhibited in the wild. As a result, analyses performed on data obtained in a laboratory may be misleading or limited in scope. Lastly, laboratory experiments tend to be performed on a small number of bats, one at a time, which is of limited use for studies on group behavior.

Our proposed work bridges the gap between methods that use 3D articulated bat models and methods that work on data of bats in the wild. In particular, we propose a model-based framework for estimating the 3D pose of bats in the wild, given multiple low resolution camera views. We call the system we designed based on this framework 3D-PEB for 3D Pose Estimation of Bats in the wild.

The contribution described in this chapter is the 3D-PEB system for estimating the articulated 3D pose of bats in the wild, a problem for which we are the first to offer a vision-based approach. The second contribution described in this chapter is a description of the challenges that arise when estimating 3D pose for non-human articulated bodies, which often lack good 3D models and training data, and solutions to address them. The third contribution is a set of 3D pose estimates over time, describing what the body and wings of individual bats are doing during an emergence of a group of bats from a cave. Our fourth contribution is a 3D graphics model of the *Tadarida brasiliensis* bat, along with calibrated images of bats, both of which are available online for public use.[1] Lastly, we introduce a part-based feature for use with low resolution images of bats.

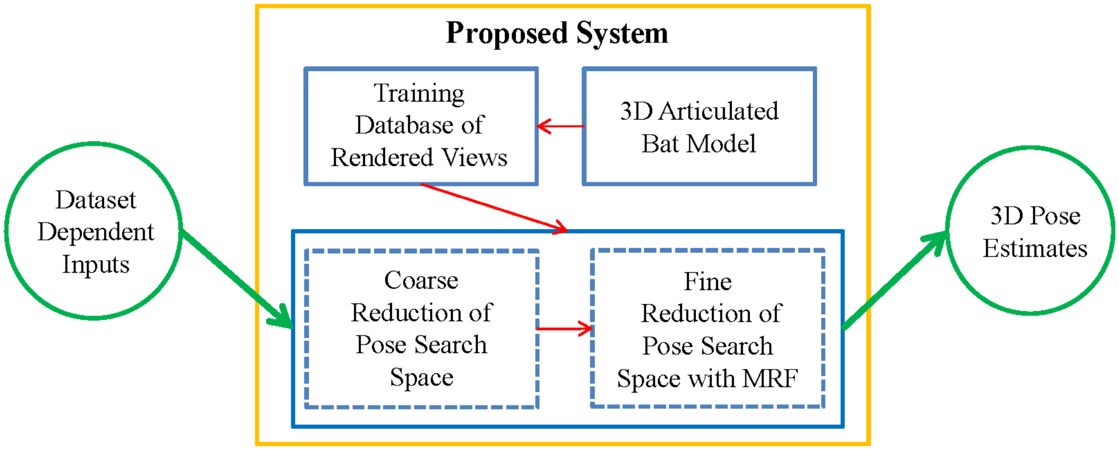

Proposed System
Training Database of Rendered Views
3D Articulated Bat Model
Dataset Dependent Inputs
Coarse Reduction of Pose Search Space
Fine Reduction of Pose Search Space with MRF
3D Pose Estimates

Figure 4·1: Overview of the proposed 3D-PEB system. Input and output are shown with green circles, system components with blue rectangles, and system dependencies with arrows.

4.1 Methods

Our proposed system 3D-PEB is made up of four components, shown in Figure 4·1. The system takes dataset-dependent inputs and produces articulated 3D bat pose estimates as output. The training database of rendered views only depends on the 3D articulated bat model, and is created once with the purpose of being reused. The training database of rendered views and the dataset-dependent inputs jointly serve as input to the subsystem responsible for reducing the pose search space. This subsystem includes one component that yields a coarse reduction of the pose search space, and another component which uses a Markov Random Field (MRF) to produce a fine reduction of the pose search space. The final system output is a set of articulated 3D pose estimates. This framework is detailed in the following subsections.

[1] http://www.cs.bu.edu/faculty/betke/research/3dpeb/

4.1.1 Dataset Dependent Inputs

3D-PEB operates on inputs extracted from video datasets. For the purposes of our specific application, datasets are videos of bats recorded by two or three cameras. We make the assumption that among the many bats flying through the field of view of the cameras, some remain unoccluded in each view for some interval of time. The task of 3D-PEB is to estimate the 3D pose of these individual bats for an interval of time represented by n frames. To reduce the scope of this work, we assume the following to be inputs to 3D-PEB:

- Segmentations $\{S_{i,j}\}$ of the bat for cameras $\{i\} \subset \{1, 2, 3\}$, for all frames $j \in [1 \cdots n]$.
- A description of the relative camera geometry, typically obtained from a calibration procedure.
- A trajectory of the 3D position of the bat $\{\mathcal{T}_j\}$, $j \in [1 \cdots n]$.

These inputs can be obtained using a camera calibration tool [40], and algorithms for detecting bats in thermal video [77] and producing 3D tracks [83]. A sample input segmentation and a typical camera setup are shown in Fig 4·4.

4.1.2 3D Graphics Model

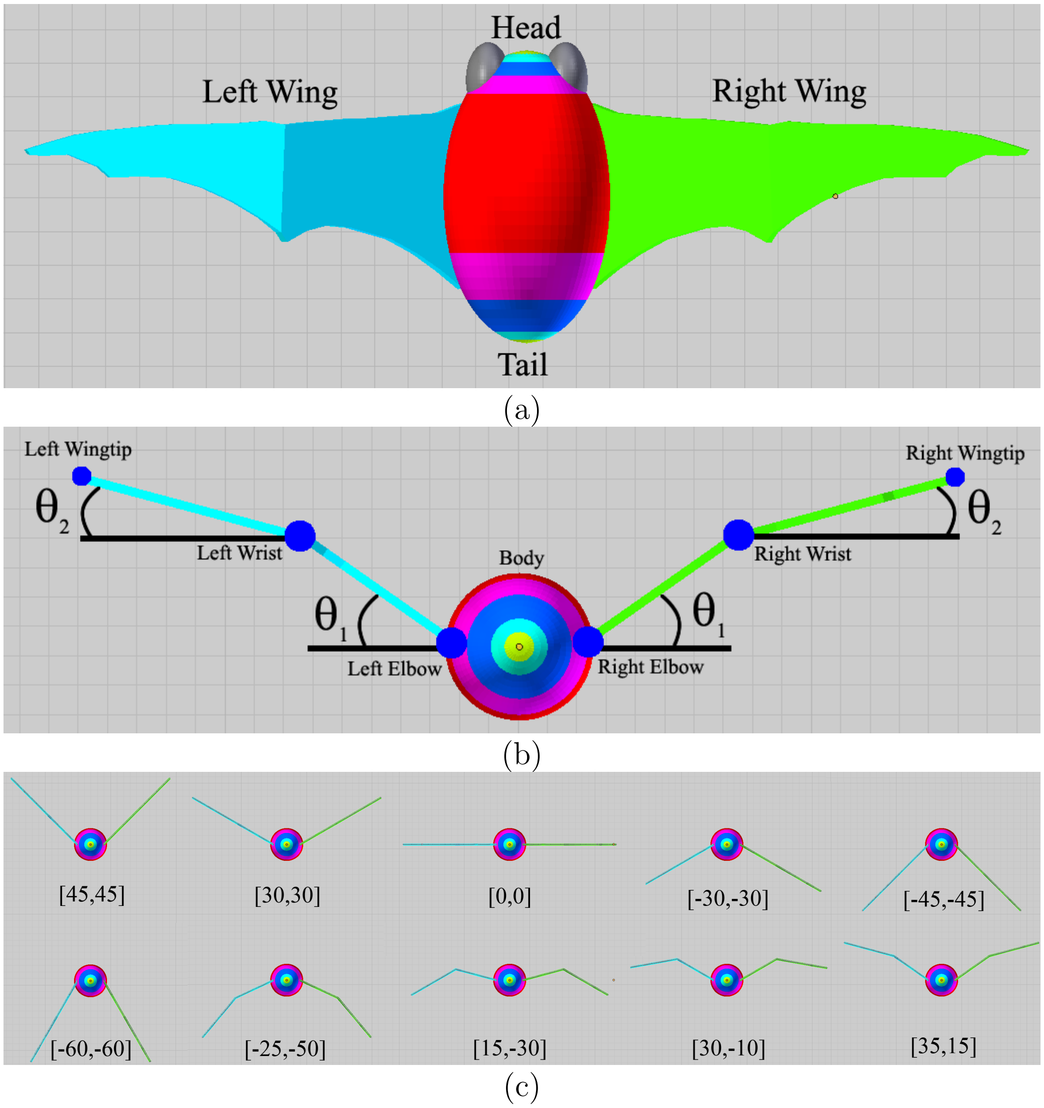
Model-based methods that estimate the 3D pose of an articulated body require a 3D graphics model of that articulated body. In the case of bats, there are many different species, so each would potentially require its own 3D graphics model. In this work, we focused on the bat species *Tadarida brasiliensis* for which 3D graphics models are not readily available. To acquire a model, we built our own using Blender [12], a free 3D modeling tool.

We based the overall shape of our model, seen in Figure 4·2a, on illustrations by biologists [10, 43]. The size of the model was determined by the average wingspan (≈ 284 mm) and aspect ratio (9) for *Tadarida brasiliensis*, measured by biologists [10]. To aid in modeling the articulation of the wings, we referenced illustrations [43, 81] and video materials [43]. We designed a model with two joints per wing, each having one degree of freedom, roughly corresponding to the elbow and the wrist, see Fig. 4·2b. Our model is sufficiently powerful to approximate the flying motion of a bat, typically characterized by a wingbeat cycle consisting of a downstroke and an upstroke. We define ten key wing configurations that represent a sampling of a full wingbeat cycle, see Figure 4·2c. The wings are assumed to move in a symmetric manner, so one pair of angles $[\theta_1, \theta_2]$ is sufficient to describe the configuration (articulation) of both wings.

4.1.3 Training Database of Rendered Views

We used our 3D graphics model to render a large set of labeled views, collectively referred to as the 'training database.' A good training database should capture the variation in appearance of a bat as its articulated 3D pose changes. In this paper, the term '3D pose,' which we use interchangeably with 'articulated 3D pose,' means a description of how the body of a bat is oriented in 3D, together with the configuration of its wings. Specifically, the 3D model is assumed to be centered at the origin of a world coordinate system where its 3D orientation is described by a unit quaternion. The wing configuration is specified by a pair of angles $[\theta_1, \theta_2]$ (Sect. 4.1.2). A fixed orthographic camera model is used to render the appearance of the bat. This rendered view v_i is labeled with the 3D pose of a bat p_i, and constitutes a single training sample $d_i = (v_i, p_i)$.

Ideally, our training database would need to satisfy the following property: for any possible view v_{in}, generated by a camera observing 3D pose p_{in} at test time, there



(a)

(b)

(c)

Figure 4·2: Proposed 3D graphics model of a *Tadarida brasiliensis* bat (a) Top down view (b) Back view of the model where the articulation of the elbow determines angle θ_1 and the wrist determines angle θ_2. (c) Video-based observations were used to manually define 10 key wing configurations. These 10 configurations, represented by a pair of angles (in degrees), describe a sampling of a typical wingbeat cycle. Shown from left to right, and top to bottom, is the downstroke phase followed by the upstroke phase.

should be at least one training sample d_j with a similar view v_j produced by a similar 3D pose p_j. In essence, this property can be satisfied for some definition of ‘similar’ if the training database acts as a good approximation to an ideal database, which would be infinite in size and contain all possible views of a flying bat.

To approximate an ideal training database, we densely sampled the space of 3D orientations and the space of wing configurations. If a sphere of views is considered, then sampling the polar angle every 10° for 360°, elevation every 10° for 180°, and camera roll every 10° for 360° yields a total of 23,328 views. Instead of moving the camera, we equivalently kept the virtual camera fixed and rotated the bat according to 23,328 randomly [50] generated unit quaternions. To keep the database size relatively small, but sufficient, we kept the wing configurations to one of ten discrete parameters (Fig. 4·2c). All together, 23,328 orientations for each of the 10 different wing configurations yielded a database of 233,280 views. In practice, the view v_i associated with each training sample d_i can be represented by a d-dimensional feature vector f_i (Sect. 4.2.1). The training samples stored have the form $d_i = (f_i, p_i)$.

4.1.4 Pose Estimation via Graphical Model

We model the problem of articulated 3D bat pose estimation with an undirected graphical model. For each unique pair of cameras and frame numbers (i, j), $i \in \{1, 2, 3\}$, $j \in [1, \cdots, n]$, there are two types of nodes in the graph. The first type of node is labeled by the random variable $X_{i,j}$ and it represents the true 3D pose of the bat at frame number j. An $X_{i,j}$ is associated with every camera i. Since the true 3D pose of a bat is not directly accessible, $X_{i,j}$ is considered to be a hidden variable or state. The second type of node is labeled by the random variable $Y_{i,j}$ and it represents the appearance or observation of a bat as imaged by camera i at frame number j. Typically $Y_{i,j}$ will be a feature vector extracted from a segmented bat. All together, this model will have cn hidden state nodes $\{X_{i,j}\}$, and cn observation nodes $\{Y_{i,j}\}$ (c is the number of cameras used); for convenience, we write $X = \{X_{i,j}\}$, and

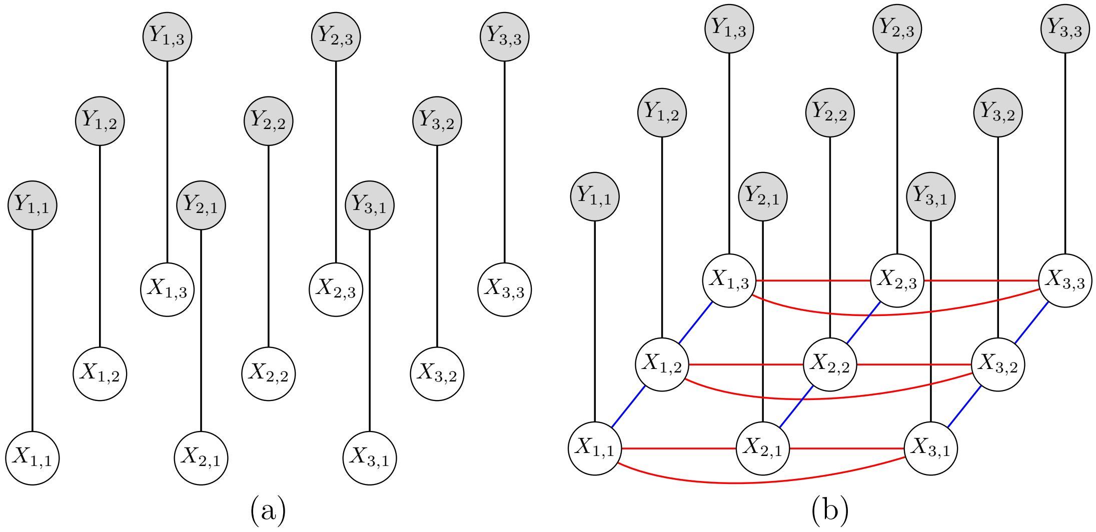


Figure 4·3: Undirected graphical models are shown for the case of 3 time frames and 3 cameras, where observed nodes are gray and hidden nodes are white. (a) Basic model. (b) Basic model with additional constraints represented by edges.

$Y = \{Y_{i,j}\}$. Figure 4·3a shows an example of this undirected graphical model for the case of three frames and three cameras.

Each hidden state $X_{i,j}$ in the model has a state space $Z_{i,j}$ initialized to be Z, the set of 3D poses spanned by the training database D, where $|Z| = 233,280$. From here on out we will express the state space $Z_{i,j}$ by a set of training samples, and it is to be understood that the state space is equivalent to the 3D poses spanned by those samples. Additionally, we assume that the training samples and features Y are represented by the same feature and that a suitable distance measure is available to compare them.

Our goal is to use our observations (features) Y to infer the most likely set of hidden states (3D poses) X. However, two problems arise that prevent the direct use of this graphical model. First, the state space is quite large which is problematic when the number of frames n grows. Second, appearance information alone can be highly ambiguous; for example, a bat flying away from the camera may appear similar to a bat flying towards the camera. To deal with problems of search size and ambiguities, we propose two strategies to progressively shrink the state space. These strategies, detailed below, leverage the dataset dependent inputs along with the training database.

4.1.5 Rule-based Coarse Reduction of Search Space

To reduce the state space of each node $X_{i,j}$, we designed a rule-based system. The first rule takes advantage of the input 3D trajectory $\mathcal{T}$ to bias the state space of a hidden state $X_{i,j}$. The heading of a bat at frame number j is given by the vector $h_j = [\mathcal{T}_{j+1} - \mathcal{T}_j]$ for $j = 1$ and $h_j = [\mathcal{T}_j - \mathcal{T}_{j-1}]$ for $j > 1$. The first rule eliminates 3D poses that represent a bat flying in a direction very different than the heading h_j. The second rule, defined for frame numbers $j > 1$, biases the state space of a hidden state $X_{i,j}$ to be nearby in pose to those from the previous time $X_{i,j-1}$. The third rule eliminates upside down poses from the state space, since they do not occur in our datasets. The fourth rule reduces the state space to only have 3D poses which viewed from camera i look similar to $Y_{i,j}$. This represents the assumption that nearby poses in pose space appear similar in image space, and feature space, when viewed from the same camera.

The second rule requires that the state space is initialized well at frame one. To obtain a sufficiently accurate initialization, we use the first rule with the assumption that the back of the bat points as much towards the sky as possible while maintaining its heading. Application of the last rule reduces the state space to size k, by choosing the k samples with appearance closest to $Y_{i,j}$, as defined by a feature distance measure (details in Sect. 4.2.1). After application of the four rules to all frames, all hidden states $X_{i,j}$ will have a state space of size k.

4.1.6 Fine Reduction with a Markov Random Field

After our method has reduced the state space of each node to k candidates, it chooses the single best candidate for each node. To help guide the choice of best candidates, we propose two constraints. The first constraint is a temporal smoothness constraint which introduces a penalty when the estimated 3D pose of a bat changes too much from one time frame to the next. The second constraint is a camera geometry constraint which favors 3D pose estimates that are consistent with the camera geometry. The undirected graphical model in Figure 4·3b models these added constraints; vertical edges in blue represent temporal constraints, and horizontal edges in red represent camera geometry constraints.

One way to find the most probable 3D poses for the hidden states in this undirected graphical model is to compute a maximum a posteriori (MAP) estimate. This can be formulated as finding the values of X that maximize $P(X|Y)$, where $P(X|Y) \propto P(Y|X)P(X)$. Here $P(X)$ is a Markov Random Field (MRF) prior, which encodes our two constraints, and $P(Y|X)$ is the likelihood that observations Y came from hidden states X. The forms we selected for the MRF prior and likelihood function depend on our design of the MRF model [11, 15], which we first introduce.

MRF Model. A Markov Random Field model is characterized by the Markov property which states that a node is conditionally independent of all other nodes in the graph given its neighbors. Additionally, a Markov Random Field model may be described by a joint distribution that factors in a special way. In particular, if C is a clique and the set of random variables belonging to that clique is x_C, then the joint distribution $P(x)$ factors as $P(x) = \frac{1}{Z}\prod_c \psi_C(x_C)$, where Z is a normalization constant, and $\psi_C(x_C)$ is a potential function defined on the maximal cliques of the graph. Maximal cliques in this work are of size 2, so potential functions are defined on pairs of hidden states connected by edges. We chose the potential function for temporal edges $e_t = (x_{i,j}, x_{i,j'})$ to be $\psi^T(e_t) = e^{-D_T(e_t)}$ and for camera edges $e_c = (x_{i,j}, x_{i',j})$ to be $\psi^C(e_c) = e^{-D_C(e_c)}$.

MRF Prior. Our choice of potential functions yields the prior $P(x) \propto e^{-(\sum_{e_t} D_T(e_t) + \sum_{e_c} D_C(e_c))}$, where the temporal distance D_T is chosen to reflect the change D_O in orientation over time and the change D_W not-consistent with a typical wing beat cycle. The sum $D_T(P_i, P_j) = D_O(P_i, P_j) + D_W(P_i, P_j, \theta)$ expresses this distance, where P_i and P_j are 3D poses consecutive in time and θ is a tuning parameter. If unit quaternion q_1 represents the first orientation, and unit quaternion q_2 the following, we define D_O by the angle between the two unit quaternions. The distance D_W between two wing configurations is given by a cost matrix that encodes a typical wing beat cycle by penalizing two wing configurations which either do not follow the correct order or are not nearby in the wing beat cycle. The distance D_C due to camera geometry is defined similarly to D_T, but with a different cost matrix that penalizes wing configurations proportional to their difference.

MRF Likelihood Function. The likelihood function $P(Y|X)$ is defined on pairs of variables connected by edges e_o. From conditional independence assumptions, the likelihood simplifies to $P(Y|X) = \prod_{v \in \mathcal{V}} P(Y_v|X_v)$. The individual likelihoods corresponding to each node are chosen as $P(Y_v|X_v) = e^{-D_f}$, where D_f is the difference between the feature representation of the observation and the feature representation of the view corresponding to the 3D pose X_v. The feature and feature distance used in our experiments are discussed in Section 4.2.1. Combining the likelihood and prior the posterior is given by $P(X|Y) \propto e^{-(\sum_{e_t} D_T(e_t) + \sum_{e_c} D_C(e_c) + \sum_{e_o} D_f(e_o))}$. Final 3D pose estimates are obtained using max-product loopy belief propagation (messages initialized to 0) to yield an approximate MAP estimate. Additional implementation details are available in [68, 82].

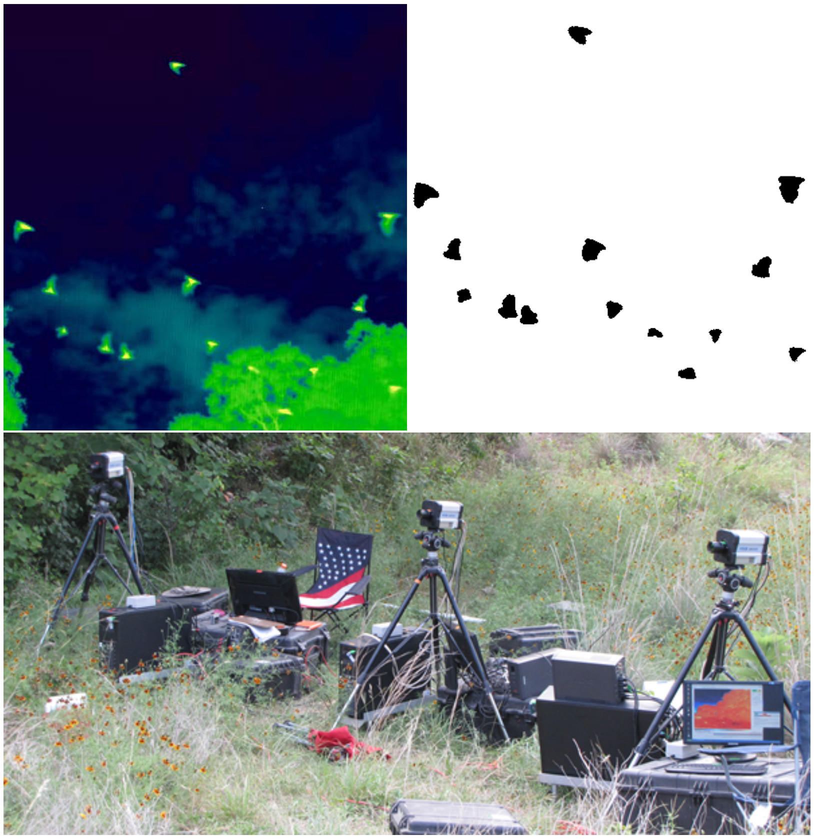
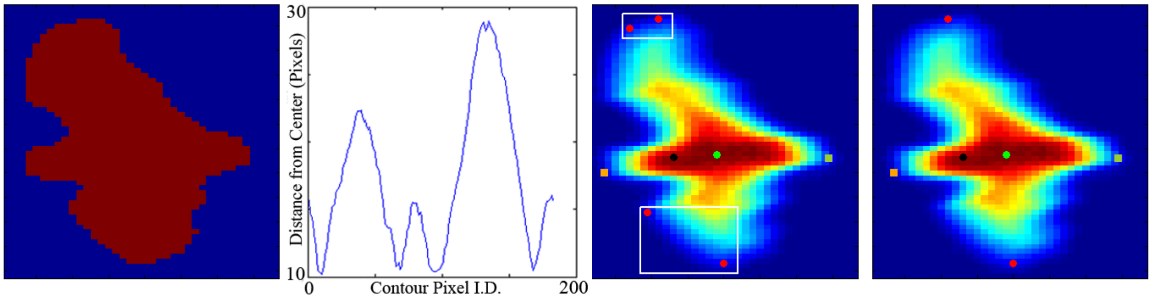


Figure 4·4: Sample data of bats in the wild. **Top:** A frame from an infrared video showing bats flying in the wild and the corresponding segmentation to the right. **Bottom:** A typical field setup of 3 cameras with baselines between 1 m and 2 m.

4.2 Experiments

Our experiments were based on videos taken of *Tadarida brasiliensis* emerging from a cave in Texas. Three FLIR SC8000 thermal infrared cameras were used to record data at a frame rate of 131.5 fps at a resolution of 1024×1024 pixels through a 25 mm lens. Typical camera baselines for field experiments were in the range of 1 to 2 m (Fig. 4·4 bottom). Segmentation of the bats from video was performed by modeling the background intensity with a single Gaussian component per pixel. Pixels with outlier intensities were labeled as belonging to a bat. An infrared image with its corresponding segmentation is shown in Fig. 4·4 top. After segmenting a bat, our



(a) (b) (c) (d)

Figure 4·5: Part-based feature: (a) Segmented bat. (b) Boundary contour unwrapped into a 1D signal from which points of high curvature are extracted. (c) Overlaying the thermal view are clusters of wingtip points shown as red dots enclosed by white rectangles. The green dot is the body center. The 2D heading is depicted by a vector from the green dot to the black dot. (The head and tail of the bat are also detected but not used in this work.) (d) Non-maximum suppression yields one wingtip per wing.

method computes a corresponding part-based feature vector, as described in the next section.

4.2.1 Comparing Part-based Features

To reliably compare real thermal infrared images to training samples from our approximate 3D graphics model, we chose the "high level representation" of a part-based feature. It is an 8-dimensional vector that encodes the overall structure of the bat body by storing the 2D position of the wingtips relative to the body center. The body center is approximated by the warmest pixel on the bat [63] and can easily be detected across poses. To detect wingtips, our method extracts the boundary contour of a bat and chooses the points with large curvature, characteristic of wingtips, as candidates. Candidates are then clustered and non-maximum suppression is used to select up to two wingtip positions. Our part-based feature is augmented with a 2D projection of the 3D heading of the bat, obtained from the 3D trajectory associated with that individual bat. The 3D trajectories were generated by using a Kalman filter to track the 2D position of a bat in each view. These 2D positions were combined with camera geometry information (relative rotation and translation) obtained from a DLT based calibration routine [40] to reconstruct a 3D trajectory. The process of feature extraction is shown in Fig. 4·5. It should be noted that the same feature representation is extracted for all training samples. In this case, the process is trivial since the color of the wingtips can be specified and the 3D heading of the model is always known.

To compare features, we chose a simple distance measure that combines a wingtip position distance d_{wt} with a 2D heading distance d_h. The distance between two wingtips is a combination of the difference in their angles (relative to the body center) and the difference in their distances from the body center. The 2D heading distance is the angle between the two vectors. When one of the features has a different number of wingtips detected than the other, a penalty is added for the mismatch. When two wingtips are detected in each feature, the resulting data association problem is solved by choosing the pair among the two possible pairs that minimizes the feature distance.

4.2.2 Experimental Methodology and Results

We designed an experiment where we compared the results of a baseline algorithm and the results of 3D-PEB to those obtained from manual annotations. The baseline algorithm B is derived from 3D-PEB by removing the first three rules from the coarse reduction stage of 3D-PEB and keeping the MRF model unmodified. In this experiment, the state space for each node was set to $k = 100$. For this value of k our Matlab implementation took ≈ 10 minutes to run, with most of the time spent on k-nearest neighbor (k-nn) retrieval, and computation of the potential functions. Relative to this, the time taken by the max-product algorithm to converge was negligible.

Table 4.1: E_o, E_{θ_1}, and E_{θ_2} represent differences in orientation and wing angles, computed by comparing baseline algorithm B and the proposed system 3D-PEB (noted by P), on two datasets with 183 and 93 frames (cn) respectively, with the gold standards established by two annotators, expert annotator A_1 and non-expert annotator A_2.

Id	# fr.	Alg.	Expert Ann. A_1			Annotator A_2		
			E_o	E_{θ_1}	E_{θ_2}	E_o	E_{θ_1}	E_{θ_2}
1	183	B	70°	45°	47°	51°	43°	53°
		P	17°	17°	14°	16°	16°	25°
2	93	B	59°	55°	53°	59°	63°	63°
		P	20°	22°	18°	21°	26°	26°

The experiment was designed to compare the automatic 3D pose estimates from 3D-PEB to a gold standard and thus examine the accuracy of our proposed algorithm. To obtain quantitative gold standard annotations of real image data, we built an annotation tool and asked two annotators to label 3D pose in an image of a bat by manipulating the 3D pose of our bat model until it matched a presented image. All controls on the annotation tool were discretized to 5° increments to make annotation less labor intensive. Annotator A_1 was an expert bat biologist, annotator A_2 a graduate student with less expertise.

As a measure of potential inaccuracies in the automated estimation of 3D pose, we report the differences in orientation and wing joint angles between automatically-produced estimates and those that were based on manual annotations. If unit quaternion q_e is the estimated orientation and q_a the gold-standard orientation, then the difference E_o is defined by the angle, in degrees, between the two unit quaternions. The maximum possible difference in orientation is 90°. The differences in the wing angles are defined by the difference $E_{\theta_1} = |\theta_{1e} - \theta_{1a}|$ in the elbow angle and the difference $E_{\theta_2} = |\theta_{2e} - \theta_{2a}|$ in the wrist angle. The maximum possible difference in wrist or elbow angle is 180°.

Experimental results are based on estimating the 3D pose of 276 frames, taken from 2 different videos, representing the 3D pose of several bats across time and cameras. The first video (Id 1), is a bat emergence where the 3D pose of bats change steadily over time. The second video, (Id 2), is a more chaotic bat emergence where the bats are changing their 3D pose rapidly.

Quantitative results are summarized in Table 5.1. Our proposed algorithm yielded a mean difference in orientation in the range of $16^{\circ}-21^{\circ}$, or $\approx 17\%-23\%$, considering the maximum of 90° difference to be 100%. For the wing angles, the mean differences range from $14^{\circ}-26^{\circ}$, which is $\approx 7\%-14\%$. The baseline algorithm has a significantly higher difference on average, with the mean orientation difference in the range of $51^{\circ}-70^{\circ}$, or $\approx 56\%-77\%$, and mean wing angle difference in the range of $35^{\circ}-71^{\circ}$, which is $\approx 25\%-35\%$.

In addition to quantitative results, we provide qualitative results in Fig. 4·6. The 3D pose estimates our system produced for 2 particular bats, selected from two different videos, are shown on even numbered rows and can be compared visually to the 3D poses observed in the input, shown on odd numbered rows. Input thermal infrared images are at their original resolution. In the model, recall that the left wing is colored blue and the right wing is colored green, with the head of the bat colored black.

4.3 Discussion

Our results are the product of many system components and reflect challenges in both the general problem of 3D pose estimation and those specific to bats and the datasets used for our experiments.

A general 3D pose estimation method must find a way to deal with ambiguities

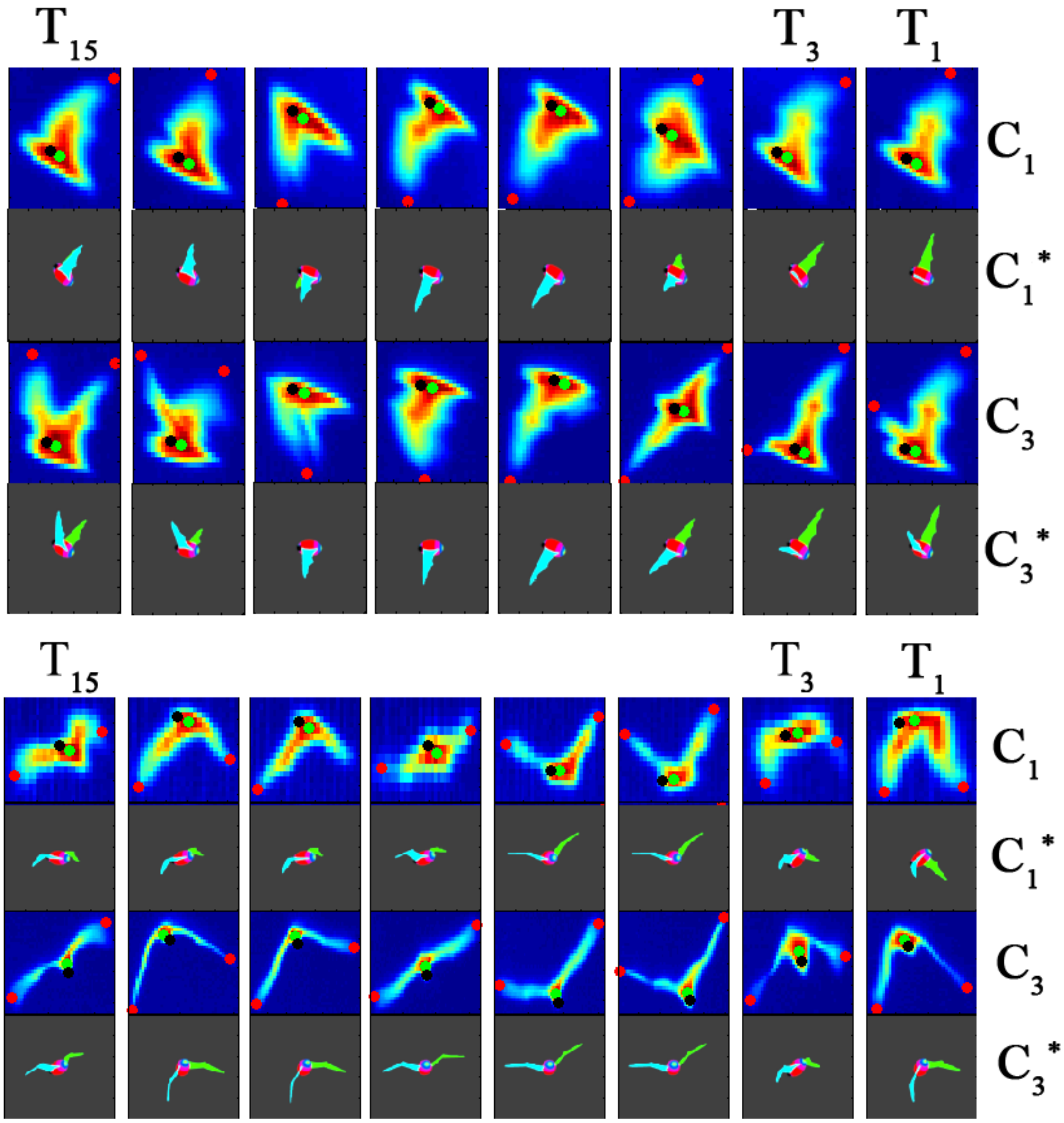


Figure 4·6: **Top:** The rows labeled C_1 and C_3 show frames of a flight sequence from video 1, seen from camera 1 and 3, respectively, with the part-based feature of the bat overlaid. The 3D pose estimates produced by 3D-PEB are shown in rows C_1^* and C_3^*. From right to left, every other frame is shown ($T_1 - T_{15}$). **Bottom:** Estimates for a flight sequence from video 2.

arising when multiple 3D poses map to nearly identical projections. In our work, ambiguities are dealt with by filtering out upside down poses and 3D poses that disagree with the heading of the bat. The errors obtained by baseline algorithm B help validate the need for these particular rules. The baseline algorithm performs significantly (≈ 3 times) worse than 3D-PEB because, without filtering, many incorrect 3D poses remain in the state space, and the MRF model is forced to find the best solution among mostly incorrect candidates.

Many of our design decisions have been influenced by the application domain of bats. Bats in the wild fly quickly, so keeping them in the field of view for a reasonable amount of time requires placing the cameras at a distance from their flight paths, yielding relatively low spatial and temporal recording resolutions. At a spatial resolution of approximately 30 by 30 pixels per bat, its overall body shape can be seen with some coarse detail on the wings, head, and tail (Fig 4·6). At a frame rate of 131.5 fps, the wingbeat cycle can be seen in a choppy fashion. This level of detail in the input video influenced both the 3D bat model we developed and the features we used.

Consequently, a fundamental challenge has been to compare images generated from our 3D bat model with images generated from a real, fast moving, articulated bat. We initially chose low level features based on moments, and contours, but these failed to capture semantic similarity across these two domains. We observed this failure when a k-nn retrieval produced few good matches and many bad ones. Our part-based feature, which captures the overall body and wing structure, was more successful at matching across domains. Detecting the body parts of bats in sequences of low resolution images is a largely unexplored problem and solutions here could accelerate progress in this application domain. More generally, accurate graphics models may not be available for a variety of animals and other researchers may find part-based features beneficial for relating projections from approximate graphics models to real image data.

Explicit experiments were not carried out to determine robustness of our estimates with respect to input segmentations. Segmentation quality was not a major concern for the datasets used because the background was mostly sky and slow moving.

The frames chosen from both videos, for evaluation, are typical of the thousands of other frames in those videos. The bats sampled for our experiments, span a variety of 3D poses, including less complex and more complex ones.

The evaluation of our method relies on the labor-intensive process of obtaining a gold standard. A domain expert was required, who had limited time, making a larger-scale experiment difficult to carry out. This hurdle should be considered by anyone wishing to analyze video of wild animals, whose motions and articulations are complex and not amenable to typical marker based motion capture approaches.

4.4 Summary

In this chapter, we have described the design of a system which can automatically estimate the articulated 3D pose of bats flying in the wild, as well as important considerations taken during the development of this system. In the context of the challenges discussed, as well as the novelty of our results, we find our automated estimates more than suitable for describing coarse 3D pose of flying bats in the wild. Furthermore, the results presented in this chapter represent the first in the literature to show quantitative and qualitative results for the automatic 3D pose estimation of bats in the wild. Future research directions to consider include: looking at how sensitive our system is to errors in camera calibration and experimenting with different potential functions.

Chapter 5

Discovering Useful Parts for Pose Estimation in Sparsely Annotated Datasets

The work described in this thesis chapter is inspired by the question of how hawkmoths (*Manduca sexta*) fly in varying wind conditions, a problem recently studied by Ortega-Jimenez et al[59]. In their work, hawkmoths were placed into a vortex chamber where their flight was captured using multiple high-resolution, high frame-rate cameras. Computer vision was used to analyze the flight sequences of hawkmoths. First key body landmarks were localized across multiple camera views and time. Secondly, 3D positions of these landmarks were reconstructed across time. While Ortega-Jimenez et al. [59] obtained interesting results, their approach to landmark localization only works on datasets where the hawkmoth is observed from a particular view point, thus limiting the general applicability of their approach. By contrast, we propose an approach for landmark localization that does not place any restrictions on the view point. For the rest of the paper we will use the term landmark localization interchangeably with 'pose estimation', as the pose of a hawkmoth, and animals in general, can be specified by the position (localization) of key landmarks.

In the field of computer vision, pose estimation is a fundamental research problem that has received a lot of attention. Among the large body of works that exist, part-based models in particular have shown great success in both 2D and 3D human

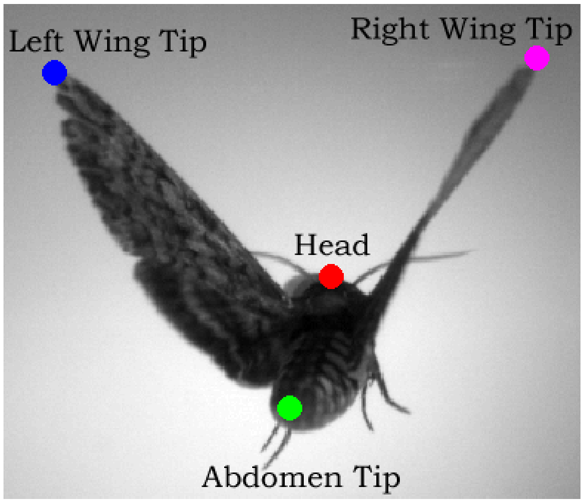


Figure 5·1: A Hawkmoth (*Manduca sexta*) is viewed from behind. Four key body landmarks sufficient for describing the pose of the moth are labeled with text alongside a colored circle at the location of the landmark.

pose estimation [2, 3, 14, 13, 20, 28, 29, 34, 41, 47, 62, 75, 80, 84, 86]. Part-based models have a common approach of modeling an object by a collection of parts. The definition of what a part is varies, but common to all of the mentioned approaches, the representation of a part is learned from annotations provided with training images.

We argue that the complete dependence of part-based models on annotations is a weakness, especially limiting in applications where training data is sparsely annotated. Consider the problem of localizing the positions of four landmarks of interest, shown in Figure 5·1, on a hawkmoth test image. Assume training images are given with those same landmarks annotated. Part-based models would do their job of modeling parts based on annotations, while regions of the training images without annotations, including most of the wings, abdomen, and antennae, risk being thrown away. Thrown away regions may contain parts which are helpful for localizing the landmarks of interest. We hypothesize that augmenting traditional part-based models with parts discovered from the unannotated regions of training images can improve the localization accuracy of landmarks of interest, especially in sparsely annotated datasets.

We now summarize the main contributions of our work.

1. We propose a novel approach to pose estimation in sparsely annotated datasets. Our approach augments traditional part-based models with useful information derived from parts that are discovered automatically from unannotated regions of training images.
2. We demonstrate experimentally that our approach leads to better pose estimation accuracy compared with a baseline representative of traditional part-based models.
3. We show that our approach is well suited for the problem of hawkmoth pose estimation and is more general and more accurate than the recent work by Ortega-Jimenez et al. [60].
4. We introduce the HRMF (High-Resolution Moth Flight) dataset, which as far as we know will be the first high-resolution, high frame-rate, video dataset capturing detailed flight of a flying animal that is made publicly available with part annotations and segmentations.

In Ortega-Jimenez et al. [59, 60], work we aim to improve over, a hawkmoth is segmented in multiple camera views and then various heuristics are used to localize the image location of the head, abdomen tip, left wing tip, and right wing tip (Figure 5·1). Specifically, the left and right wing tips were localized in one of the camera views for frames where the moth was in a particular phase of its wingbeat cycle. The head and abdomen tip were localized by removing the wings from the segmented moth using temporal information and then using morphological operations to remove the antenna and proboscis of the moth. Extrema along the boundary of the remaining

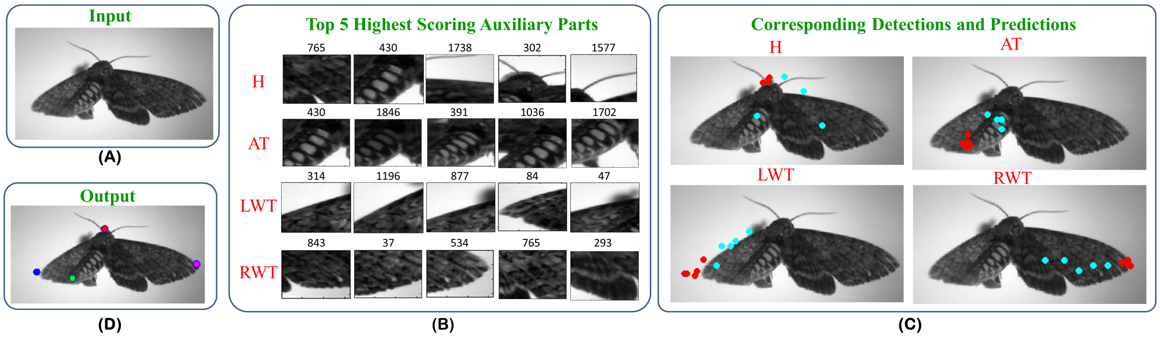


Figure 5·2: Illustrating example of how auxiliary parts are leveraged: (a) Input test image. (b) For each semantic part (**H**ead, **A**bdomen **T**ip, **L**eft **W**ing **T**ip, and **R**ight **W**ing **T**ip) we show the top 5 highest scoring auxiliary parts. Each auxiliary part is labeled by an index and represented by one example patch. (c) For each semantic part we show where the top 5 auxiliary parts were detected on the test image (cyan circles) and where their predictions or votes went (red circles). Notice that the predictions are visually close to the location of the semantic part. Also note that the votes are weighted but for simplicity we do not reflect that in our visualization. (d) After integrating votes from *all* auxiliary parts with the Mixture of Pictorial Structures model we obtain the final pose estimate (part localizations) shown.

connected component were then classified as the head and abdomen tip. To localize landmarks across all camera views epipolar geometry was leveraged.

Looking beyond pose estimation, there have been recent works on unsupervised and weakly supervised part discovery [24, 46, 74]. These works showed the utility of the parts they discovered by using them as feature representations of scenes for supervised scene classification. Our work takes inspiration from these methods and uses a simpler part discovery approach for the problem of pose estimation.

5.1 Methods

We begin this section by specifying our research problem and giving an overview of the proposed solution.

Problem: Given a test image of an object or animal of interest, estimate the image locations $L = \{L_1, L_2, \cdots, L_n\}$ of a set of predetermined landmarks $S = \{S_1, S_2, \cdots, S_n\}$. The estimated locations $L_i = (x_i, y_i)$ of landmarks S_i should be as close as possible to ground truth locations L_i^*. We assume that a training set is available as input, where images containing the object or animal of interest have the image locations of landmarks S annotated.

While this problem formulation is quite general, our application targets the study of flying animals like hawkmoths. As a result, the landmarks to be localized are parts of the animal's body that are meaningful for the analysis of flight. Therefore, for the remainder of the paper we will refer to the landmarks to be localized as semantic parts.

Proposed Solution (Overview): Our proposed solution takes a representative part-based model (Section 5.1.1) and modifies the part appearance likelihood terms so that semantic parts can be localized with higher accuracy. Part appearance likelihoods are improved by incorporating useful information obtained from parts discovered automatically from *unannotated* regions of training images (Section 5.1.2).

Sparsely Annotated Datasets: Our work targets datasets that are sparsely annotated like the hawkmoth dataset, a sample of which is shown in Figure 5·1. Formally, a dataset is sparsely annotated when there exist one ore more unannotated 'parts' present in training images, whose detection would be predictive of the location of one or more semantic parts. This loose definition means many datasets are sparsely annotated. In the case that a dataset contains annotations for all useful 'parts' and is therefore not sparsely annotated, our approach will not have any useful parts to discover and it will default to a standard part-based model.

5.1.1 Mixture of Pictorial Structures

The basis of our approach is a mixture of pictorial structures (MPS) model. A pictorial structures (PS) model [28] represents an object or animal by a collection of parts whose relationships are modeled with a tree structured graph. Mathematically a PS model can be written as

$$p(L|I) \propto p(I|L)\, p(L) \tag{5.1}$$

where $p(L)$ is a prior distribution on the locations of n parts, and $p(I|L)$ is the appearance likelihood term describing how well image evidence I agrees with part configuration (localization) L. The posterior distribution $p(L|I)$ describes the most probable part configuration(s) given the image evidence. The spatial relationship between parts is encoded by a tree structured graph with vertices V and edges E. If (i, j) is an edge in E then there is a preferred spatial relationship between parts i and j in the model. Using this information, along with the assumption that parts do not overlap in the image evidence, the posterior can be rewritten as:

$$p(L|I) \propto \left[\prod_{i=1}^{n} p(I|L_i) \prod_{(i,j)\in E} p(L_i, L_j) \right]. \tag{5.2}$$

PS models can also be used as individual components of a mixture model, which is an effective way to capture variation in the appearance and relationship of parts due to changes in pose [2, 27, 45, 86]. A mixture of PS can be written as:

$$p(L^k|I) \propto \left[\prod_{i=1}^{n} p(I|L_i^k) \prod_{(i,j)\in E^k} p(L_i^k, L_j^k) \right] \tag{5.3}$$

where $p(L^k|I)$ is the posterior given by the k^{th} pictorial structure in the mixture, $k \in \{1, \cdots, m\}$. The objective for the MPS model can then be stated as finding the most probable part configuration among all PS components:

$$\arg\max_{L^k} \quad p(L^k|I) \ . \tag{5.4}$$

Further design and implementation details of the MPS model are provided in Section 5.2.2.

5.1.2 Discovering Auxiliary Parts

The MPS model described in Section 5.1.1 is a good starting point for localizing semantic parts. However, there are potentially useful parts in the training images that are not annotated, which a MPS model (like most other part-based models) cannot make use of. Instead of letting these potentially useful parts go to waste we think of them as auxiliary parts that can be discovered and incorporated into a MPS model. We say an auxiliary part is useful if its presence in an image can be used to predict where one or more semantic parts are located. Auxiliary parts are not required to have semantic meaning.

To discover *useful* auxiliary parts, our method first discovers auxiliary parts and then determines which are sufficiently useful (predictive) to keep. Discovery of auxiliary parts begins by an image patch generation step. All training images are segmented and image patches are extracted from the segmented regions. To avoid generating too many patches, no patches can be extracted too near to an already extracted patch. The large set of patches generated by this step is then represented by a feature and clustered into visually similar clusters. A single auxiliary part can then be thought of as a model of the appearance of patches belonging to a particular cluster. In Section 5.2.2, we detail our choices of features, as well as segmentation and clustering algorithms.

Every patch in a cluster has associated with it the training image it came from, the image location it was extracted from, and the image locations of annotated semantic parts. Using this information, each auxiliary part (corresponding to some cluster C) can be evaluated by how well it predicts one or more semantic parts. Suppose cluster C contains patches $\{P_k\}, k \in \{1, \cdots, K\}$, and each patch P_k is associated with image location $L_k = (x_k, y_k)$, from which it was extracted. Also, let $S_i^k = (x_i^k, y_i^k)$ be the image location of the i^{th} semantic part annotation in the same training image as patch P_k was extracted from. Then, the disagreement $D_i(C)$ on the relative position of semantic part i relative to patch center L_k can be computed across all patches in a cluster by:

$$D_i(C) = \frac{1}{K} \sum_{k=1}^{K} ||(S_i^k - L_k) - \mu_i^k|| \tag{5.5}$$

with

$$\mu_i^k = \frac{1}{K} \sum_{k=1}^{K} (S_i^k - L_k) \; . \tag{5.6}$$

The smaller the disagreement $D_i(C)$, the more cluster C is in agreement on the relative location of semantic part i. If disagreement $D_i(C)$ is less than some chosen threshold τ_i, then the auxiliary part modeling cluster C is considered predictive of semantic part i. Our method obtains the set of useful auxiliary parts A by keeping all auxiliary parts that are predictive of at least one semantic part: $A = \{C : \exists\, i,\ D_i(C) <= \tau_i\}$

5.1.3 Leveraging Auxiliary Parts

Once useful auxiliary parts are discovered, they are used to update the appearance likelihoods of semantic parts in the MPS model. Each auxiliary part (we now drop the adjective useful and consider it implied) is realized as a discriminative classifier learned from patches belonging to its cluster. For a given test image, all auxiliary part detectors are evaluated on the image, and those scoring higher than a threshold are allowed to give a weighted vote for the locations of semantic parts that they are predictive of. The weight of the vote corresponds to the output of the detector, with a larger vote indicating a more confident detection. The prediction an auxiliary part makes is computed by taking the location where the auxiliary part is detected and adding μ_i^k, the average displacement of semantic part i with respect to that part. After all votes (predictions) are in, new appearance likelihoods are obtained for each semantic part by a linear combination of the existing appearance likelihoods with the weighted votes. The new appearance likelihoods are used in the MPS model to obtain final pose estimates.

We illustrate how discovered auxiliary parts are leveraged on a test image in Figure 5·2. Specifically, for each semantic part we show exemplars from just the top 5 highest scoring auxiliary parts. These auxiliary parts are then detected in the test image and their votes for semantic parts are recorded. The votes obtained for each semantic part can be thought of as a new response map or appearance likelihood which is integrated with an existing MPS model. The evaluation of the updated MPS model on the test image yields the final pose estimate shown in Figure 5·2d. Note the top 5 auxiliary parts are shown for ease of visualization but many more are used to obtain the output.

5.2 System Design and Implementation

The methods described in Section 5.1 explain the underlying approach that we apply to hawkmoth pose estimation. However, the specific design and implementation of these methods is heavily influenced by the hawkmoth dataset, so in this section we first introduce the hawkmoth dataset and then discuss implementation design and details.

5.2.1 Dataset

For our experiments we use a hawkmoth dataset from Ortega-Jimenez et al. [60] which captures an individual hawkmoth (*Manduca sexta*) hovering in a vortex chamber where the wind intensity is high. The hawkmoth dataset comes from a camera equipped with a 28 mm lens which records at 400 frames per second and has a resolution of 600 × 800 pixels. For all our experiments, we consider the semantic parts of a hawkmoth to be the left wing tip, right wing tip, abdomen tip, and head, the same parts that were identified as meaningful for 3D pose by biologists in [59, 60]. The experiments we perform in Section 5.3 evaluate how accurately different algorithms localize these four parts.

To facilitate the evaluation of machine learning based algorithms on this dataset we annotate the image location of the four semantic parts in 421 images. The high-resolution hawkmoth image data along with part annotations and segmentations is being made publicly available[1] to encourage more computer vision researchers to evaluate their approaches on this unique biology dataset.

5.2.2 Design and Implementation Details

The MPS model introduced in Section 5.1.1 serves as a baseline algorithm in our work. In this section we give implementation details.

Mixture of Pictorial Structures Baseline

The individual components of the mixture model are PS models with spatial terms learned from 2D pose clusters. Clustering of poses is done by first gathering annotated image locations of the four semantic parts across all training images. If a semantic part is occluded, an annotation is still provided using an educated guess. The 2D pose

[1] `http://www.cs.bu.edu/~betke/research/HRMF/`

of a hawkmoth in a training image is then described by the 8 dimensional vector that contains (x, y) annotations for the four semantic parts. These vectors are clustered using affinity propagation [33] which requires an affinity (similarity) matrix as input. We define the distance between 2D poses $D(p_i, p_j)$ to be the Euclidean distance. The similarity is then computed by $S(p_i, p_j) = e^{-(\alpha\ D(p_i,p_j))}$, where α is a scaling parameter. In our experiments we have 26 pose clusters so m, the number of PS in the mixture, is also 26.

We design the PS appearance terms to be shared across mixture components, since a part can appear similar across different poses. For a given part type ($k \in \{1, \cdots, K\}$), patches of size 64 × 64 centered on annotations of that part type are extracted from all training images and clustered into visually similar clusters. Patch appearances are represented with whitened HOG [38] features (WHOG), using a cell size of 8 × 8 pixels. Affinity propagation [33] is used to cluster patches of the same part-type. The similarity of two image patches is computed as the dot product of their respective WHOG features, $S(p_i, p_j) = f_i \cdot f_j$. The appearance of a part cluster is modeled by learning a Linear Discriminant Analysis (LDA) classifier on HOG [22] features, with the positive samples being patches in the cluster, and negative samples being all other patches as well as background patches. In our experiments the number of appearance terms we obtained for each part were: head: 32, abdomen tip: 27, left wing tip: 26, and right wing tip: 17.

We determine which appearance terms are assigned to which visual clusters, the following logic is used: Let Y_l^k, be the training image indices that are assigned to visual cluster k for part-type $l \in \{1, \cdots, n\}$, and let X_i be the training image indices that are assigned to 2D pose cluster i (equivalently the i^{th} PS component), $i \in \{1, \cdots, m\}$. Then, the part appearance represented by the visual cluster Y_l^k is shared with the i^{th} PS model if Y_l^k and X_i have a non-empty intersection.

When evaluating the resulting MPS model on a test image, all appearance terms assigned to a PS component are evaluated and the one scoring highest gives the score for the overall PS model. The evaluation of each PS model in the mixture on a test image (inference of the tree model) is done by dynamic programming, implemented using the generalized distance transform [26] for Gaussian spatial relationships.

Auxiliary Parts

Segmentation and Features

The first part of the patch generation step described in Section 5.1.2 is segmentation of all training images. We perform segmentation of the hawkmoth by observing that most of a training image is brightly colored background. We use a histogram of image intensities to find the threshold where a fixed percentage of pixels are brighter than it. This threshold does a good job of segmenting most of the hawkmoth but tends to miss the antennae. To recover the antennae we add regions of the image that have a large gradient magnitude. Finally, connected components are computed and those that are too large or too small are removed.

From the segmented region of the image, which corresponds to the hawkmoth, we uniformly sample patches of size 64 × 64 pixels, subject to the constraint that no patches are extracted from within 8 pixels of a previously extracted patch. Across all training images this results in approximately 36,000 patches. For each patch, dense SIFT [55] is extracted and used to compute a bag of words (BOW) feature. The BOW dictionary is built using k-means on dense SIFT keypoints with $k = 500$. To preserve some spatial information the BOW feature is computed for a two level spatial pyramid. The resulting feature is the concatenation of a 500 dimensional histogram for the whole patch (first level), and 500 dimensional histograms for each of the four quadrants of the patch (second level). The total feature dimension is 2500.

Clustering

We cluster patches using a greedy strategy where clusters are formed one at a time until all patches have been considered. Algorithm 1 shows our algorithm in pseudocode. To form a cluster, first a seed patch i is randomly selected from unclustered seeds S. Then from available patches P, the k patches that are most similar to the seed are found using histogram intersection and used as the initial cluster Q. To ensure the cluster Q is visually similar to and in agreement with the seed, several pruning steps are performed.

The first pruning step involves computing an alignment energy (SIFT Flow [55]) E between all patches in Q and the seed. If the alignment energy for any patch is above some threshold β, it is considered not visually similar enough to the seed and thus discarded from the cluster. The second pruning step involves computing the disagreement (Section 5.1.2) in semantic part prediction between each patch in Q and the seed. If any patch disagrees with the seed by more than a threshold γ, the patch is considered to be an outlier not representing the same part in a similar pose as the seed and thus it is discarded. If the resulting cluster is too small (size less than α) the seed patch is removed from S and the process repeats. Otherwise, the remaining patches in Q are joined with seed patch i forming a cluster C^* and added to clusterings C. Patches in C^* are removed from P and the process repeats until all seeds have been processed.

To make use of the clusters obtained by our clustering approach we model their appearance with discriminatively trained classifiers. Specifically, a cluster is modeled by extracting HOG features from its patches and then training an LDA classifier. The resulting LDA classifier can be thought of as a detector of an auxiliary part. Furthermore, each auxiliary part is associated with scores indicating how predictive it is of each of the semantic parts. Recall, Section 5.1.3 explains how these auxiliary parts are leveraged.

Algorithm 1 Auxiliary Part Clustering

```
function GreedyClustering($k$,$\alpha$,$\beta$,$\gamma$)
    ▷ Initialization
    $P = \{1, \cdots, n\}$                              ▷ Unclustered patches
    $S = \{1, \cdots, n\}$                              ▷ Unclustered seeds
    $C = \emptyset$                                     ▷ Clusterings initially empty
    while $S \mathrel{!}= \emptyset$ do
        Let $i$ be a random element of $S$
        $C^* =$ getCluster(i,P,$k$,$\beta$,$\gamma$)
        if $|C^*| >= \alpha$ then
            $P = P \setminus C^*$
            $S = S \setminus C^*$
            $C = C \bigcup \{C^*\}$                     ▷ Add cluster to clusterings
        else
            $S = S \setminus i$                         ▷ Bad seed patch
    return $C$
function getCluster(i,P,$k$,$\beta$,$\gamma$)
    $Q =$ getKMostSimilarPatchesToSeed(P,i,$k$)
    $E =$ getAlignmentEnergyToSeed(i,Q)
    $H = \{q \in Q : E(q) >= \beta\}$
    $Q = Q \setminus H$
    $D =$ getPatchesDisagreeWithSeed(i,Q,$\gamma$)
    $Q = Q \setminus D$
    return $\{Q \bigcup i\}$
```

5.3 Experiments

All experiments are performed on the hawkmoth dataset described in Section 5.2.1. To facilitate our machine-learning based approach we randomly split the 421 annotated images into a training set (211) and testing set (210). All results are based on evaluation on the testing set.

Pose estimation performance on a given test image is measured by localization error for each semantic part S_i. In particular, for semantic part S_i the localization error is measured by the Euclidean distance between the algorithm's part localization $L_i = (x_i, y_i)$ and human annotated ground truth $L_i^* = (x_i^*, y_i^*)$.

We performed three experiments, each evaluating the accuracy of different algorithms on the hawkmoth dataset. The first experiment establishes a baseline level of performance by applying the MPS model. The second experiment determines the performance gained when using auxiliary parts to update the appearance likelihoods

Table 5.1: Summary of quantitative experimental results. For each semantic part type, the mean μ, standard deviation σ, and Mean Squared Error (MSE) of the error distribution associated with each algorithm (**O**rtega-Jimenez, **B**aseline, and **P**roposed). Note: all values are rounded.

Alg.	H			AT			LWT			RWT		
	μ_h	σ_h	MSE_h	μ_a	σ_a	MSE_a	μ_l	σ_l	MSE_l	μ_r	σ_r	MSE_r
O	22	17	765	12	8	201	28	46	2856	19	21	777
B	19	11	478	23	36	1783	10	10	191	12	17	419
P	8	3	72	9	4	106	9	6	115	10	9	187

in the MPS baseline. The third experiment establishes how the existing approach of Ortega-Jimenez et al. [60] performs on this dataset.

Quantitative results for each algorithm are summarized in Figure 5·3 and Table 5.1. Specifically, Figure 5·3 gives a more visual representation of the distribution of errors (purple/magenta squares) for each algorithm on each semantic part. The mean errors are represented by the width of the bar graphs with the numeric value also displayed just to the right of the bar. To help compare the overall distribution of errors across algorithms, Table 5.1 gives the mean, standard deviation, and the mean squared error (MSE).

Qualitative results are shown on 8 test images in Figure 5·4, and 4 test images containing occlusions in Figure 5·5. The localizations output automatically by each algorithm are shown as colored circles and ground truth annotations are shown as orange stars. Figure 5·6 helps connect quantitative error to qualitative error by visualizing what localizations that are 10, 20, 30, 40, and 50 pixels from ground truth look like.

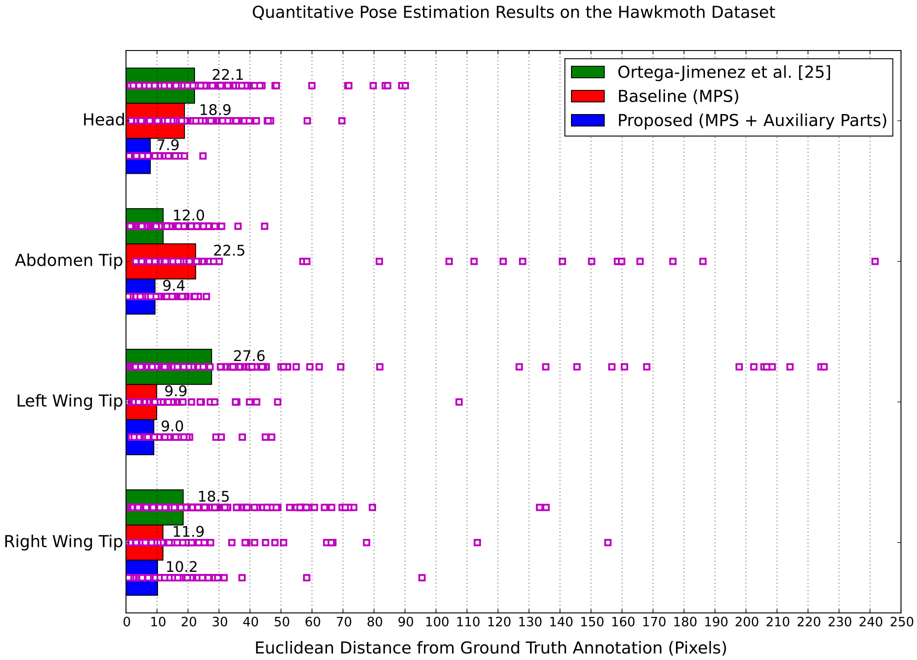


Figure 5·3: Quantitative results which summarize error distributions of the baseline, proposed approach, and Ortega-Jimenez et al. [60], on a hawkmoth test set of 210 images.

5.4 Discussion

Our experimental results show quantitatively and qualitatively that our proposed algorithm outperforms the MPS baseline and the work of Ortega-Jimenez et al. [60] across all semantic parts. Table 5.1 makes this clear because the MSE of the proposed approach is not only the lowest among algorithms but it has no more than approximately half the MSE of the next best approach.

When comparing the proposed approach with the MPS baseline we gain an insight into how much and where discovered parts are helping. The largest improvements of the proposed approach over the baseline happen for the abdomen tip and the head. We believe the reason for this gap is that our proposed approach is able to discover that there exist antennae and abdomens (discovered parts), and that they are predictive of where the head and abdomen tip are. This process is demonstrated in

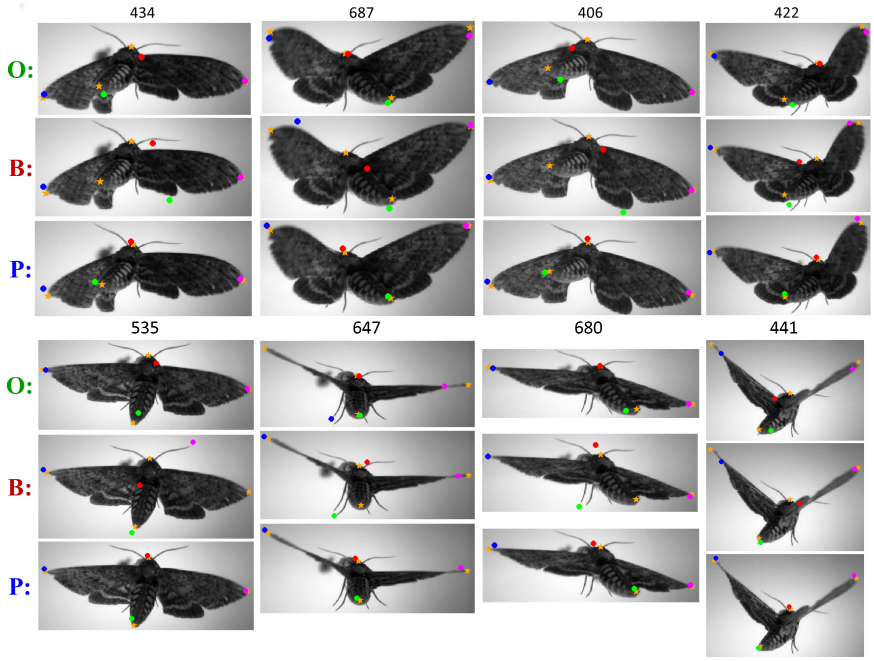


Figure 5·4: Qualitative results for the **B**aseline, **P**roposed approach, and **O**rtega-Jimenez et al. [60] on 8 test images. Orange stars are ground truth annotations. Circles represent part localizations output by the corresponding algorithm. Red, Green, Blue, and Magenta for the Head, Abdomen Tip, Left Wing Tip, and Right Wing Tip respectively.

Figure 5·2. Furthermore, the head and abdomen tip are not very discriminative due to their lack of texture, which greatly hinders the performance of the baseline. For cases where body parts are occluded, as in Figure 5·5, both approaches are able to guess where the occluded part should be located. This can be attributed to the spatial terms of the PS models that learn common configurations of body parts. The proposed approach is also potentially advantageous in occlusion cases as is demonstrated by the fact that antennae can help predict an occluded head.

The MPS baseline is a baseline we created to represent works that extend pictorial structures both with global mixtures and local (part-level) mixtures. We feel our comparison with this baseline accurately reflects the advantage our proposed approach over these types of part-based models.

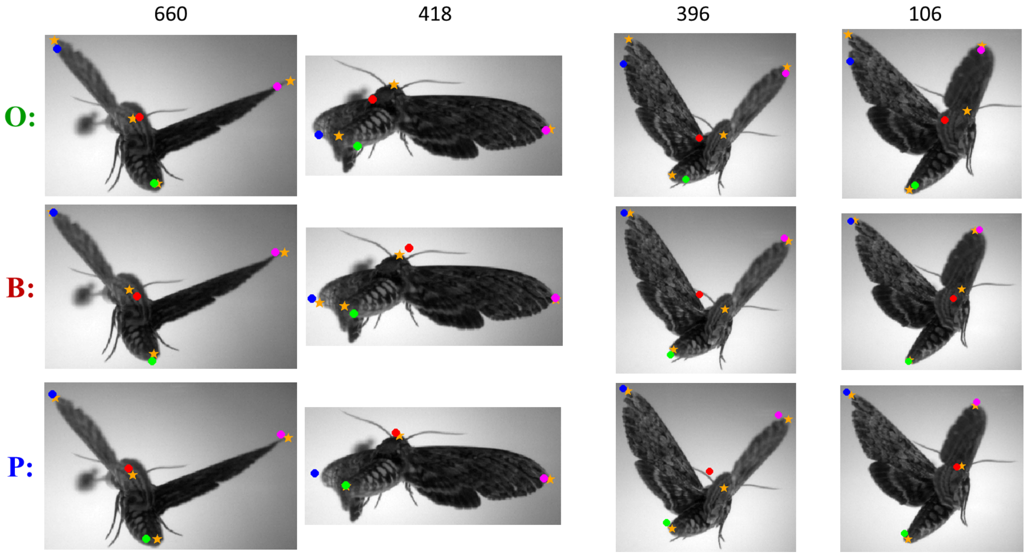


Figure 5·5: Qualitative results for the **B**aseline, **P**roposed approach, and **O**rtega-Jimenez et al. [60] on test images where a semantic part is occluded. The head is occluded by the left wing in frame 660, and by the right wing in frames 396 and 106. In frame 418 the left wing tip is occluded due to its deformation.

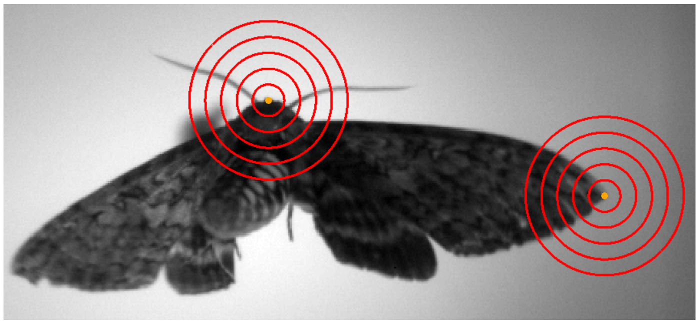


Figure 5·6: Visualization of different levels of localization error. Red rings are drawn with radiuses increasing from 10 pixels to 50 pixels showing how that much localization error looks like relative to the ground truth annotations for the head and right wing tip (orange circles).

One of the core challenges faced in our work is the discovery of parts from unannotated regions of hawkmoth images. Part discovery required both figuring out which features and clustering algorithms would work well with hawkmoth data, while simultaneously considering the computational challenges involved with clustering tens of thousands of patches, and storing their respective feature representations in memory. We found that dense SIFT was a useful feature to represent patches as it could capture the finer details of the hawkmoth's texture. For clustering patches, we chose a greedy algorithm as to reduce the time required to produce clusters. We also found it important to use an alignment energy, computed with SIFT Flow [55], as a way to remove outliers from clusters. Unfortunately, the computation of alignment energy is slow, resulting in a clustering algorithm that takes on the order of a full day to run. Despite our efforts in reducing the time complexity of part discovery, we find that our approach is not suitable for use with training sets that are large (those leading to hundreds of thousands of patches).

5.5 Summary

In this chapter we have proposed a novel way to increase pose estimation accuracy by using automatically discovered auxiliary parts to generate better appearance likelihoods which can then be fed into traditional part-based models like the MPS model. Our experiments on the hawkmoth dataset give quantitative and qualitative support to the value of our proposed approach over traditional part-based models. Our approach also yields significantly more accurate hawkmoth part localizations than previous work [60], while being more general in applicability. Potential future research directions include: extending our proposed approach to a multi-view dataset to obtain more accurate analyses of 3D hawkmoth flight.

Chapter 6

Automating Image Analysis by Annotating Landmarks with Deep Neural Networks

We begin this chapter by motivating the importance of landmark localization approaches in the context of estimating 3D quantities of an animal from video. The estimation of 3D quantities of an animal, typically requires that one or more landmarks from the body of the animal are annotated (marked) in video recordings from multiple cameras. Figure 6·1, illustrates this process where landmarks from a hawkmoth are annotated and then used to estimate the 3D configuration of a hawkmoth. From left to right, the first two plots in Figure 6·1 show landmark annotations for four different landmarks, and the third plot shows the estimated 3D positions of the landmarks.

In this thesis chapter, we focus on the general problem of *automatic* annotation of landmarks, which we apply to the study of the 3D flight of hawkmoths. Ideally, an automatic landmark annotation method should take an image as input and subsequently output the 2D image location(s) of one or more landmarks. In the analysis of animal flight, annotating landmarks has traditionally involved manual labor from one or more people. In the case of annotating numerous landmarks in hundreds or thousands of frames across multiple cameras, this process can become laborious and time intensive, which motivates the need for automatic methods. Existing approaches

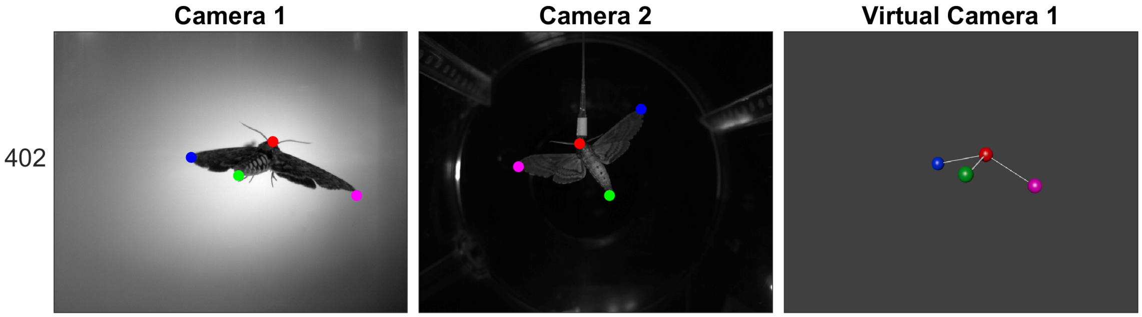


Figure 6·1: **Example of obtaining 3D quantities from video data**. Four landmarks from a hawkmoth are annotated in two images taken simultaneously by two calibrated cameras, denoted Camera 1 and Camera 2. The four landmarks consist of the head (red circle), abdomen tip (green circle), left wing tip (blue circle), and right wing tip (magenta circle). After processing, these annotations are used to estimate the 3D positions of the four landmarks, shown in the right-most plot.

to landmark annotation range in how automated they are. In the work of Tobalske et al. [78], landmarks on a hummingbird were marked with white paint and subsequently manually annotated in video data. Shelton et al. [70] manually annotated natural landmarks on cliff swallows. Bergou et al. [8] reduced the amount of annotation necessary by manually annotating landmarks of bats in an initial set of frames and then using tracking algorithms, which can be corrected with user input, to follow and annotate the landmarks over time. Ortega-Jimenez et al. [60] achieved a mostly automated approach to landmark annotation by setting the view point of the camera in a way that facilitates the use of simple image processing steps for locating parts of a moth.

Some alternative approaches obtain 3D flight kinematics without the need to annotate landmarks. These approaches rely on a 3D graphics model of the animal and use a "registration" method to align the model to 2D image data. Fontaine et al. [32] built a 3D graphics model of a *Drosophila* fly and estimated its 3D motion by aligning the model to 2D image features. In the work by Breslav et al. [16], a 3D graphics model of a *Tadarida brasiliensis* bat was built and used along with a

Markov Random Field to find a 3D flight sequence that most agreed with the image data. The reliance of these approaches on accurate 3D graphics models hinders their use in other domains where such models are not readily available or are too costly to construct.

In the field of computer vision, researchers have extensively studied problems that require automatically annotating landmarks (also referred to as localizing landmarks) in images and videos. Many works in computer vision focus on solving these problems in videos of people. Zhu and Ramanan [86], for instance, proposed an approach for localizing facial landmarks. Felzenszwalb and Huttenlocher [28] proposed an approach to the problem of 2D pose estimation of humans where the goal is to estimate the 2D position, rotation, and scale of various body parts. More recently, deep neural networks (DNNs) have gained fame in computer vision for producing top results on a variety of tasks including: object recognition and image classification [49, 73], landmark localization [85], and 2D pose estimation [79].

Inspired by the success of DNNs, our work aims to evaluate how suitable DNNs are for accurate and automatic localization of landmarks in the challenging case where the amount of labeled training data is small, as is typical in video data collected for analysis of animals. This is in contrast to the typical usage of DNNs in computer vision where large labeled training sets exist; see, for example ImageNet [23], which contains more than 1 million images, each annotated with bounding boxes for all objects in the image. To facilitate our study we perform experiments on published high speed hawkmoth (*Manduca sexta*) video data [18, 60].

The remaining sections in this chapter address the following questions:

- What are DNNs and how do they work? (Materials & Methods)
- What kind of DNN design can be used to automatically annotate landmarks?

(Materials & Methods)

- What decisions and considerations must take place before training a DNN? (Materials & Methods)
- What level of performance can be obtained using DNNs for automatic landmark localization in hawkmoth videos? (Results)
- How is the level of performance impacted by various factors including: dataset size, dataset augmentation, network architecture, parameter values, and more? (Results)
- How do I go about using DNNs for my own data/application? (Appendix B).

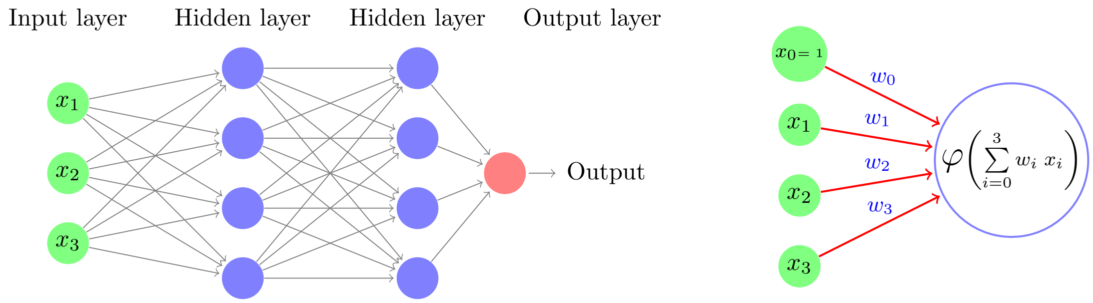
6.1 Materials and Methods

6.1.1 Background

In this subsection we provide a brief introduction to several aspects of *deep* neural networks (DNNs), which are foundational to our work. For a more comprehensive introduction to DNNs, see for example: the work of Goodfellow et al. [37].

Deep Neural Networks

To explain what a *deep* neural network is, we begin by introducing neural networks. A neural network is a model that can be trained to predict a label for a given input. Commonly, neural networks are used to perform tasks such as classification where the predicted label is discrete valued, and regression where the predicted label is continuous valued. A neural network is typically composed of an input layer, one or more hidden layers, and an output layer. Figure 6·2a shows an example neural network which has two hidden layers. The input layer represents the input to the network which in the context of images is often the raw pixel values of an image.



(a) (b)

Figure 6·2: **An example neural network. (a)** This example neural network of depth 3 contains two hidden layers and has an output layer with a single output neuron. Most generally a neural network can have many hidden layers, each containing any number of neurons. The output layer can also have any amount of neurons (outputs). Gray edges represent the weighted connections in the network. The bias terms are not shown for visual clarity but would normally be connected to all neurons. **(b)** This is an illustration of what an individual artificial neuron does. The artificial neuron denoted by the blue circle computes a linear combination of 3 inputs (x_1, x_2, x_3), weighted by the weights (w_1, w_2, w_3) and adds a bias term $w_0 \times x_0 = w_0$. The resulting sum is then input to a non linear function φ that computes a final value which is the output of the artificial neuron.

The hidden layers consist of artificial neurons which collectively map the input to an intermediate representation known as a "feature" vector. The output layer also consists of artificial neurons and it maps these feature vectors to one or more outputs. The depth of a neural network can be defined as the length of the shortest path from the input to the output, which is equal to the number of hidden layers plus one for the output [6]. A *deep* neural network is then defined as a neural network whose depth is large.[1] Each artificial neuron in the network contributes to the overall mapping by computing a weighted combination of its inputs, summed together with a bias term, and evaluated with a non linear function. Figure 6·2b illustrates the action of a single artificial neuron, denoted with a blue circle. The neuron (we drop the qualifier artificial for brevity) computes a weighted sum over 3 inputs (x_1, x_2, x_3), and adds

[1] The minimum depth for a network to be considered *deep* is not well defined in the literature [67].

a bias term represented as $w_0\ x_0$, with $x_0 = 1$. The final output of the neuron is $\varphi\left(\sum_{i=0}^{3} w_i\ x_i\right)$, where φ is a non linear function. More generally a neuron can take n inputs which will either be the input to the network or the output of a hidden layer. Note that in Figure 6·2a the gray edges represent the weights of the network. The biases, which are not shown for visual clarity, should be connected to each neuron.

Given a neural network with initial values for the weights and biases, an input sample can be "forward propagated" through the network. In forward propagation, each neuron performs a computation on its inputs before neurons in subsequent layers perform theirs. The process ends once the output is computed. To produce meaningful predictions, a neural network must be trained. The goal of training is to modify the initially set weights and biases of a neural network so that it predicts labels as accurately as possible for a set of labeled input samples known as the training set. To evaluate how well a network performs prediction on a training set, a "loss function" is defined. The loss function, or "loss" for short, is a measure of how much the predictions for a training set differ from the true labels and it is a function of the weights and biases of the network. Generally the loss function is designed to reflect the loss averaged over the training set. In our work we will refer to the evaluation of a loss function on the training set, as the "training loss", and the evaluation of a loss function on the testing set, as the "test loss" or "testing loss".

From an optimization point of view, the goal of training is to find the weights and biases that minimize the loss. The standard approach to this optimization is based on the method of gradient descent using back propagation [66]. In the gradient descent method, for a given function $F(\theta)$, where θ is a parameter vector and F is differentiable and defined in a neighborhood of some parameter setting θ_n, one can obtain a new function value $F(\theta_{n+1})$ where $F(\theta_{n+1}) \leq F(\theta_n)$. The new function value is obtained by updating the parameter vector so that it moves in the direction of the negative gradient of F evaluated at the current parameter value. The update in the parameter vector is expressed as $\theta_{n+1} = \theta_n - \eta \nabla F(\theta_n)$, where η is a small scalar. By repeatedly updating the parameter vector in this way the gradient descent method will converge to a local minimum of F, which is also the global minimum if F is convex. In the context of neural networks, the function F is the loss function, and the parameter vector θ is the weights and biases of the network. To compute the gradient of the loss function with respect to the weights and biases of the network, back propagation is used which first computes gradients with respect to weights and biases at the end of the network and then uses the "chain rule" from calculus to compute gradients with respect to the weights and biases in earlier layers. Traditional gradient descent requires that the whole training set is forward propagated before the loss function is evaluated and a single update to the weights and biases is performed, which in practice is expensive since datasets can be large. Instead, stochastic gradient descent (SGD) or mini-batch SGD are used, where the loss function is evaluated on either a single training sample or a subset of training samples, and an update to the weights and biases can be performed much quicker. A more formal discussion on gradient descent and SGD is provided in the work of LeCun et al. [53].

Deep Neural Network Architecture

The neural network in Figure 6·2a has a "shallow" (not deep) architecture, which is useful as an illustration, but is not representative of the kinds of networks being used in practice. Instead, the networks used to solve many real world problems have deeper and more complex architectures. Theoretical results support the idea that deeper architectures can more efficiently represent a function as compared to shallower ones [6]. Today, researchers in computer vision and machine learning continue to design and study DNN architectures with the goal of obtaining the top results on a variety

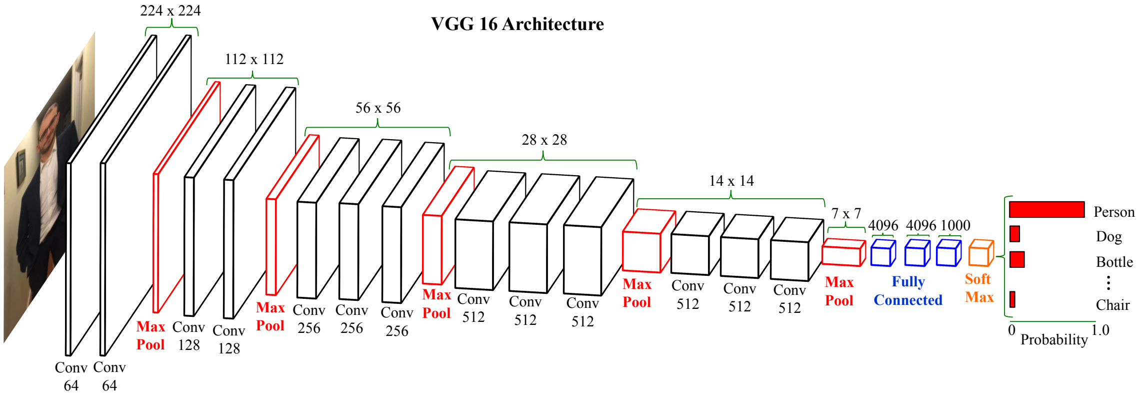


Figure 6·3: **Illustration of the VGG 16 [73] network architecture, layer by layer**. The VGG 16 network consists of 16 layers that have weights and biases. These layers include convolutional layers ("Conv") denoted by black rectangular volumes and fully connected layers denoted by blue rectangular volumes. The height and width of each convolutional layer is specified above the layer and the depth is specified below the layer. Max pooling layers, denoted as red rectangular volumes, have the same depth as the layer that precedes them, and their height and width is the same as the layer that succeeds them. The fully connected layers are 1D columns of neurons with the number of neurons specified above the layer. A soft max layer denoted as an orange rectangular volume is used to convert the output of the neurons in the last fully connected layer to class probabilities. The weights and biases of the network are not explicitly shown. A more in depth explanation of the VGG 16 network and its layers is given in Appendix A. The VGG 16 network was originally trained for the task of image classification. Here we illustrate an example classification where the network is given an input image of a person and the output probability distribution is largest for the output corresponding to the label "person".

of problems.

One type of DNN, known as a Convolutional Neural Network (CNN or ConvNet), has succeeded on various computer vision problems by leveraging a particular architecture. The architecture of a CNN uses "convolutional" layers, which perform the mathematical operation of convolution. The use of convolution is motivated by the idea that local patterns in an image are informative and that informative local patterns should influence the network regardless of their absolute location in an image [52]. Convolutional layers are also beneficial in that they reduce the total number of weights and biases in the network when used instead of traditional layers, which are also called fully connected layers. One of the earliest successes of CNNs was in handwritten character recognition [52]. More recently Simonyan and Zisserman [73] proposed 16 and 19 layer CNNs, called VGG 16 and VGG 19, which achieved the best results in 2014 on the task of classifying and locating objects in images. Figure 6·3 shows the layer by layer composition of the VGG 16 network, and illustrates what a reasonable network output would be for an input image of a person. Unlike the neural network introduced in Figure 6·2a, where all layers containing neurons are arranged as 1D columns, the VGG 16 network has layers consisting of 3D volumes (width x height x depth) of neurons. These layers are the convolutional layers and the max pooling layers, which are labeled in Figure 6·3 with 'Conv' and 'Max Pool', respectively. As before neurons in the same layer do not connect to each other. The VGG 16 network plays an important role in our work because we use architectures that are directly based on this network. For a more in depth explanation of the different types of layers used in the VGG 16 network, which are also common to many other CNNs, see Appendix A.

6.1.2 Experimental Setup

In this section we describe some of the key parameters that need to be considered and set prior to training and testing a DNN. The goal of our experiments is to obtain quantitative results detailing how different values and settings of these parameters impact landmark localization accuracy. Preliminary experiments were performed to find an initial set of parameter values that resulted in a reasonably low test loss, indicating that the network learned sufficiently well to generalize. Many parameter values could be quickly rejected when the training loss was observed to either diverge or to decrease too slowly from a large value. We will refer to the parameter values found in our preliminary experiments as the "default" parameter values. These default parameter values are also used as a guide for what parameter value ranges we consider in our experiments. Exploring how all parameters jointly influence landmark localization is a combinatorial problem and is not feasible to perform due to the large amount of time (tens of hours) it takes to train a *single* DNN. Instead, our experiments evaluate how deviations from default parameter values influence results[2]. In the rest of this subsection we will describe what the parameters are, and where appropriate, we will state what the default values are. Conceptually our experiments can be thought of as an exploration of the parameter space around a point (default parameter values), exploring one dimension at a time.

Datasets

For our experiments we use hawkmoth video data from Ortega-Jimenez et al. [60]. The video data captures an individual hawkmoth (*Manduca sexta*), from multiple cameras, while it hovers in a vortex chamber where the wind speed is high. Specifically, we use videos obtained from cameras 1 and 2, which simultaneously recorded the flight of the hawkmoth. Cameras 1 and 2, which are identical, are equipped with a 28 mm lens, record at 400 frames per second and have a resolution of 600 x 800 pixels. Both of these cameras have been calibrated [76], so their relative positions and orientations in 3D are known. Landmark annotations for 800 frames in each video were obtained by using annotations published by Breslav et al. [18] and performing additional annotations ourselves. Landmark annotations consist of the 2D image position of the head, abdomen tip, left wing tip, and right wing tip. We make the hawkmoth data used in our experiments freely available along with landmark annotations[3].

[2] Alternative approaches for exploring parameter values are discussed in Bengio et al. [7].

[3] `http://www.cs.bu.edu/~betke/research/HRMF2/`

DNN Architecture

The DNN architectures used in our experiments are directly based on the VGG 16 network of Simonyan and Zisserman [73]. The default network architecture used in our experiments is a network we call "VGG 7 + FC8". VGG 7 is what we call the network obtained by taking VGG 16 and removing everything after the max pooling layer that follows the 7^{th} convolutional layer; counting starts from the layer closest to the input. On top of VGG 7 we added a single fully connected layer consisting of 8 neurons (FC8), which produces the output of the network. The quantity of 8 neurons is chosen so that the network outputs a pair of (x, y) image locations for four hawkmoth landmarks of interest: the head, abdomen tip, left wing tip, and right wing tip.

In our experiments we also investigated the performance of architectures similar to VGG 7 + FC8, but with different depths. We will refer to these alternative architectures as VGG X + FC8, where VGG X corresponds to the subset of VGG 16 that remains when removing everything after the max pooling layer that follows the X^{th} convolutional layer.

Network Initialization

There are two primary ways to initialize the weights and biases of a network. The first way initializes the weights and biases to values that were already learned from training the same network, or a superset[4] of it, on a different dataset. A network initialized using this approach is said to be "pretrained". The second way is to initialize the weights and biases manually by choosing a constant or generating values randomly

[4] Here we define the superset of a network a as another network b that contains a as part of its architecture.

from a distribution. Training a network that has been initialized using the second approach is also referred to as training from “scratch”.

In our experiments the first approach is the default way we initialized an architecture. Specifically, for an architecture VGG X + FC8, we initialized the weights and biases of the VGG X portion of the network to the values learned from training VGG 16 on ImageNet, which are freely available for download[5]. For the weights and biases of the fully connected portion of the network (FC8), which is not part of VGG 16, a constant of 0 was used.

We also performed experiments to evaluate how our network performed when trained from scratch. In these experiments the weights were initialized either from a Gaussian distribution (0 mean, 0.01 standard deviation), or using “xavier” initialization, where the weights are drawn from a distribution whose variance is determined by the number of inputs and outputs a particular neuron has [35].

Finetuning

Our default network architecture VGG 7 + FC8 was chosen so that the VGG 7 portion of the network can be pretrained and used as a feature extractor, while the FC8 part of the network performs linear regression and needs to be trained from scratch. As a result, for most experiments training only involves the fully connected layer. Finetuning, however, is an approach where pretrained layers can be further trained to “tune” the network for a particular dataset. In our case, when we perform finetuning all layers of the network are trained. It is worth noting that in libraries like Caffe [44] a learning rate can be specified for each layer of the network. In our experiments we investigated the impact of finetuning VGG 7 + FC8.

[5] http://www.robots.ox.ac.uk/ vgg/research/very_deep/

Dataset Augmentation

Data augmentation is an approach used for increasing the size of a training set by applying transformations to images in the training set [72, 49]. In our experiments we investigate how data augmentation using combinations of translation, rotation, and scale, impact network performance. Additionally, we also investigate how the amount of data augmentation performed, which determines the total number of training samples available, influences performance. Our default data augmentation uses translation alone and results in 200,000 training samples in total. As a frame of reference we also evaluated network performance when no data augmentation is performed.

Translational data augmentation was implemented by taking a 400 x 600 pixel crop from an original 600 x 800 pixel image and subsequently resizing the crop to 224 x 224 pixels. The 400 x 600 pixel crops were taken so that the position of the hawkmoth inside the crop was distributed uniformly at random. The crop was constrained to contain the full hawkmoth body. A bounding box of the hawkmoth body was generated from the hawkmoth segmentations provided by Breslav et al. [18]. Rotational data augmentation was implemented by rotating the hawkmoth around the center of its bounding box, randomly in the range of -45 degrees to 45 degrees. Scaling data augmentation was implemented using the “imresize” function in Matlab which performs downsampling or upsampling to the image using bilinear interpolation. The amount of scaling was chosen randomly in the range of 0.5 to 1.5.

Deep Learning Library

To train and evaluate DNNs we used the publicly available deep learning framework Caffe, developed by Jia et al. [44]. The Caffe library implements mini-batch SGD [53] using back propagation and is able to leverage GPU resources. See Appendix B for an overview of how to use Caffe to train and test DNNs. We provide free software to help facilitate training and testing of DNNs with Caffe[6].

Learning Rate

The learning rate is one of the parameters that greatly impact training of a DNN. The value of the learning rate determines how large of a step is taken during SGD. In the expression $\theta_{n+1} = \theta_n - \eta \nabla F(\theta_n)$, η is the learning rate. In our experiments we used a default learning rate of $100 \times 10^{-12} = 10^{-10}$ for training the fully connected layer (FC8). Our experiments investigating the influence of network architecture and network initialization on DNN performance also consider a range of learning rates.

Batch Size

The batch size is the number of training samples used for a single iteration of training which results in a single update of the weights and biases in the network. The loss function is evaluated on the batch of training samples and as a result subsequent gradient computations depend on these samples. The default batch size we used is 32. Additional experiments were performed to evaluate the influence of batch size on network performance.

Training Iterations

The number of training iterations determines in combination with batch size how many training sample are seen during training, which also determines the amount of time training takes. The default number of training iterations we performed is 10,000. Additional experiments were performed to evaluate how the number of training iterations impacts performance.

[6] `http://www.cs.bu.edu/~betke/research/ALADNN/`

Training and Testing Split

For our experiments (not including multi-view experiments) we had available a total of 800 consecutive frames which were annotated. By default we used the first 400 frames for training (not including any data augmentation performed) and the latter 400 frames for testing. We also performed two additional experiments to study alternative training and test splits. The first experiment studies the impact of making the training and testing sets more similar to each other. This is accomplished by using odd numbered frames for training and even numbered frames for testing. We refer to this alternative split as “interleaved”. The second experiment studies how different sizes of the training set (prior to data augmentation) impacts performance on a test set. Specifically, we used 200, 400, and 600 training images and a random test set of 200 frames.

Evaluation

After a DNN is trained its weights and biases are finalized, and the network can be used on new inputs. For evaluation a set of test images (inputs) are forward propagated through the network and a loss function is evaluated. In all of our experiments we used the loss function defined by: $\frac{1}{n}\sum_{i=1}^{n} ||\mathbf{y_i} - \mathbf{y_i^*}||^2$, where: i denotes the i^{th} test sample out of n, $\mathbf{y_i}$ is the output of the network on the i^{th} test sample represented as an 8 dimensional vector, and $\mathbf{y_i^*}$ is the 8 dimensional vector consisting of ground truth annotations for the i^{th} test sample. We will refer to this loss as the *squared loss* for convenience, though more accurately the loss reflects the expectation (average) of the squared loss due to the $\frac{1}{n}$ term. In all of our experiments we evaluated the performance of a DNN by evaluating this loss function on the test set. Recall, that this is the same loss function that is integral in training and influences the derivatives computed during backpropagation.

For some of our experiments we also report the more interpretable mean absolute error (MAE), which represents the average distance between a predicted landmark location and its corresponding ground truth location. The MAE can be defined as follows: let y_i be the 2D image location of a particular landmark in the i^{th} test image, let y_i^* be the ground truth 2D image location of that landmark in the same test image, and let the error be defined as $e_i = ||\mathbf{y_i} - \mathbf{y_i^*}||$, then MAE is given by: $\frac{1}{n} \sum\limits_{i=1}^{n} |e_i|$.

It is also important that we clarify the resolution of the image upon which these evaluation metrics are computed. In all experiments where squared loss is reported, it is computed on images of size 224 × 224 pixels. The dimensions 224 × 224 represent the width and height of all images used for training DNNs in our experiments. Images of these dimensions are obtained from our original images of size 600 × 800 by taking a 400 × 600 crop, and then rescaling it to 224 × 224. The ground truth annotations (labels) associated with training images are also transformed to be consistent with these dimensions. As a result, after training, our DNNs will output landmark locations relative to an image with dimensions 224 × 224. All training and testing losses reported in our work were computed on images of this resolution. In contrast, in all experiments where MAE is reported, the MAE was computed on images at the original resolution of 600 × 800.

Multi-view

In our multi-view experiments we traind one DNN (DNN$_1$) using video from camera 1, and another DNN (DNN$_2$) using video from camera 2. Both DNNs were trained using our default split, where the first four hundred frames from a video are used for training and the last four hundred frames are used for testing. Once DNN$_1$ and DNN$_2$ are trained they can be used to predict landmark locations for their respective test sets. Given predicted landmark locations $\mathbf{y}_1^i$ for frame i of video 1, and predicted landmark locations $\mathbf{y}_2^i$ for frame i of video 2, 3D triangulation [39] can be used to estimate ("reconstruct") the 3D position of the landmarks at time i, which can be thought of as a representation of the 3D pose of the hawkmoth at time i.

6.2 Results

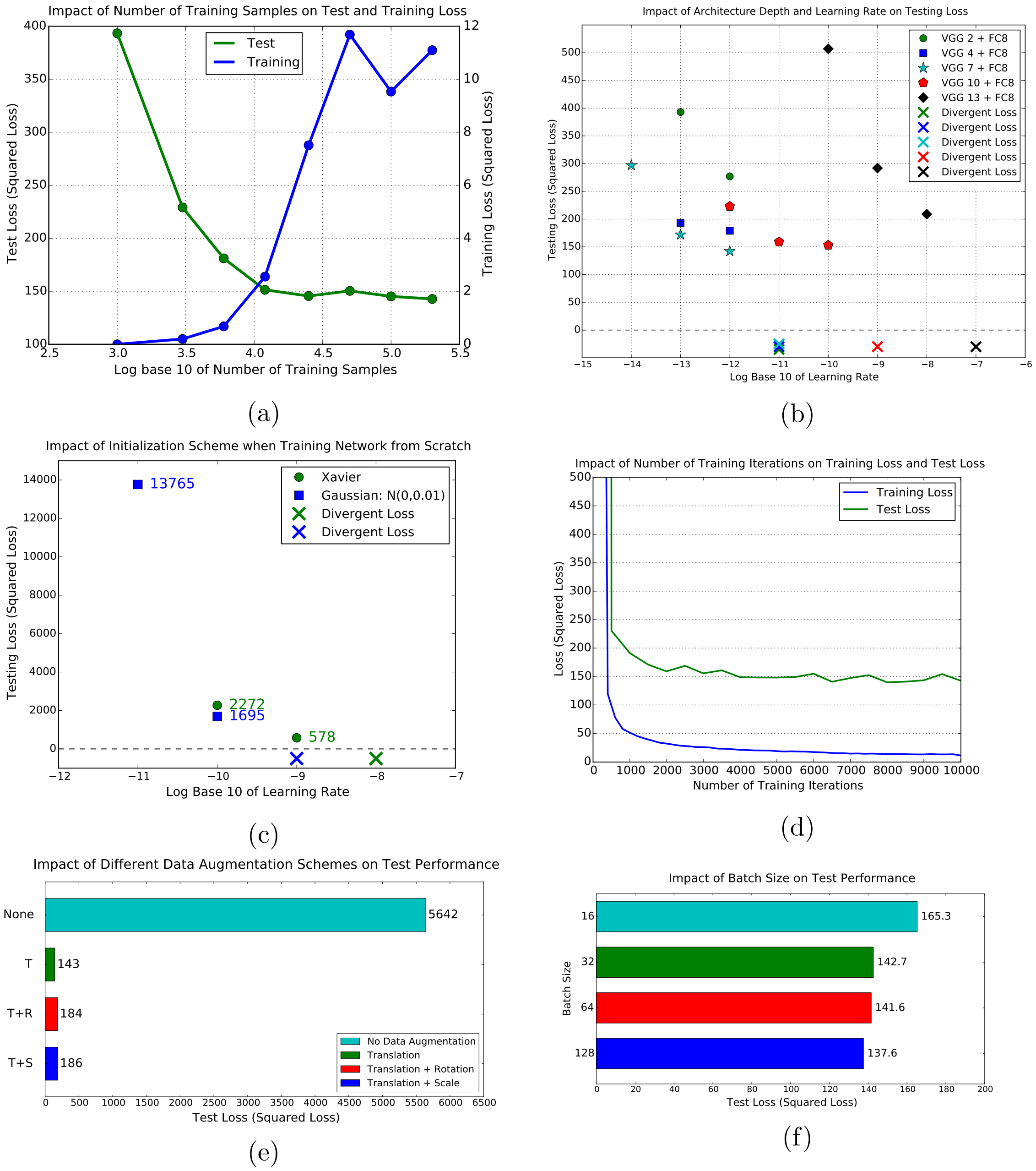
Default Network Parameters

As we have noted earlier, our experimental results show how changes to parameter values influence network performance measured by the test loss, with a point of reference being a DNN trained using default parameter values. The default parameter values are compactly provided in Table 6.1. Also recall that all test loss and training loss values reported in the results were computed on images with dimension 224 × 224, and all MAE values reported are computed on images with the original dimensions of 600 × 800.

Table 6.1: **Default parameter values for:** the number of training samples (NTS), data augmentation type (DA), architecture (Arch.), base learning rate (BLR), learning rate multiplier for VGG layers (VGG LRM), learning rate multiplier for fully connected layer (FC LRM), pretraining of the VGG layers (VGG PT), number of training iterations (NI), batch size (BS), and finetuning (FT). **Note:** to be consistent with Caffe the learning rates for the VGG and FC layers are provided as constants that are relative to the base learning rate. The actual learning rate for the VGG and FC layers is obtained by taking the base learning rate and multiplying it by the VGG and FC multipliers respectively.

NTS	DA	Arch.	BLR	VGG LRM	FC LRM	VGG PT	NI	BS	FT
$2 \cdot 10^5$	T	VGG 7 + FC8	10^{-12}	0	100	✓	10^4	32	X

Number of Training Samples: Figure 6·4a shows how the number of training samples used for training impacts both the final training loss (blue plot) and the final testing loss (green plot). Note the total samples seen during training is equal to the number of iterations performed multiplied by the batch size, which are both



(a)

(b)

(c)

(d)

(e)

(f)

Figure 6·4: **Experimental results show how different parameter values impact the test performance of the trained network.** These results summarize the quantitative impact of (a) the number of training samples, (b) different architectures and fully connected layer learning rates, (c) network initialization from scratch schemes, (d) number of training iterations, (e) data augmentation type, and (f) batch size.

fixed. Thus, the total samples seen during training with default parameter values is $10,000 \times 32 = 320,000$. For the default training set size of 200,000 this means that each sample is seen at least once during training, but no more than twice.

Architecture Type: Figure 6·4b shows the impact of several different architectures on test performance. The architectures tested differ in the subset of VGG 16 used for feature extraction. The smallest network is VGG 2 + FC8 and the largest is VGG 13 + FC8. Each architecture is evaluated over several different learning rates for the fully connected layer. The crosses plotted below the $y = 0$ line indicate the pairs of architectures and learning rates that resulted in the test loss *diverging*.

Training from Scratch: Figure 6·4c shows how different initializations for all the weights and biases of the network impact the final testing loss. Each initialization scheme is evaluated for several fully connected layer learning rates.

Number of Iterations: Figure 6·4d shows how training loss (blue plot) and testing loss (green plot) vary with the number of training iterations performed. The training loss plotted reflects the average training loss over a window of 200 iterations, computed every 200 iterations. Testing loss was computed every 500 iterations.

Data Augmentation: Figure 6·4e shows how different data augmentation schemes influenced test performance. The green plot corresponds to data augmentation done using translations alone (denoted by T). The red plot corresponds to data augmentation done using both translation and rotation (denoted by T + R). The blue plot corresponds to data augmentation done using both translation and scale (denoted by T + S). In all cases data augmentation was used to increase the initial 400 training samples to 200,000 training samples. The final training set consists of 400 original training samples and 199,600 training samples generated from data augmentation. As a point of comparison, the black plot shows the performance when no data augmentation is performed.

Batch Size: Figure 6·4f shows how the batch size impacts test performance. An additional experiment was performed where the batch size was 8 which resulted in the loss diverging.

Finetuning: Finetuning the default network resulted in a negligible difference in testing loss. One experiment performed finetuning by training all layers of the network with the VGG LRM set to 1 and resulted in a test loss of 142.75. A second finetuning experiment was performed where the VGG LRM was set to 10 and resulted in a test loss of 142.79. Increasing the BLR to 10^{-11} and setting both the VGG LRM and FC LRM to 1000 resulted in a test loss of 514.66. The default network which does not use finetuning resulted in a loss of 142.7.

Training and Test Split: Using the default split resulted in a test loss of 142.7 and a training loss of 11.0. The alternative split, where the training and test sets interleave, resulted in a test loss of 33.6 with a training loss of 19.3. Figure 6·6 shows qualitative and quantitative results comparing these two splits. The quantitative results consist of the MAE associated with localizing each landmark. Figure 6·7 shows quantitative results depicting how using training sets of size 200, 400, and 600 influence performance on a test set of size 200, measured with MAE.

Comparison to Previous Works: Figure 6·5 shows how our default DNN, denoted

VGG 7 + FC8, performs on automatic landmark localization on a hawkmoth dataset, relative to competing approaches whose performance was published in the work of Breslav et al. [18]. Performance is measured for each landmark by computing MAE on the test set. The error for each landmark is defined as the Euclidean distance between the automatically generated landmark position and the corresponding "ground truth" (manually annotated) position. To be consistent with results reported in the work of Breslav et al. [18], we used the same training/test split and do not include frames where one or more landmarks were occluded in the quantitative results we report. Specifically, 421 images were used with 211 for training and 210 for test.

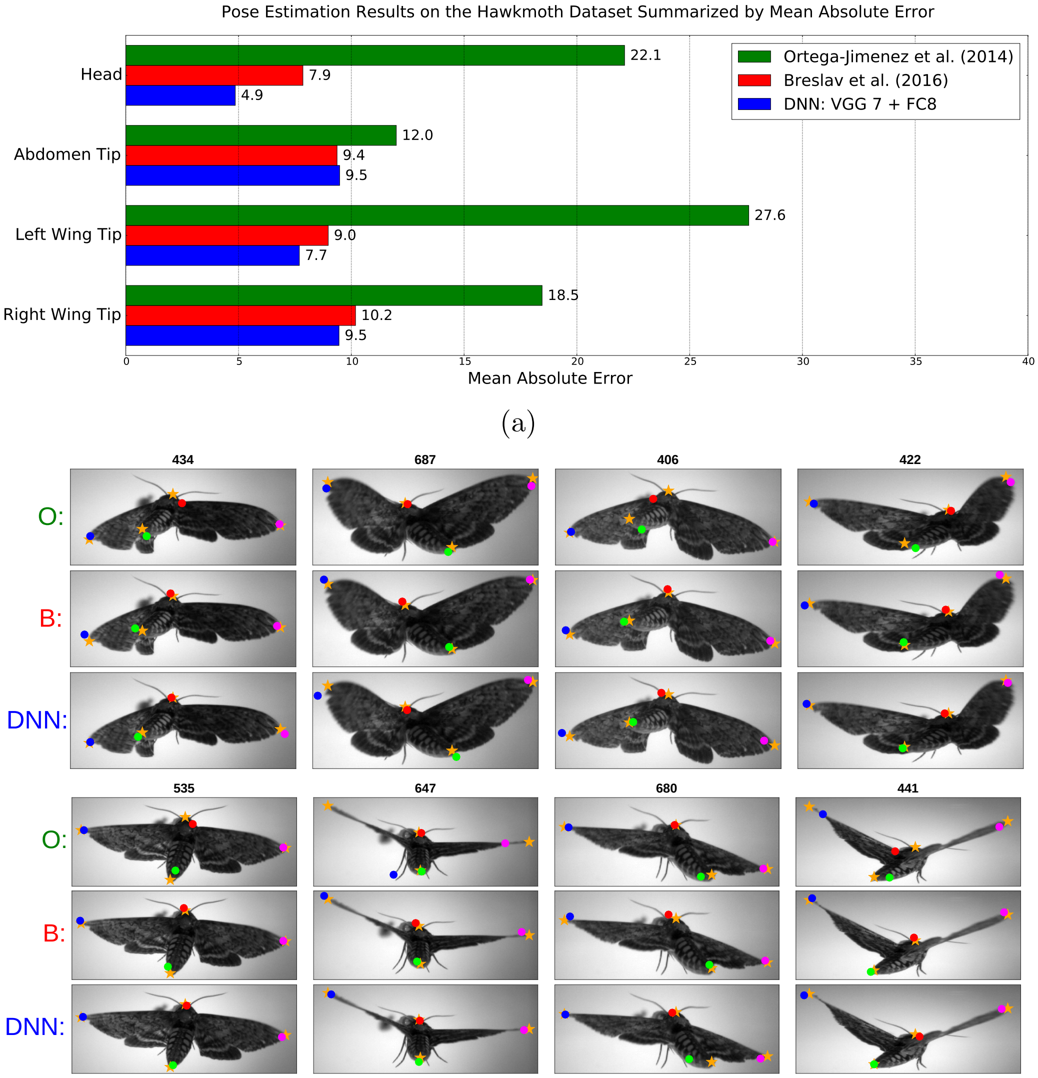
Multi-view: DNN$_2$, our default DNN trained on video from camera 2 with the default train/test split, obtained a test loss of 191.8. Figure 6·8 shows the qualitative performance of DNN$_2$ on 12 different frames from the test set. Figure 6·9, shows the qualitative performance of both DNN$_1$ and DNN$_2$ on landmark localization for frames simultaneously captured by both cameras 1 and 2. The resulting landmark localizations are used to reconstruct the 3D positions of the landmarks. The right-most image in each subplot, labeled "Virtual Camera 1", illustrates the reconstructed 3D configuration (pose) of the hawkmoth.

To assess how accurate 3D reconstructed positions (based on our DNN predictions) are, we compare them with reconstructed 3D positions based on ground truth landmark annotations. Let HA_i be the distance in 3D, at time i, between the predicted head and abdomen tip, and let HA_i^* be the distance in 3D, at time i, between the ground truth head and abdomen tip. Similarly, let LR_i be the distance in 3D, at time i, between the predicted left wing tip and the right wing tip, and let LR_i^* be the distance in 3D, at time i, between the ground truth left wing tip and right wing tip. Then, the ratios $\frac{HA_i}{HA_i^*}$, and $\frac{LR_i}{LR_i^*}$, indicate at time i, how closely a 3D measurement of the distance between landmarks is to the ground truth distance, with a perfect match resulting in ratios of 1. Here we report the mean of these ratio taken over the test set. The mean ratio for the head to abdomen tip distance is 1.0096 and the mean ratio for the left wing tip to right wing tip distance is 0.9847.

6.3 Discussion

Number of Training Samples: Our experiments show that increasing the number of training samples from 10^3 to 10^4 has a significant impact on reducing the test loss from approximately 400 to approximately 150. Beyond 10^4 samples the test loss stays around 150, indicating that the benefit of additional training samples diminishes. Recall that these training samples are a result of performing data augmentation with translations on an initial set of 400 samples. This suggests that creating a training set that is 25 times larger than the original dataset, using data augmentation with translation alone, is sufficient. It is worth emphasizing that these results do not tell us how changes in the original number of training samples would impact the results. The trends observed in this experiment agree with the idea that when the training set is relatively small (10^3), it is easier for a DNN to learn or potentially memorize the data (also referred to as "overfitting"), leading to a small training loss. This overfitting comes at the expense of the ability of the network to generalize which is reflected in larger test losses. The saturation in test loss as the number of training samples goes beyond 10^4 also appears to be consistent with our hypothesis that the benefit of data augmentation is limited and cannot replace more original training samples.

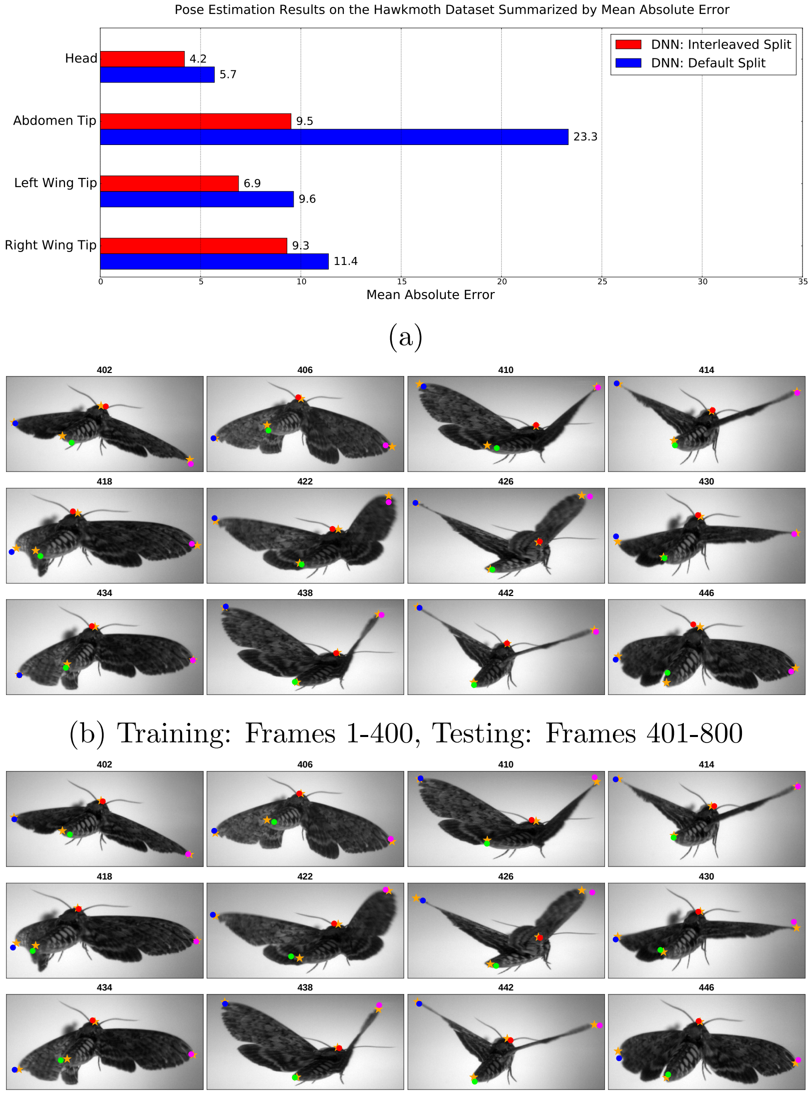
DNN Architecture: Our experiments show that the feature extraction portion (VGG X) of the DNN makes a difference in test performance, but not as much as hypothesized. Our initial hypothesis was that the best performing architecture



(a)

(b)

Figure 6·5: **Quantitative and qualitative comparison of our default DNN, denoted as VGG 7 + FC8, with the two best published approaches for hawkmoth landmark localization.** These approaches include the work of Ortega-Jimenez et al. [60] denoted by "O" and Breslav et al. [18] denoted by "B". (a) Plot showing the mean absolute error associated with each landmark, for each of the methods. (b) Plot showing the predicted landmark localizations (colored circles) along with ground truth landmark localizations (gold stars), for 8 different frames. Each column, labeled with a frame number, shows the performance of the three approaches on that frame.



(a)

(b) Training: Frames 1-400, Testing: Frames 401-800

(c) Training: Odd Numbered, Testing: Even Numbered

Figure 6·6: **Quantitative and qualitative results demonstrating the performance obtained when using the default training split as compared to an interleaved one.** (a) Plot showing the mean absolute error associated with each landmark, computed for both splits, using a 400 frame test set. (b) Qualitative results are shown for 12 frames that are common to the test sets of both splits. Predicted landmark localizations for the default split are shown as colored circles and ground truth landmark localizations are shown as gold stars. (c) Same type of plot as (b), but for the case of the interleaved split.

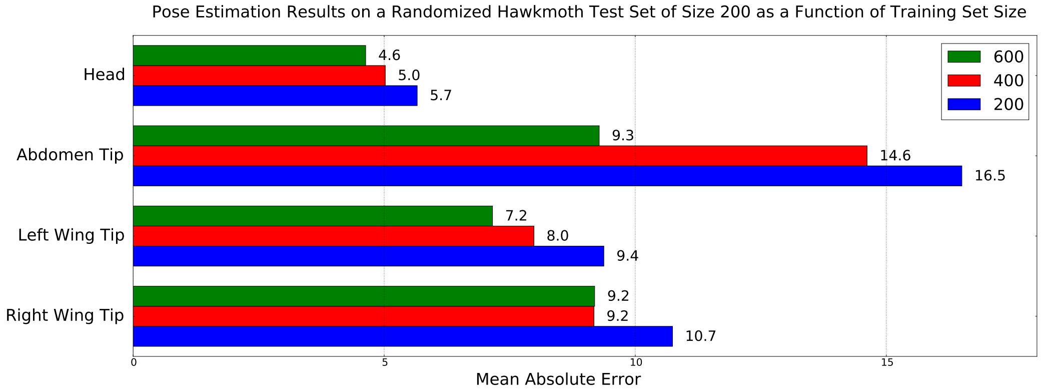


Figure 6·7: **Quantitative results demonstrating the influence of the training set size, prior to data augmentation, on test performance.** For each training set size, MAE is computed for each landmark type across a test set consisting of 200 randomly chosen images from an initial set of 800 frames. The training set sizes used in this experiment include 200, 400, and 600 images, which respectively represent 25%, 50%, and 75% of the dataset.

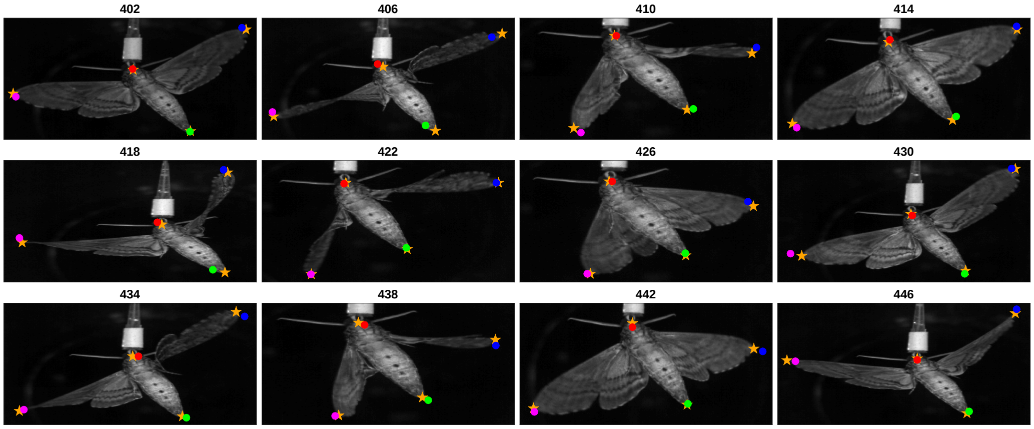


Figure 6·8: **Qualitative results of DNN$_2$ on the test set from camera 2.** Results are shown for 12 different frames with the predicted landmark localizations (colored circles) and the ground truth landmark localizations (gold stars).

would be the one which uses the lowest level features (VGG 2 + FC8), and the worst performing architecture would be the one which uses the highest level features (VGG 13 + FC8), and that the difference between the two would be large. The rationale for this hypothesis is that the VGG layers are pretrained on ImageNet which contains many different scenes and objects that at a high level have little in common with

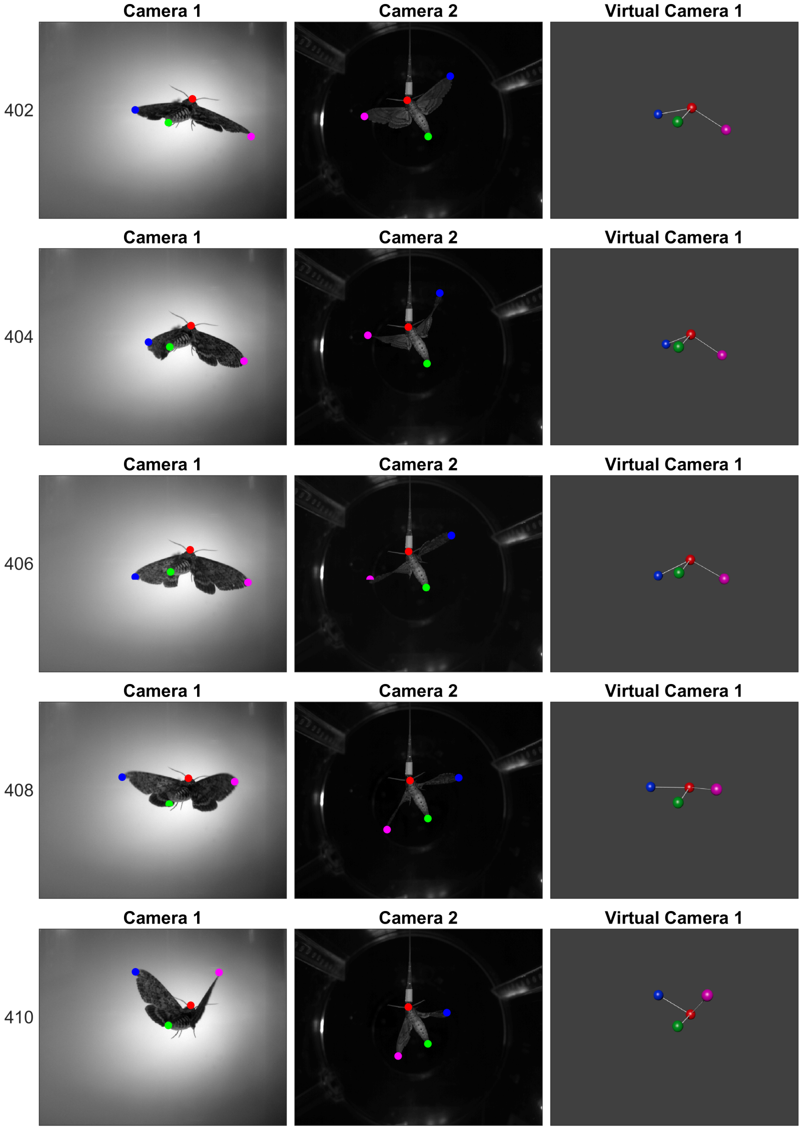


Figure 6·9: **Qualitative results of both DNN$_1$ and DNN$_2$ on landmark localization for frames simultaneously captured by both cameras 1 and 2.** The resulting landmark localizations are used to reconstruct the 3D positions of the landmarks. The right-most image in each subplot, labeled "Virtual Camera 1", illustrates the reconstructed 3D configuration (pose) of the hawkmoth. Results are shown for four frames with predicted landmark localizations denoted with colored circles and ground truth landmark localizations denoted with gold stars.

the hawkmoth dataset, and so only the lowest level features would be useful to our application. Our hypothesis was incorrect as the best performing architecture was

VGG 7 + FC8, which can be interpreted as an architecture which uses features that are not as low level as VGG 2 + FC8, nor as high level as VGG 13 + FC8. Consistent with our hypothesis the worst performing architecture was VGG 13 + FC8. Surprisingly, when comparing across learning rates the difference in test loss between the best performing architecture and the worst is only approximately 50.

One potential reason why the architecture may not make as large of a difference as hypothesized is that the features computed by the VGG X portion of the network are hierarchical in nature. This means that features output in one layer depend on the features output in earlier layers. If the low level features alone are sufficient for learning a linear regression model that accurately localizes landmarks then higher level features which depend on these low level features may preserve enough low level information to also be of use. In this case, the FC8 fully connected layer is able to learn a linear regression model that works across different feature spaces.

We also performed experiments to determine whether adding an additional fully connected layer to VGG 7 + FC8 would improve test performance by learning a non linear regression model. The result of these experiments was that the training loss converged to values that were too large to be of any use. Our results here also support the idea that training success is strongly dependent on the learning rate. For each network studied we found training loss diverged once the learning rate was set too large.

Network Initialization: Our experiments show that when training a network from scratch “xavier” initialization greatly outperforms Gaussian (0 mean, 0.01 standard deviation) initialization. Still, training a network from scratch results in inferior performance compared to using pretrained VGG X layers. These results support the hypothesis that a small training set, even augmented, is insufficient for learning quality feature maps. There maybe no choice for small datasets but to rely on pretrained networks which allow one to leverage feature maps learned on large datasets [69].

Number of Iterations: Our experiments show that performing 4000 iterations of training was enough to achieve a loss close to the local minimum, and that additional training did not significantly lower the loss. Relative to the default batch size of 32, 4000 iterations results in the network seeing 128,000 training samples during training. At 6250 iterations the network would perform 1 "epoch" meaning it would see the whole training set of 200,000 samples once.

Data Augmentation: Our experiments show that performing data augmentation with translation only outperforms combinations of translation with rotation and translation with scale. Our initial hypothesis was that by adding rotated and scaled versions of the hawkmoth to the training set, the network would be forced to learn a more general representation of hawkmoth landmarks, which would result in better generalization and performance on test images. In practice, our test set did not have significant in-plane rotations or scale changes compared to the original (unaugmented) training set, so we believe that augmentation with rotation and scale acted as a distraction to the network which ultimately hurt performance on the test set. We also want to emphasize that data augmentation is absolutely crucial to preventing our architectures from overfitting to the training set, as shown by the enormous loss that results from no data augmentation being performed.

Batch Size: Our experiments show that increasing the batch size results in a lower test loss, but has diminishing returns beyond a batch size of around 32. These results agree with the notion that a larger batch size allows for a more accurate approximation of the gradient of the loss function with respect to the weights and biases of the network, which allows SGD to descend in a direction closer to the optimal direction. However, the time it takes to train a network is linear in the batch size, so a doubling in the batch size will result in a doubling of the training time. To train a DNN with a batch size of 128 took more than 11 hours. We also performed an experiment with a batch size of 8 which resulted in a diverging loss during training.

Training and Test Split: Our experiments show that an alternative split, where the training and test sets interleave, results in a lower test loss. This supports the hypothesis that the more similar the training and testing distributions are, the closer the training loss will be to the test loss. In our application of landmark localization, given a fixed size training set, it is clearly favorable to choose the images in the training set so they overlap as much as possible with the testing data.

Another experiment we performed addresses the question of how increasing the size of the original training set, prior to any data augmentation, influences performance on a test set containing 200 randomly chosen images. Our hypothesis is that the more original training data available the better the test performance will be. The results mostly confirm this where the test loss decreases as the original training set size increases. An interesting question that arises is how much of a difference does it make when using a training set of 200 images or 25% of the original dataset, as compared to 600 or 75% of the original dataset? Our experiments reveal that for some landmarks the difference can be as little as approximately 1 pixel in terms of MAE, or as large as approximately 7 pixels in terms of MAE. The abdomen tip in our experiment is the landmark whose localization performance benefits the most from additional training data. This can be explained by the fact that the pose of the abdomen tip varies significantly across the video making it more difficult to learn a model as compared to other landmarks. Our results suggest that annotating as little as 25% of a dataset can be sufficient to localize some landmarks reasonably well, but other landmarks may have unacceptably high error. Increases in the training set size do help but must be weighed against the cost of producing more annotated training data.

Comparison with Previous Works: The focus of our experiments has largely been on studying the impact of parameter values on testing loss. Now we aim to put the results obtained with our default DNN (VGG 7 + FC8) into context. One way we contextualize our results is by comparing them to already published results on hawkmoth landmark localization [18]. The results presented in Figure 6·5a show the MAE that our default DNN achieves per landmark relative to two other approaches. Our default DNN outperformed the next best approach on localizing the head, left wing tip, and right wing tip. However, for the abdomen tip our DNN is outperformed by the approach of Breslav et al. [18]. As an overall measure of performance for an approach we compute the sum of MAEs across landmark types. This resulted in the method of Ortega-Jimenez et al. [60] obtaining a total MAE of 80.2, the approach of Breslav et al. [18] obtaining a total MAE of 36.5, and our default DNN obtaining a total MAE of 31.6. The conclusion here is that our proposed default DNN is the best performing among published works for this application.

The landmark that is easiest for our DNN to localize is the head, then followed by the left wing tip, abdomen tip, and the right wing tip. It is not surprising that the head is easiest to localize since its position and orientation relative to the camera are the least variable due to the hawkmoth being engaged in feeding during the video capture. Specifically the hawkmoth is feeding on artificial nectar contained in a plastic pipette tip that is fixed in position. The abdomen tip is expected to be difficult to localize partly because it does not have a distinct appearance compared to say the abdomen or other more textured regions. It is not clear why the right wing tip is more difficult to localize than the left wing tip, but overall the wing tips are also expected to be relatively difficult to localize due to the large changes in position they undergo during flight. It is also worth noting that the quantitative results discussed here do not include occlusion cases. However, we have labeled occluded landmarks in our training set and we have observed that our DNN is able to learn to predict the position of occluded landmarks.

Multi-view: Our multi-view experiments are included to emphasize one of the original motivations of our work: the quantitative study of 3D flight kinematics of a flying animal. In Figure 6·9 we demonstrate the result of two DNNs (DNN_1 and DNN_2) automatically annotating landmarks in their respective camera views, and how the 3D positions of the landmarks can subsequently be reconstructed. The ratios we have computed in the results section are very close to 1 and suggest that using the output of our DNNs to measure quantities in 3D will result in measurements that are very close to ground truth. Looking at Figure 6·9 we note that the virtual camera subplot shows the 3D flight of the hawkmoth. This use case is very exciting as our DNNs can enable scientists to obtain more accurate 3D positions of flying animal landmarks in less time, ultimately leading to analyses that better reflect the true kinematics of the animal.

Recommendations: Given our experimental results we make the following recommendations to our readers who would like to train DNNs for their own data and application.

1. Use as large of an initial training set as possible and definitely perform data

augmentation with translation alone to obtain a training set that is about 25 times larger than the original set. Consider other data augmentation types based on the invariances you want the network to learn.

2. As a starting point, use the architecture VGG 7 (pretrained) with a single fully connected layer attached. The number of neurons in the fully connected layer should be set to twice the number of landmarks that you wish to localize (a single neuron per dimension).

3. Choose the learning rate to be the largest value that still allows the training loss to converge.

4. Choose the number of training iterations by taking some of your training set and using it as a validation set (analogous to how our test set has been used here). Set the number of iterations to the value where validation set loss stops decreasing significantly. For a quick starting point try 10,000 iterations.

5. A batch size of 32 is a good starting point as it balances performance, accuracy of the gradient computation, and total training time.

6. Choose the training set so that it is as similar to the images that need to be annotated as possible.

6.4 Summary

In this chapter we have evaluated the use of DNNs for the problem of landmark localization in the challenging case where the amount of labeled training data is small. We have performed comprehensive experiments studying how numerous parameters influence the training of DNNs and the resulting test performance. Our experiments show that DNNs can be used for landmark localization and that they outperform other leading approaches for the problem of landmark localization in hawkmoth videos. We also show how DNNs used on multiple camera views can enable 3D pose estimation of flying animals.

6.5 Appendix A

Here we provide a more in depth explanation of the different types of layers used in the VGG 16 network, which are also common to many other CNNs.

Convolutional: A convolutional layer is a volume of neurons with the special property that individual neurons only connects to a subset of neurons in the previous layer, and neurons at the same depth in the volume share the same weights and biases. As an example, the first convolutional layer of VGG 16 has a height of 224 neurons, a width of 224 neurons, and a depth of 64. Each neuron in the first convolutional layer is connected with a 3 x 3 pixel region in each of the 3 channels (RGB) of the input image. As a result a single neuron in the first convolutional layer must learn 3 x 3 x 3 weights and 1 bias. Since all neurons at the same depth (depth slice) share the same weights and bias, then each depth slice requires learning only 27 weights and 1 bias, resulting in a total of 1728 (27 x 64) weights and 64 (1 x 64) biases for the whole layer. Conceptually, each depth slice is learning a filter that is sensitive to some pattern. Collectively this means the first convolutional layer learns 64 filters. The second convolutional layer in VGG 16 also has a height of 224, width of 224, and depth of 64, but now each neuron connects to a 3 x 3 region of each of the 64 depth slices in the previous layer, resulting in 3 x 3 x 64 weights and 1 bias per neuron. In Figure 6·3, the convolutional layers of VGG 16 are depicted as black rectangular volumes with a label ‘Conv,’ short for convolution, and a number indicating the depth of the volume. Notice that layers towards the end of the network (output side) have larger depths but smaller heights and widths.

Fully Connected: A fully connected layer is a layer where each neuron is connected with all outputs from the previous layer. In VGG 16 there are 3 fully connected layers, shown in blue in Figure 6·3. The first fully connected layer has 4,096 neurons each of which connect to all outputs from the previous layer. Since neurons in the fully connected layers connect to all outputs of the previous layer, instead of just a subset, fully connected layers account for a large percentage of the weights and biases present in a typical CNN.

Non Linearity: As previously mentioned, a neuron applies a non linear function to a weighted combination of inputs. The choice of function to apply is a design decision. Since each neuron applies a non linearity, it is natural to consider it as a parameter of the convolutional layers and also the fully connected layers. This is the case in Figure 6·3 where the non linearity is not explicitly shown. However, to make the choice of non linearity more explicit, it can be thought of as its own layer. From this point of view each convolutional and fully connected layer are immediately followed by a non linearity layer. The output dimension of the non linearity layer will be the same as its input dimension. One popular choice for non linearity is the rectified linear unit (ReLU), defined as: $f(x) = max(0, x)$. The ReLU is used in VGG 16 and was shown to be advantageous for training deep networks [36].

Pooling: Pooling layers apply a pooling function to reduce the dimensionality of the input. A common pooling function is max pooling where the output of a node is the maximum over a region from the input. In VGG 16, max pooling is performed over a 2 x 2 window, with a “stride” of 2 which means that an input volume of size 224 x 224 x 64 will be transformed into an output volume of size 112 x 112 x 64. In Figure 6·3 all max pooling layers are denoted in red.

It is worth noting that the VGG 16 network shown in Figure 6·3 also has a final layer in orange named ‘Soft Max.’ This soft max layer facilitates the computation of a loss function by remapping the outputs of the last fully convolutional layer to class probabilities between 0 and 1. Figure 6·3 illustrates not only the architecture of the VGG 16 network but also the action of the network on an input image of a person. The network output is a list of probabilities, one for each class, and as desired the class “person” has the highest probability.

6.6 Appendix B

Here we describe how to accomplish two tasks using the deep learning library Caffe [44]. The first task is to train or finetune a DNN using some training data. The second task is to run a trained DNN on some testing data to get output. To facilitate these two tasks we provide example code that can freely be used[7].

6.6.1 Training Set Preparation

Before training or finetuning a DNN one needs to prepare a training set and ensure it is formatted properly for use with Caffe. A training set should consist of training samples and their labels. Since the focus of this work is on image annotation, we will think of training samples as individual images. The label associated with an image depends on the task that we are training the DNN to perform. In the case of classification where there are N classes, the label should be a single integer in the range of 0 and $N-1$ inclusive. For a multi-label classification problem where K labels are predicted per training sample, the overall label will be a vector of dimensionality K, with each dimension containing an integer indicating the true class for that label. In the case of regression problems the label is continuous valued and can either be a single value, or a vector of values for the case of multiple regression. Example: The

[7] `http://www.cs.bu.edu/~betke/research/ALADNN/`

training set used in our experiments consists of images of a moth where the label for each image is an 8 dimensional continuous valued vector containing the x and y coordinates of four landmarks.

Preprocessing and Data Format

When training or finetuning a DNN, the training images need to conform to the architecture of the DNN. The first layer of the DNN will determine what the input to the network should be. To obtain training images that are suitable for input it is common to perform one or more preprocessing steps. In the case of the VGG 16 network, all input images needs to have 3 channels with a width and height of 224 pixels. Images that have a different width and height should be resized to 224×224. In the case that the images is a single channel or grayscale image, one can create a 3 channel images by replicating the original image in each channel. If finetuning a pre-trained network, additional preprocessing steps may need to be performed. For example, if the original network was trained on 8 bit images, one should ensure their images are also 8 bit. Additionally, it is common to normalize the input image values by subtracting a mean value. To finetune VGG 16, the authors provide per channel (blue, green, red) constants which are to be subtracted from each corresponding channel in the input.

For Caffe to be able to use one's training data, the data should be stored in two 4D arrays, one containing the training images, and the other containing the labels. These two 4D arrays are then written to disk as an HDF5 file. The 4D array storing images should have dimensions $N \times C \times H \times W$, where N is the number of training images, C is the number of channels per image, H is the height of the image, and W is the width of the image. The same format is used for the labels, but C and H

will be 1, with the last dimension W containing the label. Example: In our work the data 4D array has dimensions $1000 \times 3 \times 224 \times 224$, and the label has dimensions $1000 \times 1 \times 1 \times 8$. Note that here N is 1000, but the total number of training samples we use is much larger. The reason we do not put all training samples into a single 4D array is because that would create a single file that is enormous. Instead the training set is broken into smaller batches, and each batch is represented by a pair of 4D arrays that are written to a single HDF5 file. Thus in our example, our 100,000 training samples are broken into 100 batches each containing a pair of 4D arrays which store $N = 1000$ training images and their labels. Since each batch is written to a separate HDF5 file, we obtain 100 HDF5 files.

The data format for the 4D arrays previously described is consistent with how Caffe thinks of data. However, when writing to HDF5 files the 4D arrays need to be modified so that the order of their dimensions is reversed, e.g $W \times H \times C \times N$. After the 4D data and label arrays are in this format they can be written to disk using an appropriate library that supports writing HDF5 files. In our work we use the built in Matlab functions h5create and h5write for this, however libraries for Python and other languages should also be readily available. HDF5 files can have different fields in the same file, so in our case a single HDF5 file will have a /data field storing the training image 4D array, and the /label field storing the label 4D array. For specifics refer to our example code moth_dataset_augmentation.m in the supplemental materials. Note that the reversal of dimensions of the 4D arrays is an extra step that strictly speaking is not required but it makes it easier to differentiate the format used by Caffe with the one used by HDF5.

Lastly, it is worth remembering that data augmentation can be beneficial for training a DNN. Any data augmentation that a user wishes to perform on an original data set should be done *prior* to the preprocessing and data format steps above. Note that with relatively large datasets the storage for HDF5 files can be large. For example 100 HDF5 files, each containing 1000, $224 \times 224 \times 3$ images takes up about 100 GB.

6.6.2 Training a DNN with Caffe

Training a DNN with Caffe requires the creation of 3 text files which collectively specify the input data, the network architecture, and the training and optional testing parameters.

The main text file called train_val.prototxt contains a specification of the DNN architecture, layer by layer, starting from the input data and ending at the loss function. Each layer is denoted in the text file with the word layer followed by an open and close parentheses. Within the parentheses is a specification of the type of layer and any pertinent parameter values. The first layer in this file will be a data layer, and in the case of HDF5 files, it will have a type of "HDF5Data". In this case the layer will need to reference a text file that contains the file paths of all the HDF5 files used for training. Layers with learnable parameters, like the convolutional and fully connected layers, will have text specifying the learning rate and initial values for the weights and biases in that layer. The second text file called deploy.prototxt is the same as train_val.prototxt but with the data and loss layers removed. See our example code in the supplemental material.

So far we have made the assumption that "training" means training all of the layers of a DNN from scratch, where all weights and biases are initialized by the user. There are two variations to this kind of training which are also useful in practice, and require some modifications to the train_val and deploy text files. The first variation is when a user only wants to train *some* of the layers in the network. This can be accomplished by setting the learning rate of all other layers to zero. A second variation occurs when a user wants to initialize the weights and biases of some layers of their network to the weights and biases of an already trained network. This commonly occurs when someone wants to take a piece of a network that has already been trained and add their own layers on top of it. To initialize the weights and biases of some layers in the network to values learned from another network, one should not include any weight or bias initialization in the train_val or deploy text files. Additionally, the name of the layers, which are to be initialized in this way, should match those used in the already trained network. Lastly, the weights and biases of the already trained network will be stored in a .caffemodel file which will be specified as an additional argument during training. See supplemental materials for an example.

The third text file called solver.prototxt, specifies training parameters, and optional parameters if one wishes to evaluate the currently trained model on a test set, as the training process goes on. The solver text file will reference the train_val.prototxt file, so that the solver knows what data and architecture to use for training. One of the most important parameters in the solver file is the base learning rate. The base learning rate is multiplied by the learning rate values (multipliers) specified in the train_val.prototxt to obtain the final learning rate used for any layer. For example if the base learning rate in the solver is 10^{-5} and the learning rate specified in a certain fully connected layer is 100 then the learning rate used for that fully connected layer will be $100 \times 10^{-5} = 10^{-3}$. Other parameters in the solver file determine how the base learning rate is changed (its "schedule") as training proceeds. The total number of training iterations to perform is specified by the parameter max_iter. As training occurs it is wise to have Caffe save the weights and biases of the network to disk in the form of a .caffemodel. The parameter snapshot specifies that every snapshot iterations a .caffemodel will be written to disk. If all goes well the very last .caffemodel written is the one that contains the weights and biases learned after max_iter iterations of training. See supplemental materials for an example.

After these 3 text files are created and specified, Caffe can be run from the command line. The following is an example call: `caffe train -gpu 0 -solver solver.prototxt`. If you want to initialize some of the weights and biases in the network to those that were learned for another network you would use the following command line command: `caffe train -gpu 0 -solver solver.prototxt -weights name.caffemodel`. Additional details on layer types, solver parameters, training and testing usage from the command line, are available on the Caffe website[8].

6.6.3 Testing a DNN with Caffe

The Caffe library provides interfaces for Matlab (MatCaffe) and Python (PyCaffe), and either can be used to run a trained DNN on test data. In our work we use the MatCaffe interface and create a simple Matlab script to run our trained DNNs on test data. The script creates a caffe network object which is initialized by specifying a deploy.prototxt file and the .caffemodel containing the weights and biases learned during training. This object, which we call net, has the forward function called which takes in a cell array of size 1, containing a 3D array of dimension $W \times H \times C$, which contains the test data. The output of the function is a cell array of size 1 containing the output of the network, which is just the input forward propagated through the

[8] http://caffe.berkeleyvision.org/tutorial/

network. The output can be used for evaluating performance and visualizing the result of the network. Note that any preprocessing steps performed on the training data, should also be performed on the test data. Typically, testing is fast since a single input only requires a single forward pass over the network to obtain an output. See supplemental materials for an example of testing.

Chapter 7

Conclusion

Flying animals have been, and will continue to be, actively studied by researchers working to better understand these animals' behaviors and flight characteristics. As more scientists continue to leverage multi-view camera systems to record video data of these flying animals, the opportunity for computer vision based algorithms to make an impact grows. Algorithms that can automatically and accurately estimate the 3D pose of flying animals can save scientists lots of time and manual labor, affording them more time to spend on analysis. The focus of this thesis is on the problem of 3D pose estimation of flying animals in multi-view video datasets, a problem that has received little attention in the computer vision community where few flying animal datasets exist. Additionally, published approaches from researchers in the natural sciences have yet to take full advantage of advancements in computer vision research, resulting in approaches that are not broadly applicable. Our thesis represents an effort to revitalize the bridge between the computer vision community and the natural science community where these algorithms are needed. The remainder of this chapter summarizes our key contributions, limitations of our work, and potentially interesting future research directions.

7.1 Main Contributions

Our thesis proposes three different novel approaches for estimating the 3D pose of flying animals in multi-view video datasets. The three proposed approaches are inspired by three successful pose estimation paradigms developed in computer vision, as well as our formative experience working with low-resolution video of bats from which we estimate wing beat frequencies of individual bats [17].

The first contribution of this thesis is our work on estimating wing beat frequencies of individual bats [17], which to the best of our knowledge is the first work to estimate the wing beat frequency of bats in the wild using shape-based computer-vision methods on thermal infrared video data. Our method obtained wing beat frequency estimates that agreed with ranges that were previously reported in the biology literature [31]. In the context of our thesis, this work represents our first experience trying to leverage shape information in flying animal data, leading to us thinking about how shape could be used from multiple views to estimate 3D pose.

Our second contribution is the design and creation of our system named 3D-PEB [16], which can automatically estimate the articulated 3D pose of bats flying in the wild given low resolution multi-view video data. Our method has resulted in the first published quantitative and qualitative results in the literature on automatically generated 3D pose estimates of bats in the wild. In the context of the challenges faced due to the low resolution nature of the data, as well as the novelty of our results, we find our automated estimates more than suitable for describing coarse 3D pose of flying bats in the wild. Our proposed approach approach leverages the use of a 3D graphics models of a *Tadarida brasiliensis* bat which we have built. We have used a Markov Random Field model to frame 3D pose estimation over time as one large discrete optimization problem. To encourage more computer vision researchers to branch out to less established applications we have made both our 3D graphics model and some multi-view low resolution bat data freely available[1].

Our third contribution is a novel way to increase pose estimation accuracy by using

[1] `http://www.cs.bu.edu/~betke/research/3dpeb/`

automatically discovered auxiliary parts to generate better appearance likelihoods which can then be fed into traditional part-based models like the MPS model [18]. Our experiments on hawkmoth data show that our proposed approach significantly improves over existing work [60], while also being more generally applicable. We once again support computer vision research on the domain of flying animals by making freely available a unique hawkmoth dataset that comes with landmark annotations and segmentations[2].

The last contribution of this thesis is our work evaluating the use of DNNs for the problem of landmark localization in the challenging case where the amount of labeled training data is small. We present comprehensive experiments studying how numerous parameters influence the training of DNNs and the resulting test performance. Our experiments show that DNNs can be used for landmark localization and they outperform other leading approaches for the problem of landmark localization in hawkmoth videos. We demonstrate how DNNs used on multiple camera views can enable 3D pose estimation of flying animals. To facilitate the use of DNNs by scientists from many different fields we provide a self contained explanation of what DNNs are, how they work, and how to apply them to other datasets using the freely available library Caffe, and additional code[3] that we provide. Furthermore, we make freely available an additional high resolution hawkmoth video along with landmark annotations and segmentations to provide the first published high resolution multi-view dataset of a flying animal, which we call HRMF2.0.

7.2 Limitations and Interesting Future Research Directions

The proposed approaches in this thesis motivate several interesting directions for future research. Our work on improving part-based models by discovering auxiliary

[2] http://www.cs.bu.edu/~betke/research/HRMF/

[3] http://www.cs.bu.edu/~betke/research/HRMF2/

parts from unannotated regions of training images did not take advantage of DNNs anywhere in the pipeline. An interesting research direction is to examine where in the standard part discovery pipeline DNNs can be of use. Additionally, we think strategies that try to explicitly make use of all the available image evidence for landmark localization are promising. The recent work of Lifshitz et al. [54] supports this idea by learning DNNs that learn to predict landmark locations for all patches in an image, while taking an additional step to form a consensus on landmark locations.

Our study of the use of DNNs for end to end landmark localization in the domain of flying animals can be seen as an important stepping stone for further study of the application of DNNs to 3D pose estimation of flying animals. One interesting question is whether with DNNs it is beneficial to model the individual appearance of a landmark with a single DNN and more spatial dependencies of landmarks with an additional DNN? In these cases what kinds of data augmentation strategies might be helpful for learning? In the case of multiple views, it would be interesting to see whether a DNN trained on multiple-views would perform better at landmark localization when compared to DNNs trained for individual views.

One underlying challenge in the application of DNNs to datasets commonly used in the study of animal flight is their relatively small size. In our work we address this by taking advantage of pre-trained DNNs and leveraging simple data augmentation. More research to yield best practices of using DNNs in such regimes would be of great practical value so that questions such as “At what size of data should researchers and practitioners look for alternatives to DNNs?” can be answered.

References

[1] Agarwal, A., and Triggs, B. Recovering 3D human pose from monocular images. *IEEE Transactions on Pattern Analysis and Machine Intelligence 28*, 1 (2006), 44–58.

[2] Amin, S., Andriluka, M., Rohrbach, M., and Schiele, B. Multi-view pictorial structures for 3D human pose estimation. In *British Machine Vision Conference* (2013), vol. 2, p. 12 pp.

[3] Andriluka, M., Roth, S., and Schiele, B. Discriminative appearance models for pictorial structures. *International Journal of Computer Vision 99*, 3 (2012), 259–280.

[4] Athitsos, V., and Sclaroff, S. 3D hand pose estimation by finding appearance-based matches in a large database of training views. In *IEEE Workshop Cues in Communication* (2001), vol. 9, pp. 100–106.

[5] Belongie, S., Malik, J., and Puzicha, J. Shape matching and object recognition using shape contexts. *IEEE Transactions on Pattern Analysis and Machine Intelligence 24*, 4 (2002), 509–522.

[6] Bengio, Y. Learning deep architectures for AI. *Foundations and Trends in Machine Learning 2*, 1 (2009), 1–127.

[7] Bengio, Y. Practical recommendations for gradient-based training of deep architectures. In *Neural Networks: Tricks of the Trade*. Springer, 2012, pp. 437–478.

[8] Bergou, A. J., Swartz, S., Breuer, K., and Taubin, G. 3D reconstruction of bat flight kinematics from sparse multiple views. In *International Conference on Computer Vison Workshop on Dynamic Shape Capture and Analysis* (2011), pp. 1618–1625.

[9] Betke, M., Hirsh, D. E., Makris, N. C., McCracken, G. F., Procopio, M., Hristov, N. I., Tang, S., Bagchi, A., Reichard, J. D., Horn, J. W., Crampton, S., Cleveland, C. J., and Kunz, T. H. Thermal imaging reveals significantly smaller Brazilian free-tailed bat colonies than previously estimated. *Journal of Mammalogy 89*, 1 (Feb. 2008), 18–24.

[10] Birch, J. M. Comparing wing shape of bats: The merits of principal-components analysis and relative-warp analysis. *Journal of Mammalogy 78*, 4 (1997), pp. 1187–1198.

[11] Bishop, C. M. *Pattern Recognition and Machine Learning*. Springer-Verlag New York, 2006.

[12] Blender. `http://www.blender.org`.

[13] Bourdev, L., Maji, S., Brox, T., and Malik, J. Detecting people using mutually consistent poselet activations. In *European Conference on Computer Vision*. Springer, 2010, pp. 168–181.

[14] Bourdev, L., and Malik, J. Poselets: Body part detectors trained using 3D human pose annotations. In *International Conference on Computer Vision* (2009), IEEE, pp. 1365–1372.

[15] Boykov, Y., Veksler, O., and Zabih, R. Markov random fields with efficient approximations. In *Computer Vision and Pattern Recognition* (1998), pp. 648–655.

[16] Breslav, M., Fuller, N., Sclaroff, S., and Betke, M. 3D pose estimation of bats in the wild. In *Winter Conference on Applications of Computer Vision* (2014), pp. 91–98.

[17] Breslav, M., Fuller, N. W., and Betke, M. Vision system for wing beat analysis of bats in the wild. In *Workshop on Visual Observation and Analysis of Animals and Insect behavior held in conjunction with International Conference of Pattern Recognition* (Tsukuba, Japan, 2012), p. 4 pp.

[18] Breslav, M., Hedrick, T., Sclaroff, S., and Betke, M. Discovering useful parts for pose estimation in sparsely annotated datasets. In *Winter Conference on Applications of Computer Vision* (2016), IEEE, p. 9 pp.

[19] Briassouli, A., and Ahuja, N. Extraction and analysis of multiple periodic motions in video sequences. *IEEE Transactions on Pattern Analysis and Machine Intelligence 29*, 7 (July 2007), 1244–1261.

[20] Burenius, M., Sullivan, J., and Carlsson, S. 3D pictorial structures for multiple view articulated pose estimation. In *Computer Vision and Pattern Recognition* (2013), IEEE, pp. 3618–3625.

[21] Cutler, R., and Davis, L. View-based detection and analysis of periodic motion. In *International Conference on Patern Recognition* (1998), p. 4 pp.

[22] Dalal, N., and Triggs, B. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition* (2005), pp. 886–893.

[23] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition* (2009), pp. 248–255.

[24] Doersch, C., Gupta, A., and Efros, A. A. Mid-level visual element discovery as discriminative mode seeking. In *Neural Information Processing Systems* (2013), pp. 494–502.

[25] Erol, A., Bebis, G., Nicolescu, M., Boyle, R. D., and Twombly, X. Vision-based hand pose estimation: A review. *Computer Vision and Image Understanding 108* (2007), 52–73.

[26] Felzenszwalb, P., and Huttenlocher, D. Distance transforms of sampled functions. Tech. rep., Cornell University, 2004.

[27] Felzenszwalb, P. F., Girshick, R. B., McAllester, D., and Ramanan, D. Object detection with discriminatively trained part-based models. *IEEE Transactions on Pattern Analysis and Machine Intelligence 32*, 9 (2010), 1627–1645.

[28] Felzenszwalb, P. F., and Huttenlocher, D. P. Pictorial structures for object recognition. *International Journal of Computer Vision 61*, 1 (2005), 55–79.

[29] Fischler, M. A., and Elschlager, R. The representation and matching of pictorial structures. *IEEE Transactions on Computers C-22*, 1 (Jan 1973), 67–92.

[30] Fisher, R., and Xiao, Y. Bat echolocation behavior from highspeed 3D video. In *Workshop on Visual Observation and Analysis of Animals and Insect behavior held in conjunction with International Conference of Pattern Recognition* (2010), p. 4 pp.

[31] Foehring, R. C., and Hermanson, J. W. Morphology and histochemistry of flight muscles in free-tailed bats, Tadarida brasiliensis. *Journal of Mammalogy 65*, 3 (1984), pp. 388–394.

[32] Fontaine, E. I., Zabala, F., Dickinson, M. H., and Burdick, J. W. Wing and body motion during flight initiation in Drosophila revealed by automated visual tracking. *Journal of Experimental Biology 212* (2009), 1307–1323.

[33] Frey, B. J., and Dueck, D. Clustering by passing messages between data points. *Science 315* (2007), 972–976.

[34] Gkioxari, G., Hariharan, B., Girshick, R., and Malik, J. Using k-poselets for detecting people and localizing their keypoints. In *Computer Vision and Pattern Recognition* (June 2014), pp. 3582–3589.

[35] Glorot, X., and Bengio, Y. Understanding the difficulty of training deep feedforward neural networks. In *International Conference on Artificial Intelligence and Statistics* (2010), vol. 9, pp. 249–256.

[36] Glorot, X., Bordes, A., and Bengio, Y. Deep sparse rectifier neural networks. In *International Conference on Artificial Intelligence and Statistics* (2011), pp. 315–323.

[37] Goodfellow, I., Bengio, Y., and Courville, A. Deep learning. Book in preparation for MIT Press, 2016.

[38] Hariharan, B., Malik, J., and Ramanan, D. Discriminative decorrelation for clustering and classification. In *European Conference on Computer Vision*. Springer, 2012, pp. 459–472.

[39] Hartley, R. I., and Zisserman, A. *Multiple View Geometry in Computer Vision*, second ed. Cambridge University Press, ISBN: 0521540518, 2004.

[40] Hedrick, T. L. Software techniques for two- and three-dimensional kinematic measurements of biological and biomimetic systems. *Bioinspiration & Biomimetics 3*, 3 (2008), 6 pp.

[41] Hernandez-Vela, A., Escalera, S., and Sclaroff, S. Contextual rescoring for human pose estimation. *British Machine Vision Conference* (2014), 11 pp.

[42] Hofmann, M., and Gavrila, D. Multi-view 3D human pose estimation in complex environment. *International Journal of Computer Vision 96*, 1 (2012), 103–124.

[43] Hubel, T. Y., Hristov, N. I., Swartz, S. M., and Breuer, K. S. Changes in kinematics and aerodynamics over a range of speeds in Tadarida brasiliensis, the Brazilian free-tailed bat. *Journal of The Royal Society Interface* (2012), 1120–1130.

[44] Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., and Darrell, T. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093* (2014), 4 pp.

[45] Johnson, S., and Everingham, M. Clustered pose and nonlinear appearance models for human pose estimation. *British Machine Vision Conference 2*, 4 (2010), 11 pp.

[46] Juneja, M., Vedaldi, A., Jawahar, C., and Zisserman, A. Blocks that shout: Distinctive parts for scene classification. In *Computer Vision and Pattern Recognition* (2013), IEEE, pp. 923–930.

[47] Kazemi, V., Burenius, M., Azizpour, H., and Sullivan, J. Multi-view body part recognition with random forests. *British Machine Vision Conference* (2013), 11 pp.

[48] Kong, Z., Ozcimder, K., Fuller, N., Greco, A., Theriault, D., Wu, Z., Kunz, T., Betke, M., and Baillieul, J. Optical flow sensing and the inverse perception problem for flying bats. In *Conference on Decision and Control* (Dec 2013), pp. 1608–1615.

[49] Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *Neural Information Processing Systems*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105.

[50] Kuffner, J. J. Effective sampling and distance metrics for 3D rigid body path planning. In *International Conference on Robotics and Automation* (2004), pp. 3993–3998.

[51] Kuhn, H. W. The Hungarian method for the assignment problem. *Naval Research Logistics Quarterly 2*, 1–2 (1955), 83–97.

[52] LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE 86*, 11 (November 1998), 2278–2324.

[53] LeCun, Y. A., Bottou, L., Orr, G. B., and Müller, K.-R. Efficient backprop. In *Neural Networks: Tricks of the trade*. Springer Berlin Heidelberg, 2012, pp. 9–48.

[54] Lifshitz, I., Fetaya, E., and Ullman, S. Human pose estimation using deep consensus voting. *Computing Research Repository abs/1603.08212* (2016), 16 pp.

[55] Liu, C., Yuen, J., and Torralba, A. Sift flow: Dense correspondence across scenes and its applications. *IEEE Transactions on Pattern Analysis and Machine Intelligence 33*, 5 (May 2011), 978–994.

[56] Moeslund, T. B., and Granum, E. A survey of computer vision-based human motion capture. *Computer Vision and Image Understanding 81*, 3 (2001), 231 – 268.

[57] Moeslund, T. B., Hilton, A., and Krger, V. A survey of advances in vision-based human motion capture and analysis. *Computer Vision and Image Understanding 104*, 23 (2006), 90 – 126.

[58] Murphy-Chutorian, E., and Trivedi, M. Head pose estimation in computer vision: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence 31* (2009), 607–626.

[59] Ortega-Jimenez, V. M., Greeter, J. S. M., Mittal, R., and Hedrick, T. L. Hawkmoth flight stability in turbulent vortex streets. *Journal of Experimental Biology 216*, 24 (2013), 4567–4579.

[60] Ortega-Jimenez, V. M., Mittal, R., and Hedrick, T. L. Hawkmoth flight performance in tornado-like whirlwind vortices. *Bioinspiration & Biomimetics 9*, 2 (2014), 025003, 11 pp.

[61] Ott, P., and Everingham, M. Shared parts for deformable part-based models. In *Computer Vision and Pattern Recognition* (June 2011), pp. 1513–1520.

[62] Pishchulin, L., Andriluka, M., Gehler, P., and Schiele, B. Poselet conditioned pictorial structures. In *Computer Vision and Pattern Recognition* (2013), IEEE, pp. 588–595.

[63] Reichard, J. D., and Fellows, S. R. Thermoregulation during flight: Body temperature and sensible heat transfer in free-ranging Brazilian free-tailed bats (*Tadarida brasiliensis*). *Physiological and Biochemical Zoology 83*, 6 (2010), pp. 885–897.

[64] Riskin, D. K., Iriarte-Daz, J., Middleton, K. M., Breuer, K. S., and Swartz, S. M. The effect of body size on the wing movements of pteropodid bats, with insights into thrust and lift production. *Journal of Experimental Biology 213*, 23 (2010), 4110–4122.

[65] Ristroph, L., Berman, G. J., Bergou, A. J., Wang, Z. J., and Cohen, I. Automated hull reconstruction motion tracking (hrmt) applied to sideways maneuvers of free-flying insects. *Journal of Experimental Biology 212*, 9 (2009), 1324–1335.

[66] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. Learning representations by back-propagating errors. *Cognitive Modeling 5*, 3 (1988), 4 pp.

[67] Schmidhuber, J. Deep learning in neural networks: An overview. *Neural Networks 61* (2015), 85–117.

[68] Schmidt, M. Ugm: Matlab code for undirected graphical models. `http://www.di.ens.fr/~mschmidt/Software/UGM.html`, 2013.

[69] Sharif Razavian, A., Azizpour, H., Sullivan, J., and Carlsson, S. Cnn features off-the-shelf: an astounding baseline for recognition. In *Computer Vision and Pattern Recognition Workshop* (2014), pp. 806–813.

[70] Shelton, R. M., Jackson, B. E., and Hedrick, T. L. The mechanics and behavior of cliff swallows during tandem flights. *Journal of Experimental Biology* (2014), 2717–2725.

[71] Shotton, J., Sharp, T., Kipman, A., Fitzgibbon, A., Finocchio, M., Blake, A., Cook, M., and Moore, R. Real-time human pose recognition in parts from single depth images. *Communications of the ACM 56*, 1 (Jan. 2013), 116–124.

[72] Simard, P. Y., Steinkraus, D., and Platt, J. C. Best practices for convolutional neural networks applied to visual document analysis. In *International Conference on Document Analysis and Recognition* (2003), IEEE, p. 6 pp.

[73] Simonyan, K., and Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556 abs/1409.1556* (2014), 14 pp.

[74] Singh, S., Gupta, A., and Efros, A. A. Unsupervised discovery of mid-level discriminative patches. In *European Conference on Computer Vision.* Springer, 2012, pp. 73–86.

[75] Sun, M., and Savarese, S. Articulated part-based model for joint object detection and pose estimation. In *International Conference on Computer Vision* (2011), IEEE, pp. 723–730.

[76] Theriault, D. H., Fuller, N. W., Jackson, B. E., Bluhm, E., Evangelista, D., Wu, Z., Betke, M., and Hedrick, T. L. A protocol and calibration method for accurate multi-camera field videography. *Journal of Experimental Biology 217*, 11 (2014), 1843–1848.

[77] Theriault, D. H., Wu, Z., Hristov, N. I., Swartz, S. M., Breuer, K. S., Kunz, T. H., and Betke, M. Reconstruction and analysis of 3D trajectories of Brazilian free-tailed bats in flight. Tech. Rep. BUCS-2010-027, Boston University, 2010.

[78] Tobalske, B. W., Warrick, D. R., Clark, C. J., Powers, D. R., Hedrick, T. L., Hyder, G. A., and Biewener, A. A. Three-dimensional kinematics of hummingbird flight. *Journal of Experimental Biology 210*, 13 (2007), 2368–2382.

[79] Toshev, A., and Szegedy, C. Deeppose: Human pose estimation via deep neural networks. In *Computer Vision and Pattern Recognition* (2014), pp. 1653–1660.

[80] Wang, Y., Tran, D., and Liao, Z. Learning hierarchical poselets for human parsing. In *Computer Vision and Pattern Recognition* (2011), IEEE, pp. 1705–1712.

[81] Watts, P., Mitchell, E. J., and Swartz, S. M. A computational model for estimating the mechanics of horizontal flapping flight in bats: Model description and validation. *Journal of Experimental Biology 204*, 16 (2001), 2873–2898.

[82] Weiss, Y. *Comparing the mean field method and belief propagation for approximate inference in MRFs.* The MIT Press, 2001.

[83] Wu, Z., Hristov, N. I., Hedrick, T. L., Kunz, T. H., and Betke, M. Tracking a large number of objects from multiple views. In *International Conference on Computer Vision* (2009). 8 pp.

[84] Yang, Y., and Ramanan, D. Articulated pose estimation with flexible mixtures-of-parts. In *Computer Vision and Pattern Recognition* (2011), IEEE, pp. 1385–1392.

[85] Zhang, Z., Luo, P., Loy, C. C., and Tang, X. Facial landmark detection by deep multi-task learning. In *European Conference on Computer Vision.* Springer, 2014, pp. 94–108.

[86] Zhu, X., and Ramanan, D. Face detection, pose estimation, and landmark localization in the wild. In *Computer Vision and Pattern Recognition* (2012), IEEE, pp. 2879–2886.

CURRICULUM VITAE

Mikhail Breslav

Address	MCS 211, 111 Cummington Mall Department of Computer Science, Boston University Boston, MA 02215
Email	breslav@bu.edu
Website	http://www.breslav.org/

Education

Boston University, Department of Computer Science
Ph.D. Candidate. Fall 2010 - Summer 2016
Thesis topic: 3D Pose Estimation of Flying Animals.
Advisor: Dr. Margrit Betke (betke@cs.bu.edu)
Image and Video Computing (IVC) Group

The Pennsylvania State University - University Park
Master of Science in Electrical Engineering, 2010
Thesis: 3D Reconstruction of 2D Endobronchial Ultrasound.
Advisor: Dr. William Higgins (weh2@psu.edu)
Multidimensional Image Processing Lab (MIPL)

The Pennsylvania State University - University Park
Bachelor of Science in Electrical Engineering, 2008

Professional Experience

Nokia HERE, Berkeley, CA.
Intern, May – August 2014

MIT Lincoln Laboratory, Lexington, MA.
Intern, June – August 2010

Lockheed Martin, Owego, NY.
Intern, June – August 2008

Publications

- Breslav, M., Hedrick, T., Sclaroff, S., and Betke, M. "Discovering useful parts for pose estimation in sparsely annotated datasets." In *Winter Conference on Applications of Computer Vision* (2016), IEEE, 9 pp.

- Breslav, M., Fuller, N., Sclaroff, S., and Betke, M. "3D pose estimation of bats in the wild." In *Winter Conference on Applications of Computer Vision* (2014), pp. 91–98.

- Zang, X., Breslav, M., and Higgins, W. E., "3D segmentation and reconstruction of endobronchial ultrasound," in *SPIE Medical Imaging 2013: Ultrasound Imaging, Tomography, and Therapy*, (2013), 867505, 15 pp.

- Breslav, M., Fuller, N. W., and Betke, M. "Vision system for wing beat analysis of bats in the wild." In *Workshop on Visual Observation and Analysis of Animals and Insect behavior held in conjunction with International Conference of Pattern Recognition* (Tsukuba, Japan, 2012), 4 pp.

- Breslav, M., "3D Reconstruction of 2D Endobronchial Ultrasound," Masters thesis, The Pennsylvania State University (2010).