UNIVERSITY OF CALIFORNIA, SAN DIEGO

Tracking Multiple Mice through Severe Occlusions

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy

in

Computer Science

by

Kristin Branson

2007

To my family, for always supporting me.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

First, I would like to acknowledge my adviser, Serge Belongie. Serge's passion for computer vision and the real-world problems it can solve was truly inspirational to me, and has very much shaped my professional goals in life. Many researchers in computer vision begin with a fancy algorithm, then create a problem for it to solve. Serge, on the other hand, seeks out interesting, novel, and usually difficult problems which computer vision can benefit. I am also grateful to Serge for giving me the freedom to investigate whatever I felt like. As someone who has had three advisers in graduate school, my research interests have meandered over many topics in machine learning, computer vision, even classical artificial intelligence. The trust Serge has had in me throughout my stay at UCSD has given me the confidence to have ambitious goals.

I would also like to acknowledge my other advisers at UCSD, Sanjoy Dasgupta and Gary Cottrell. Sanjoy is unlike any other person I have met. When I would get very stuck on a problem, I would meet with Sanjoy. In 15 minutes, I could explain *any* problem to Sanjoy using my awful, complex notation. He could understand the problem in depth immediately, and provide insightful suggestions. Gary introduced me to the field of machine learning. As my adviser during my first couple years of graduate school, he taught me how to do research.

I would also like to acknowledge my pseudo-advisers, Sameer Agarwal and Eric Wiewiora. Sameer has always been the first person I go to with any question in any field. He is interested in *every* problem, and is always willing to devote his full attention to any crazy problem or algorithm I propose. Sameer is the first person I ever met who is truly passionate about mathematics and statistics. He has taught me to see the beauty in an algorithm. Eric, besides instructing me in the way of arts and crafts, has also always been willing to listen to my crazy ideas. Faults in my ideas found by Eric have saved me countless wasted hours, and approval of an idea by Eric gave me great confidence.

I would like to acknowledge the younger folk with whom I have discussed my research, Vincent Rabaud, Daniel Hsu, Lawrence Cayton, and Piotr Dollar. Vincent is always willing to help anybody with anything; he is possibly the nicest person I have ever met. I would also like to acknowledge my colleagues who have been there for moral support, Bianca Zadrozny, Ben Ochoa, Satya Mallick, Manmohan Chandraker, Dave Kauchak, Jongwoo Lim, Kuang-chih Lee, Andrew Rabinovich, Josh Wills, Craig Donner, William Beaver, Neel Joshi, Neil Alldrin, Greg Hamerly, Gyozo Gidofalvi, Matt Tong, and Doug Turnbull.

I would also like to acknowledge professors who have been very influential to me through their classes: Professor de Sa, Professor Kriegman, Charles Elkan, Professor Freund, Alex Orailoglu, Professor Bellare, and Professor Micciancio.

I would also like to acknowledge the Smart Vivarium group, without whom I would not have had such an interesting problem to work on: John Wesson, Keith Jenne, Phil Richter, and Geert Shmid-Schoenbein. I would like to acknowledge my source of funding for the last three years – the NASA Graduate Student Researcher Program.

Finally, I would like to acknowledge my family. That they would love me even if I don't make it through graduate school is a constant source of comfort. My mother Mary Woo, father Jim Branson, brother Steven Branson, and boyfriend Kyle Scheihagen have always been willing to do anything for me. I owe everything to them.

Portions of this dissertation are based on papers that I have co-authored with others. Listed below are my contributions to each of these papers.

1. Parts of Chapters 3, 4, 5, and 6 are based on the paper "Tracking Multiple Mouse Contours (without Too Many Samples)" by K. Branson and S. Belongie [17]. I developed the algorithm, performed the experiments, and wrote the paper.

2. Parts of Chapters 3 and 7 is based on the paper "Three Brown Mice: See How They Run" by K. Branson, V. Rabaud, and S. Belongie. With insights from S. Belongie, I was responsible for developing the algorithm, performing the experiments, and writing the paper.

VITA

1979	Born, Hamburg, Germany.
2000	A. B., Harvard University.
2002	M. S. University of California, San Diego.
2007	Ph. D., University of California, San Diego.

PUBLICATIONS

S. Agarwal, K. Branson, and S. Belongie, "Higher-Order Learning with Graphs", *Proceedings of the International Conference on Machine Learning*, 2006.

K. Branson and S. Belongie, "Tracking Multiple Mouse Contours (without Too Many Samples)", *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2005.

K. Branson, V. Rabaud, and S. Belongie, "Three blind mice: See how they run", *Proceedings of the Joint IEEE Workshop on Visual Surveillance and Performance Evaluation of Tracking and Surveillance at the International Conference on Computer Vision*, 2003.

G. Cottrell, K. Branson, and A. Calder, "Do expression and identity need separate representations?", *Proceedings of the 24th Annual Cognitive Science Conference*, 2002.

ABSTRACT OF THE DISSERTATION

Tracking Multiple Mice through Severe Occlusions

by

Kristin Branson
Doctor of Philosophy in Computer Science

University of California San Diego, 2007

Professor Serge Belongie, Co-Chair

Professor Sanjoy Dasgupta, Co-Chair

In this thesis, I address the problem of tracking multiple identical mice through severe occlusions from video of a side of their cage. A solution to this problem would greatly benefit medical research because of the key role animal testing plays in medical research. As the majority of visual tracking algorithms are intended for tracking people or cars, they are not directly applicable to the mouse tracking problem. Mice are extremely deformable, unconstrained three-dimensional objects. They have few trackable features, and their motion is extremely erratic. In addition, the mice are visually indistinguishable. Thus, to keep track of identities one must track the mice through sometimes complete occlusions. Furthermore, because of the constraints of the application, we had little control over the content of the video sequence. Thus, the bedding of the cage was hard to distinguish from the mice, and the mice were also occluded by scratches on the front of the cage.

In this thesis, I break up the tracking problem into parts: defining a state representation for the mice, defining an appearance model, defining a motion model, and inferring statistics of the positions of the mice for a novel video sequence given the defined model. Because of the difficulty and uniqueness of the mouse tracking problem, it was necessary to understand, modify, combine, and invent state-of-the-art approaches to each of these parts of the tracking problem. In this thesis, I describe and motivate the solutions chosen for each of these subproblems.

1

Introduction

In this thesis, I discuss the problem of visual tracking. A visual tracking algorithm inputs a video sequence and outputs the positions of the targets of interest in every frame. Tracking is currently one of the most popular fields in computer vision. Most research focuses on tracking people or cars in very controlled settings. In fact, in the computer vision literature, the test bed for most tracking algorithms is video of the authors and their co-workers walking around their lab.

I address a very different tracking problem in this thesis. I discuss the application of computer vision algorithms to tracking multiple identical mice from a side view of a cage. Example video frames are shown in Figure 1.1. As mice appear and move in manners significantly different from the standard objects of interest, most existing tracking algorithms were not directly applicable to this problem. Besides being different, I claim that the mouse tracking problem proposed is one of the most difficult addressed in the computer vision literature. Mice are extremely deformable, unconstrained three-dimensional objects. They have few trackable features, and their motion is extremely erratic. In addition, the mice are visually indistinguishable. Thus, to keep track of identities one must track the mice through sometimes complete occlusions. Furthermore, because of the constraints of the application, we had little control over the content of the video sequence. Thus, the bedding of the cage was hard to distinguish from the mice, and the mice were also occluded by scratches on the front of the cage.

Because of the uniqueness and difficulty of the problem, it was necessary to understand, modify, combine, and invent state-of-the-art approaches to every part of the tracking problem. I document in this thesis the successes and failures of algorithms tested for each subproblem.

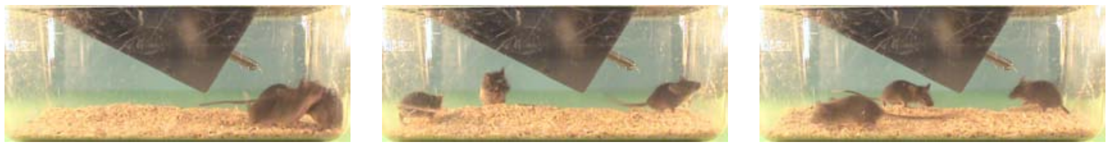


Figure 1.1: Example video frames of a side view of a mouse cage containing three identical mice.

Besides being interesting from a computer vision standpoint, a solution to the mouse tracking problem would be of great benefit to the biology and medical research community. Animal subjects, and mice in particular, are key in modern medical research. The Foundation for Biomedical Research contends that "animal research has played a vital role in virtually every major medical advance of the last century" [40]. Mice are the most popular research subject because they are genetically similar to humans, small and easy to maintain, and have relatively short life-spans and reproductive cycles [113]. UCSD currently houses about 100,000 mice for research purposes. Figure 1.2 shows an image of a vivarium at UCSD. A vivarium is a lab for housing animal subjects. In the world, approximately 25 million mice are used annually.

Because of the huge number of mice used, and the relatively small number of staff monitoring them (UCSD employs about 100 people to take care of the mice), manual, close monitoring of the health and behavior of mice is impossible. An automatic method for closely monitoring the health and behavior of mice would have several benefits. First, it would decrease the number of mice wasted and improve their quality of life. For example, sick mice could be detected before they have infected other mice in the vivarium. Second, new kinds of data could be collected and analyzed because the behavior of the mice could be analyzed 24 hours per day. This is particularly important because mice are nocturnal, thus most of their interesting behavior occurs when lab technicians are absent. Currently, I am working on applying automatic behavior analysis to detecting seizures in mice. These are rare occurrences, thus this application greatly benefits from the 24 hour nature of automatic surveillance.

Our goal in this project was to construct a noninvasive system that could automatically monitor the health and behavior of mice and interface easily with existing vivaria. Thus, we were restricted to placing a camera outside a standard Static Microisolator cage, an example of which is shown in Figure 1.3. Because of the feeder and water bottle at the top of the cage, the best view was obtained from a side of the cage. While this view

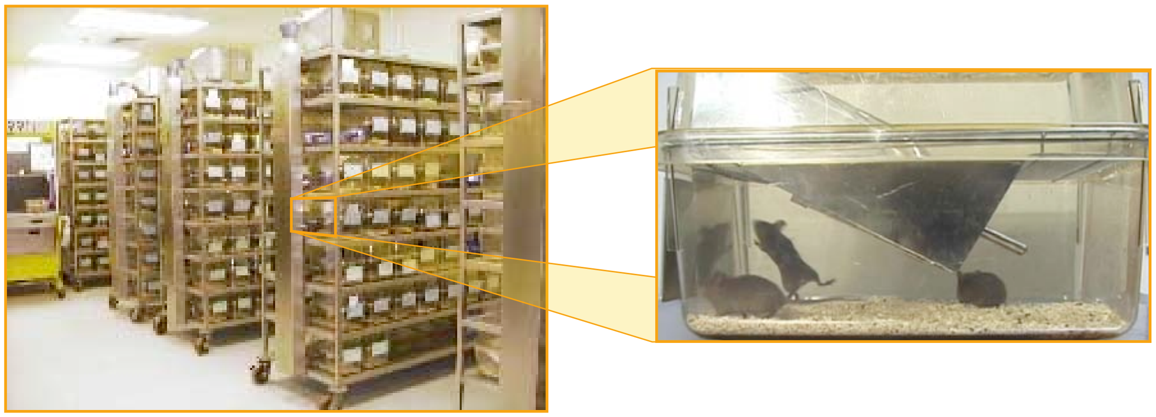


Figure 1.2: An example vivarium at UCSD. Each rack contains tens of mouse cages. Each mouse cage is about the size of a shoe box, and can contain between 1 and 6 mice.

Figure 1.3: An image of a Static MicroIsolator cage common in research vivaria. Image taken from [2].

is perhaps ideal for behavior analysis, it is difficult for tracking purposes when there are multiple mice in the cage. This is because one mouse may be completely hidden from view by another, a complete occlusion. An example occlusion is shown in the left image in Figure 1.1.

Tracking is a first and essential step to behavior analysis. This is first because it can be used to obtain many statistics of interest, such as the position and pose of the mouse. Second, it is necessary because the mice in a single cage are often visually (and sometimes even genetically) indistinguishable. Thus, the only noninvasive way to keep track of the identities of the mice is to successfully track them through occlusions.

As tracking the mice through occlusions proved to be a very difficult problem, it became the focus of my thesis. In the following chapters, I discuss my solutions to the mouse tracking problem. First, in Chapter 2, I introduce a mathematical formulation of the tracking problem. This formulation breaks up the tracking problem into the following parts: choosing a hidden state representation (Chapter 3), choosing an appearance model (Chapter 4), choosing a motion model (Chapter 5), and inferring statistics of the state of the mice given a novel video sequence and the chosen model (Chapter 6). In Chapter 7, I discuss acausal inference algorithms. Finally, in Chapter 8, I discuss future directions for research.

2

A Mathematical Formulation of Tracking

In tracking, our goal is to estimate some properties of the targets at every instant given a video sequence. Let $\mathbf{z}_t$ denote the properties we would like to estimate at time t and let $\mathbf{y}_t$ denote the video observation at time t. We break up the tracking problem into two parts.

The first part is done offline, and involves defining the relationship between the video sequence and the desired properties of the targets. In online tracking, we would like to define a function that tells us how certain we are that the targets currently have properties $\mathbf{z}_t$ given the video sequence up to the current frame $\mathbf{y}_{1:t}$. We will discuss other types of tracking in Section 7. We usually give a probabilistic interpretation to this function. We define a model of the conditional distribution of the desired properties of the targets $\mathbf{z}_t$ given the observations $\mathbf{y}_{1:t}$, $p(\mathbf{z}_t|\mathbf{y}_{1:t})$.

The second problem in tracking is, given the model $p(\mathbf{z}_t|\mathbf{y}_{1:t})$, inferring statistics of this distribution for a novel video sequence $\mathbf{y}_{1:t}$. Statistics we are usually interested in are the *maximum-a-posteriori* (*MAP*) properties of the targets, the expected properties of the targets, and the variance of the estimate of the target properties. Computing these statistics is nontrivial because there is often no analytical expression for them, or there is no efficient way of calculating them exactly. Computing these statistics is referred to as inference.

2.1 Defining the Model

Specifying the posterior distribution $p(\mathbf{z}_t|\mathbf{y}_{1:t})$ involves specifying the relationship between $t+1$ variables, at least t of which are high-dimensional vectors. To make this tractable, we first introduce a hidden state representation of the targets. We denote the hidden state at time t by $\mathbf{x}_t$. One can think of the hidden state $\mathbf{x}_t$ as a sufficient statistic of the desired properties of the targets $\mathbf{z}_t$ and the video sequence $\mathbf{y}_{1:t}$. The hidden state $\mathbf{x}_t$ must contain enough information so that we can deterministically compute the properties of the position desired for our application, $\mathbf{z}_t$. In addition, the hidden state at time t should contain all information about the video sequence up to time t, $\mathbf{y}_{1:t}$, relevant for predicting the hidden state at time $t+1$, $\mathbf{x}_{t+1}$. Finally, the hidden state representation should be low-dimensional to make inference and modeling tractable.

If we have such a hidden state representation, then we can break down the relationship between all variables $\mathbf{z}_t$ and $\mathbf{y}_{1:t}$ into the combination of the relationship between pairs of variables. The relationships that must be defined are the following. First, we must define the relationship between the hidden state $\mathbf{x}_t$ and the desired properties $\mathbf{z}_t$. We will assume that this is a simple, deterministic relationship, and will ignore $\mathbf{z}_t$ in the rest of this thesis. We will consider our goal to be to estimate statistics of the posterior distribution of the hidden state, $p(\mathbf{x}_t|\mathbf{y}_{1:t})$. Second, we must define the relationship between the hidden state at the current frame $\mathbf{x}_t$ and the current video frame $\mathbf{y}_t$. This is called the *appearance model* and is sometimes given the probabilistic interpretation of the likelihood of the observation $\mathbf{y}_t$ given the hidden state $\mathbf{x}_t$, $p(\mathbf{y}_t|\mathbf{x}_t)$. Third, we must define the relationship between consecutive hidden states, $\mathbf{x}_{t-1}$ and $\mathbf{x}_t$. This is called the *motion model* and is sometimes given the probabilistic interpretation of the distribution of the current hidden state $\mathbf{x}_t$ given the previous hidden state $\mathbf{x}_{t-1}$, $p(\mathbf{x}_t|\mathbf{x}_{t-1})$.

The assumption that the hidden state $\mathbf{x}_t$ is truly a sufficient statistic is called the first-order Markov assumption. More specifically, the first-order Markov assumption is that (1) the current observation $\mathbf{y}_t$ is conditionally independent of all the other variables given the current hidden state $\mathbf{x}_t$ and (2) variables in the future are conditionally independent of variables in the past given the current hidden state $\mathbf{x}_t$.

Given the first-order Markov assumption, we can decompose the posterior distribution as

$$\begin{aligned}
p(\mathbf{x}_t|\mathbf{y}_{1:t}) &= p(\mathbf{y}_t, \mathbf{x}_t|\mathbf{y}_{1:t-1})/p(\mathbf{y}_t|\mathbf{y}_{1:t-1}) && (2.1)\\
&= p(\mathbf{y}_t|\mathbf{x}_t, \mathbf{y}_{1:t-1})p(\mathbf{x}_t|\mathbf{y}_{1:t-1})/p(\mathbf{y}_t|\mathbf{y}_{1:t-1}) && (2.2)\\
&= p(\mathbf{y}_t|\mathbf{x}_t, \mathbf{y}_{1:t-1}) \int p(\mathbf{x}_t, \mathbf{x}_{t-1}|\mathbf{y}_{1:t-1})d\mathbf{x}_{t-1}/p(\mathbf{y}_t|\mathbf{y}_{1:t-1}) && (2.3)\\
&= p(\mathbf{y}_t|\mathbf{x}_t, \mathbf{y}_{1:t-1}) \int p(\mathbf{x}_t|\mathbf{x}_{t-1}, \mathbf{y}_{1:t-1})p(\mathbf{x}_{t-1}|\mathbf{y}_{1:t-1})d\mathbf{x}_{t-1}/p(\mathbf{y}_t|\mathbf{y}_{1:t-1}) && (2.4)\\
&= p(\mathbf{y}_t|\mathbf{x}_t) \int p(\mathbf{x}_t|\mathbf{x}_{t-1})p(\mathbf{x}_{t-1}|\mathbf{y}_{1:t-1})d\mathbf{x}_{t-1}/p(\mathbf{y}_t|\mathbf{y}_{1:t-1}). && (2.5)
\end{aligned}$$

Since $p(\mathbf{y}_t|\mathbf{y}_{1:t-1})$ is a constant that does not depend on the hidden state $\mathbf{x}_t$, Equation (2.5) is a recursive definition of the posterior distribution $p(\mathbf{x}_t|\mathbf{y}_{1:t})$ in terms of the appearance model $p(\mathbf{y}_t|\mathbf{x}_t)$ and the motion model $p(\mathbf{x}_t|\mathbf{x}_{t-1}$. Thus, to define the posterior distribution, we need only specify the appearance and motion models.

For the frequentists, I note that if we choose not to believe the first-order Markov assumption, we can think of this approach as just assuming a sparse parametric form for the posterior distribution.

2.2 Inferring Statistics of the Hidden State

The decomposition of the posterior distribution in Equation (2.5) translates into an iterative algorithm. Given the posterior distribution of the hidden state in the previous frame $p(\mathbf{x}_{t-1}|\mathbf{y}_{1:t-1})$, we can use the motion model to compute a prior on the current hidden state position:

$$p(\mathbf{x}_t|\mathbf{y}_{1:t-1}) = \int p(\mathbf{x}_t|\mathbf{x}_{t-1})p(\mathbf{x}_{t-1}|\mathbf{y}_{1:t-1})d\mathbf{x}_{t-1} \qquad (2.6)$$

We call this a prior because this distribution does not use information in the current frame, $\mathbf{y}_t$. It is easy to compute this prior if the motion model has a form simple enough to allow us to estimate this integral. This is usually the case in visual tracking algorithms.

The support of this prior distribution $p(\mathbf{x}_t|\mathbf{y}_{1:t-1})$ will be small relative to the hidden state space as the targets cannot, for example, teleport. The regions of the hidden state space where the prior probability is significant can be thought of as the space we must search to find the hidden state of the mice at time t. This is the advantage of object tracking over object detection algorithms, which consider each frame independently. The prior distribution also defines a preference for one reasonable hidden state over another reasonable state.

Once we have defined a search space, we can compare (most of) the hidden states in the search space to the current video frame. This is done using the appearance model. The final posterior score is the product of the appearance score and the motion score.

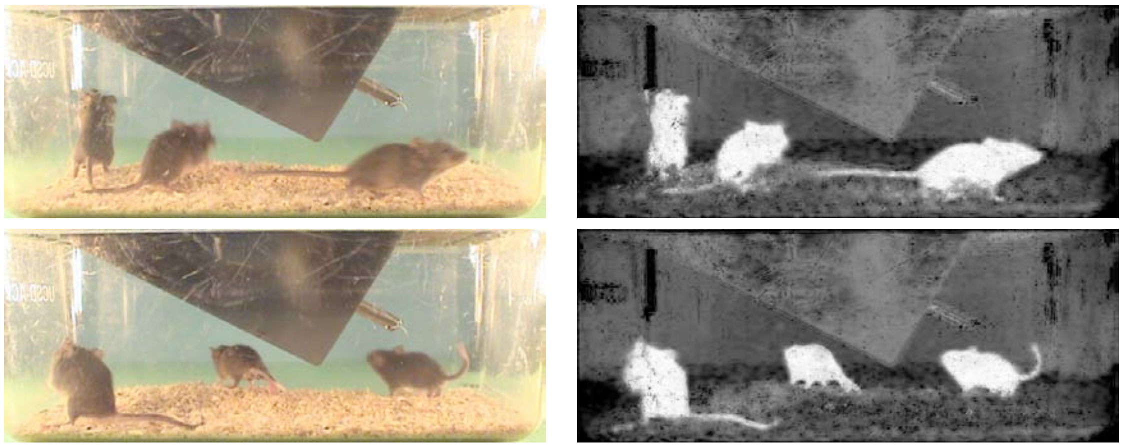
2.3 Considerations Specific to Mouse Tracking

When we are choosing our hidden state representation, model, and inference algorithm, it is important to consider the following property of mouse tracking. When the mice are separated, that is, when no mouse is in front of another mouse, tracking is easy. One approach that works pretty well is background subtraction. When the mice are separated, as in the frames in Figure 2.1(a), the output of background subtraction can easily be used for tracking. When the mice are occluding one another, that is, when one mouse is in front of another mouse, tracking is very difficult. As is evident in the frames in Figure 2.1(b), the output of background subtraction is hard to interpret during occlusions.

However, when we look at static images of the mice during an occlusion, we can tell where the mice are pretty well. How are we able to do this? We have a very accurate model of what shapes are possible mouse shapes, and what a given mouse shape looks like. In addition, we have a strong model of motions that can occur from one frame to the next.

2.4 Outline

To summarize our progress so far, we have decomposed the tracking problem into the following subproblems. First, what is the hidden state of the targets? The next two questions define the relationship between the hidden state at time t and the video sequence up to time t: What are the appearance and motion models? Finally, given the state representation and model, how do we estimate statistics of the posterior distribution for a particular video sequence. That is, how do we perform inference? In this thesis, I address each of these issues in relation to the mouse tracking application in separate chapters. I then discuss acausal inference algorithms that estimate the current state given observations in both the past and future.


(a) No occlusions.

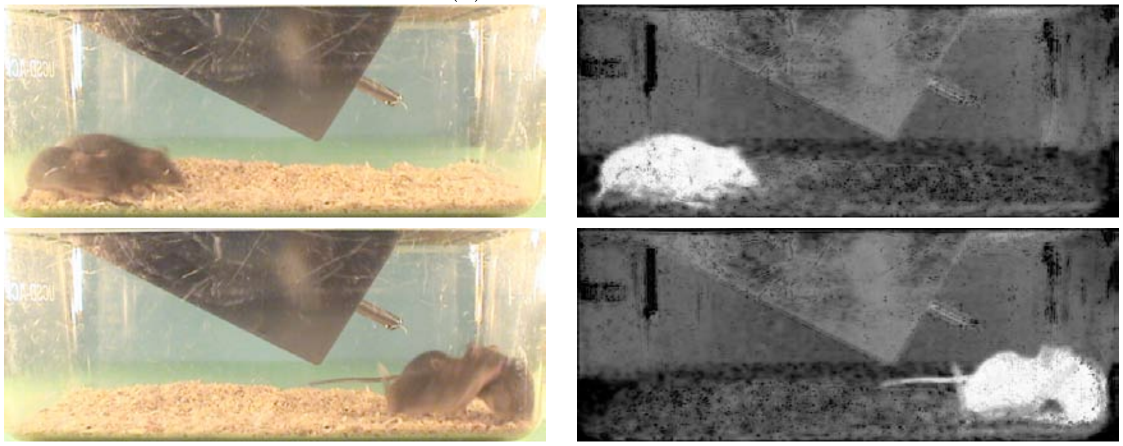

(b) Occlusions.

Figure 2.1: In the left column, we show the input video frame. In the right column, we show the results of simple processing based on models of the foreground and background appearance. When the mice are not occluding one another, we can easily use the images on the right to track the mice. We therefore focus our attention on occlusions.

3

Hidden State Representation

One of the most difficult parts of applying tracking algorithms to a new domain is choosing the representation for the target position. First, the hidden state must contain a lot of information. This is because of the sparse functional form assumed for the distribution. Recall that we assume that the current state is a sufficient statistic of the current and past images. Second, the hidden state must be low-dimensional. This is because the amount of time and space required to accurately infer statistics of the hidden state generally increases exponentially with dimensionality. Satisfying both these conflicting criteria is difficult.

The majority of shape modeling algorithms model the 2-D image projection of the target shape. Mice are very different than other, more standard types of targets, in terms of the types of shapes and deformations they can take. As a mouse is extremely deformable in 3 dimensions and is free to rotate in 3 dimensions, there is a huge amount of variability in the 2-D mouse shape. See Figure 3.1 for example shapes of a mouse.

In this chapter, we first describe state representations used in previous research.

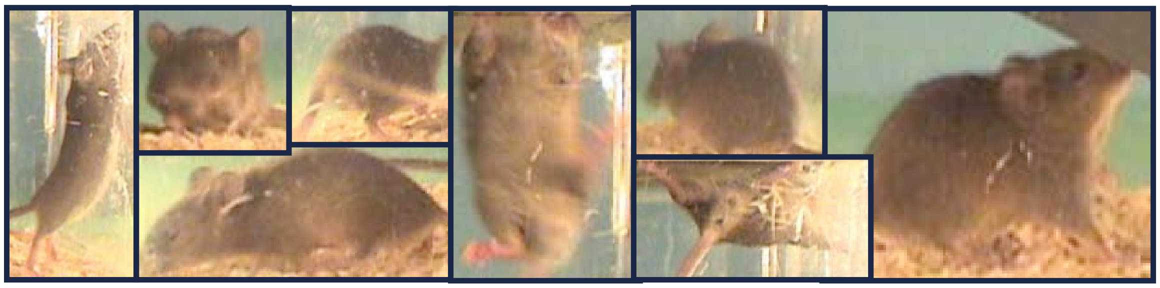


Figure 3.1: Example images of a mouse deforming and rotating in three dimensions. This figure illustrates the extreme amount of deformation the mouse shape undergoes.

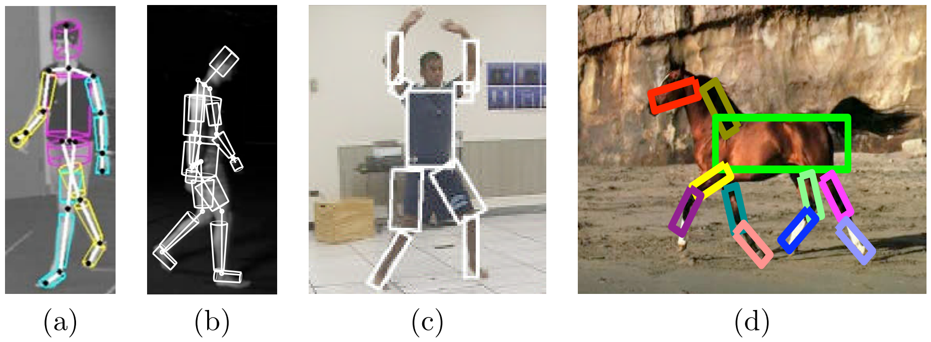


Figure 3.2: Example articulated structures. (a-c) model a person, while (d) models a horse. Images are taken from (a) [106] (b) [35] (c) [91] (d) [90].

Then, we describe the state representations we used for modeling mice. Our approach for choosing a hidden state representation was to begin with a very simple representation. When/if it was determined that this state representation was lacking in information, we added information to the representation. We began our research with manually-determined state representations. These manually determined state representations were used in the research described in Sections 6.9 and 7. Our recent research has explored automatically learned state representations.

3.1 Previous Work

3.1.1 Articulated Structures

While a person is also a highly deformable object, a person is well-modeled by an articulated structure [94]. An articulated structure models the target as a collection of rigid objects that are connected at joints. Figure 3.2 shows example articulated representations of a person. Unfortunately, an articulated model does not work well for a mouse because the limbs of the mouse are relatively small and difficult to detect, and the body of the mouse is very blob-like, as is evident in Figure 3.3.

3.1.2 Active Shape Models

For targets which are more blob-like, such as faces, lips, and objects tracked in medical imaging such as the heart and lungs, a common approach is to learn an active shape model from training data [26]. In active shape models, the target is modeled in terms of the positions of landmark points. Corresponding landmark points are first labeled in

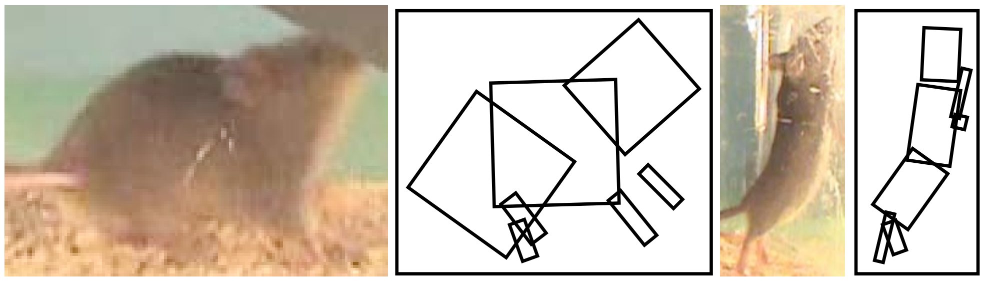


Figure 3.3: An attempt at finding an articulated model of a mouse. The legs of the mouse are difficult to detect, and the body and head are very blob-like.

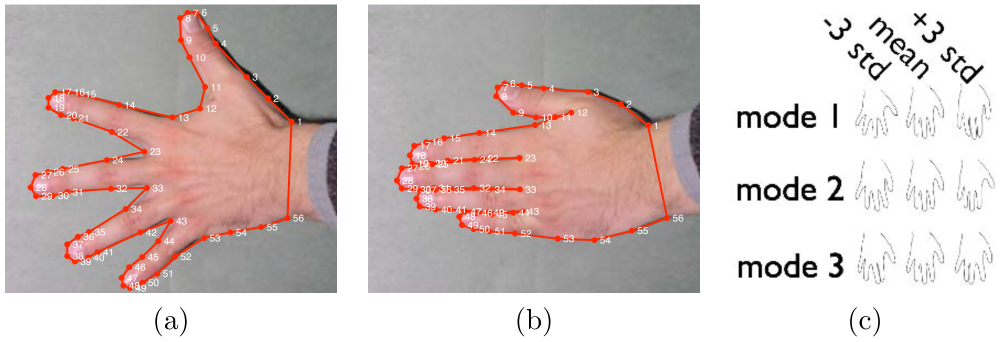


Figure 3.4: An active shape model learned for a hand. In each training image, 56 corresponding parts were manually labeled. Two training images are shown in (a) and (b). (c) shows the learned active shape model. The middle column of (c) shows the mean landmark positions. Each row corresponds to a different dimension of the subspace learned. The left and right columns show the variation in the landmark positions along the given dimension. Note that the hand modeled can only deform; it is not allowed to rotate out-of-plane.

a set of training images. A low-dimensional manifold of the landmark positions is found, and the hidden state space is then any point on this low-dimensional manifold. Figure 3.4 shows the 56 corresponding parts manually labeled for learning an active shape model representation of a hand. Note that the hand modeled only deforms – it is not allowed to rotate out-of-plane.

The original active shape model [26] finds the optimal d-dimensional linear subspace of the landmark positions using PCA. This approach assumes (among other things) that the space of feasible landmark positions is unimodal. If this is not the case, we get an inefficient representation of shape, and implausible shapes can occur. This proved to be a problem with modeling mice, as the number of dimensions required to represent the variability in mouse shape using this approach is very large. This huge amount of variabil-

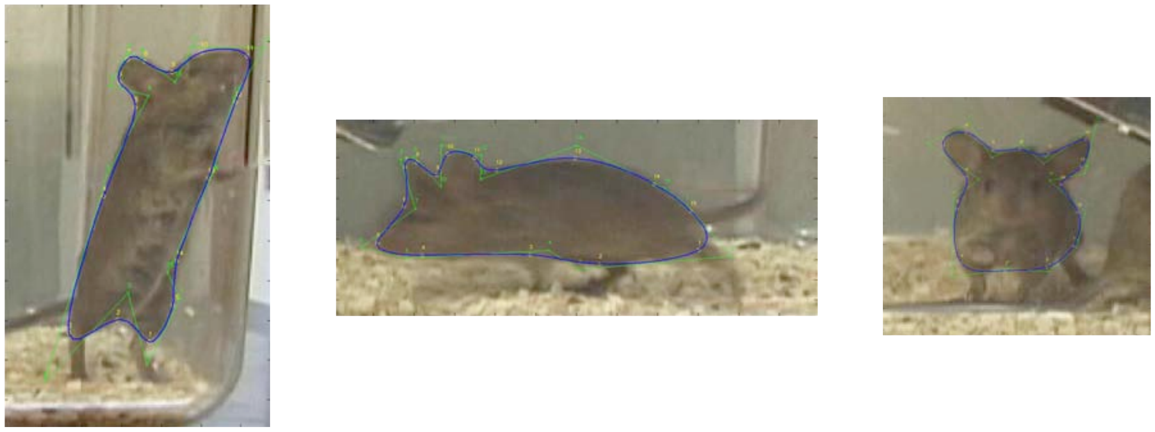


Figure 3.5: An attempt to label corresponding landmark points in training images of mice. Even in this small set of examples in which the mouse is on the floor of the cage, it is difficult to define corresponding parts.

ity stems from the correspondence problem. Active shape model approaches assume that there are landmark points that appear and are easily identifiable (by the human labeler) in every possible shape. When a target is free to perform out-of-plane rotation and is very deformable, as is the case for mice, a given landmark point may only appear in a small subset of frames. See, for example, the mouse shapes in Figure 3.5. Even for this set of examples in which the mouse is only on the floor of the cage, correspondences are difficult to find. We attempted to find corresponding landmark points for the mice, but many of the decisions made were arbitrary, and corresponding points did not in fact correspond to the same part of the mouse. For example, in the left image of Figure 3.5, only one ear of the mouse is visible, while in the other two images, both ears are visible. Thus, the nose of the mouse in the left image corresponds to an ear of the mouse in the other two images.

A number of approaches exist for extending active shape models to multimodal shape distributions [19, 52, 48, 47]. [48] assumes that correspondences can only be found between subsets of training examples. [48] first uses an integrated registration and clustering approach to partition the shapes, establish point correspondences between similar shapes, and align them with respect to a similarity transform. Then, a separate linear subspace is found for each cluster. Finally, transition probabilities between different parameterizations are determined. We explore a similar approach in Section 3.4.

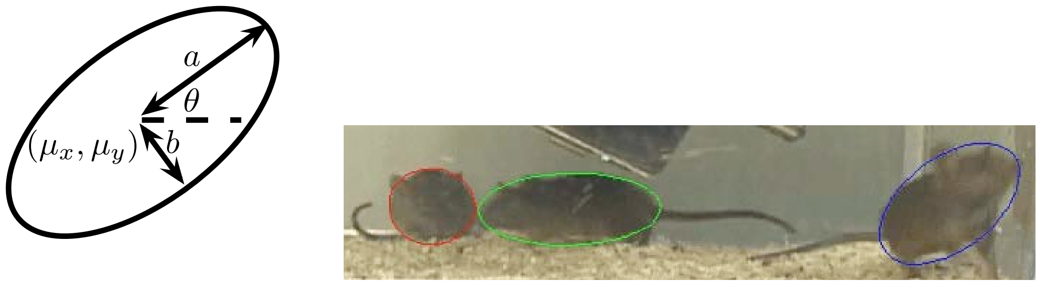


Figure 3.6: Ellipse representation of a mouse. The figure on the left shows the five parameters of the ellipse. The figure on the right shows an example ellipse fit.

3.2 Ellipse-shaped Mice

Our first model of a mouse was as the five parameters of an ellipse. The hidden state representation for K mice contained $5K$ parameters, obtained by concatenating the ellipse parameters for each mouse. An ellipse can be defined by the x-coordinate of the center μ_x, the y-coordinate of the center μ_y, the semi-major axis length a, the semi-minor axis length b, and the rotation θ, as shown in Figure 3.6. Alternatively, an ellipse can be defined by the two parameters defining the center of the ellipse and the three parameters defining the 2-D covariance Σ of the ellipse:

$$(\mathbf{x}-\mu)^\top \Sigma^{-1} (\mathbf{x}-\mu) \leq 4 \tag{3.1}$$

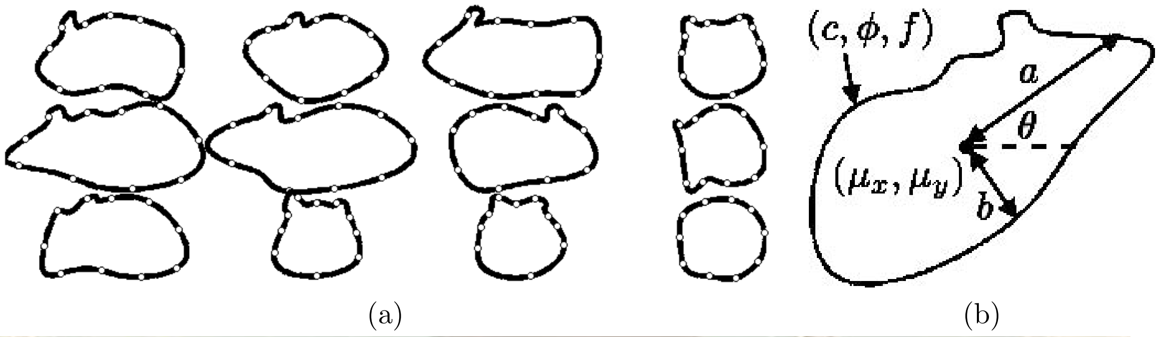
The relationship between the covariance Σ and the ellipse parameters (a, b, θ) is

$$\Sigma = \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix}^\top \begin{pmatrix} a^2 & 0 \\ 0 & b^2 \end{pmatrix} \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix} \tag{3.2}$$

We found that the ellipse well-modeled the mouse for blob-based tracking (Section 4.11.1), although little signal was available during occlusions.

3.3 Contour Templates

A more complex representation is necessary for contour-based tracking (Section 4.9.1), as the edge feature is much sparser than the blob feature. We found decent performance by allowing any affine transformation of any of 12 manually drawn contour templates,



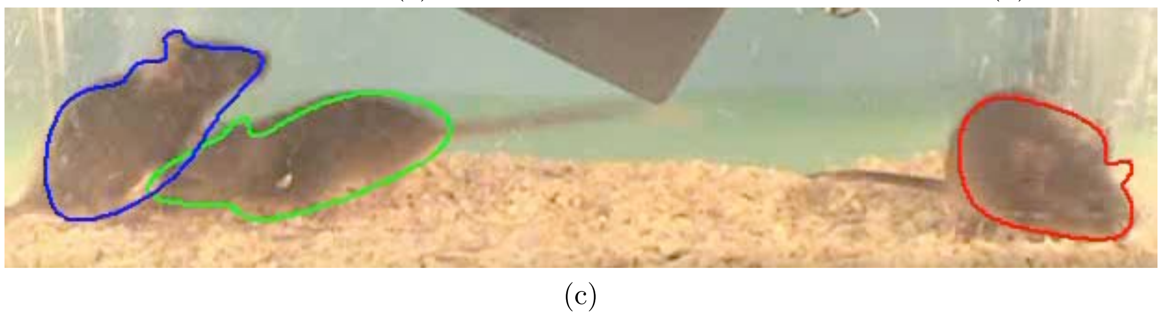


Figure 3.7: Contour template representation of a mouse. (a) shows the 12 manually drawn templates. (b) shows the parameterization of the template and affine transformation. (c) shows an example fit.

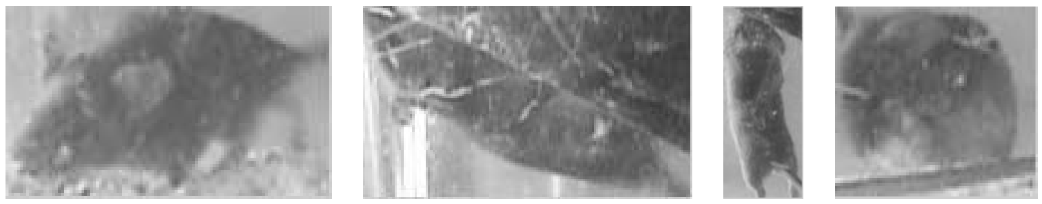
shown in Figure 3.7.

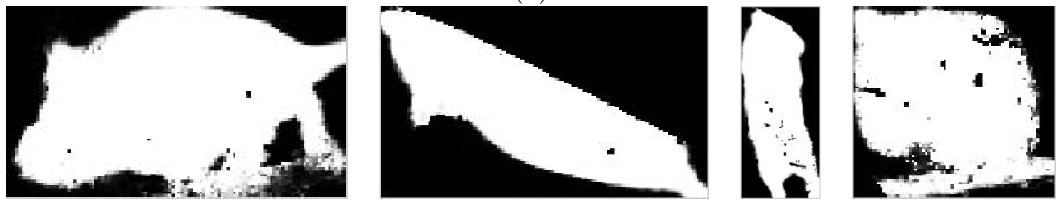
These templates were chosen in a fairly arbitrary manner. After much study of video of mice, I chose 12 images in which the mice seemed to be in representative poses. I then outlined these mice and fit smooth B-spline functions to the outlines. It was necessary to have a simple, analytical expression for each contour template for the work presented in Section 6.9.

3.4 Learning a Hidden State Representation

While the ellipse and contour template representations worked much of the time, as discussed in Section 6.9, we felt that better results could be obtained with a more precise state representation. We have therefore recently been exploring methods for learning a hidden state representation from video. Our approach is most similar to [44], the basis for [48] described earlier.

Training data is collected in a mostly unsupervised manner. The manual labels given are the approximate starts and ends of occlusions in a video sequence. This is a fairly simple problem, and we plan on automating this step in the future. As tracking mice is


(a)


(b)

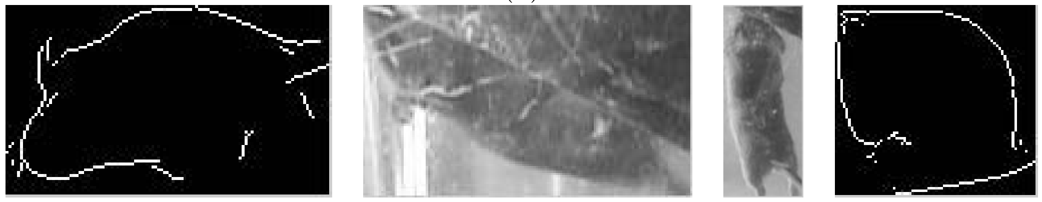

(c)

Figure 3.8: Automatically collected training data. The mice are tracked during non-occlusions using the background subtraction data. Around the predicted position of the mouse, we crop subimages. (a) shows the subimage of the original video frame cropped around the predicted mouse position. (b) shows the cropped subimage of the foreground probability image. Each pixel represents the probability that the pixel belongs to the foreground, with white representing 1 and black representing 0. (c) shows the cropped subimage of the edge detection image.

fairly simple in the frames in which there is no occlusion, we automatically track the mice in these frames using an ellipse model of a mouse and the background subtraction images (see Section 4.1). From the background subtraction images, we crop subimages around the predicted position of each mouse. As our background subtraction image contains the log-likelihood ratio of foreground to background, we can use this image to compute the probability that each pixel belongs to the foreground. Figure 3.8 (b) shows examples of the probability that each pixel belongs to foreground for example training images. We also crop subimages around the predicted mouse positions from the edge detection images. Figure 3.8 (c) show the edge pixels for example training images. We collect training data for each of three mice from 3000 training images.

We then define a distance between a pair of shapes. First, the two shapes are aligned by finding the best affine transformation aligning the two shapes in terms of the shape context distance [10]. Then, the Chamfer distance between the aligned edge training images and the sum-squared distance between the aligned foreground images are averaged.

Because of the computational requirements of computing the pairwise distance between all shapes, we make two simplifications. First, we break up the training data into four disjoint sets based on the ellipse positions of the mice. These correspond to mice on the ceiling, round mice on the floor, horizontal mice on the floor, and vertical mice on the floor. Only pairwise distances within each set are computed. Second, we only compute the pairwise distance between 225 samples from each class. This is approximately one sample from every 10 frames.

We use the k-medoids algorithm to cluster the 225 samples into 20 initial classes. We then greedily compute a hierarchical clustering using agglomerative clustering. At each iteration, we merge the two clusters whose average distance to the combined center is smallest. At each level of the hierarchy, each of the $3000 \cdot 3$ training examples is given a cluster label. Each training example is aligned to the center of its cluster, and the mean probability that a pixel is foreground is computed. In addition, the mean distance to an edge from each pixel is computed. We call the mean probability of foreground the blob template for the cluster, and the mean distance to an edge the contour template for the cluster.

As the ellipse model of the mouse is symmetric according to reflections and rotation by π, we perform a post-processing step to determine whether to flip the templates over the x-axis and/or rotate the templates by π. We choose to rotate and flip so that the total rotation and the total number of flips are minimized. The optimal setting is computed using a variant of the WalkSAT algorithm [101]. The resulting blob and contour templates are shown in Figures 3.9, 3.10, 3.11, 3.12.

Representative examples of the fits of the templates to the 9000 training instances are shown in Figure 3.13. Here, we use only the contour template, and plot the local minima of the average distance below a threshold. In future work, we plan to incorporate this model of mouse shape.

3.5 Acknowledgements

Portions of this chapter are based on the paper “Tracking Multiple Mouse Contours (without Too Many Samples)” by K. Branson and S. Belongie [17]. I developed the algorithm, performed the experiments, and wrote the paper.

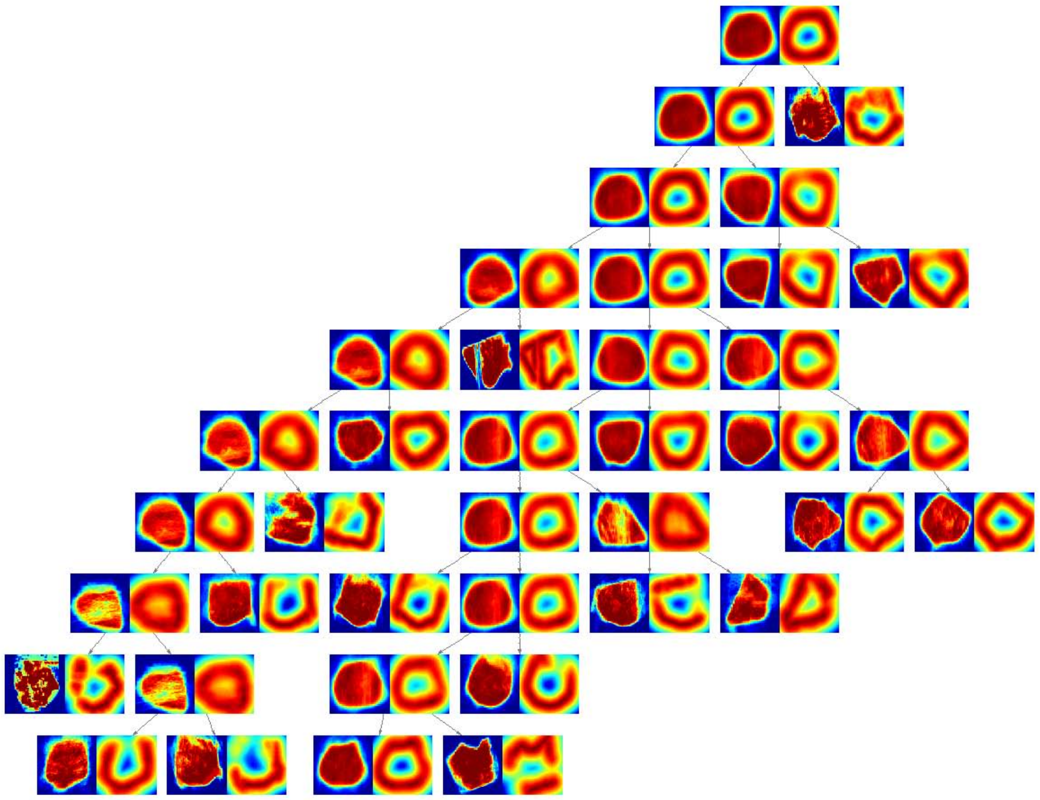


Figure 3.9: Blob and contour templates for each level of the hierarchy for mice on the ceiling. The left image for each cluster is the blob template; each pixel shows the probability of foreground. The right image for each cluster is the contour template. Each pixel shows the negative average distance to a detected edge.

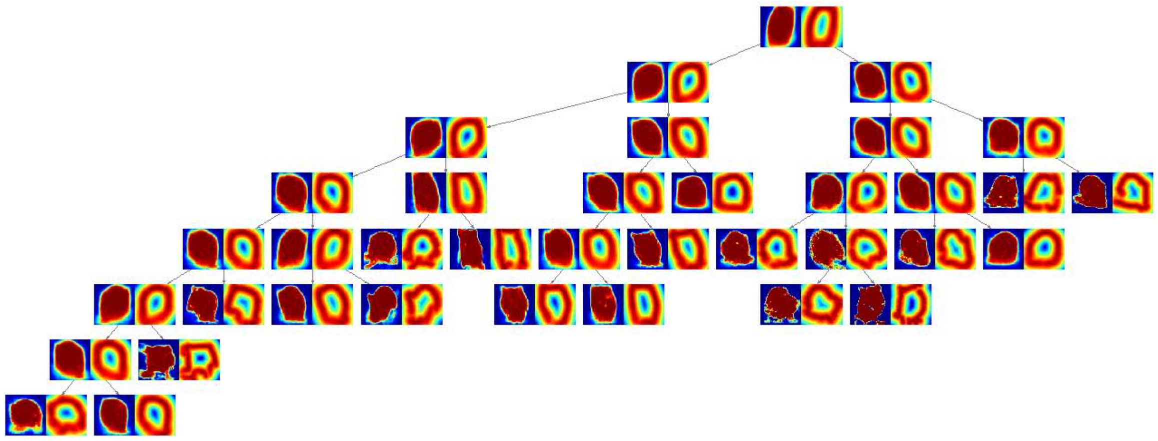


Figure 3.10: Blob and contour templates for each level of the hierarchy for round mice on the floor. The left image for each cluster is the blob template; each pixel shows the probability of foreground. The right image for each cluster is the contour template. Each pixel shows the negative average distance to a detected edge.

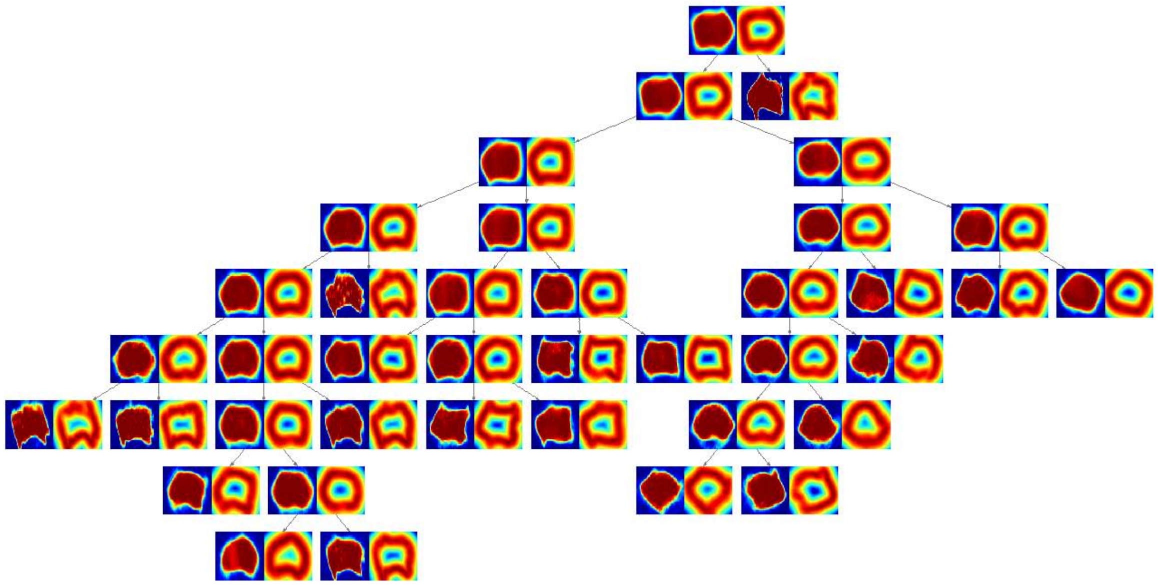


Figure 3.11: Blob and contour templates for each level of the hierarchy for vertical mice on the floor. The left image for each cluster is the blob template; each pixel shows the probability of foreground. The right image for each cluster is the contour template. Each pixel shows the negative average distance to a detected edge.

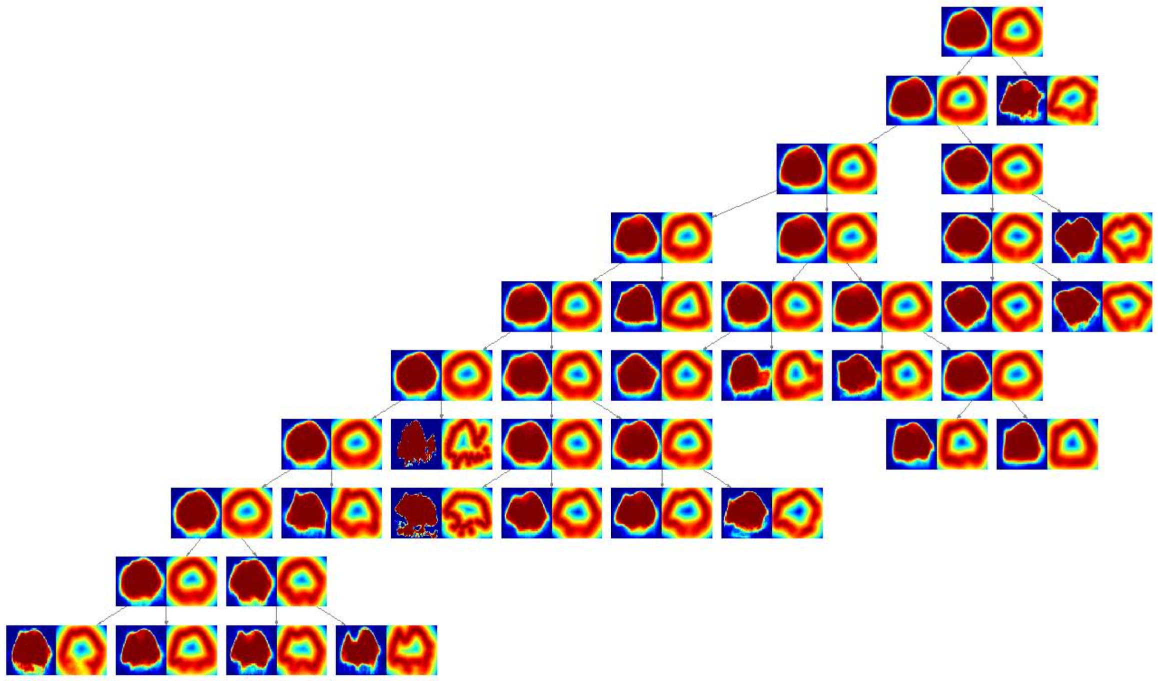


Figure 3.12: Blob and contour templates for each level of the hierarchy for horizontal mice on the floor. The left image for each cluster is the blob template; each pixel shows the probability of foreground. The right image for each cluster is the contour template. Each pixel shows the negative average distance to a detected edge.

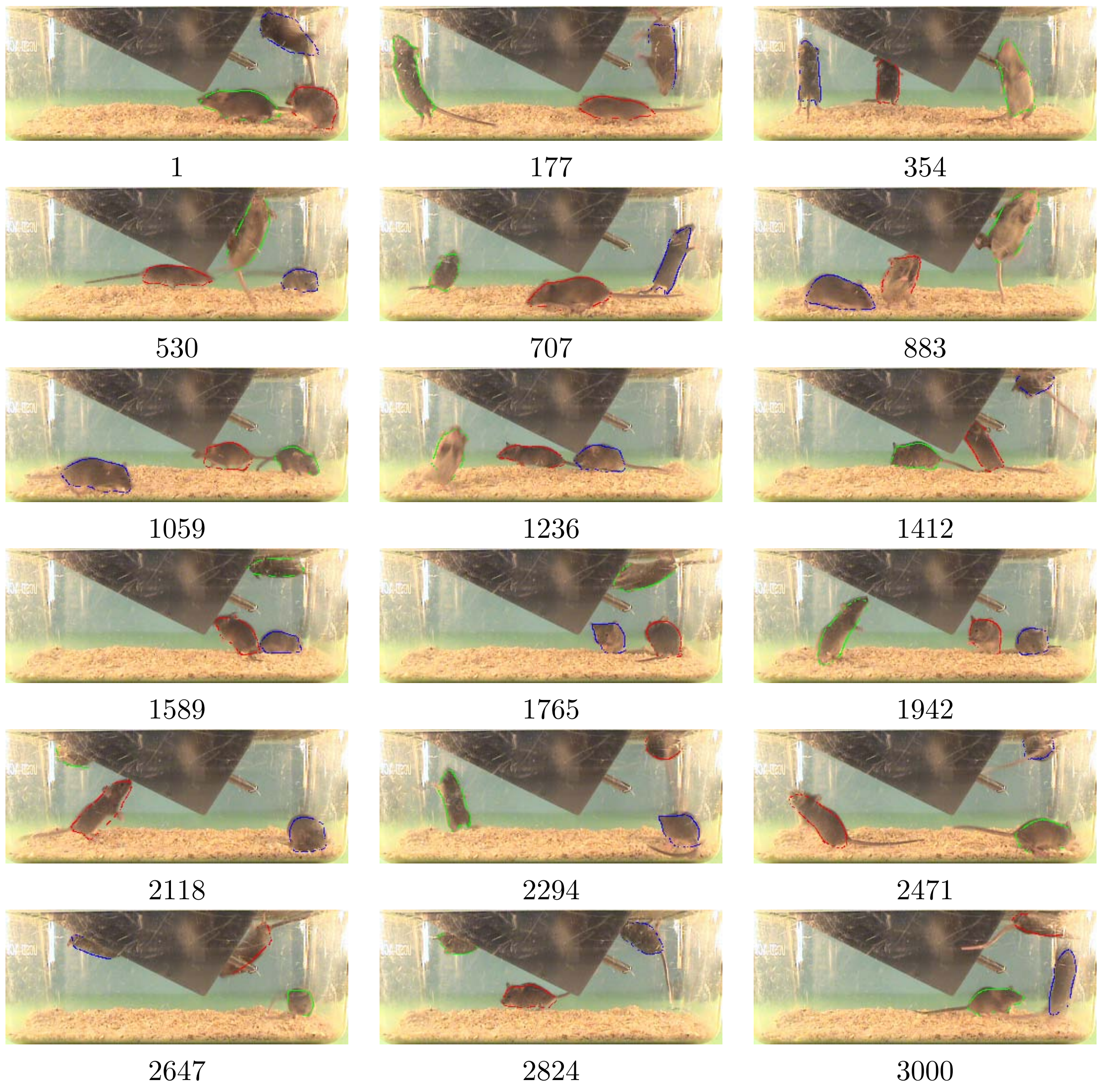


Figure 3.13: Example results of fitting the $20 \cdot 4$ training images to the $3000 \cdot 3$ training instances. We plot the local minima of the average distance to an edge below a threshold for evenly spaced example images.

4

Appearance Models

Perhaps the most essential part of a visual tracking algorithm is the model of the relationship between a video frame image and the positions of the targets: the *appearance model*. An appearance model can be viewed mathematically as a function $f_{app} : \mathcal{Y} \times \mathcal{X} \rightarrow \mathbb{R}$ that inputs the video frame image $\mathbf{y}_t$ and the hypothesized positions of the targets $\mathbf{x}_t$ and outputs a score $f_{app}(\mathbf{y}_t, \mathbf{x}_t)$ that is higher the better the image and target positions correspond with one another. Often, this function is given a probabilistic interpretation. Generative models interpret the appearance scoring function $f_{app}(\mathbf{y}_t, \mathbf{x}_t)$ as the likelihood of the video image given the state of the targets, $p(\mathbf{y}_t|\mathbf{x}_t)$.

In this chapter, I discuss appearance models for tracking. The appearance models discussed make different assumptions about the nature of the video and target appearance. Each type works with different statistics of the video sequence, and assumes that enough information is available in these statistics to determine the targets' positions. To choose an appearance model reasonable for one's application, one must use domain-specific knowledge. Existing appearance models had previously only been applied to tracking targets quite different from mice, for instance people and cars. In this chapter, I discuss the applicability of these appearance models to the mouse tracking application with respect to the following questions. Do the assumptions made by the model reasonable? Are the statistics of the video sequence modeled sufficient for tracking?

Before we begin reviewing appearance models for tracking, let us note that object detection and recognition are very related fields. Recent algorithms have been bridging the gap between tracking and detection [5, 6, 124]. However, I focus on appearance models specific to tracking in this thesis.

4.1 Background Modeling

In many tracking applications, the camera is assumed to be still. In this case, it is useful to learn the appearance of the video frame image if the targets were absent. For example, in the mouse tracking application, this is the appearance of an empty cage, as in Figure 4.1. The majority of background models model each local region of the image

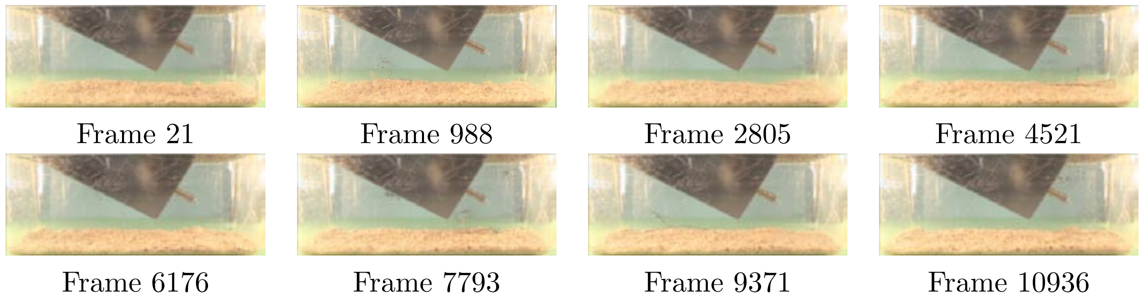


Figure 4.1: Expected background appearance of the mouse cage computed automatically at different frames in the sequence. These images were computed from the expected value of the Gaussian components of the background model.

independently, i.e. for each location g on a dense grid, we learn a separate model

$$score_{back}(\mathbf{y}_g; \theta_g) \tag{4.1}$$

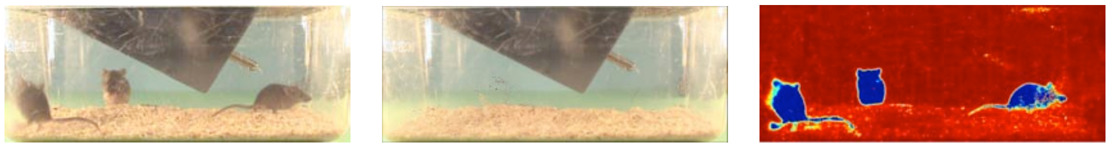
where $\mathbf{y}_g$ are the image observations in a small patch centered at location g and θ_g are the parameters of the function specific to the background model at location g. The value $score_{back}(\mathbf{y}_g)$ should be high if the image patch at location g is contained mostly in the background, and low if it is contained mostly in the foreground. It is common to give a probabilistic interpretation to this score:

$$score_{back}(\mathbf{y}_g) = p(\mathbf{y}_g | label_g = background; \theta_g). \tag{4.2}$$

Thus, the $score_{back}(\cdot)$ function can be viewed as a statistical description of the background appearance at location g.

At every location g in the current video frame, we compute $score_{back}(\mathbf{y}_g)$. The background likelihood image of these scores, as can be seen in Figure 4.2, is a very useful feature for tracking.

Research on background modeling has has focused on three questions. First, what representation $\mathbf{z}_g$ should be used for a patch of background $\mathbf{y}_g$? Second, what parametric



(a) Original image.

(b) Expected value of current background model.

(c) Log of the background likelihood score for each pixel.

Figure 4.2: Background likelihood model applied to a video frame.

form of $score_{back}(\cdot)$ should be assumed [85]? Finally, how do we estimate the parameters of the background appearance model? More specifically, as the appearance of the background generally changes over time, how do we update the background model?

Because background modeling proved to be one of the most successful features for mouse tracking, I discuss in some detail a smattering of approaches to background modeling. I focus on background models that (1) model background appearance at each pixel location independently and (2) treat each frame independently. I focus on this type of background model because they are simpler and faster to compute. After all, the background likelihood image is just one appearance feature used in my appearance model. For a more general review of background modeling algorithms, see e.g. [85, 38, 65, 103]. In Section 4.2, I discuss the approach to background modeling I chose for the mouse tracking application.

4.1.1 Patch Representation

The majority of background modeling algorithms operate on single pixels, i.e. patches of size 1. Choosing the patch representation then boils down to choosing a color space representation. A number of color spaces have been experimented with, including RGB, normalized RGB, and HSV. When choosing a color space, one of the main concerns of prior research has been invariance to lighting changes. For an experimental comparison of different color spaces for background modeling under illumination changes, see [68]. [126, 59] propose an edge-based representation of a patch, again for invariance to illumination changes.

A number of approaches for change detection that operate on patches of size greater than 1 interpret the pixel values in the patch as a vector of random variables. The background score $score_{back}(\mathbf{y}_g)$ is a function of the sum-squared difference between the current patch $\mathbf{y}_g$ and a reference patch [87].

A few approaches convolve each color channel of the patch with a filter, for instance a Gaussian filter, resulting in a single value to represent each channel of the patch [110].

Texture features can be captured using a Laplacian of Gaussian (LoG) filter [57].

4.1.2 Parametric Form

As in most machine learning problems, there is a bias versus variance tradeoff when selecting the parametric form of the background model. If the parametric form is too simple, then the model will not represent the data well and results will be poor. If the parametric form is too complex, the amount of data necessary to learn the parameters is excessive, and results again will be poor.

Possibly the simplest model is a single Gaussian [85]:

$$p(\mathbf{y}_g | label_g = background; \theta_g) = \mathcal{N}(\mathbf{z}_g; \mu_g; \Sigma_g) \tag{4.3}$$

where $\mathbf{z}_g$ is a chosen representation of the patch of pixels $\mathbf{y}_g$.

In many applications including mouse tracking, the appearance of a background patch can change periodically. In the mouse tracking application, this is primarily due to

- Shadows cast by the mice.
- Reflections cast by the mice.
- Noise from the camera and compression artifacts.

See Figure 4.3(d-f) for illustrations of these periodic changes. In more standard applications, periodic changes in appearance are also due to objects moving in the background, for example trees moving in the wind [107]. See Figure 4.3(a-c) for illustrations of these types of periodic changes.

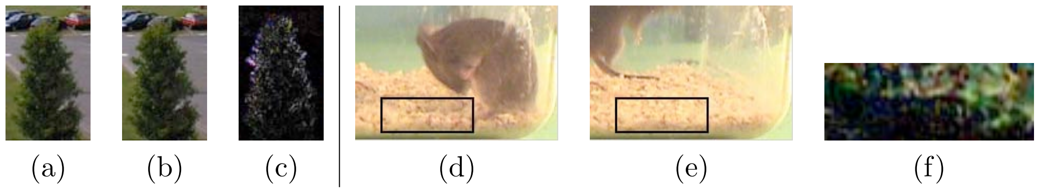


Figure 4.3: (a-c) Example video frames of an outdoor scene from [58]. The tree moves in the wind, causing variance in the background appearance at a given location. (a) Video frame 1. (b) Video frame 30 (≈ 1 second later). (c) Absolute difference of (a) and (b) (image intensity is scaled). (d-f) Example images of the right bottom corner of the mouse cage. The appearance of the bedding is different in the shadow of the mouse. (d) Video frame 200. (e) Video frame 400 (≈ 7 seconds later). (f) Absolute difference of the rectangles plotted in (d) and (e) (image intensity is scaled).

Because of periodic changes in the background appearance, a multimodal model of the background patch appearance is useful. One such model is a Gaussian mixture model (GMM) [107, 42, 63, 70]:

$$p(\mathbf{y}_g | label_g = background; \theta_g) = \sum_{i=1}^{nmodes} \pi_{gi} \mathcal{N}(\mathbf{z}_g; \mu_{gi}; \Sigma_{gi}), \tag{4.4}$$

for a prespecified number of mixture components, $nmodes$. The computational requirements for a GMM are only slightly more than for a single Gaussian, while the representational power of the GMM is much larger.

An even more general model of the background appearance is based on kernel density estimation (KDE) [38]:

$$p(\mathbf{y}_g | label_g = background; \theta_g) = \frac{1}{nkernels} \sum_{i=1}^{nkernels} \mathcal{N}(\mathbf{z}_g; \mu_{gi}; \Sigma). \tag{4.5}$$

Note that Equation (4.5) differs from (4.4) in that (1) the prior for each component is uniform, (2) the same variance Σ is used for each component, and (3) most importantly, the number of components $nkernels$ is much larger than the number of components $nmodes$. Each component of the GMM is meant to represent a mode of the distribution, while KDE is more like a smoothed version of a histogram density approximation.

Finally, some algorithms incorporate components intended to detect shadows [63, 32]

4.1.3 Updating the Background Model

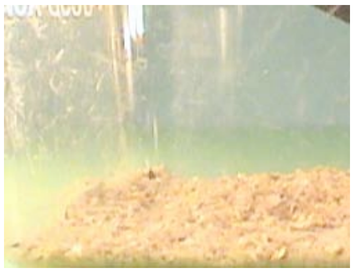
Once we have chosen a parametric model of the background appearance, we must devise a method for updating estimates of its parameters. These methods learn the optimal values of the parameters that fit the sequence of background patch appearances in

$$\mathbf{z}_{window(t)} \triangleq (\mathbf{z}_{t-w,g}, \mathbf{z}_{t-w+1,g}, \mathbf{z}_{t-w+2,g}, ..., \mathbf{z}_{t-1,g}), \tag{4.6}$$

where w is the number of frames in the sliding window of frames fit and t is the current frame.

We desire the following properties of our update algorithm. First, the update method selected must be fast since it will be called often by a real-time algorithm. Second, it must be robust to foreground patches in the data window. Third, it must adapt quickly to changes in the background appearance. Of course, the second and third requirements are often contradictory, since it is difficult to distinguish a foreground patch from a change in the background appearance. Changes in background appearance in the mouse tracking application primarily involve movement of the bedding, for instance when mice walk or dig. In less controlled scenes than the mouse cage, appearance changes can include drastic lighting changes in outdoor scenes and the addition or removal of an object from the scene. See Figures 4.4 and 4.5 for examples.

(a) Lower left corner of the cage in frame 2.


(b) Lower left corner of the cage in frame 2999.

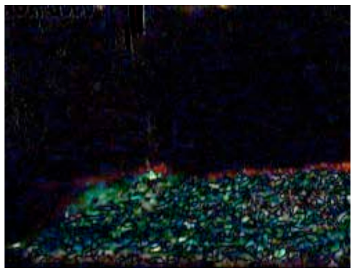

(c) Absolute difference of (a) and (b), multiplied by 2.125 to enhance differences.

Figure 4.4: The background appearance of the mouse cage changes as the bedding is moved around by the mice. The background model must adapt to these changes.

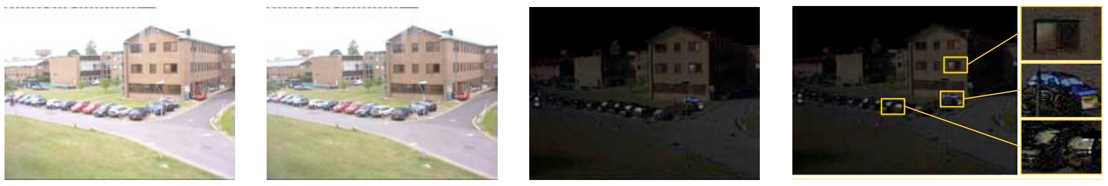

(a) (b) (c) (d)

Figure 4.5: Example video frames of an outdoor scene from [58]. (a) Video frame 1. (b) Video frame 6789 ($\approx$ 4 minutes later). (c) Absolute difference of (a) and (b). (d) Although the camera is still, the background appearance changes. First, as highlighted in the top box, a car that was parked in frame 1 has left the scene in frame 6789. Second, lighting changes have caused slight differences in the background appearance in nearly all locations. Effects of lighting change are most evident in the windows of the cars and building, as highlighted in the middle and bottom boxes.

Computational Efficiency.

Recomputing the optimal parameters at each update requires storing the relevant window of frames and performing a computation that is $O(w)$ in running time. To make the update fast, most background modeling algorithms compute an online estimate of the parameters. For example, if we choose the single Gaussian parametric model (Equation (4.3)), and we choose to estimate the parameter μ_g as the mean of the window of patch values, the online update of the mean is

$$\mu_{tg} = \alpha \mathbf{z}_{t-1,g} + (1-\alpha)\mu_{t-1,g} \tag{4.7}$$

where a large α is analogous to a small window length w.

Robustness versus Adaptiveness.

The background model parameters for location g are estimated from the sequence of patches $\mathbf{z}_{t-w:t-1,g}$. Often, some patches in this sequence will correspond to foreground. Ideally, these foreground patches would be ignored by the estimation algorithm.

One approach to making the parameter estimates robust to foreground patches in the window $\mathbf{z}_{window(t)}$ is to make the window length w large. If foreground patches are a small fraction of the window, then they will not have much effect on the parameter estimates. The minimum necessary length of the window depends on the application. If the targets always move quickly, then at any given time t, one can expect at most a few frames in the window to contain foreground patches. However, if the targets can stay still for long periods of time, as is the case in the mouse tracking application, the window length would have to be huge to ensure robustness. As noted above, increasing the window length w causes the modeling algorithm to adapt more slowly to changes in the background scene.

A second approach is to use robust estimates of the parameters instead of the maximum likelihood estimates. For example, the median patch value of the window $\mathbf{z}_{window(t)}$ (where the median is taken in each dimension independently) is a more robust estimate of μ_{tg} than the sample mean [32]. Assuming that more than half of the patches in the window are background, then the median will be robust to foreground patches. One of the drawbacks of using robust statistics is that they are computationally expensive. For example, to compute the median, one must store a sorted list of all the patches in the window. This prohibits increasing the window length w to accommodate targets that stay still for long periods of time.

A third approach is to try to determine, based on the current parametric model of background (and possibly higher level information), if the newest patch is likely to be background or not. If it is not likely to be background, then the patch is not added to the background model [67]. A number of approaches instead add the patch to a separate model, to aid in classification [42]. In machine learning applications, it is often risky to use data labeled by an inexact classifier as though it were ground truth. Errors in classification will compound, resulting in drift.

A related approach is to add every patch in the window to the background model, but only use some of the background patches to compute the background scores for the current frame, $score_{back}(\mathbf{y}_g)$. [107] uses the GMM parametric model. Every new patch $\mathbf{z}_{tg}$ is used to update the mean and variance of only the single closest mixture component. The score $score_{back}(\mathbf{y}_g)$ is computed using only the highest weight mixture components. Thus, in general there will be some low weight components corresponding to foreground, while the rest correspond to background. Only the background components are used to compute the background score, $score_{back}(\mathbf{y}_g)$.

4.2 Background Modeling for Mouse Tracking

The likelihood that a pixel in the video was produced by the background model is a very useful feature for mouse tracking, as is evident in the example in Figure 4.2. Because we have control over the mouse cage environment, we can make many assumptions about the appearance of the scene background. First, the camera can be assumed to be stationary. Second, we can assume illumination is constant. This makes background modeling for mouse tracking easier than background modeling for outdoor scenes. Third, we can assume that the only movement in the video sequence is caused by the mice. Thus, the periodic motion in our video sequence is much less a problem than in other tracking applications.

One of the main difficulties with applying background modeling techniques to mouse tracking is that the mice can and often do occlude a given pixel of the background for many frames in a row. Consider, for instance, if a mouse goes to sleep. Our background modeling procedure must not let this stationary mouse become part of the background. At the same time, we would like our model to adapt to changes in the scene appearance, as, for instance, the bedding of the cage is often shifted by the mice. A second difficulty with our data is the similarity of mouse color and the color of the bedding, particularly when the bedding is in the shadow of the mouse. A similar difficulty occurs at the ceiling of the cage. Figure 4.6.

Following [57], I chose to represent an image patch of size 11×11 (recall that each video frame is 480×720) by six features – two features per color channel. I chose the

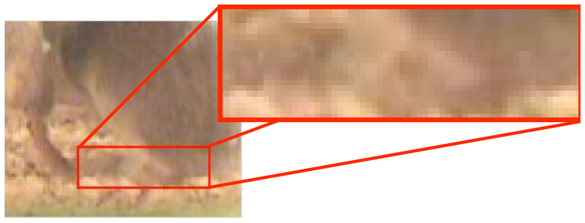


Figure 4.6: Illustration of the similarity in color of the bedding and a mouse.

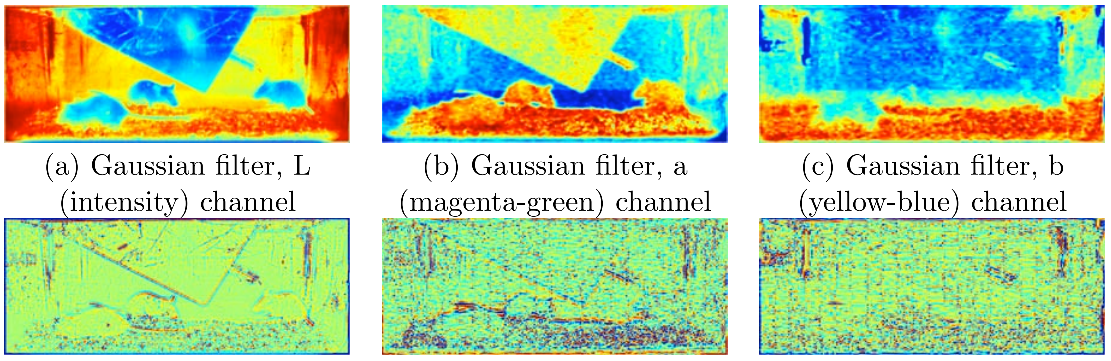

(a) Gaussian filter, L (intensity) channel
(b) Gaussian filter, a (magenta-green) channel
(c) Gaussian filter, b (yellow-blue) channel
(d) LoG filter, L channel
(e) LoG filter, a channel
(b) LoG filter, b channel

Figure 4.7: The six features used to represent each patch. Each pixel of the above images corresponds to a patch.

CIELAB color space representation. The CIELAB color space was constructed using human perception as a model. Small changes in human perception of a color correspond to small changes in the CIELAB color value, and large changes in human perception correspond to large changes in CIELAB color value [123]. While there is no guarantee that the color space used in human perception is a good color space for the specific task of mouse tracking, we do know that humans are able to easily track the mice, thus motivating the choice of the CIELAB color space.

Independently for each color channel, we convolve the image patch with a Gaussian filter (with standard deviation 1.5) to create the first feature. We create the second feature by convolving with a Laplacian of Gaussian (LoG) filter (with standard deviation 1.5). Figure 4.7 shows the six filter representation of a video frame. The LoG filter is useful distinguishing the bedding, ceiling, and lixit (the tube protruding from the water bottle) from the foreground. Figure 4.8 illustrates the decrease in misclassification error for each pixel location with the LoG filters included.

We chose to follow the background modeling algorithm of [107]. The main modi-

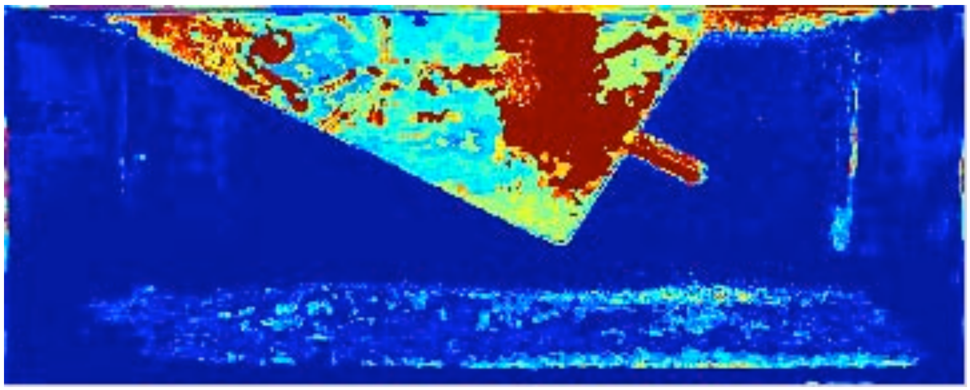


Figure 4.8: Illustration of the decrease in misclassification error when LoG filters are incorporated. A linear classifier was trained independently at each location in the image to distinguish foreground and background patches using linear discriminant analysis [51]. The training data consisted of 100 time-separated hand-labeled frames, created for learning the foreground distribution (see Section 4.4). Foreground and background classes were set to have equal priors. The misclassification error takes into account these priors. This was done for data containing the Gaussian and LoG filters and for data containing just the Gaussian filters. The difference in misclassification rate, thresholded at .1, is plotted. The main areas of improvement are the bedding, scratches at the back right corner of the cage, and the ceiling.

fication we made to this algorithm involved using a very rough, static foreground classifier (see Section 4.4 for details). Pixels that are classified by this foreground classifier as foreground are not used to estimate parameters of the background. The exception to this rule is any pixel location that was always classified as foreground. There were 27 such pixel locations in our video sequence. At these locations, all the data was pooled to learn a foreground classifier. Figure 4.13 shows example results of this rough foreground classifier.

Because a mouse often stays still for many frames in a row, a mouse will sometimes become part of the background without this modification. This modification is resistant to drift because the foreground classifier is never modified. It is reasonable to use a static model because there are no permanent changes in the foreground appearance e.g. due to illumination changes.

As this foreground classifier is very rough, there will be a number of false positives and false negatives. False negatives, i.e. classifying a pixel as background when it is truly foreground, result in parts of the foreground being incorporated into the background model. The method of [107] was designed to be resistant to a reasonable fraction of false negatives, since all foreground pixels are added to the background model in their method. In Figure

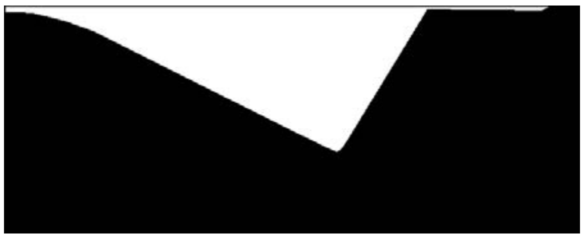


Figure 4.9: Pixel locations that can never contain foreground because of the structure of the cage. These locations are always classified as background by our system. The mask was manually input.

4.1, we see some effects of false negatives where foreground is very evident in the background model.

The main disadvantage of using our rough foreground classifier is false positives, i.e. pixels that are classified as foreground when they are truly background. If all the pixels at a location are classified as foreground, then all pixels are added to the background model, thus there is no disadvantage. There are two possible failure modes. First, if some appearance of the background at a location are classified as foreground and some are classified as background, then only those that are classified as background will be used in the background model. Second, consider a background location always is classified as foreground. If at some point when a mouse is in front of this location true foreground is classified as background, then this background location will always be modeled as foreground. While we did not qualitatively notice failures of these sorts in our results, we have not experimented on a long enough video sequence to be confident. We plan to implement safeguards against these failure modes in future work.

A second modification we made to the background modeling algorithm of [107] was to manually input a mask outlining the feeder. There is not enough space between the feeder and the cage wall for the mice to occlude the feeder, thus these locations can always be classified as background. Figure 4.9 shows the mask of pixel locations always classified as background. This was helpful in speeding up our implementation. As we feel the feeder is very easy to identify automatically, in future implementations this step will be automated.

Besides these modifications, our background modeling algorithm followed [107]. We made no attempt to optimize the parameters of this algorithm. The first set of values we chose worked fairly well. Table 4.1 contains the parameters used in our implementation. Below, we briefly describe the background modeling algorithm of [107]. For full details, please see the original paper.

We compute initial estimates of the background model by learning independent GMMs at each location in every fifth frame of the first 1000 frames (excluding pixels in these frames classified as foreground) using EM for GMMs. The standard EM algorithm was modified to force all covariance matrices to be diagonal and to enforce a minimum variance. The minimum variance in dimension d was set to be one tenth the sample variance of that feature in all locations in 100 time-spaced frames:

$$\sigma^2_{min,d} = \left(\frac{1}{10}\right)\left(\frac{1}{N_{images}N_{pixels}}\right)\left[\sum_{i=1}^{N_{images}}\sum_{g=1}^{N_{pixels}} y^2_{igd} - \left(\sum_{n=1}^{N_{images}}\sum_{g=1}^{N_{pixels}} y_{igd}\right)^2\right] \tag{4.8}$$

where $N_{images} = 100$ is the number of training images, $N_{pixels} = 283 \cdot 720$ is the number of pixels in each image, y_{igd} is the value of feature d in location g in training image i. This came out to

$$\Sigma_{min} = \text{diag}[(30.6608, 4.5499, 10.1498, 0.2496, 0.0261, 0.0483)]. \tag{4.9}$$

Once the background model is initialized, we update the parameters every fifth frame. The update step involves the following steps. First, we apply the rough foreground classifier at each location. No updates are performed at locations classified as foreground. Second, we compute the closest mixture component to each new pixel using the Mahalanobis distance.

If the Mahalanobis distance is at most $\sigma_{max} = 2.5 \cdot \sqrt{6} = 6.1237$, we add the pixel $\mathbf{y}_{new}$ to the closest mixture component i with the following update rules:

$$\mu_i \leftarrow (1-\alpha)\mu_i + \alpha \mathbf{y}_{new} \tag{4.10}$$

$$\Sigma_i \leftarrow (1-\alpha)\Sigma_i + \alpha \text{diag}(\mathbf{y}_{new} - \mu_i)^2 \tag{4.11}$$

$$\pi \leftarrow (1-\alpha)\pi + \alpha \mathbf{e}(i) \tag{4.12}$$

$$\pi \leftarrow \pi / \|\pi\| \tag{4.13}$$

where μ_i is the mean of the closest mixture component, Σ_i is the variance of the closest mixture component, π is the vector of priors for all components, $\mathbf{e}(i)$ is a vector that is 1 for the closest component and 0 for the other components, and $\alpha = 0.01$.

Otherwise, if the Mahalanobis distance to all mixture components is larger than σ_{max}, then we remove the mixture component with the lowest prior and replace it with a mixture component centered at the new pixel $\mathbf{y}_{new}$. The prior of the new mixture component

Table 4.1: Parameters used for estimating the background model and background likelihood for mouse tracking.

Parameter Description	Parameter value
The width and height of an image patch.	11
The color space used to represent an image.	CIELAB [46]
First filter convolved with image patch	Gaussian
Standard deviation of Gaussian filter	1.5
Second filter convolved with image patch	LoG
Standard deviation of LoG	1.5
Number of frames skipped between updates.	4
Number of frames skipped between initialization frames.	4
Number of frames used to initialize.	200
Minimum variance of image patch, where features are represented in the order: (L, Gaussian), (a, Gaussian), (b, Gaussian), (L, LoG), (a, LoG), (b, LoG).	(30.6608, 4.5499, 10.1498, 0.2496, 0.0261, 0.0483)
Maximum Mahalanobis distance for membership to existing component.	6.1237
Update weight.	0.05
Initial prior of new mixture component.	0.01
Initial variance of new mixture component.	(153.3042, 22.7495, 50.7490, 1.2482, 0.1308, 0.2412)
Minimum weight of background mixture components	0.8

is set to $\pi_{init} = .01$. The variance of the new mixture component is set to one half the sample variance of that feature in all locations in 100 time-spaced frames (Equation (4.8)). This came out to:

$$\Sigma_{init} = \text{diag}[(153.3042, 22.7495, 50.7490, 1.2482, 0.1308, 0.2412)]. \tag{4.14}$$

At each new frame, we compute the background likelihood using only some of the mixture components from the un-updated background model. We sort the background mixture components from highest prior to lowest prior, and use the first mixture components whose total weight is at least $min_{weight} = 0.8$. We set the weight of the other mixture component(s) to 0 to compute the background likelihood.

4.3 Foreground Density Tracking

One common model of target appearance involves determining how well image patches within a region match a model of foreground patch appearance. These models take

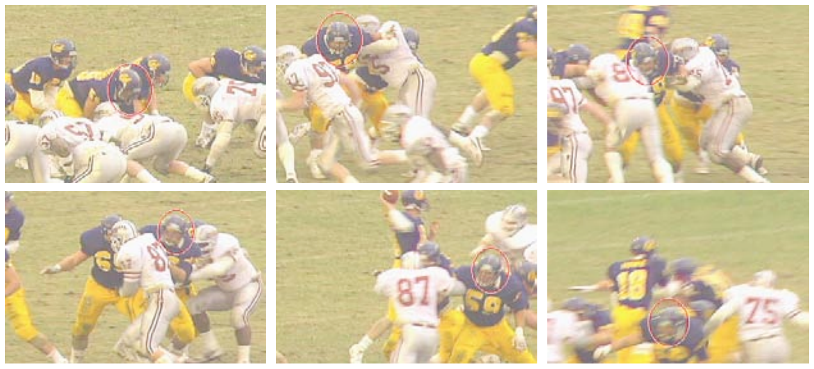


Figure 4.10: Results of tracking a football player through a complicated sequence involving occlusion and complex transformations. The color histogram describing the region tracked is fairly uniform through this sequence. Images taken from [25].

basically the same form as the patch distributions for a single background location, described in Section 4.1. Two types of scoring functions exist for foreground density tracking. We discuss these next.

4.3.1 Kernel-Based Tracking

The first type of scoring function assumes that the distribution of pixels in a foreground region does not change over time. These algorithms are termed kernel-based tracking algorithms [25] because they use kernel density estimation (KDE) to model the foreground density. The score for a hypothesized hidden state is computed from the distance between the a static model of foreground density and the empirical density of the pixels within the target region corresponding to the hypothesized hidden state. This form of tracking is similar to region-based template matching (Section 4.5), except instead of treating an image as a vector, an image is treated as a bag of pixels. That is, the locations of the individual locations of the pixels within the region are ignored. The benefit of this bag of pixels representation is its robustness to projective transformation, partial occlusion, deformation, and a reasonable amount of out-of-plane rotation. Figure 4.10 for an illustration of the power of these kernel-based techniques. Besides its robustness to common deformations during tracking, kernel-based tracking is popular because there are fast algorithms for optimizing the appearance score [24, 50].

4.3.2 Foreground Likelihood Maximization

A second type of scoring function assumes that each pixel in a region is drawn i.i.d. from the same foreground distribution. The score for a hypothesized region is the likelihood of the observed foreground region given a model of foreground density [39, 125, 57, 88]. Because this method of foreground tracking is so similar to background tracking, the two are usually combined.

The foreground density models used in this type of algorithm can be either static or updated online. Static and online foreground density models have the same advantages and disadvantages and static and adaptive background models. While online foreground density models may adapt to changes in the foreground appearance, they are also prone to drift.

4.4 Foreground Density Modeling for Mouse Tracking

The foreground patch appearance for mice is very homogeneous. This means that both offline and online models of foreground density are usually useless in distinguishing one region of foreground from another. Thus, during occlusions, foreground density modeling cannot be used to separate one mouse from another. However, the foreground model is extremely useful for distinguishing foreground from background, much more so than in other applications. We can therefore use foreground modeling, much like background modeling, to fit the shapes of the targets during occlusions such that pixels inside the targets look like foreground and pixels outside the targets do not.

The fast optimization algorithms for kernel-based tracking such as [25, 50] can only estimate the position and scale of the target, not the shape, size, and rotation. Thus, these algorithms cannot be directly used to estimate the positions of the mice from the shape of the foreground mask during occlusions, as in blob tracking algorithms such as [57, 125]. In future work, we plan to explore using kernel-based tracking algorithms for initializing searches such as those used in [57].

As in our background model (Section 4.2), we represent an image patch of size 11×11 by six features – two features for each of three color channels. Again, we use the CIELAB color space and the Gaussian and LoG filters. Figure 4.7 shows the six filter representation of a video frame.

Since the general foreground appearance is static over time, we learn a static model of appearance from hand-labeled training data. We fit an eight-component Gaussian

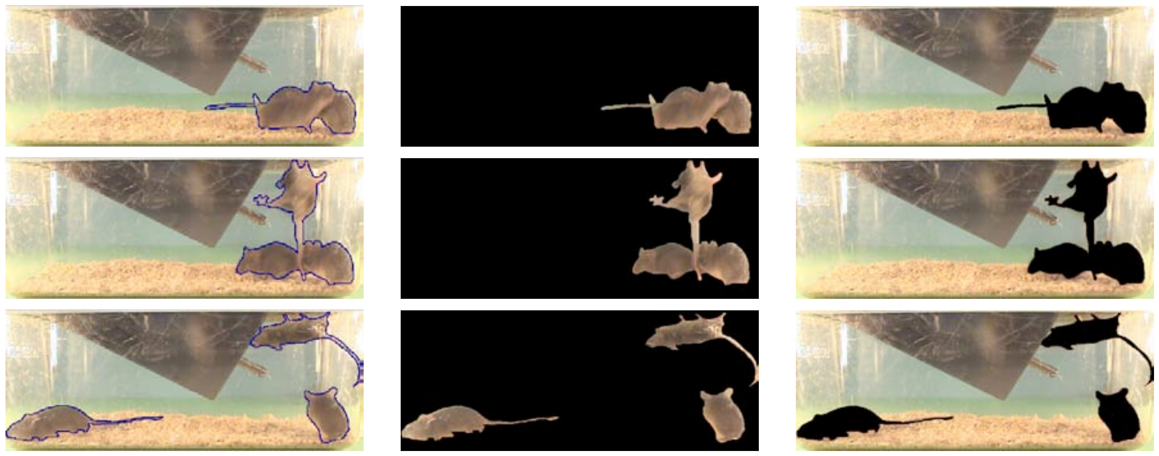
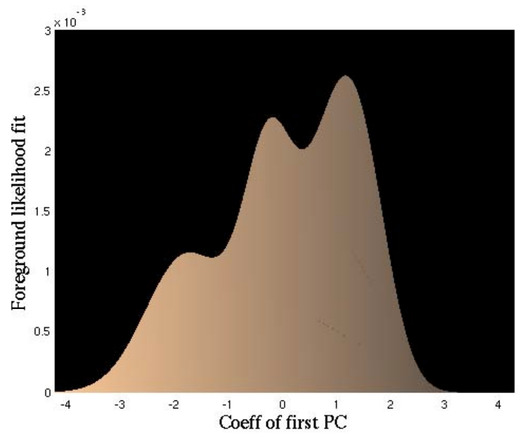

(a) Labeled image (b) Foreground training data (c) Bkgd data excluded.

Figure 4.11: Examples of the labeled data from which the foreground GMM was learned.

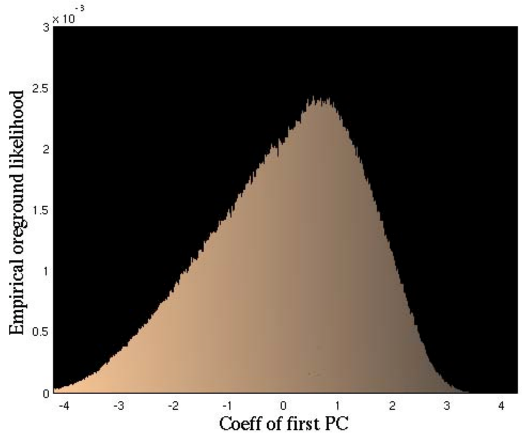
mixture model to the foreground data of 100 labeled frames, spaced equally in time. See Figure 4.11 for example training images. Empirically, we found that a Gaussian mixture model fit the foreground likelihood well. See Figure 4.12 for a visual comparison of the GMM fit and the empirical distribution of the data.

The rough foreground classifier used to filter pixels before updating the background model (see Section 4.2) was created from the foreground GMM. From the 100 training images, we also learn a threshold for the foreground likelihood. If the foreground likelihood of a patch is above the threshold, we classify the patch as foreground. The threshold was chosen to minimize the classification error on the training set. Because the number of background pixels in the training set is so much higher than the number of foreground pixels, we (arbitrarily) set the misclassification cost for foreground to be 5 times the misclassification cost for background. This is equivalent to setting the foreground class prior to 0.2886 and the background class prior to 0.7114. Figure 4.13 shows example results of this rough foreground classifier.

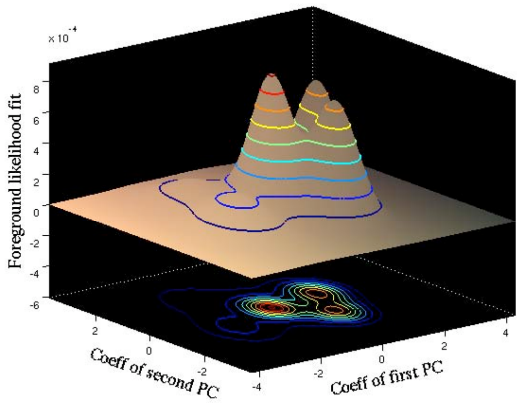
Patches in the video that are classified as foreground based on foreground likelihood alone are not used to estimate or update the background model, with one exception. There were 27 locations in the image, all on the right side of the ceiling of the cage, that always looked like foreground, i.e. the background appearance at these 27 locations had foreground likelihood greater than the learned threshold. At these locations, we ignored the foreground label and estimated the background parameters using all the data.



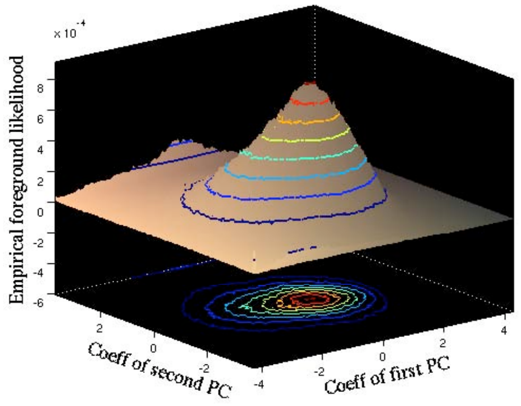
(a)



(b)



(c)



(d)

Figure 4.12: Plot of the foreground likelihood learned from the labeled training data. (a) The projection of the parametric likelihood model onto the first principal component of the foreground data. (b) The projection of the empirical likelihood of the labeled training data on the first principal component. (c) The projection of the parametric likelihood model onto the first two principal components of the foreground data. (d) The projection of the empirical likelihood of the labeled training data onto the first two principal components. The axes of (a) and (b) and the axes of (c) and (d) are the same. We can see that the learned mixture model is close to the empirical distribution. In all plots, the color of the surface plotted is the RGB equivalent of the Gaussian filter features of the patch. Thus, the color of the plot is a visualization of the foreground color.

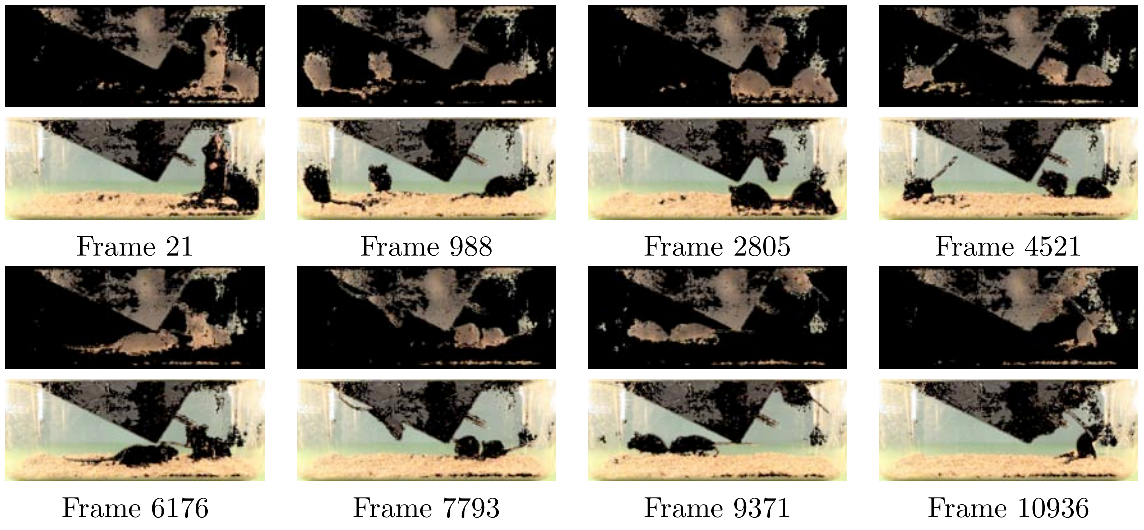


Figure 4.13: Results of the rough foreground classifier on example video frames. The top row shows pixels classified as foreground and masks out pixels classified as background. The bottom row shows pixels classified as background and masks out pixels classified as foreground.

4.5 Region-based Template Matching

A popular approach to tracking is to maintain a template model of the global appearance of the foreground region. In this section, I briefly discuss algorithms that find the transformation of an image texture map that best matches the input image in terms of sum-squared distance or normalized cross-correlation. I do not go into much detail because these techniques do not work very well for mouse tracking. These techniques require the target to have parts with different texture appearances, whereas the image texture of the mouse is uniform.

4.5.1 Static Image Prototype

One of the earliest methods for tracking models the global appearance by a static image prototype. The target state contains the transformation of the image prototype that matches the current video frame. The sum-squared difference or the normalized cross-correlation between the current video frame and the image prototype under the hypothesized transform is used to score the hypothesized transform. In rigid template matching, the state corresponds to a rigid translation (e.g. translation, rotation, scaling) [23]. Influenced by research on active shape models (see Section 3.1.2), larger classes of transformations such as affine transformations and free-vibration modes have been considered [100]. See Figure 4.14 for example transformations of a static image prototype from [100].

The target appearances that can be represented by a static image prototype model are very limited. For instance, this approach cannot model significant out of plane rotation. Thus, it is not directly applicable to mouse tracking.

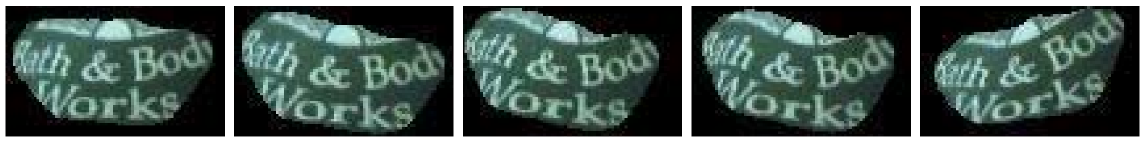


Figure 4.14: Example warps of an Active Blob template image. Images from [100].

4.5.2 Active Appearance Models

For static image prototype tracking to work in non-rigid tracking, all possible target deformations must be approximately representable by the class of image transformations chosen. It is very difficult to manually describe a concise yet complete set of possible deformations for non-rigid targets. Active appearance models [27, 109] attempt to learn the appearances the target can take. Instead of to a single image prototype under a transformation, the image is matched to a space of image textures under a transformation. The space of image textures is the top principal components of registered and aligned training images. Besides a shape transformation, the target state contains the coordinates of a texture in the principal subspace. The score of a hypothesized shape transform and image texture is based on the sum-squared difference or normalized cross-correlation between the current video frame and the hypothesized image texture warped by the hypothesized shape transform. See Figure 4.15 for an illustration of the space of image textures and appearances generated by this model.

Like active shape models (Section 3.1.2), active appearance models assume that the correspondence problem between every possible target appearance is solved – one must be able to identify corresponding points in *every* training image. If one uses a 2-D model of target shape and appearance, then this is impossible if the targets exhibit significant out-of-plane rotation. As discussed in Section 3.1.2, 3-D models of shape are not applicable to targets that deform as much as the mice. Even if one solves the correspondence problem, the features present in the mouse texture are small in size, thus small errors in registration cause the appearance model to fail. Figure 4.16 illustrates this. This example suggests that

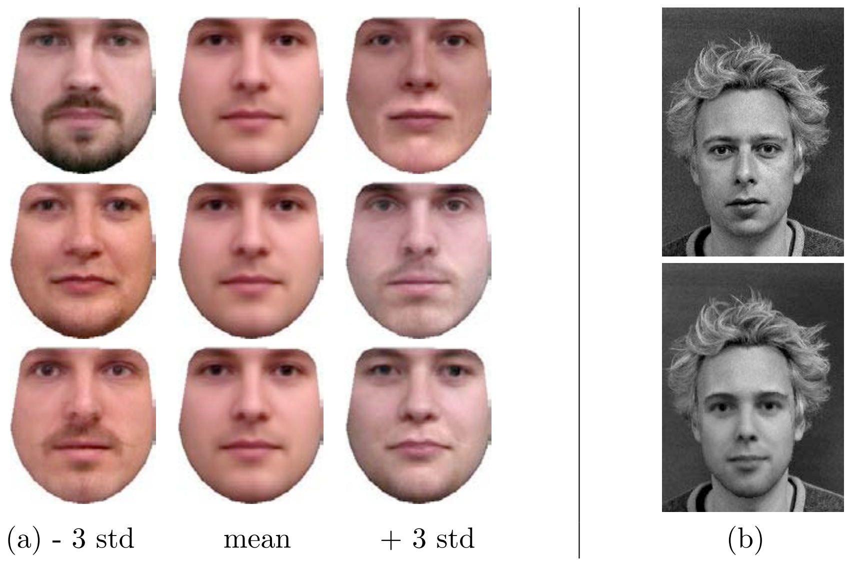


Figure 4.15: (a) Principal components of an active appearance model representing faces. Each row corresponds to a different principal component. The left column is the image texture -3 standard deviations along a principal component, the middle column is the mean image texture, and the right column is the image texture +3 standard deviations along a principal component. Images from [109]. (b) The active appearance model fit to a novel input video frame. The top image is the original input image. The bottom image contains the fit active appearance model overlaid on the original image. Images from [108].

the texture of a mouse is rich enough in features to warrant a complex, static model of texture.

4.5.3 Part-Based Models

To deal with large amounts of shape variation and occluded/missing parts, the above appearance models have been generalized to part-based models [89, 106, 61] (see Section 4.5.3). The target is broken into multiple parts. Each part has an individual image template or active appearance model. See Figure 4.17 for an illustration.

As discussed in Section 4.5.3, it is not clear how to break a mouse into parts, as they are not as articulated as most of the subjects of part-based models.

4.5.4 Online Appearance Models

The above methods use static templates that are learned or set before tracking begins, and never updated. This template must be invariant to any lighting changes or out-of-plane rotation that occurs in the video sequence. Instead of trying to develop an image

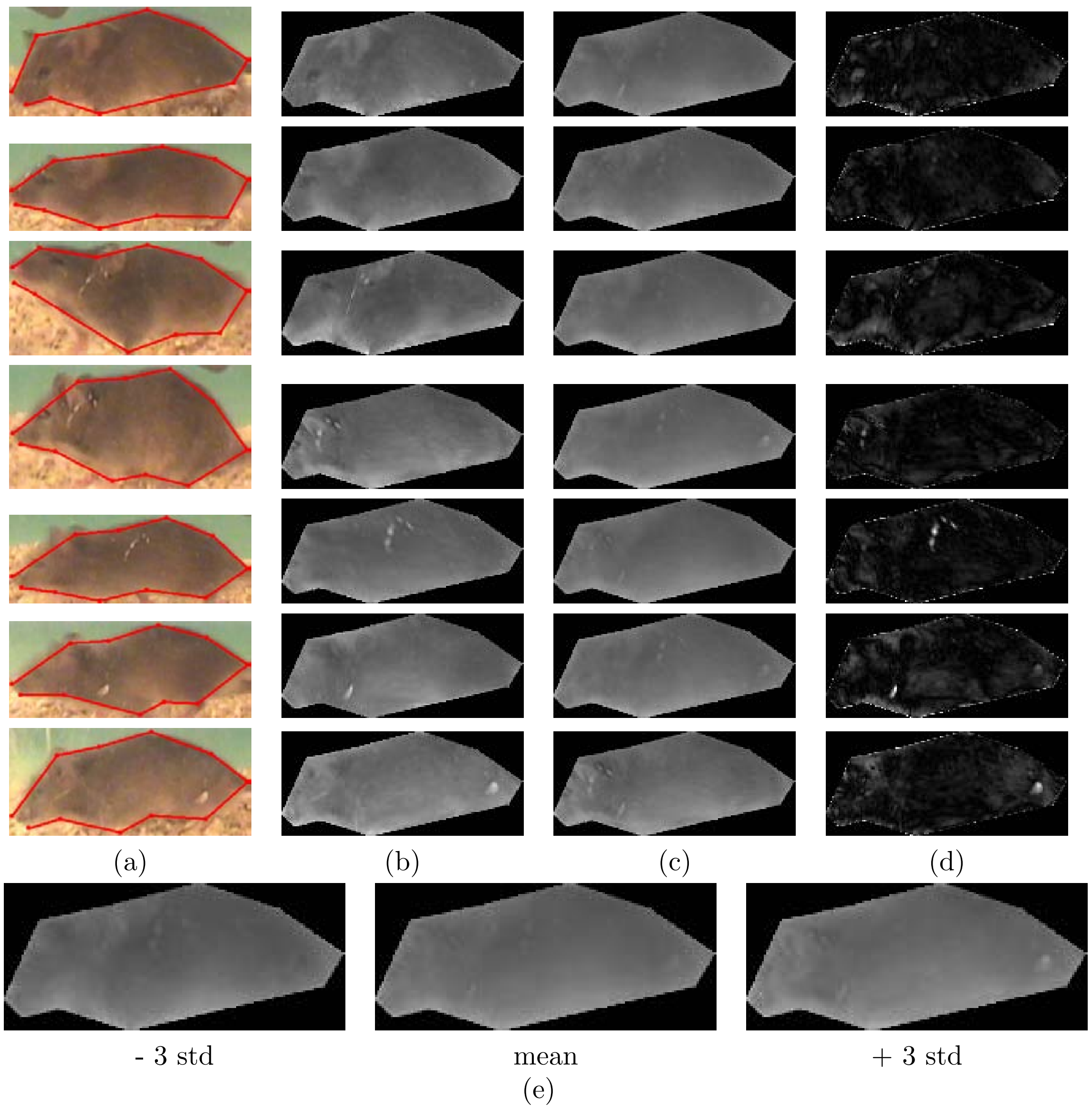


Figure 4.16: Active appearance models for mice. We manually labeled 11 corresponding points in 7 images, each 10 frames apart. This corresponds to about 2 seconds of video in which the amount of out-of-plane rotation and deformation is small enough that correspondences could be defined. Using leave-one-out cross validation, we illustrate the performance of the learned active appearance model from this training data set. (a) Manually entered correspondences. (b) Training images warped so that correspondence points match locations in first training image. (c) Projection of each training image on the 5 dimensional principal subspace computed from the other 6 training images. (d) Absolute difference between (b) and (c). (e) First principal component of active appearance model learned. We see that the projections in (c) looks basically like the mean image in (e), and most of the texture is lost, even in this short sequence. The texture that is captured is due mainly to lighting changes. The hair of these mice is blonde at the ends and brown at the roots, thus it is lighter when seen at an angle. In future work, we plan to investigate if this type of texture information can benefit tracking through occlusions.

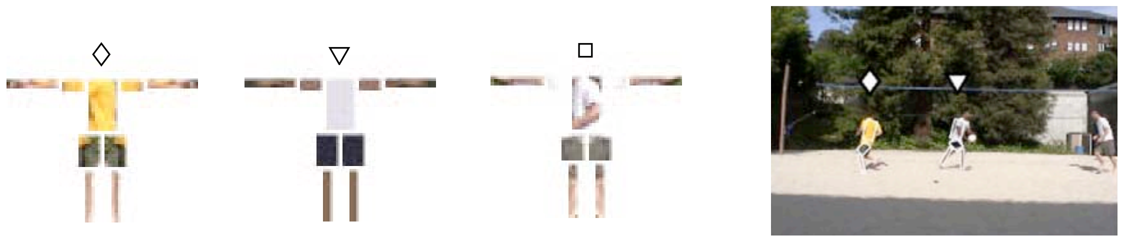


Figure 4.17: (a) Examples of part-based image templates. (b) These part-based image templates fit to a video frame. Images from [90].

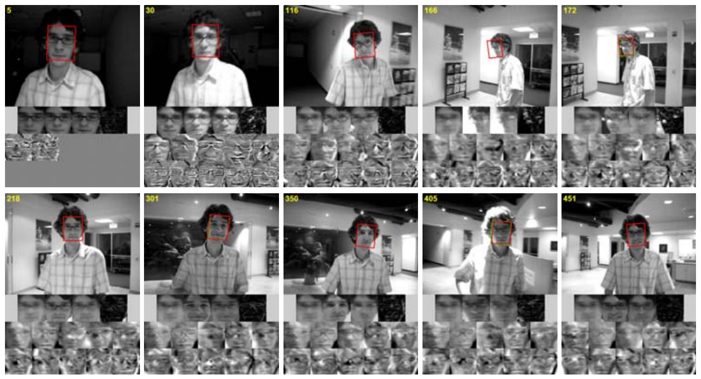


Figure 4.18: Example of the updates to the eigenspace template from [72]. The large image shows face tracking results in a video sequence with lighting changes and out-of-plane rotation. The second row shows the current sample mean, the part of the video frame matched to the template, the closest image in the appearance manifold, and the error of the match. The last two rows show images from the top 10 principal components. Image taken in full from [72].

texture that applies in all situations, many approaches update the image texture online. That is, after every video frame, the part of the video frame that matched the previous template is incorporated into the new template. For example, [9] replaces the previous template with the part of the current video frame matched. [72] and [71] add the matched part of the video to the image manifold representing the target. See Figure 4.18 for an example of the changes to the appearance template over time.

We tried an approach similar to [9] for mouse tracking. We represented the hidden state of a mouse by an ellipse. Given the position of the ellipse in the first frame, we searched for the affine transformation of the image region within the ellipse that resulted in the largest normalized cross-correlation with the second image. The hidden state in the second frame is then the affine transformation applied to the hidden state in the first frame.

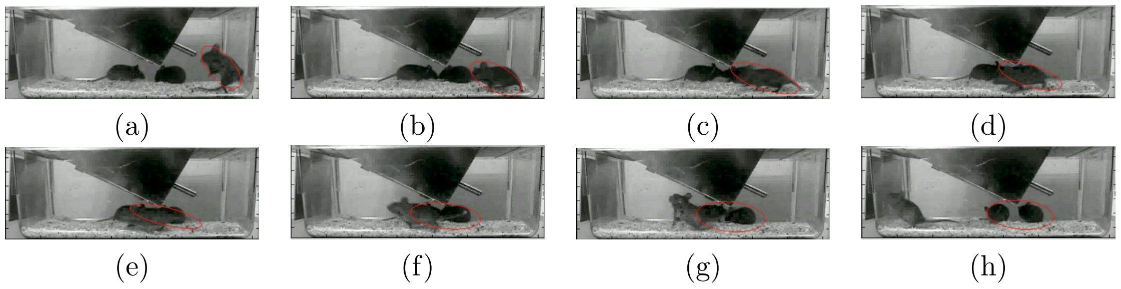


Figure 4.19: Selected frames of an online tracking algorithm applied to mouse tracking. The position and therefore the texture of the mouse can change quickly. As the algorithm implemented uses a local search method to find the best fit, even from frame (a) to frame (b) the position of the mouse is not fit well. The front mouse is completely lost during the occlusion. This could either be because there are not enough features available, because of the local search procedure, or both.

We tried to track the front mouse in this way. This method failed during the first occlusion. See Figure 4.19 for an illustration of the results.

From discussion with the authors of [72] and [71], we determined that the appearance of a mouse changed too rapidly for tracking to be aided by the use of more complicated appearance manifolds.

4.6 Feature Point-Based Tracking

When few assumptions can be made about the shape or appearance of the target, feature point-based tracking is popular. Instead of matching a template describing the global appearance of the target, or even the appearance of a few specific object parts, feature point-based tracking matches small, selected windows around feature points in consecutive frames. These feature points are detected automatically using criteria that describe how trackable the features are. For example, the criterion used in [105] is the minimum eigenvalue of the 2×2 gradient matrix, a quantity directly related to the effectiveness of the tracking algorithm employed in [105]. Other criteria include cornerness and other interest descriptors [97]. These features are tracked from the first frame to the second by finding the motion parameters (e.g. translation or affine) that lead to the best match in the second frame in terms of sum-squared distance.

4.7 Feature Point-Based Mouse Tracking

We experimented with [105] on the mouse video. We used the implementation or KLT available at [14]. All parameters were set to their default values. In our experiments, we tracked 150 features in each frame. If a feature was deemed to be lost, we detected a new feature to replace it. In Figures 4.20 and 4.21, we plot some of the features tracked. We plot only those features that move more than four pixels within their lifetimes.

There are three occlusions in this sequence. Let's interpret the results frame by frame with respect to these occlusions. In frames 1-7, no features are tracked on mouse 3 as it occludes mouse 2, most likely because of motion blur. However, the heads of mice 1 and 3 are (partially) tracked as mouse 3 occludes mouse 1 in frames 13-30. In frames 65-77, no features are reliably tracked on mice 1 and 2 as the mouse 2 occludes mouse 1. In frames 83-88, a point at the intersection of two mice is tracked. This is not useful for inferring the position of either mouse. From frames 83-100, some some features on mice 1 and 2 are successfully tracked.

Thus, useful signal for tracking through occlusions was available in one of the three occlusions. In addition, the results are extremely noisy. Finally, points that were successfully tracked were usually only tracked for a few frames. This is because most of the features were on the boundary of a mouse. Features on the boundary are often not long-lasting because of self-occlusion. Our conclusion was that KLT was not well-suited to the mouse tracking application because of the lack of distinguishing texture in the interior of the mouse.

However, this investigation into KLT led us to try a number of more successful appearance features. Since points on the boundary of the mouse were somewhat successfully tracked by KLT, we experimented with edge tracking algorithms, as discussed in Section 4.9. Because there was not enough texture in the interior of a mouse to track using small feature windows, we experimented with affine flow, which pools texture information in the interior of an entire mouse (see Section 4.8.3).

4.8 Optical Flow

Instead of selecting only a sparse set of good features to track, optical flow algorithms approximately track every pixel in the image. This is similar in goal to the online region-based correlation approaches described in Section 4.5, e.g. [9]. However, since searching for the closest match to every point in the first frame is prohibitive, optical flow

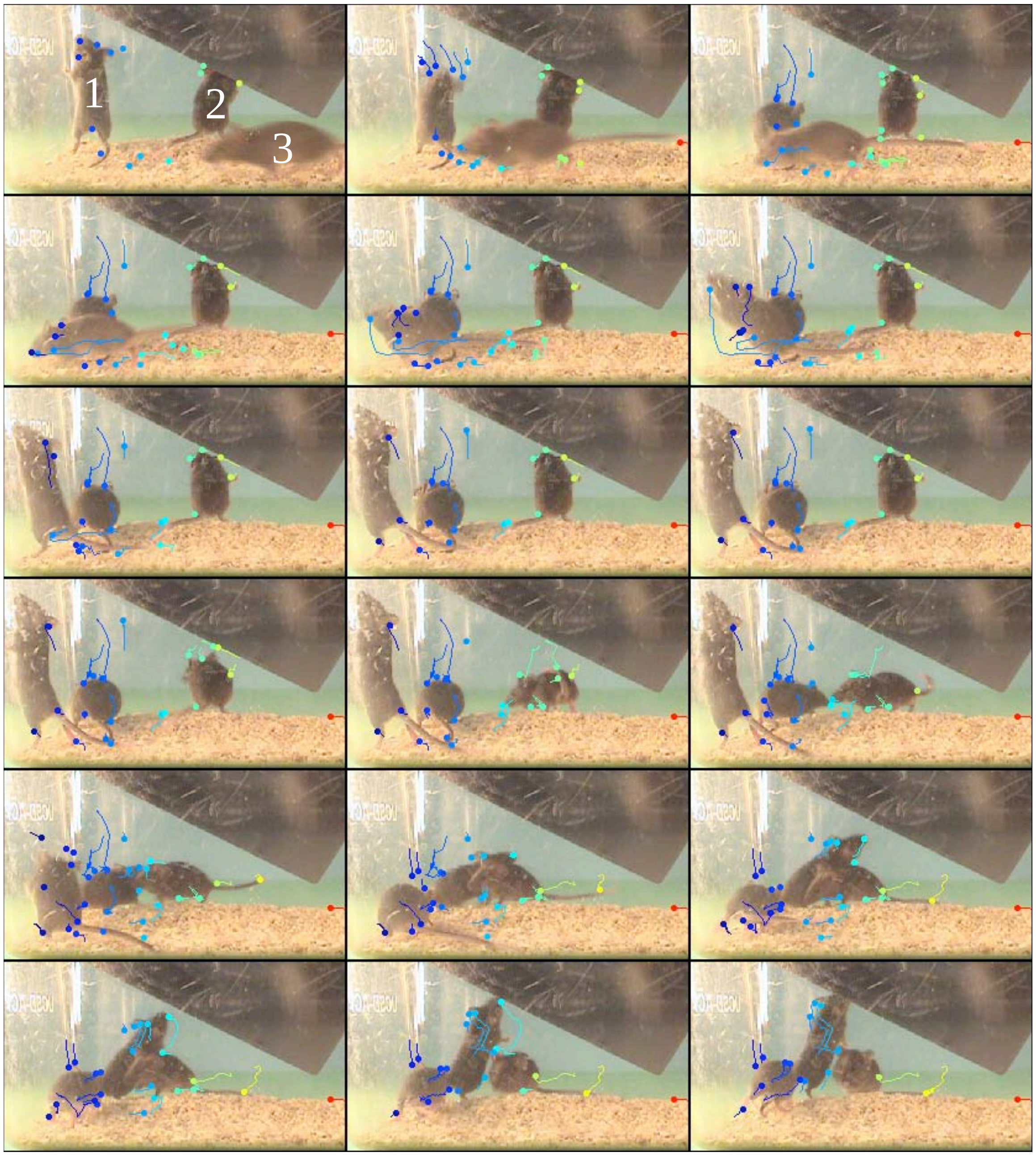


Figure 4.20: Results of running the KLT tracker [105] on the mouse tracking data. Here, we plot 18 equally-spaced frames of the 100 frame test sequence (frame numbers are shown for each row at the right of the image). So that we can reference different mice, the mice are numbered in the first frame. In a given frame, we plot the tracks of some detected features over time. The end of the track corresponds to the current location of the feature, while the start of the track corresponds to the location in which the feature was first detected. The location of the feature in the current frame is indicated by a circle. Thus, we see chains of feature detections following mouse 1 as its head drops.

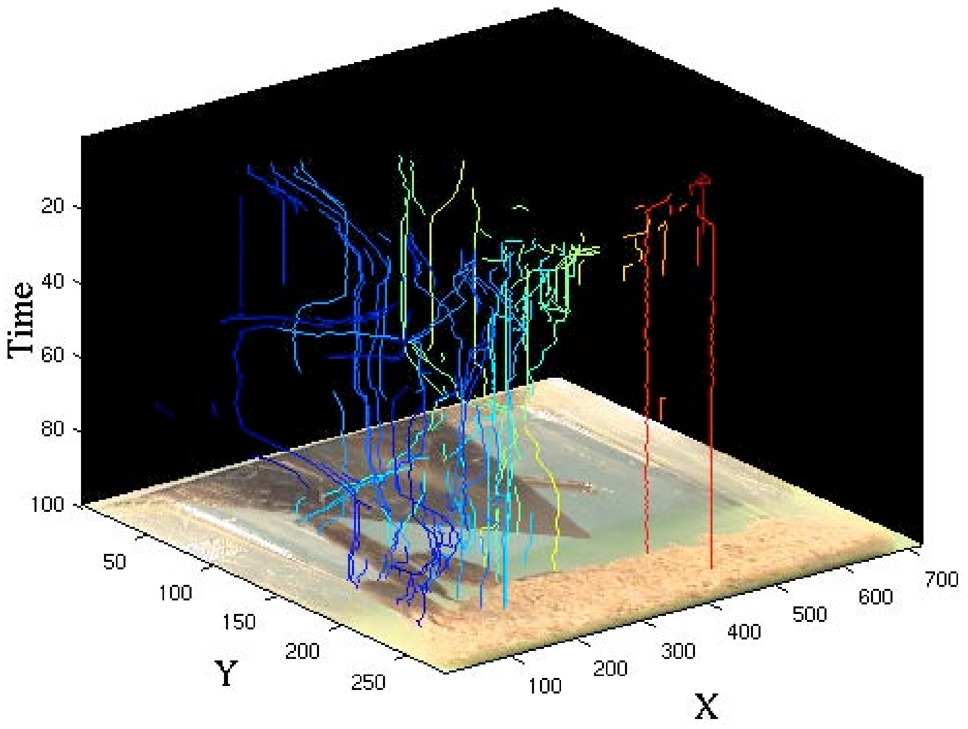


Figure 4.21: All the tracks detected in the sequence illustrated in Figure 4.20 over time. The z-axis corresponds to time. For comparison, tracks are plotted with the same colors in Figure 4.20 as in this figure.

optimizes a first-order approximation to the brightness constancy assumption. This is the assumption that every pixel in the first frame I_1 is present in the next video frame I_2, but may have translated:

$$\forall \text{ pixel locations } \mathbf{p},\ \exists \text{ translation } \Delta\mathbf{p} \text{ such that } I_1(\mathbf{p}) = I_2(\mathbf{p} + \Delta\mathbf{p}) \tag{4.15}$$

The brightness constancy assumption does not hold for most image sequences. It is violated when parts of the scene are occluded or unoccluded. For instance, as a target translates across a scene, part of the scene at the front of the target is occluded while part of the scene at the back of the target is unoccluded. It is also violated if lighting conditions change, for instance if the target moves from a well-lighted area to a less lighted area. Nonetheless, optical flow has been extremely successful in practice.

The Taylor series first-order approximation to the brightness constancy assumption at location $\mathbf{p}$ is:

$$\frac{\partial I_1(\mathbf{p})}{\partial x} u + \frac{\partial I_1(\mathbf{p})}{\partial y} v + (I_2(\mathbf{p}) - I_1(\mathbf{p})) = 0. \tag{4.16}$$

where u and v are the x- and y-components of the translation $\mathbf{p}$, respectively. This approximation is reasonable if the translation $\Delta\mathbf{p} = (u, v)^\top$ is small.

The brightness constancy assumption does not fully constrain the translation $(u, v)^\top$ for each pixel location $\mathbf{p}$, as there are two unknowns in Equation (4.16). The solution is to constrain (usually overconstrain) the system by assuming the optical flow field is spatially coherent. There are two common types of spatial coherence. Both types are incorrect at motion boundaries, thus the optical flow estimates will be incorrect at these locations.

4.8.1 Horn-Schunck Flow

The first method is to add a regularization term to encourage spatial coherence:

$$J(\mathbf{u}, \mathbf{v}) = \sum_{\mathbf{p}} \left[error\left(I_{1x}(\mathbf{p}) u + I_{1y}(\mathbf{p}) v + (I_2(\mathbf{p}) - I_1(\mathbf{p})) \right) + \lambda J_{smooth}(\mathbf{u}, \mathbf{v}) \right] \tag{4.17}$$

where $\mathbf{u}$ and $\mathbf{v}$ are the vectors of flow at every location, the sum is over all locations in the image, $I_{1x}(\mathbf{p})$ and $I_{1y}(\mathbf{p})$ are discrete approximations of the x- and y-gradients of the first image at location $\mathbf{p}$, respectively, $error(\cdot)$ is any error function, $J_{smooth}(\cdot)$ is a smoothness criterion, and λ is the weight of the smoothness term relative to the error term. The error term proposed in [53] is the squared-error:

$$error_{sq}(z) = z^2. \tag{4.18}$$

The smoothness term proposed in [53] penalizes the first-order gradient:

$$J_{smooth}(\mathbf{u}, \mathbf{v}) = \sum_{\mathbf{p}} \frac{1}{2|\mathcal{N}_{\mathbf{p}}|} \sum_{\mathbf{q} \in \mathcal{N}_{\mathbf{p}}} \left[(u_{\mathbf{p}} - u_{\mathbf{q}})^2 + (v_{\mathbf{p}} - v_{\mathbf{q}})^2 \right] \tag{4.19}$$

where again the first sum is over all locations $\mathbf{p}$ in the image and $\mathcal{N}_{\mathbf{p}}$ is the set of pixel locations neighboring $\mathbf{p}$ to the north, south, east, and west. Putting together Equations (4.17–4.19), the optical flow criterion proposed in [53] is

$$\begin{aligned} J_{HS}(\mathbf{u}, \mathbf{v}) = \sum_{\mathbf{p}} & \left[(I_{1x}(\mathbf{p})u + I_{1y}(\mathbf{p})v + (I_2(\mathbf{p}) - I_1(\mathbf{p})))^2 \right] \\ & + \lambda \sum_{\mathbf{p}} \left[\frac{1}{2|\mathcal{N}_{\mathbf{p}}|} \sum_{\mathbf{q} \in \mathcal{N}_{\mathbf{p}}} \left((u_{\mathbf{p}} - u_{\mathbf{q}})^2 + (v_{\mathbf{p}} - v_{\mathbf{q}})^2 \right) \right] \end{aligned} \tag{4.20}$$

The heuristic of penalizing the first-order gradient of the flow may cause errors wherever the true motion in a neighborhood is not uniform. This will occur in general occur even within moving objects. However, it causes the largest errors at the boundaries of moving objects, where the true difference in flow is large. Estimation errors at motion boundaries can also be propagated to other locations.

4.8.2 Lucas-Kanade Flow

The second method of incorporating spatial coherence into the brightness constancy assumption is to assume that the motion within a small region $\mathcal{R}$ of the image is uniform [76]. Then, the goal is to find the single motion vector $(u, v)^\top$ that minimizes the error of Equation (4.16) summed over the entire region $\mathcal{R}$:

$$J(u, v) \triangleq \sum_{\mathbf{p} \in \mathcal{R}} error\left(I_{1x}(\mathbf{p})u + I_{1y}(\mathbf{p})v + (I_2(\mathbf{p}) - I_1(\mathbf{p})) \right) \tag{4.21}$$

where $error(\cdot)$ is the chosen error function. In [76], the squared error defined in Equation (4.18) is used. The optical flow criterion proposed in [76] is

$$J_{LK}(u,v) = \sum_{\mathbf{p}\in\mathcal{R}} \left(I_{1x}(\mathbf{p})u + I_{1y}(\mathbf{p})v + (I_2(\mathbf{p}) - I_1(\mathbf{p}))\right)^2 , \tag{4.22}$$

One advantage is that the optimal solution can be found in closed form:

$$\begin{pmatrix} u \\ v \end{pmatrix} = -(\mathbf{M}_{LK}^\top \mathbf{M}_{LK})^{-1}\mathbf{M}_{LK}^\top \begin{pmatrix} \sum_{\mathbf{p}} I_x(\mathbf{p})(I_2(\mathbf{p}) - I_1(\mathbf{p})) \\ \sum_{\mathbf{p}} I_y(\mathbf{p})(I_2(\mathbf{p}) - I_1(\mathbf{p})) \end{pmatrix} \tag{4.23}$$

where

$$\mathbf{M}_{LK} = \begin{pmatrix} \sum_{\mathbf{p}} I_x^2(\mathbf{p}) & \sum_{\mathbf{p}} I_x(\mathbf{p})I_y(\mathbf{p}) \\ \sum_{\mathbf{p}} I_x(\mathbf{p})I_y(\mathbf{p}) & \sum_{\mathbf{p}} I_y(\mathbf{p})^2 \end{pmatrix}. \tag{4.24}$$

To overconstrain the system, the number of pixels in region $\mathcal{R}$ must be large enough to make the matrix $\mathbf{M}_{LK}^\top \mathbf{M}_{LK}$ well-conditioned. This is called the aperture problem, and will occur if the region is uniform in intensity or has a non-zero gradient in only one direction. If the gradient in one of the direction of the patch is due mainly to noise or other features that do not obey the brightness constancy assumption, then the computed optical flow may be arbitrarily incorrect. In addition, the estimate will be incorrect if there are multiple motions in the region, violating the spatial coherence assumption.

4.8.3 Affine Flow

To be applicable on larger regions more likely to contain enough signal to constrain the problem, the Lucas-Kanade optical flow criterion can be extended to more complex parametric motion models. For example, one could model the motion as an affine transformation:

$$\begin{pmatrix} u_{\mathbf{p}} \\ v_{\mathbf{p}} \end{pmatrix} = \begin{pmatrix} a_1 \\ a_4 \end{pmatrix} + \begin{pmatrix} a_2 & a_3 \\ a_5 & a_6 \end{pmatrix} \mathbf{p}, \tag{4.25}$$

where $\mathbf{a} = (a_1, ..., a_6)^\top$ are the parameters of the affine transformation. Affine flow finds the parameters $\mathbf{a}$ that best satisfy Equation (4.21):

$$\begin{aligned} J(\mathbf{a}) &= \sum_{\mathbf{p}\in\mathcal{R}} error\left(I_{1x}(\mathbf{p})u_\mathbf{p} + I_{1y}(\mathbf{p})v_\mathbf{p} + (I_2(\mathbf{p}) - I_1(\mathbf{p}))\right) \\ &= \sum_{\mathbf{p}\in\mathcal{R}} error\left(I_{1x}(\mathbf{p})(a_1 + (a_2 \quad a_3)\mathbf{p}) + I_{1y}(\mathbf{p})(a_4 + (a_5 \quad a_6)\mathbf{p}) + (I_2(\mathbf{p}) - I_1(\mathbf{p}))\right) \\ &= \sum_{\mathbf{p}\in\mathcal{R}} error\left(\mathbf{z}_\mathbf{p}^\top \mathbf{a} + (I_2(\mathbf{p}) - I_1(\mathbf{p}))\right) \end{aligned} \tag{4.26}$$

where

$$\mathbf{z}_\mathbf{p} \triangleq (I_{1x}(\mathbf{p}), I_{1x}(\mathbf{p})x, I_{1x}(\mathbf{p})y, I_{1y}(\mathbf{p}), I_{1y}(\mathbf{p})p_x, I_{1y}(\mathbf{p})p_y)^\top \tag{4.27}$$

and $\mathbf{p} = (p_x, p_y)^\top$.

If the squared error is used, then this can be written as

$$J(\mathbf{a}) = \sum_{\mathbf{p}\in\mathcal{R}} \left[\mathbf{z}_\mathbf{p}^\top \mathbf{a} + (I_2(\mathbf{p}) - I_1(\mathbf{p}))\right]^2 \tag{4.28}$$

Since the affine transformation is linear, the optimal parameters $\mathbf{a}$ can again be found in closed form:

$$\mathbf{a} = -(\mathbf{Z}^\top \mathbf{Z})^{-1} \mathbf{Z} \mathbf{I}_t \tag{4.29}$$

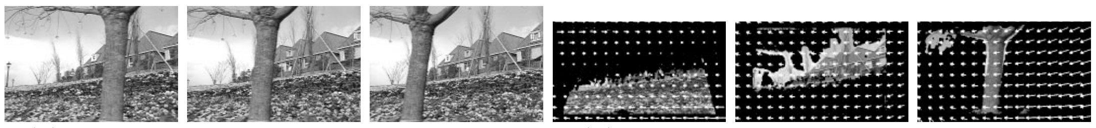
where $\mathbf{Z}$ is the matrix whose columns are $\mathbf{z}_\mathbf{p}$ and $\mathbf{I}_t$ is the column vector whose entries are $I_2(\mathbf{p}) - I_1(\mathbf{p})$.

4.8.4 Robust Flow

The spatial coherence constraints of Equations (4.17) and (4.26) are not always correct, [15] suggests using robust error functions $error(\cdot)$ instead of the squared error. Their experiments estimate the affine flow using the Geman-McClure norm with parameter σ:

$$error(z; \sigma) = \frac{z^2}{\sigma + z^2} \tag{4.30}$$

Their estimation algorithm is an iterative algorithm that using the simultaneous over-relaxation technique.



(a) Frames 0, 15, and 30 of the "flower garden" sequence.

(b) Three motion segments computed.

Figure 4.22: Example of segmentation of a video sequence into different objects. (a) shows example images from the video sequence in which the camera moves. Because the tree, flower bed, and house are at different depths, their image motions are distinct. (b) shows the three motion segments computed. The first motion segment corresponds to the flower bed, the second to the sky and house, the third to the tree. The affine motion estimated for each segment is also plotted in (b). Images taken from [120].

4.8.5 Motion Segmentation

A number of approaches, some of the earliest being [1, 33], use affine flow to segment a short sequence of images into different objects, the hypothesis being that one object has a single motion and different objects have different motions. See Figure 4.22 for an example. These techniques are called *layer-based* because they model an image in the sequence as the superposition of 2-D layers, each layer representing an object.

[1, 120] describe a k-means-based iterative method for segmenting an image into regions, each of whose individual motion is well-described by an affine transformation. At each iteration, from the current affine motion parameters of the segments, the support of the segments is estimated. Then, from the support of the segments, the affine motion parameters are re-estimated. [33] describes a similar technique for segmenting an image into regions whose motion is well described by individual translations using the more robust truncated squared error instead of simply squared error. [55] presents an algorithm that sequentially estimates the motion and support of the object with the dominant motion, removes the pixels following this motion, and continues to the next object. [7] formalizes an optimization criterion and presents an EM-based alternative to these algorithms for different motion models.

The above motion segmentation algorithms only take into account the motion of a pixel, not the location of the pixel. [121, 3, 81, 104] present methods of incorporating spatial coherence into the segmentation criterion.

[111, 112] describe their region-based algorithm in the motion segmentation framework. At every frame [111], simultaneously estimates the positions of the targets and assigns membership of every pixel location to a target.

In more detail, $\mathbf{x}_{tk} = (x_{tk}, y_{tk}, \theta_{tk}, \mathbf{A}_{tk}, \dot{x}_{tk}, \dot{y}_{tk}, \dot{\theta}_{tk})^\top$, the hidden state at frame t for object k, contains the following properties: the target center (x_{tk}, y_{tk}), the rotation of the target θ_{tk}, the target center velocity $(\dot{x}_{tk}, \dot{y}_{tk})$, the rotational velocity $\dot{\theta}_{tk}$, and the appearance, represented by a texture map $\mathbf{A}_{tk}$. To enforce the constraint that each segment's shape, appearance, and motion change gradually, [111] places a prior on the state at time t given the state at time $t-1$. The prior distribution on the state of target k at time t is Gaussian around the state of target k at time $t-1$, with fixed, diagonal covariance. Soft pixel memberships to targets are assigned based on the distance between the appearance of the pixel and the appearance hypothesized for the pixel by the hidden state. The MAP state is estimated through a generalized EM algorithm which iteratively assigns pixel memberships then updates some of the hidden state parameters.

The Pfinder algorithm [125] is similar in many respects to [111]. Pfinder represents a target by the target center position and position covariance, the target velocity, and its appearance. In addition, Pfinder also uses an iterative EM-like algorithm to simultaneously estimate the target's state and the pixel memberships to targets. One of the main differences is that the Pfinder appearance model is the mean color and covariance of the color of the target, instead of a texture map. The two algorithms also differ in the iterative update steps.

Affine flow tracking combined with motion segmentation proved effective at tracking mice through occlusions [18]. Our algorithm used acausal inference to successfully track mice through occlusions. Because the inference scheme is vital to the success of this algorithm, we discuss it in Section 7.

4.9 Edge-Based Tracking

All the previously discussed features focus on the appearance of features *inside* the target. A very popular approach in tracking is to instead score a hypothesized hidden state by computing the distance between the predicted boundary of the target and edges detected in the image [78, 16]. Edge-based approaches have the advantage of being robust to many types of appearance changes. For example, we can fit a model of a person boundary to different people wearing different clothes. Edge-based approaches are also invariant to illumination changes. An additional advantage of edge-based approaches is that appearance scores can be computed quickly. Instead of needing to examine every pixel within the target, one only needs to examine pixels on the boundary.

Given the hypothesized target boundary, there are a few common methods for computing how well this boundary matches edges in the image. Perhaps the simplest matching score is the "edginess" of the pixels directly under the boundary [64]. More specifically, we compute some edge score, for instance the gradient magnitude of the image at the locations of pixels in the target boundary, and sum these edge scores. If the edges are blurred, then the edge score will increase the closer a pixel is to the true location of the edge.

A similar approach is to directly compute the minimum distance from a predicted boundary pixel location to any edge location. The average distance for each predicted boundary pixel location is called the Chamfer distance [45]. Since edges corresponding to a boundary may not be detected, a more robust measure is the Hausdorff distance [45], in which the maximum distance is thresholded.

These approaches do not take into account the correlation between the predicted direction of the boundary at a location and the direction of edges detected in the image. One method for taking this correlation into account is to compute edges near the predicted boundary pixel from image gradient in the direction normal to the boundary at that location [16].

4.9.1 Contour Likelihood Model

[78] describes a generative model of edges detected in the image given the boundary of the target. The advantage of this generative model is that it makes use of more information in the image. While the Chamfer score only uses the closest edge detected, this method uses all edges detected. This work focuses on a measurement line model of the boundary. Instead of modeling the probability of the entire edge image, this work models the distribution of detected edges along line segments normal to the boundary at a few discrete points. Figure 4.23 illustrates these measurement lines.

Let $\{\mathbf{p}_1, ..., \mathbf{p}_m\}$ be the locations of the m measurement lines in the image. Let $\{\mathbf{n}_1, ...\mathbf{n}_m\}$ be the unit normals to the predicted boundary at locations $\{\mathbf{p}_1, ..., \mathbf{p}_m\}$. Let L be the length of each measurement line. [78] constructs the following generative model of the edge detection image. For each location $\mathbf{p}$ on the boundary, there is some probability q_{01} that the edge corresponding to the boundary is not detected. If it is detected, the location of the detection is chosen randomly from a Gaussian distribution on the line normal to the boundary centered at the boundary location $\mathbf{p}$. There will also be spurious edge detections, caused by clutter in the background and features in the target. [78] assumes that the

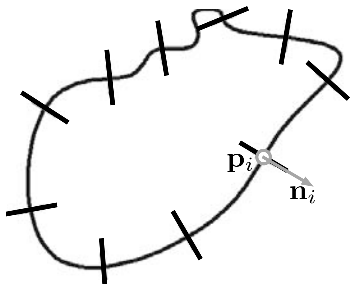


Figure 4.23: Illustration of measurement lines on a mouse contour. The number and locations of edge detections along a measurement line are modeled. $\mathbf{p}_i$ is the center of the measurement line and $\mathbf{n}_i$ is the unit normal of the contour at the measurement line location.

distributions of clutter in the foreground and background are the same and fixed. Let $b_L(n)$ be the probability of detecting n edges from clutter. Finally, assume that detections from clutter are distributed uniformly along the measurement line. Let $\mathbf{z}_i = (z_{i1}, ..., z_{in_i})$ be the 1-D locations of detected edges along the measurement line where the origin is at the measurement line center. Given these assumptions, we can evaluate the likelihood of detecting $\mathbf{z}_i$ as

$$p(\mathbf{z}_i|\mathbf{x}) = \sum_{j=1}^{n_i}(1-q_{01})\left(\frac{1}{L^{n_i-1}}\right)b(n_i-1)\mathcal{N}(z_{ij};0,\sigma^2) + q_{01}\left(\frac{1}{L^n}\right)b(n_i). \tag{4.31}$$

Term j of the sum corresponds to the probability that detection z_{ij} was produced by the edge and all the other detections were produced by clutter. The last term in the equation corresponds to the probability that all the detections were produced by clutter. [78] assumes the measurement locations of edges detected along measurement lines are conditionally independent, thus $p(\mathbf{z}_{1:m}|\mathbf{x}) = \prod_{i=1}^{m} p(\mathbf{z}_i|\mathbf{x})$. Generalizations and perturbations of this model can be found in [78].

One of the main difficulties with this approach is setting the parameters. The parameters must be set to account for inaccuracies in the target shape model and the shortcomings of the inference algorithm. It is difficult to quantify these inaccuracies using intuition and domain knowledge. Since we multiply the measurement line likelihoods, if even one of these measurement line likelihoods is close to zero, the entire likelihood is zero. Parameters must therefore be set so that measurement line likelihoods are fairly flat. In addition, we must consider the joint effect of all the parameters. For example, the parameters dictate precisely how far away a detected edge must be for it to be more likely to be produced by clutter than by the boundary.

4.10 Contour Tracking for Mice

Contour tracking relies on an accurate edge detection algorithm. This is a challenge for our video sequence because there is a large amount of clutter in the scene. Much of the bedding has a high image intensity gradient and there are scratches on the cage. It is difficult to detect edges between the mice and bedding because the bedding in the shadow of the mice is very similar in color to the mouse. In addition, the edges between pairs of mice are subtle, if visible at all. We tried numerous edge detection methods, and the only one that gave reasonable results was the boundary detection algorithm used by the Berkeley Segmentation Engine (BSE) [80, 41]. BSE computes the posterior probability of an edge based on the brightness, texture, and color gradient using a classifier trained on 12,000 manually labeled images. We credit BSE's superior performance in part to the texture gradient, which is robust to the types of clutter described. Figure 4.24 shows example images illustrating BSE's performance. The major downside of the BSE boundary detector is it is expensive – processing one entire image took over five minutes on a 2.8 GHz machine. We hypothesize that this algorithm can be optimized for tracking applications to reduce this cost. In addition, we plan to try faster methods such as [36] that can be tuned specifically to the mouse tracking task.

Because the BSE boundary detector outputs soft probabilities rather than hard edge classifications, we used a soft version of the generic contour likelihood. This was essential for detecting edges between pairs of mice, as BSE often outputs a weak response for these edges (see Figure 4.24(b)). The BSE output for location g is $p(edge|\mathbf{y}_g)$, the probability that g is an edge given the observations $\mathbf{y}_g$. We model the probability of a binary classification of each measurement line pixel $\mathbf{z}$ given the edge features $\mathbf{y}$ as:

$$p(\mathbf{z}|\mathbf{y}) = \prod_{i=0}^{L} p(edge|\mathbf{y}_i)^{z_i}(1 - p(edge|\mathbf{y}_i))^{1-z_i}. \tag{4.32}$$

We assume equal priors for all $\mathbf{z}$, so this is equal to $p(\mathbf{y}|\mathbf{z})$. The probability of observing

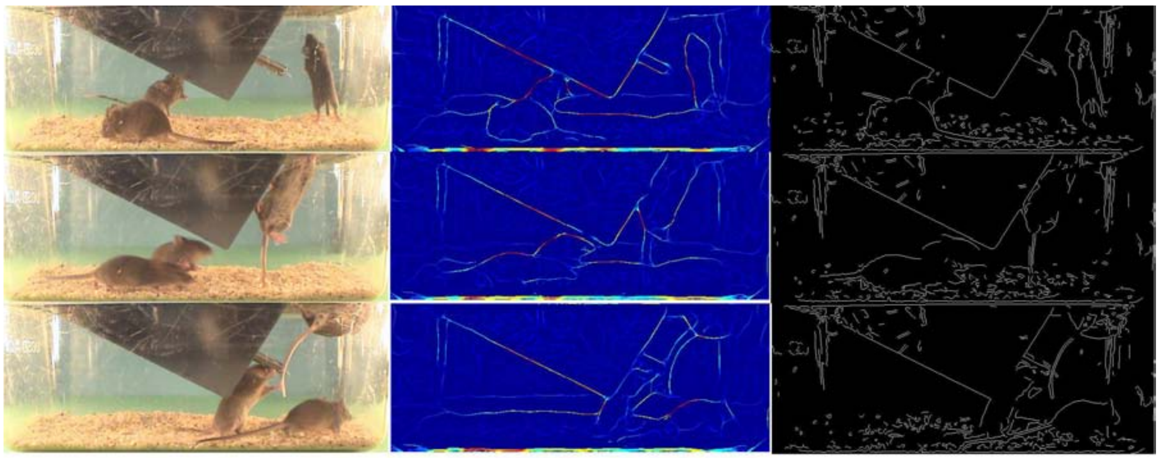


Figure 4.24: Comparison of BSE boundary detection and Canny edge detection on example frames. The left column shows the original video frame, the middle column shows the soft scores output by BSE, and the third column shows the Canny edge detection results. We can see that edges in the bedding and edges in the corners from scratches on the cage have a lower score than real edges in the image, whereas in the Canny detection results all edges have equal weight. This is due to the texture gradient computed by BSE. In the first row, we see that both methods detect the edge between the two mice on the left. In the second row, BSE gives a weak edge score at the boundary between the two left mice, while Canny does not detect this edge. In addition, edges between the back mouse and the green background are missed by Canny. In the third row, we see scratches on the cage in front of the mouse on the ceiling. While BSE only gives a weak response to some of these edges, many edges are detected by Canny, occluding the contour of the mouse.

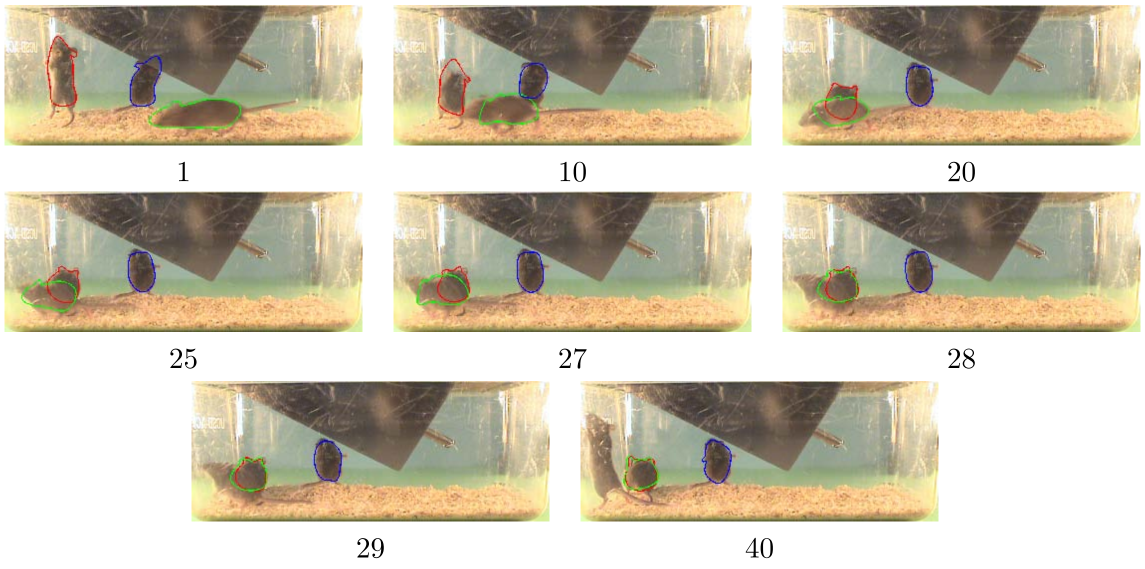


Figure 4.25: Results of using independent contour likelihood appearance models for tracking multiple mice. In frame 28, the two target boundaries follow a single mouse.

measurement line $\mathbf{y}$ given the hypothesized contour is the sum over all these possibilities:

$$p(\mathbf{y}|\mathbf{x}) = \sum_{z \in \{0,1\}^{L+1}} p(\mathbf{y}|\mathbf{z})p(\mathbf{z}, n|\mathbf{x}), \tag{4.33}$$

where $p(\mathbf{z}|\mathbf{x})$ is the generic contour likelihood described in Equation (4.31). While this computation is extremely fast for small L, it grows exponentially with L. To combat this, the sum can be taken only over $\mathbf{z}$ such that $p(\mathbf{y}|\mathbf{z})$ is significant.

4.11 Multitarget Likelihoods

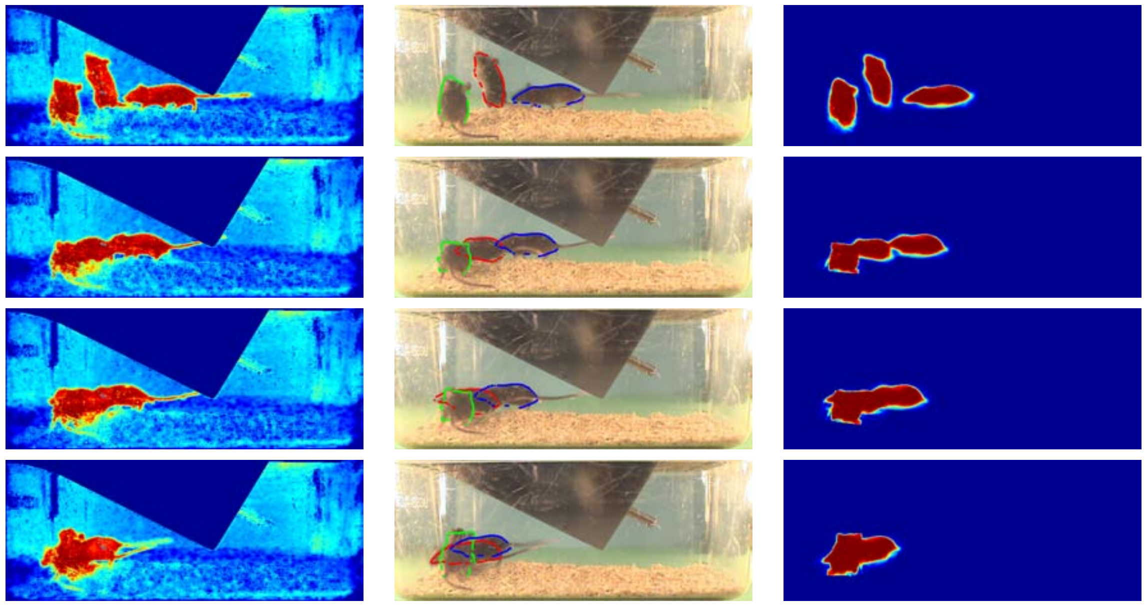
We have discussed a number of appearance models for single target tracking. In this section, we describe methods of constructing multitarget appearance models. An obvious approach is to treat the targets independently. For instance, suppose one is using the contour likelihood appearance model just described in Section 4.9.1 . For each target, we compute and combine the contour likelihood of each target's boundary. We could combine these likelihoods by multiplying them. Inevitably, this approach will result in multiple target boundaries following the single target that matches the likelihood model best. This is illustrated in Figure 4.25. The problem with this approach is that the appearance score of multiple targets can be reenforced by a single observation (a single edge detection).

[111] and [125] describe motion segmentation algorithms for tracking multiple targets (see Section 4.8.5). However, the goal of these algorithms is really to track the visible pixels of the targets. If, for instance, half of a target is occluded, both techniques estimate the position of the target as just the position of the unoccluded part of the target. This is intentional in Pfinder; it is meant to track blobs of pixels in the image. If a target is occluded, its covariance matrix will just shrink. [111] does not address this issue, as the semi-minor and semi-major axis lengths for a target are fixed. Thus, if the algorithm were applied to video in which a target was occluded, it would most likely fail. While these approaches do not allow the appearance score of multiple targets to be reenforced by a single pixel appearance, they accomplish this in a way that penalizes multiple targets containing the same pixel locations.

To obtain the desired multitarget behavior, one must consider the data association problem: which observations were produced by each target? In addition, one must have a reasonable model of occlusions. This model must take into account the states of the other targets when estimating the observations expected for a given target.

Data association is a term originating in research on tracking from radar observations [102]. In radar tracking, at any time t, we are given a set of (x, y) coordinates of detections. It is assumed that these detections are either produced by a target close to the detection or are produced by noise. Each target is assumed to produce at most one detection, and each detection is assumed to be produced by at most one target. The data association problem is then matching targets with detections. This problem is much simpler (though not necessarily easier) than the data association problem in visual tracking. In visual tracking, we must associate each feature in the video frame with a target or with background.

Multitarget appearance scores are difficult to optimize because we must score the hidden states of all the targets jointly. Thus, the size of the hidden state space increases exponentially with the number of targets K. Figure 4.28 shows this curse of dimensionality. In Figure 4.28(a), we see the results of running an independent tracker for each target. In Figure 4.28(b), we see the results of running a joint multitarget tracker with the same amount of computation. As this sequence does not have an occlusion, there is no disadvantage to running independent trackers. We see that results are much worse for the multitarget tracker.



(a) Log-likelihood ratio. (b) Predicted hidden state. (c) Predicted probability of foreground.

Figure 4.26: Illustration of BraMBLe appearance model. (a) The log-likelihood ratio of foreground to background for each patch in the image. This can be thought of as BraMBLe's representation of the video frame. (b) A hypothesized joint hidden state of all the mice. (c) Probability that each pixel is within a mouse for the proposed hidden state. BraMBLe measures how well (a) and (c) match.

4.11.1 Bayesian Multiple Blob Likelihood

The Bayesian multiple blob likelihood (BraMBLe) [57] is a multitarget appearance model that uses background and foreground appearance features. Given the hypothesized target positions $\mathbf{x}$, BraMBLe first computes the label $l_g(\mathbf{x})$ for each image location g on a grid as either foreground (within *some* target) or background (outside *all* targets). The likelihood of a patch at location g given that it is background, $p_g(\mathbf{y}_g|background)$ is modeled as discussed in Section 4.1. The likelihood of a patch given that it is foreground is modeled as discussed in Section 4.3.2. The grid features are assumed to be independent given the state, so the likelihood of the entire frame is $p(\mathbf{y}|\mathbf{x}) = \prod_g p_g(\mathbf{y}_g|l_g(\mathbf{x}))$, where $p(\mathbf{y}_g|fore)$ and $p_g(\mathbf{y}_g|back)$ are the foreground and background likelihoods.

BraMBLe can be interpreted as representing the current video frame as the log-likelihood ratio of foreground to background. Then, it finds the hidden state of the targets such that the shape of the region predicted to be foreground matches the shape of the region with a high log-likelihood ratio. This is illustrated in Figure 4.26.

4.11.2 Sprites

[60] describes a multitarget region-based template appearance model. Let $\mathbf{s}_k$ be the appearance of target k in the current frame. This appearance map has an entry for each pixel location in the image. Let $\mathbf{m}_k$ be a binary mask indicating which pixels belong to target k in the current frame. Like $\mathbf{s}_k$, there is an entry in $\mathbf{m}_k$ for each pixel location in the image. Let $\mathbf{b}$ be the background appearance at the current frame. Let c_l be the identity of the l^{th} target in depth, where c_1 is the front target in the current frame and c_K is the back target in the current frame. This model of depth is called 2.1-dimensional. The expected appearance $\mathbf{y}$ of the current frame using these notations is assumed to be

$$\begin{aligned}\mathbf{y} = \mathbf{m}_{c_1} * \mathbf{s}_{c_1} + (1 - \mathbf{m}_{c_1}) * \{\mathbf{m}_{c_2}\mathbf{s}_{c_2} + (1 - \mathbf{m}_{c_2}) * [\quad & ... \\ * (\mathbf{m}_{c_K} * s_{c_K} + (1 - \mathbf{m}_{c_K})\mathbf{b})) \quad ... \quad]\} \quad & (4.34)\end{aligned}$$

[60] is actually not intended for tracking applications. It works the same on an unordered set of images as on a sequence of frames. Instead of assuming a motion model, it assumes that the target mask and target appearance map in a particular frame are each distributed according to a Gaussian around a mean target mask and a mean appearance map, respectively. This is not effective in the mouse tracking application, as the mice are visually identical and without a motion model there is no distinguishing one from another. [93] describes a similar 2.1-dimensional multitarget model for tracking. This model is explored for foreground likelihood maximization, region-based template matching, and edge tracking.

4.11.3 Multitarget Contour Likelihood

[79] describes a multitarget extension of the contour likelihood model discussed in Section 4.9.1. This model enforces that a detected edge can only belong to one target. This model is significantly more complex than the model in Equation (4.31) because there are many more configurations we must sum over. Suppose a measurement line centered on one target's predicted boundary also intersects the predicted boundary of a second target. Then the following data associations can occur for this two target case.

1. Both boundaries are undetected and all observations are caused by clutter.
2. Observation i was produced by target k, the other target's boundary is undetected, and all other observations are caused by clutter for $i = 1, ..., n$, and $k = 1, 2$.

3. Observation i_1 was produced by the first target, observation $i_2 \neq i_1$ was produced by the second target, and all other observations are produced by clutter, for $i_1 = 1, ..., n$ and $i_2 = 1, ..., n$.

More complex cases can occur with three or more targets. If two intersections are predicted with the measurement line, the likelihood is

$$\begin{aligned} p(\mathbf{z}|\mathbf{x}) = q_{02} b(n) \left(\frac{1}{L^n} \right) + \\ q_{12} b(n-1) \left(\frac{1}{L^{n-1}} \right) \left(\sum_{i=1}^{n} \mathcal{N}(z_i; \nu_1, \sigma^2) + \sum_{i=1}^{n} \mathcal{N}(z_i; \nu_2, \sigma^2) \right) \\ + q_{22} b(n-2) \left(\frac{1}{L^{n-2}} \right) \sum_{i_1 \neq i_2}^{n} \mathcal{N}(z_{i_1}; \nu_1, \sigma^2) \mathcal{N}(z_{i_2}; \nu_2, \sigma^2) \end{aligned} \quad (4.35)$$

where q_{02} is the probability of missing two detections, q_{12} is the probability of missing one detection, $q_{22} = 1 - q01 - q02$, ν_1 is the predicted location of the first target's boundary on the measurement line, and ν_2 is the predicted location of the second target's boundary on the measurement line.

4.12 Multitarget Likelihood Models for Mice

The BraMBLe model is well-suited to the mouse tracking problem, since the background is relatively static and the foreground is homogeneous in appearance. Empirically, BraMBLe performs well at estimating the mouse positions (up to a permutation of identity labels). In addition, the space searched by BraMBLe does grow exponentially with the number of targets. However, its likelihood function is smooth and well-behaved, in comparison to those used for contour tracking. Assuming the foreground and background likelihood models are correct, the log of the BraMBLe score is proportional to the area of overlap of the predicted foreground region and the true foreground region. Thus, BraMBLe is able to give a meaningful ranking of two hidden states that are far from the true hidden state. This properties make the search simple and robust. Another advantage of the BraMBLe model is that it gives meaningful rankings even if the shape model is inexact. We can reduce the size of the hidden state space by considering simple models of the mouse shape. For instance, we can use an ellipse model of the mouse.

The main failure with the BraMBLe model is the lack of signal during severe occlusions. For example, see the frame in the last row of Figure 4.26. By working only with the log-likelihood ratio of foreground to background, BraMBLe only considers the shape of the silhouette of all the targets jointly. This is ignoring all information in the interior of the foreground region. During severe occlusions, much of the signal available is in the interior of the foreground region. During occlusions, the likelihood will be higher for some reasonable fits than others based on the relationship between the model of shape and the true object shape. Because BraMBLe assumes each patch is conditionally independent, often the difference in scores will be several orders in magnitude. Thus, the posterior distribution of hidden state will be very peaked at an incorrect estimate. Other inaccuracies were produced where background or foreground modeling were inaccurate, for instance at shadows.

Nonetheless, the BraMBLe tracking algorithm produced reasonable results. Some results are shown in Figure 4.27.

The multitarget contour likelihood proved less successful on the mice. While we were able to achieve good results when the mice were not occluding one another, we could not get reasonable results during occlusions. We experimented with a number of parameter settings, but were unable to track reasonably through any occlusion. As there are a number of parameters to set in the multitarget case, we may still have not found the best parameter setting. While attempting to determine the cause of the failure, we noticed that the assumptions that one detection was caused by at most one target and one target can cause at most one detection often did not hold.

4.13 Combining Appearance Features

More robust trackers can be achieved by combining sets of the appearance features described in this chapter. Background modeling is often combined with other appearance features since it can greatly reduce the effects of background clutter. A number of algorithms combine edge-based tracking methods with tracking methods that model the interior of the targets.

[13] combines kernel-based tracking (Section 4.3.1) and edge tracking for single target tracking. This is done by simply adding together the contour score and foreground score. [22] also combines kernel-based tracking with an edge-based module for single target tracking. This is done using a Markov random field (MRF) model with a variable for each pixel location in each video frame.

[93] combines foreground likelihood maximization (Section 4.3.2) with edge tracking and region-based matching with edge tracking for multiple targets. This approach also

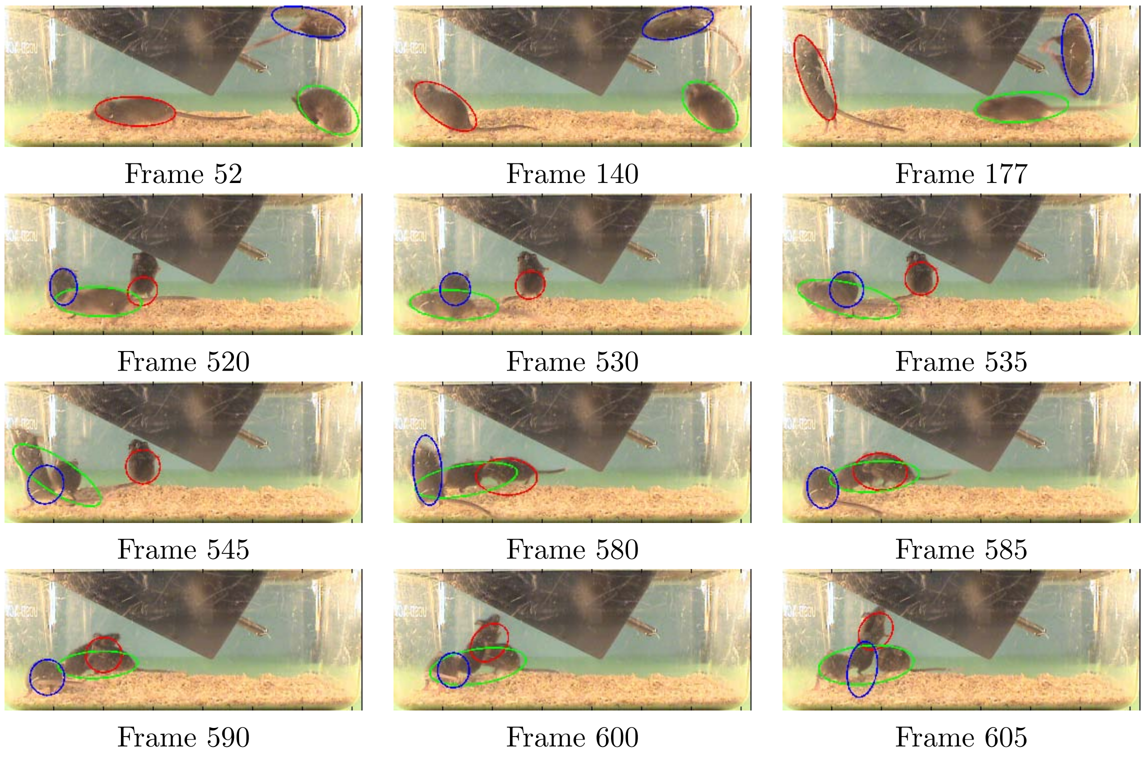
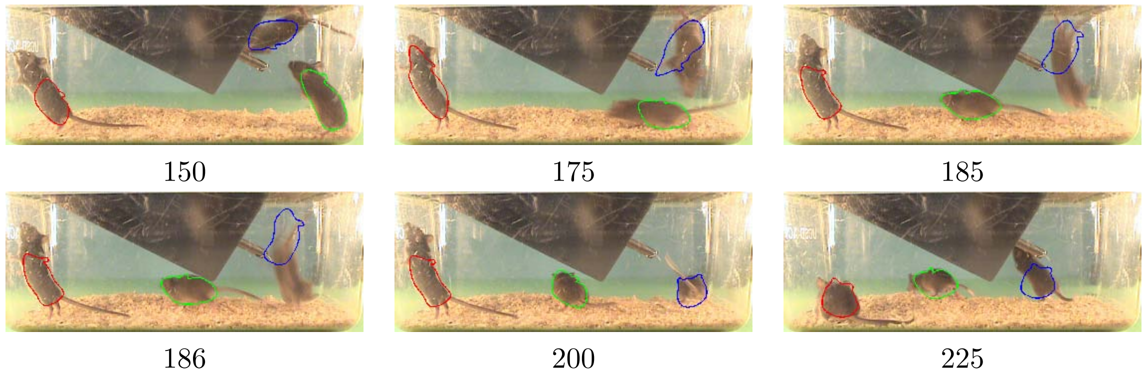
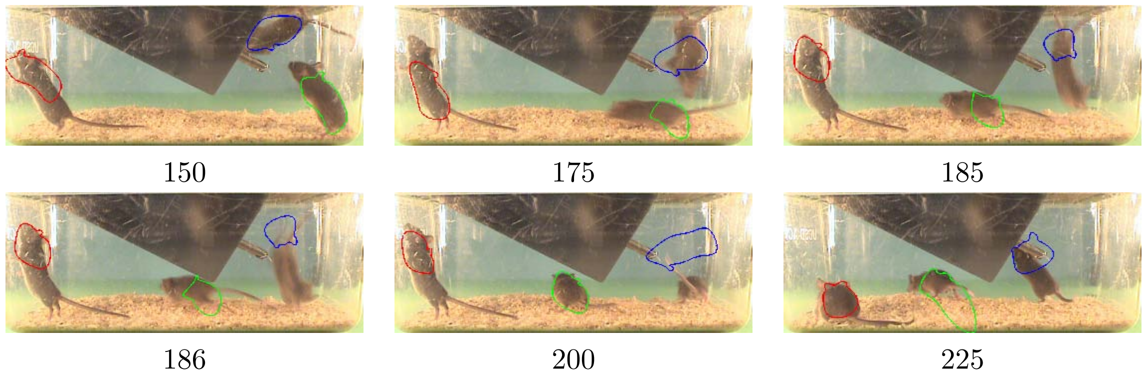


Figure 4.27: Example results of running BraMBLe on the mouse tracking application. In the first row, we see that BraMBLe is very effective at tracking the mice when there are no occlusions, even when motion is fast. Frame 52 shows that some errors can occur when background and foreground modeling is inaccurate, as is the case on the ceiling. The lower three rows show frames from an occlusion. We see in frames 520 and 530 that BraMBLe tracks the green mouse as it occludes the red mouse and the blue mouse for a few frames. In frame 540, the identities of the blue and green mouse have swapped. In the rest of the frames, we see that BraMBLe chooses a state such that most of the mouse pixels are labeled as foreground, but that this state does not correctly identify the positions of the individual mice.



(a) Independent trackers.



(b) A single joint multitarget tracker.

Figure 4.28: Illustration of the curse of dimensionality associated with multitarget tracking. In (a) the targets are tracked independently using the single target contour likelihood described in Section 4.9.1. In (b), the targets are tracked jointly using the multitarget likelihood described in Section 4.11.3. While in general we see that the estimates in (b) are more inexact than those in (a), we also see that when the mouse moves quickly, as it does from frames 175-200, the target can be lost entirely my the multitarget likelihood. The only difference between these two trackers is that the search for the optimal state is done independently in (a) and jointly in (b). In frame 225, the blue mouse stands up again and regains its tracker.

adds the edge tracking and region tracking scores.

[56] combines foreground likelihood maximization (Section 4.3.2) with edge-based tracking for single target tracking. Instead of adding the two types of scores together, [56] uses the region-based score only to improve inference. The contour likelihood is the only score that is actually reflected in the posterior distribution.

4.14 Combining Multiple Cues for Mouse Tracking

We found that combining the BraMBLe multitarget appearance likelihood (Section 4.11.1) with the single target contour likelihood (Section 4.9.1) worked well for mouse tracking. This is a natural combination because both algorithms have complementary strengths. The multiple blob tracker uses a natural multitarget model and searches a simpler space. On the other hand, contour tracking gives more fine-tuned results and utilizes cues that are available during severe occlusions.

We represent the useful information in an image observation by two sets of the features. The *blob features* $\mathbf{y}_b$ are those used by the BraMBLe tracker to determine how much each location matches the foreground and background models. The *contour features* $\mathbf{y}_c$ are the edges in the image. This representation encompasses much of the available signal, as the interior of the mouse is nearly featureless. Our current model ignores optical flow features. As discussed in Section 7, these features are useful in the mouse tracking domain, and we plan on incorporating them in the future.

To combine the BraMBLe likelihood $p(\mathbf{y}_b|\mathbf{x})$ and a soft version of the generic contour likelihood $p(\mathbf{y}_c|\mathbf{x}) = \prod_k p(\mathbf{y}_{ck}|\mathbf{x})$, we assume conditional independence given the state of the mice:

$$p(\mathbf{y}|\mathbf{x}) = p(\mathbf{y}_b|\mathbf{x}) \prod_{k=1}^{K} p(\mathbf{y}_{ck}|\mathbf{x}_k). \tag{4.36}$$

While [56] avoids this assumption, it does this by not including the region score at all. It is essential that $p(\mathbf{y}_b|\mathbf{x})$ be included in our likelihood in order to use BraMBLe to model the multitarget dependencies.

This appearance likelihood is similar to those discussed in the previous section that add together the scores from the different cues. Our approach to multicue, multitarget tracking is unique because our inference algorithm takes into consideration the complementary strengths of these two sets of features. We capitalize on the independence assumptions of our model to perform most of the search independently for each mouse. This reduces the size and complexity of the search space exponentially, and allows our Monte Carlo sampling algorithm to search the complex state parameter space with a reasonable number of samples. This is described in detail in Section 6.9. We hold discussion of experimental results using multiple cues until this section.

4.15 Acknowledgements

Portions of this chapter are based on papers that I have co-authored with others. Listed below are my contributions to each of these papers.

1. “Tracking Multiple Mouse Contours (without Too Many Samples)” by K. Branson and S. Belongie [17]. I developed the algorithm, performed the experiments, and wrote the paper.

2. “Three Brown Mice: See How They Run” by K. Branson, V. Rabaud, and S. Belongie. With insights from S. Belongie, I was responsible for developing the algorithm, performing the experiments, and writing the paper.

5

Motion Model

As with the hidden state representation, we began our research with a very simple motion model.

Recall that the motion model describes how the mice move from one frame to the next. First, we assume that the targets move independently. This means that we can factor

$$p(\mathbf{x}_{t,1:K}|\mathbf{x}_{t-1,1:K}) = \prod_k p(\mathbf{x}_{tk}|\mathbf{x}_{t,k-1}). \tag{5.1}$$

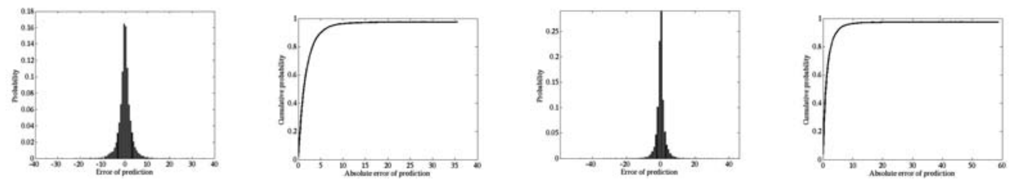
Second, we assume that all continuous position parameters follow independent constant velocity and/or autoregressive Gaussian diffusion models:

$$\mathbf{x}_{st}|\mathbf{x}_{t-1} \sim \mathcal{N}((\mathbf{I}-\mathbf{\Gamma})(\mathbf{x}_{s,t-1}+\mathbf{\Lambda}\mathbf{v}_{s,t-1})+\mathbf{\Gamma}\hat{\mathbf{x}}_s, \mathbf{\Sigma}_s),$$

where $\mathbf{\Gamma}$ is the diagonal autoregressive matrix, $\mathbf{\Lambda}$ is the diagonal dampening matrix, and $\mathbf{\Sigma}_s$ is the assumed diagonal covariance matrix. Velocities are set by subtracting the previous state from the current state.

Empirically, we found that this model fit the data well most of the time. Figure 5.1 shows the empirical distribution of the error for the constant velocity fit for the x and y coordinates of the center of a mouse. This is shown for the data collected automatically during nonocclusions, as discussed in Section 3.4. There are some instances in which the constant velocity prediction is very inaccurate. Notice in Figure 5.1 that there are a few examples in which the error is greater than 20 pixels. This means that, in order for the search space defined by the motion model to include enough samples from regions near the true position of the mice for *all* cases, the variance of the motion model must be large.

To model changes in the discrete contour template described in Section 3.3, we



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

Figure 5.1: Illustration of the error of the constant velocity model prediction of the x- and y-coordinates of the center of a mouse. (a) and (b) illustrate the error for the x-coordinate, (c) and (d) illustrate the error for the y-coordinate. (a) and (c) show a histogram approximation of the probability of an error of given magnitude and sign. (b) and (d) show the cumulatively probability that the error magnitude is less than a given magnitude.

created some ad-hoc rules that seemed sensible. In our current research, we are working on learning a similar set of rules for the learned blob and contour templates discussed in Section 3.4. There is a high probability that the contour template does not change. The probability of changing to a different contour template is based on whether the current and new contour face the same direction. We first determine which contours are allowed given the generated shape. If no contour is allowed, we rotate the shape after scaling by the minimum amount to allow at least one contour. We then decide which direction (left or right) the generated contour should face (if contours facing both directions are allowed). We flip the direction with probability proportional to the squared eccentricity. Given the direction, we choose an allowed contour facing that direction. If the direction has not changed, we choose the same contour with high probability. All other contours allowed in a given direction are given equal weight.

5.1 Acknowledgements

Portions of this chapter are based on "Tracking Multiple Mouse Contours (without Too Many Samples)" by K. Branson and S. Belongie [17]. I developed the algorithm, performed the experiments, and wrote the paper.

6

Online Inference Algorithms

Once we have assumed a statistical relationship between the hidden states and video observations, we must define a method for inferring statistics of the hidden states given a novel video sequence. In online tracking, this relationship is the posterior distribution of the current hidden state $\mathbf{x}_t$ given given all observations seen so far, $\mathbf{y}_{1:t}$. Statistics we are interested in inferring include the *MAP* estimate of the hidden state at every frame given a video sequence

$$\mathbf{x}_t^* = \arg\max_{\mathbf{x}_t} p(\mathbf{x}_t|\mathbf{y}_{1:T}), \tag{6.1}$$

and the expected value of the hidden state

$$E[\mathbf{x}_t|\mathbf{y}_{1:t}] = \int \mathbf{x}_t p(\mathbf{x}_t|\mathbf{y}_{1:t})d\mathbf{x}_t \tag{6.2}$$

Recall that we have defined our posterior distribution recursively in terms of the appearance and motion models:

$$p(\mathbf{x}_t|\mathbf{y}_{1:t}) = \frac{1}{Z}p(\mathbf{y}_t|\mathbf{x}_t)\int p(\mathbf{x}_t|\mathbf{x}_{t-1})p(\mathbf{x}_{t-1}|\mathbf{y}_{1:t-1})d\mathbf{x}_{t-1} \tag{6.3}$$

In visual tracking, usually the relationship between a hidden state and a video frame is complex, thus there is no analytical expression for the hidden state statistics given the video sequence. Thus, inference in tracking is almost always approximate.

Much of the research in tracking has focused on improving the performance of approximate inference algorithms. In this chapter, we describe a few general methods.

6.1 Greedy Inference

One common inference algorithm involves approximating the posterior distribution $p(\mathbf{x}_t|\mathbf{y}_{1:t})$ by a delta function at the approximately optimal hidden state

$$p(\mathbf{x}_t|\mathbf{y}_{1:t}) \approx \delta(\mathbf{x}_t - \hat{\mathbf{x}}_t) \tag{6.4}$$

The posterior distribution then reduces to

$$p(\mathbf{x}_t|\mathbf{y}_{1:t}) \approx \frac{1}{Z} p(\mathbf{y}_t|\mathbf{x}_t) p(\mathbf{x}_t|\hat{\mathbf{x}}_{t-1}). \tag{6.5}$$

Then, the state $\mathbf{x}_t$ that maximizes Equation (6.5) is found either approximately or exactly.

Greedily fixing the state of the target at each frame can be dangerous, for example, during occlusions. If one target is occluded, there is not enough information to accurately determine the state of this target. However, a greedy decision will be made. If the difference between the true state and the approximate state is too large, it may be impossible to recover from this error.

In Section 7, we discuss a greedy iterative algorithm that incorporates information from the future to successfully track the mice.

6.2 Approximate Kalman Filters

If the motion and appearance models are linear Gaussian, i.e.

$$p(\mathbf{y}_t|\mathbf{x}_t) = \mathcal{N}(\mathbf{y}_t; \mathbf{A}\mathbf{x}_t, \Sigma_a), \qquad p(\mathbf{x}_t|\mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \mathbf{B}\mathbf{x}_{t-1}, \Sigma_b) \tag{6.6}$$

then the posterior distribution $p(\mathbf{x}_t|\mathbf{y}_{1:t})$ will be Gaussian and we can perform inference analytically using Kalman filtering [122]. In visual tracking, the appearance model will almost never have a linear Gaussian form.

The extended Kalman filter (EKF) relaxes the assumption that the means of the Gaussians be linear functions. Instead, it assumes that these functions are approximately linear in the neighborhood around the current estimate. That is, if $E[\mathbf{x}_t|\mathbf{x}_{t-1}] = f(\mathbf{x}_{t-1}$

and $E[\mathbf{y}_t|\mathbf{x}_t] = h(\mathbf{x}_t)$ then it approximates these as

$$f(\mathbf{x}_{t-1}) \approx \hat{f}(\mathbf{x}_{t-1}) = f(\bar{\mathbf{x}}_{t-1|t-1}) + (\nabla f(\bar{\mathbf{x}}_{t-1|t-1})^\top (\mathbf{x}_{t-1} - \bar{\mathbf{x}}_{t-1|t-1}) \tag{6.7}$$

$$h(\mathbf{x}_t) \approx \hat{h}(\mathbf{x}_t) = h(\bar{\mathbf{x}}_{t|t-1}) + (\nabla h(\bar{\mathbf{x}}_{t|t-1})^\top (\mathbf{x}_t - \bar{\mathbf{x}}_{t|t-1}) \tag{6.8}$$

where

$$\bar{\mathbf{x}}_{t-1|t-1} = E[\mathbf{x}_{t-1}|\mathbf{y}_{1:t-1}], \qquad \bar{\mathbf{x}}_{t|t-1} = f(\bar{\mathbf{x}}_{t-1|t-1}). \tag{6.9}$$

These approximations are linear and can be plugged in the standard Kalman filtering equations [122].

The extended Kalman filter will fail if either of these functions are highly nonlinear in the region of interest. While the extended Kalman filter only applies the nonlinear functions f and h to the current estimates of the expected value, the unscented Kalman filter (UKF) applies these functions to a weighted sample set representation of the mean and variance.

The unscented Kalman filter [62, 119] deterministically generates $2d+1$ weighted samples, where d is the dimensionality of the hidden state. This set is chosen so that it is symmetric and the weighted mean and variance match the mean and variance of the posterior distribution at time $t-1$. The samples are propagated through the dynamics equation f, and the mean and covariance of the prior distribution $p(\mathbf{x}_t|\mathbf{y}_{1:t-1})$ are estimated as the weighted mean and covariance of the sample set. These propagated samples are then propagated again through the observation function h. The observation mean and covariance are approximated as the weighted mean and covariance of the sample set. These computed means and variances are input into the standard Kalman filtering equation to estimate the current posterior distribution.

While the extended Kalman filter is a first-order approximation, the unscented Kalman filter is a third-order approximation, and can thus deal with more nonlinearity in the dynamics and observation equations f and h.

We tried the UKF on the mouse tracking application with BraMBLe model (Section 4.11.1) and the five parameter ellipse hidden state. Our hope was that it would provide a good estimate of the uncertainty of a prediction for use in an acausal inference algorithm (see Chapter 7). The results are shown in Figure 6.1. We indicate two standard deviations of the posterior distribution of each parameter. Note that the variance does not seem correlated to be correlated to the error in the estimate. In addition, the UKF was unable to track through this non-occlusion sequence. In retrospect, the BraMBLe model and the

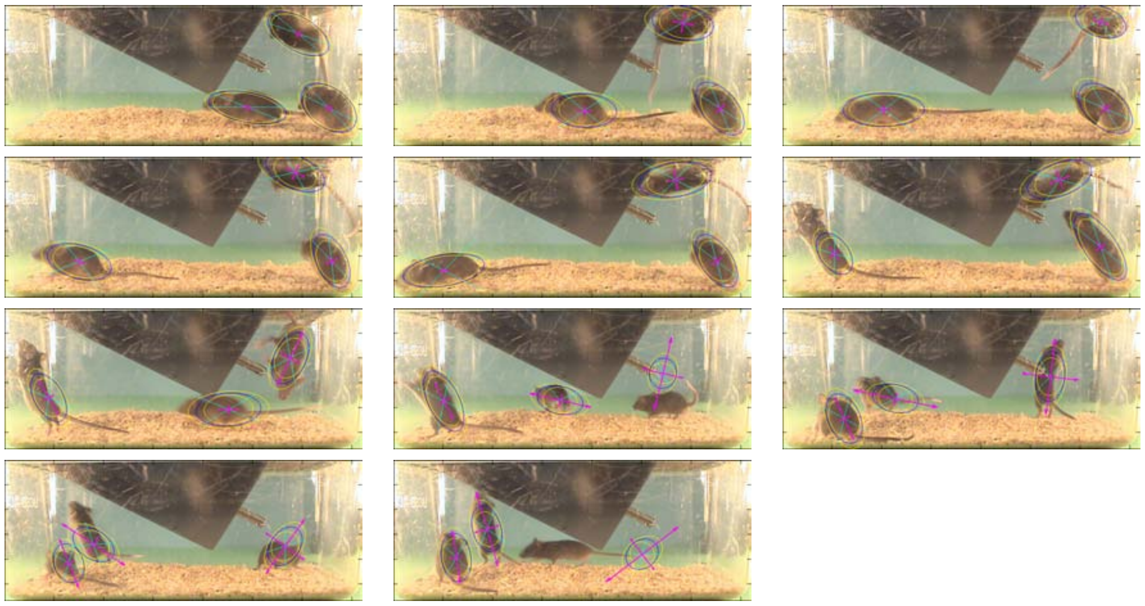


Figure 6.1: Results of trying the UKF inference algorithm for mouse tracking using the BraMBLe appearance model. We indicate two standard deviations of the x and y position by the magenta cross, two standard deviations of the minor and major axis lengths by the yellow ellipses, and two standard deviations of the angle with the cyan cross. We note that mice are lost twice in this sequence and the estimates of variance are uninformative.

UKF are probably not a good fit, as BraMBLe is highly nonlinear. Recall that we assume each grid location is independent and the likelihood is therefore a product of many Gaussian densities. It is thus very peaked and nonlinear.

We note for the reader interested in applying UKF to visual tracking that we had to tweak the UKF algorithm to deal with numerical precision issues. We found that the estimated covariance $\mathbf{P}_t$ of the Gaussian distribution was usually singular up to working precision if the UKF was performed using the original parameterization of the hidden state. At each iteration t, we found it necessary to normalize the hidden state by the standard deviation of each dimension.

6.3 Sequential Importance Sampling

Sequential importance sampling (SIS), also referred to as particle filtering and CONDENSATION relaxes both the linearity and Gaussian assumptions at the cost of additional computational expense. Instead of representing the distribution by a small set of deterministically chosen weighted samples, as is done by the unscented Kalman filter, the distribution is represented by a large set of stochastically chosen weighted samples. The larger the set of samples, the better the approximation.

6.3.1 Importance Sampling

As sequential importance sampling is an iterative sampling algorithm, we first review importance sampling. The goal in importance sampling is to approximate the distribution $p(x)$ by a finite sum

$$p(x) \approx p_N(x) = \sum_{i=1}^{N} w^{(i)} \delta(x - x^{(i)}) \tag{6.10}$$

for the purpose of approximating some integral, e.g.

$$E[f(x)] = \int f(x)p(x)dx \approx \int f(x)p_N(x)dx = \sum_{i=1}^{N} f(x^{(i)})w^{(i)} \tag{6.11}$$

Thus, we are representing the distribution $p(x)$ by the set of N weighted samples $\{(x^{(i)}, w^{(i)})\}_{i=1}^{N}$.

We assume that we can evaluate some

$$\tilde{p}(x) \propto p(x). \tag{6.12}$$

In addition, we assume that we can draw samples from some $q(x)$ that is "close" to $p(x)$. In importance sampling, we draw N i.i.d. samples from $q(x)$:

$$x^{(i)} \sim q, \quad i = 1, ..., N. \tag{6.13}$$

We then compute unnormalized weights:

$$\tilde{w}^{(i)} = \tilde{p}(x^{(i)})/q(x^{(i)}), \quad i = 1, ..., N. \tag{6.14}$$

Finally, we normalize the weights:

$$w^{(i)} = \tilde{w}^{(i)} / \sum_{j=1}^{N} w^{(j)}, \quad i = 1, ..., N. \tag{6.15}$$

We use the weighted sample set (also referred to as a particle set) $\{(x^{(i)}, w^{(i)})\}_{i=1}^{N}$

to approximate p:

$$p(x) \approx p_N(x) = \sum_{i=1}^{N} w^{(i)} \delta(x - x^{(i)}) \tag{6.16}$$

This is a reasonable approximation because, using the approximation

$$q(x) \approx q_N(x) = \frac{1}{N} \sum_{i=1}^{N} \delta(x - x^{(i)}), \tag{6.17}$$

we get an approximation of the unnormalized $\tilde{p}$:

$$\begin{aligned}
\tilde{p}(x) &= \tilde{p}(x) \frac{q(x)}{q(x)} && (6.18)\\
&= \frac{\tilde{p}(x)}{q(x)} q(x) && (6.19)\\
&\approx \frac{\tilde{p}(x)}{q(x)} q_N(x) && (6.20)\\
&= \frac{1}{N} \sum_{i=1}^{N} \frac{\tilde{p}(x)}{q(x)} \delta(x - x^{(i)}) && (6.21)\\
&= \frac{1}{N} \sum_{i=1}^{N} \tilde{w}^{(i)} \delta(x - x^{(i)}) \triangleq \tilde{p}_N(x). && (6.22)
\end{aligned}$$

We normalize this approximation to get an approximation of p:

$$\begin{aligned}
p(x) &= \tilde{p}(x) / \int \tilde{p}(x) dx && (6.23)\\
&\approx \tilde{p}_N(x) / \int \tilde{p}_N(x) dx && (6.24)\\
&= \frac{\frac{1}{N} \sum_{i=1}^{N} \tilde{w}^{(i)} \delta(x - x^{(i)})}{\frac{1}{N} \int \sum_{i=1}^{N} \tilde{w}^{(i)} \delta(x' - x^{(i)}) dx'} && (6.25)\\
&= \frac{\sum_{i=1}^{N} \tilde{w}^{(i)} \delta(x - x^{(i)})}{\int \sum_{i=1}^{N} \tilde{w}^{(i)} \delta(x' - x^{(i)}) dx'} && (6.26)\\
&= \frac{\sum_{i=1}^{N} \tilde{w}^{(i)} \delta(x - x^{(i)})}{\sum_{i=1}^{N} \tilde{w}^{(i)}} && (6.27)\\
&= \sum_{i=1}^{N} w^{(i)} \delta(x - x^{(i)}) && (6.28)\\
&\triangleq p_N(x) && (6.29)
\end{aligned}$$

Finally, we can use this approximate distribution to approximate the integral

$$E[f(x)] = \int f(x)p(x)dx \tag{6.30}$$

$$\approx \int f(x) \sum_{i=1}^{N} w^{(i)} \delta(x - x^{(i)})dx \tag{6.31}$$

$$= \sum_{i=1}^{N} w^{(i)} x^{(i)} \qquad \triangleq E_N[f(x)] \tag{6.32}$$

Above, we stated that the importance function $q(x)$ must be close to the true distribution $p(x)$. For the approximate expected value $E_N[f(x)]$ to converge almost surely to the true expected value $E[f(x)]$ as the number of samples N approaches infinity, all that is necessary is that $q(\cdot)$ be nonzero over the support of $p(\cdot)$:

$$q(x) > 0 \quad \forall x : p(x) > 0. \tag{6.33}$$

This assumption is easy to satisfy. For instance, this is satisfied for any $p(\cdot)$ if $q(\cdot)$ is a density with infinite support such as a Gaussian [37]. To be correct, we note that there are a few reasonable conditions on the true distribution $p(x)$. $p(x)$ must be finite everywhere and the variance of the function f must be finite:

$$p(x) < \infty \quad \forall x, \qquad Var_p[f(x)] < \infty. \tag{6.34}$$

In reality, the number of samples we can use is fixed by the running time and space requirements of our application. The variance of the estimated expected value (over randomness in the sampling algorithm) increases with the variance of the weights $\frac{1}{N}\sum_i (w^{(i)})^2 - 1/N^2$ [73, 74, 75, 77]. The optimal importance function is the true distribution, $p(x)$, in which case the variance of the weights is 0.

[77] quantifies this in terms of the effective sample size:

$$\mathcal{N}_N = \frac{Var_p[f(x)]}{Var_{q^N}[\sum_{i=1}^{N} w^{(i)} x^{(i)}]}. \tag{6.35}$$

The effective sample size is the number of i.i.d. samples from the true distribution needed to achieve the same approximation accuracy as $E_N[f(x)]$. [77] shows that the effective sample size can be approximated by the survival diagnostic

$$\mathcal{D}_N = \left(\sum_{i=1}^{N} (w^{(i)})^2\right)^{-1} \tag{6.36}$$

6.4 Bootstrap Filter

The most popular instantiation of sequential importance sampling is bootstrap filtering. Given a set of samples $\{\mathbf{x}_{t-1}^{(i)}\}_{i=1}^{N}$ such that

$$p(\mathbf{x}_{t-1}|\mathbf{y}_{1:t-1}) \approx p_N(\mathbf{x}_{t-1}|\mathbf{y}_{1:t-1}) = \frac{1}{N}\sum_{i=1}^{N} \delta(\mathbf{x}_{t-1} - \mathbf{x}_{t-1}^{(i)}) \tag{6.37}$$

the bootstrap filter generates a set of samples $\{\mathbf{x}_t^{(i)}\}_{i=1}^{N}$ such that

$$p(\mathbf{x}_t|\mathbf{y}_{1:t}) \approx p_N(\mathbf{x}_t|\mathbf{y}_{1:t}) = \frac{1}{N}\sum_{i=1}^{N} \delta(\mathbf{x}_t - \mathbf{x}_t^{(i)}) \tag{6.38}$$

at each iteration.

It does this by importance sampling using the importance function

$$q(\mathbf{x}_t|\mathbf{y}_{1:t}) = \int p(\mathbf{x}_t|\mathbf{x}_{t-1}) p_N(\mathbf{x}_{t-1}|\mathbf{y}_{1:t-1}) d\mathbf{x}_{t-1} \approx p(\mathbf{x}_t|\mathbf{y}_{1:t-1}) \tag{6.39}$$

First, we draw N samples

$$\tilde{\mathbf{x}}^{(i)} \sim \frac{1}{N}\sum_{j} p(\mathbf{x}_t|\mathbf{x}_{t-1}^{(i)}) \approx p(\mathbf{x}_t|\mathbf{y}_{1:t-1}) \tag{6.40}$$

This approximation holds because

$$\begin{aligned}
p(\mathbf{x}_t|\mathbf{y}_{1:t-1}) &= \int p(\mathbf{x}_t|\mathbf{x}_{t-1}) p(\mathbf{x}_{t-1}|\mathbf{y}_{1:t-1}) d\mathbf{x}_{t-1} && (6.41)\\
&\approx \int p(\mathbf{x}_t|\mathbf{x}_{t-1}) p_N(\mathbf{x}_{t-1}|\mathbf{y}_{1:t-1}) d\mathbf{x}_{t-1} && (6.42)\\
&= \int p(\mathbf{x}_t|\mathbf{x}_{t-1}) \frac{1}{N}\sum_{i} \delta(\mathbf{x}_{t-1} - \mathbf{x}_{t-1}^{(i)}) d\mathbf{x}_{t-1} && (6.43)\\
&= \frac{1}{N}\sum_{i} p(\mathbf{x}_t|\mathbf{x}_{t-1}^{(i)}) \delta(\mathbf{x}_{t-1} - \mathbf{x}_{t-1}^{(i)}) && (6.44)
\end{aligned}$$

A popular modification is to generate

$$\tilde{\mathbf{x}}_t^{(i)} \sim p(\mathbf{x}_t|\mathbf{x}_{t-1}^{(i)}), \quad i = 1, ..., N \tag{6.45}$$

to reduce the variance of the estimate.

Next, we compute the weights

$$\tilde{w}_t^{(i)} = p(\mathbf{y}_t|\tilde{\mathbf{x}}_t^{(i)}) \tag{6.46}$$

$$w_t^{(i)} = \tilde{w}_t^{(i)} / \sum_j \tilde{w}_t^{(j)} \tag{6.47}$$

This gives us an approximation of $p(\mathbf{x}_t|\mathbf{y}_{1:t})$:

$$\frac{\tilde{p}(\mathbf{x}_t|\mathbf{y}_{1:t})}{q(\mathbf{x}_t|\mathbf{y}_{1:t})} = \frac{p(\mathbf{y}_t|\mathbf{x}_t)p(\mathbf{x}_t|\mathbf{y}_{1:t-1})}{\int p(\mathbf{x}_t|\mathbf{x}_{t-1})p_N(\mathbf{x}_{t-1}|\mathbf{y}_{1:t-1})d\mathbf{x}_{t-1}} \tag{6.48}$$

$$\approx \frac{p(\mathbf{y}_t|\mathbf{x}_t)p(\mathbf{x}_t|\mathbf{y}_{1:t-1})}{p(\mathbf{x}_t|\mathbf{y}_{1:t-1})} \tag{6.49}$$

$$= p(\mathbf{y}_t|\mathbf{x}_t) \tag{6.50}$$

Now, we have an approximation of $p(\mathbf{x}_t|\mathbf{y}_{1:t})$:

$$p(\mathbf{x}_t|\mathbf{y}_{1:t}) \approx \sum_i w_t^{(i)} \delta(\mathbf{x}_t - \tilde{\mathbf{x}}_t^{(i)}) \tag{6.51}$$

as this is just importance sampling.

Finally, we resample

$$\mathbf{x}_t^{(i)} \sim \sum_j w_t^{(i)} \delta(\mathbf{x}_t - \tilde{\mathbf{x}}_t^{(j)}) \tag{6.52}$$

The sample set $\{\mathbf{x}_t^{(i)}\}_{i=1}^N$ approximates the filtering distribution

$$p(\mathbf{x}_t|\mathbf{y}_{1:t}) \approx p_N(\mathbf{x}_t|\mathbf{y}_{1:t}) = \frac{1}{N}\sum_{i=1}^N \delta(\mathbf{x}_t - \mathbf{x}_t^{(i)}) \tag{6.53}$$

6.5 Approximation Error

As we discussed in the previous section, for importance sampling to be effective for a fixed number of samples N, the importance function must be close to the true distribution in the sense that the weights must have small variance. Thus, the variance of the observation likelihood over samples drawn from the prior distribution

$$Var_{\mathbf{x}}[p(\mathbf{y}_t|\mathbf{x}_t)|y_{1:t-1}]. \tag{6.54}$$

must be small. If the variance of the motion model $p(\mathbf{x}_t|\mathbf{x}_{t-1})$ is high, as in the case of mouse tracking, then there will be many states with high density according to the motion model which will have very low density according to the appearance model. In this case, the effective sample size will be a fraction of the number of samples N, and we must choose N to be large in order to get a reasonable approximation.

Since the weights are normalized to sum to one, the highest variance of the weights is achieved if all the weight is on a single sample. We note that the variance of the motion model also usually increases exponentially with the dimensionality of the hidden state representation. Suppose, for example, the motion model $p(\mathbf{x}_t|\mathbf{x}_{t-1}$ models each dimension of the hidden state independently:

$$p(\mathbf{x}_t|\mathbf{x}_{t-1}) = \prod_{j=1}^{d} p(x_{tj}|x_{t-1,j}). \tag{6.55}$$

This is a common assumption. If we assume that the size of the hidden state space in which the observation likelihood $p(\mathbf{y}_t|\mathbf{x}_t)$ is high is constant, then the fraction of samples with high weight will decrease exponentially with dimensionality. Thus, the variance of the weights increases exponentially with dimensionality of the hidden state space.

[31] shows that the inaccuracy of the estimates will often increase with time. Assuming that the true distribution does not forget its initial condition, then the approximation errors accumulate with time. So, to ensure a constant precision, one needs an increasingly large number of samples for each frame.

To ensure that the error does not increase with time, we need to assume that any error is forgotten (exponentially) with time. More specifically, we must assume there exists some ϵ and some measure $\lambda(\cdot)$ such that

$$\epsilon\lambda(\mathbf{x}_t) \leq g(\mathbf{y}_t|\mathbf{x}_t)p(\mathbf{x}_t|\mathbf{x}_{t-1}) \leq \epsilon^{-1}\lambda(x_t) \tag{6.56}$$

for any $\mathbf{x}_{t-1}$.

These two requirements (Equations (6.54) and (6.56)) seem somewhat contradictory. The requirement that the variance in Equation (6.54) be small forces the posterior density not to depend too much on the appearance model $p(\mathbf{y}_t|\mathbf{x}_t)$. The requirement that the system be memoryless described in Equation (6.56) forces the posterior density not to depend too much on the prior distribution $p(\mathbf{x}_t|\mathbf{y}_{1:t-1})$.

In most applications, particularly visual tracking, neither of these requirements are met. Bootstrap filtering produces sample sets such that only a small number of samples will have significant weight. In addition, the accuracy of the approximation degrades with time. While bootstrap filtering has the potential to represent multimodal, non-Gaussian distributions, in practice it cannot represent many modes for many frames. Thus, bootstrap filtering degrades into an almost-greedy algorithm in practice. If things work fairly well, it may put off deciding between a pair of modes for a few frames, but it does not accurately approximate the true posterior distribution. If things do not work well, bootstrap filtering may perform worse than a greedy algorithm which approximates the distribution at each frame with a delta function at the approximate optimum. This is because bootstrap filtering not only must represent all modes of the distribution, it must find all modes of the distribution through random sampling.

6.6 Improvements to Bootstrap Filtering

SIS is a more general framework than bootstrap filtering. In general, it just means that one recursively estimates $p(\mathbf{x}_t|\mathbf{y}_{1:t})$ by a set of weighted particles $\{(\mathbf{x}_t^{(i)}, w_t^{(i)})\}_{i=1}^N$. It is not specific on how one computes these particles. One of the simplest things that we can change is to use an importance function closer to the posterior distribution than the prior distribution $q(\mathbf{x}_t|\mathbf{y}_{1:t}) \approx p(\mathbf{x}_t|\mathbf{y}_{1:t-1})$. Many algorithms try to use importance functions that incorporate the current observation, $\mathbf{y}_t$. We discuss this in Section 6.7.

As we have discussed, the number of samples required for accurate estimation often increases exponentially with the number of targets tracked. For this reason, a number of algorithms try to do as much processing as possible independently for each target. We discuss this approach in Section 6.8.

In Section 6.9, we discuss the SIS algorithm we developed for tracking mice. This algorithm combines both techniques in Sections 6.7 and 6.8.

Before we describe these techniques, we mention that there are several other approaches to minimizing the variance of SIS and increasing the effective number of particles. We briefly describe some of these approaches now.

A common technique for decreasing the variance of the estimator is to do less sampling. It is not necessary to resample in every iteration, just when the effective number of particles is small. Schemes for determining when to resample involve estimating the effective number of particles [11].

In addition, there are other methods of generating samples from the particle approximation that have lower variance than just resampling according to the weights [30].

Another problem with the resampling step is that the number of unique particles could be very small. To prevent this, the RESAMPLE-MOVE algorithm uses MCMC methods [12] set up a Markov chain in which the stationary distribution is the desired posterior $p(\mathbf{x}_t|\mathbf{y}_{1:t})$, for instance using Gibbs sampling or Metropolis-Hastings. After the resampling step, it iterates a given sample through the Markov chain. This creates diversity in the samples.

As mentioned before, the number of particles necessary to guarantee low error increases with the dimensionality of the state vector. For many applications, Rao-Blackwellization can be used to decrease the number of dimensions we need to sample over. Many distributions can be factored as

$$p(\mathbf{x}_t|\mathbf{y}_{1:t}) = p(\mathbf{x}_{t2}|\mathbf{x}_{t1}, \mathbf{y}_{1:t})p(\mathbf{x}_{t1}|\mathbf{y}_{1:t}) \tag{6.57}$$

where $\mathbf{x}_{t1}$ is one part of the hidden state at time t and $\mathbf{x}_{t2}$ is the rest of the hidden state at time t. If one of these distributions is such that integration can be performed analytically, we only need to sample over the other one. This is called Rao-Blackwellization [4].

6.7 Other Choices of Importance Function

Much research in SIS has focused on improving the similarity between the importance function and the posterior distribution. In this section, we discuss some of the proposed approaches.

6.7.1 Auxiliary Particle Filter

Instead of uniformly sampling $p_N(\mathbf{x}_{t-1}|y_{1:t-1})$ then drawing a sample from the motion model $p(\mathbf{x}_t|\mathbf{x}_{t-1})$, the auxiliary particle filter [86] first incorporates information in $\mathbf{y}_t$ into which sample $\mathbf{x}_{t-1}^{(n)}$ to choose.

To do this, it adds the auxiliary variable n to the distribution and first draw samples from

$$p(\mathbf{x}_t, n|y_{1:t}) \propto p(y_t|x_t)p(x_t|x_{t-1}^{(n)}) \tag{6.58}$$

Note that

$$\sum_{n=1}^{N} p(\mathbf{x}_t, n|\mathbf{y}_{1:t}) \propto p(\mathbf{y}_t|\mathbf{x}_t) \sum_{n=1}^{N} p(\mathbf{x}_t|\mathbf{x}_{t-1}^{(n)}) \approx p(\mathbf{y}_t|\mathbf{x}_t)p(\mathbf{x}_t|\mathbf{y}_{1:t-1}). \tag{6.59}$$

Let $\mu_t^{(n)}$ be the mean (or some other representative) of $p(\mathbf{x}_t|\mathbf{x}_{t-1}^{(n)})$. The auxiliary particle filter uses the importance function

$$q(\mathbf{x}_t, n|\mathbf{y}_{1:t}) \propto p(\mathbf{y}_t|\mu_t^{(n)})p(\mathbf{x}_t|\mathbf{x}_{t-1}^{n}). \tag{6.60}$$

Integrating $\mathbf{x}_t$ out gives

$$q(n|\mathbf{y}_{1:t}) \propto p(\mathbf{y}_t|\mu_t^{(n)}) \tag{6.61}$$

So, to sample from $q(x_t, n|y_{1:t})$, the auxiliary particle filter first chooses a sample n with probability $p(\mathbf{y}_t|\mu_t^{(n)})$. Then, it draws a sample from $p(\mathbf{x}_t|\mathbf{x}_{t-1}^{(n)})$.

We have not yet applied this particle filtering approach to mouse tracking. While we plan to experiment with this method in the future, we worry that the observation likelihood of the predicted mean may not be sufficiently informative for mouse tracking. This is because the variance of the motion model is high and the observation likelihood is peaked. We discussed the inaccuracy of low-order approximations in Section 6.2. In addition, [8] found empirically that auxiliary particle filtering did not work as well on occlusions as bootstrap filtering.

6.7.2 Combining Kalman and Particle Filtering

Instead of incorporating the current frame by computing the observation likelihood just at the predicted mean $\mu_t^{(n)}$, we can use the extended Kalman filter to incorporate information from the current frame using a linear approximation of the observation likelihood at a Gaussian approximation of the distribution [37, 34]. Extended Kalman particle filtering first resamples the particle set according to the weights, then for each new sample i computes the extended Kalman filter approximation of the posterior distribution

$$q^i(\mathbf{x}_t|\mathbf{y}_{1:t}) = p(\mathbf{y}_t|\mathbf{x}_t) \int p(\mathbf{x}_t|\mathbf{x}_{t-1})\mathcal{N}(\mathbf{x}_{t-1}; \mathbf{x}_{t-1}^{(i)}, \Sigma_{t-1}^{(i)}). \tag{6.62}$$

The new sample $\mathbf{x}_t^{(i)}$ is drawn according to this Gaussian approximation.

We emphasize an important difference between Kalman particle filtering and auxiliary particle filtering. In extended Kalman particle filtering, for *each* sample from the previous frame, an approximation incorporating the current observation is used to produce a single new sample in the current frame. In auxiliary particle filtering, we *choose* samples from the previous frame according to an approximation incorporating the current observation.

Instead of using the EKF approximation, we can use the more accurate UKF approximation [114, 115, 96]. We experimented with this modification, and found that the estimates produced by the UKF were less accurate than using just the bootstrap filter. Again, this is most likely because the appearance likelihood is very peaked and nonlinear. We plan to explore unscented particle filtering in the future with a less peaked observation likelihood.

6.7.3 Domain-Specific Methods

[83] incorporates the current observation into the importance function by first running the object detection algorithm described in [117, 118]. Samples are proposed around detected objects. This algorithm is applied to tracking many hockey players, thus the object detection algorithm mainly distinguishes dark rectangles from the white background.

A similar approach is described in [56]. This algorithm incorporates the current observation into the importance function by running a skin-color segmenter on the current video frame. Samples are proposed around large enough regions of skin-color.

In Section 6.9, we discuss a novel, related approach that was highly effective for mouse tracking.

6.8 Multitarget Sequential Importance Sampling

As discussed earlier, the number of samples required to accurately track multiple targets often grows exponentially with the number of targets. It is therefore beneficial to do as much inference as possible independently for each target. There are a number of approaches, including our own, which exploit this fact.

The multitarget tracking problem also exists in applications in which observations are from non-visual sensors such as radar, discussed briefly in Section 4.11. As the observation and motion models are simple in these applications, the *only* difficulty in these applications is the data association problem. The most popular approach in this area of research is the Joint Probabilistic Data Association Filter (JPDAF) [102], which estimates the product of the marginal posterior distributions instead of the joint posterior distribution. Monte Carlo sampling-based extensions of the JPDAF are described in [116]. While these algorithms are not directly applicable to visual tracking, we can see parallels between research in multitarget visual tracking and multitarget radar tracking. In fact, a number of papers focus on extending radar tracking algorithms to visual tracking [20, 93, 28, 29, 99, 92]. In Section 6.8.1, we discuss some popular multitarget radar tracking algorithms. In Section 6.8.2, we discuss similar approaches to improving SIS for multitarget tracking.

6.8.1 Radar-Based Tracking

Radar-based tracking algorithms represent the multitarget tracking problem in terms of the data association problem, discussed in Section 4.11. Let us describe the assumptions made by these algorithms. The observation at time t, $\mathbf{y}_t = (y_{1,t}, ..., y_{M_t})$, is made up of M_t observations. It is assumed each target generated at most one observation at a particular time step (it may go undetected). It is also assumed that each observation was produced by at most one target (it may have been produced by noise). Because of this one-to-one relationship, the association is represented as $\lambda_t = (\lambda_{1,t}, ..., \lambda_{tk})$ where

$$\lambda_{kt} = \begin{cases} 0 & \text{if target } k \text{ is undetected} \\ j \in \{1, ..., M_t\} & \text{if target } k \text{ generated observation } j \end{cases} \tag{6.63}$$

Besides the first-order Markov assumption, it is assumed that the observations are independent of one another, so the likelihood can be factored as

$$p(\mathbf{y}_t|\mathbf{x}_t, \lambda_t) = \prod_{j \in clutter} p_C(\mathbf{y}_{jt}) \prod_{k=1}^{K} p(\mathbf{y}_{\lambda_{kt},t}|\mathbf{x}_{kt}), \tag{6.64}$$

The first product is over all observations assigned by λ_t to be clutter. $p_C(\cdot)$ denotes the clutter distribution, and is assumed to be uniform. In the second product, we take $p(\mathbf{y}_{0t}|\mathbf{x}_{kt})$ to be the probability that target k is undetected. We note that in visual multitarget tracking it is not obvious how to factor the observation likelihood into independent components like this.

Standard Filtering

Without making any other assumptions or approximations, estimating $p(\mathbf{x}_t|\mathbf{y}_{1:t})$ requires first estimating $p(\mathbf{x}_t, \lambda_t|\mathbf{y}_{1:t})$. Once we have computed $p(\mathbf{x}_t, \lambda_t|\mathbf{y}_{1:t})$, we marginalize over the association variable λ_t

$$p(\mathbf{x}_t|\mathbf{y}_{1:t}) = \sum_{\lambda_t} p(\mathbf{x}_t, \lambda_t|\mathbf{y}_{1:t}). \tag{6.65}$$

Each term of the sum is computed iteratively using the recursive decomposition

$$p(\mathbf{x}_t|\mathbf{y}_{1:t}) = \sum_{\lambda_t} w(\lambda_t)p(\mathbf{y}_t|\mathbf{x}_t, \lambda_t)p(\mathbf{x}_t|\mathbf{y}_{1:t-1}), \tag{6.66}$$

where $w(\lambda_t)$ is the weight of the hypothesized data association vector λ_t. Recall that computing $p(\mathbf{x}_t|\mathbf{y}_{1:t-1})$ requires integrating over the previous posterior:

$$p(\mathbf{x}_t|\mathbf{y}_{1:t-1}) = \int p(\mathbf{x}_t|\mathbf{x}_{t-1})p(\mathbf{x}_{t-1}|\mathbf{y}_{1:t-1})d\mathbf{x}_{t-1} \tag{6.67}$$

The posterior at every frame is a mixture distribution with a component for each possible data association value. If M_t is the number of observations and K is the number of targets, the number of components is the number of binary vectors of length M_t such that at most K elements are 1:

$$N_t = \sum_{i=0}^{K} \binom{M_t}{i} \frac{K!}{(K-i)!}. \tag{6.68}$$

At each frame, the number of mixture components compounds, as there is a mixture component for every possible sequence of data association vectors. Thus, the number of mixture components at time t is $prod_{\tau=1}^{T} N_\tau$. Computing and storing this huge number of components is intractable. The multiple hypothesis tracking algorithm [95] involves storing as many components as possible and throwing away those with the lowest weight.

The Joint Probabilistic Data Association Filter

Instead of estimating the joint posterior distribution $p(\mathbf{x}_t|\mathbf{y}_{1:t})$ at each frame, the JPDAF estimates the *marginal* posteriors $p(\mathbf{x}_{tk}|\mathbf{y}_{1:t})$ for each target k. Each marginal posterior can be represented by a mixture with a small number of components, so the product of the marginals effectively has a high number of components.

To estimate the marginal posterior $p(\mathbf{x}_{tk}|\mathbf{y}_{1:t})$, the JPDAF estimates marginal motion and appearance models. It is common to assume that the targets move independently, so the motion model can be decomposed as

$$p(\mathbf{x}_t|\mathbf{x}_{t-1}) = \prod_{k=1}^{K} p(\mathbf{x}_{tk}|\mathbf{x}_{t-1,k}) \tag{6.69}$$

The marginal posterior is assumed to have the form:

$$p(\mathbf{y}_t|\mathbf{x}_{tk}) = \sum_{i=0}^{M_t} w_{tk}^i p(\mathbf{y}_{ti}|\mathbf{x}_{tk}). \tag{6.70}$$

Here, $p(\mathbf{y}_{t0}|\mathbf{x}_{tk})$ is the probability that the target is undetected. The soft assignment weight w_{tk}^i is an approximation of the probability that target k will produce observation i. It is computed from

$$p_{k'}(\mathbf{y}_{tj}|y_{1:t-1}) = \int p(\mathbf{y}_{tj}|\mathbf{x}_{tk'})p(\mathbf{x}_{tk'}|\mathbf{y}_{1:t-1})d\mathbf{x}_{tk'} \tag{6.71}$$

for all $k' = 1, ...K$ and $j = 0, ..., M_t$. This is the expected matching score between target k' and observation j. The expression for the soft assignment weight w_{tk}^i is

$$w_{tk}^i = \sum_{\lambda:\lambda_k=i} \frac{1}{Z} p(\lambda) \prod_{k'=1}^{K} p_{k'}(\mathbf{y}_{t\lambda_{k'}}|\mathbf{y}_{1:t-1}). \tag{6.72}$$

The standard approach involves enumerating all possible configurations of joint hidden state and data association vector then computing the matching score between each target and its assigned observation. This approach enumerates all possible configurations of the data association vector then computes the *expected* matching score between each target and its assigned observation.

The marginal distribution has a component for each observation, instead of a N_t components. Effectively, the product of the marginals therefore has M_t^K components, but each component never has to be computed or stored. The number of components will still compound over time and low weight components will need to be thrown away. However, there is an exponential decrease in the number of components necessary to represent similar information.

The JPDAF was originally proposed with Gaussian approximations of the distributions. At the beginning of each iteration, the prior distribution is approximated by a Gaussian distribution. [98] presented a SIS version of the JPDAF which allows a particle set representation of the data, as well as nonlinear and non-Gaussian motion and observation models. Second, because the density is represented by a sum of delta functions, the number of components remains constant.

6.8.2 Partitioned Sampling

[77] proposed an algorithm for performing the sampling independently for each target using the multitarget contour likelihood. This algorithm depends on modeling the likelihood $p(\mathbf{y}_t|\mathbf{x}_{t,1:k})$. For each target k, beginning with the target in front and ending with the target in back, [77] first draws samples from the prior distribution for the target k:

$$\mathbf{x}_{tk}^{(n)} \sim p(\mathbf{x}_{tk}|\mathbf{x}_{t-1,k}^{(n)}). \tag{6.73}$$

This sample is concatenated with the results of sampling for the first $k-1$ targets:

$$\mathbf{x}_{t,1:k}^{(n)} = (\tilde{\mathbf{x}}_{t,1:k-1}^{(n)}, \mathbf{x}_{tk}^{(n)}) \tag{6.74}$$

These concatenated samples are weighted by the target contour likelihood

$$w_{tk}^{(n)} \propto \frac{p(\mathbf{y}_t|\mathbf{x}_{t,1:k})}{p(\mathbf{y}_t|\mathbf{x}_{t,1:k-1})} \tag{6.75}$$

The particles $\{\mathbf{x}_{t,1:k}^{(n)}, w_{tk}^{(n)}\}$ are then resampled according to the weights $\{w_{tk}^{(n)}\}$ to create an unweighted sample set $\{\tilde{\mathbf{x}}_{t1}^{(n)}$. This is repeated for $k = 1, ..., K$, where $k = 1$ is the front target and $k = K$ is the back target.

The benefit of this algorithm is that we sample over each target sequentially. If we are confident that our single target likelihood $p(\mathbf{y}_t|\mathbf{x}_{t1})$ is representative, then this sampling step locates the relevant part of the state space for the dimensions of the state space corresponding to the first target. Then, for the second target, we only need to search the part of the state space corresponding to the second target.

Of course, if there is no ordering of targets known so that we can reliably locate the relevant part of the search space for one target without considering the positions of the other targets, then this approach will fail. For example, the BraMBLe model of appearance relies on the shape of all the targets to compute the likelihood score, and cannot be simply factored like the multitarget contour likelihood – there is no analytic form for $p(\mathbf{y}_t|\mathbf{x}_{t,1:k})$ for $k \neq K$.

In the next section, we propose an approach similar to the MC-JPDAF for such a situation.

6.9 Tracking Multiple Mouse Contours (without Too Many Samples)

In this section, we present a particle filtering algorithm for robustly tracking the contours of multiple deformable objects through severe occlusions. Our algorithm combines a multiple blob tracker with a contour tracker in a manner that keeps the required number of samples small. This is a natural combination because both algorithms have complementary strengths. The multiple blob tracker uses a natural multitarget model and searches a smaller and simpler space. On the other hand, contour tracking gives more fine-tuned results and relies on cues that are available during severe occlusions. Our choice of combination of these two algorithms accentuates the advantages of each. In addition, we capitalize on the independence assumptions of our model to perform most of the search independently for each mouse. This reduces the size and complexity of the search space exponentially, and allows our Monte Carlo sampling algorithm to search the complex state parameter space with a reasonable number of samples. Our algorithm works with a detailed representation of a mouse contour to achieve encouraging results on a video sequence of three mice exploring a cage.

6.9.1 Blob and Contour Model

We use a particle filtering algorithm to approximate $p(\mathbf{x}_t|\mathbf{y}_{1:t})$, the posterior distribution of the state of all K mice in frame t, $\mathbf{x}_t = \mathbf{x}_{t,1:K}$, given blob and contour observations for frames 1 to t. We use the combined blob and contour observation model described in Section 4.14. We describe our model of the position of the mouse in Section 3. We use the constant velocity motion model described in Section 5. See Table 6.1 for a summary of the motion and appearance models assumed.

6.9.2 Blob-Contour Particle Filtering

In this section, we describe our algorithm for efficiently sampling from the combined blob and contour posterior distribution of the K mice, $p(\mathbf{x}_{tk}|\mathbf{y}_{b,1:t}, \mathbf{y}_{c,1:t})$, given the models and independence assumptions described in Section 6.9.1.

At each iteration of particle filtering, we generate a set of weighted samples $\{(x_t^{(i)}, w_t^{(i)})\}_{i=1}^N$ such that $p(\mathbf{x}_t|\mathbf{y}_{1:t}) \approx \sum_{i=1}^N w_t^{(i)}\delta(\mathbf{x}_t - \mathbf{x}_t^{(i)})$ from the previous set of weighted samples $\{(x_{t-1}^{(i)}, w_{t-1}^{(i)})\}_{i=1}^N$. The most popular particle filtering algorithm is the bootstrap filter. As described in Section 6.3, the importance function used in this algorithm is

Table 6.1: Blob-Contour Model. The first column describes the model, the second column states which section the model was discussed in, the third column states which mice the model applies to, and the third column shows the mathwmatical expression for the model at frame t.

Description	Section	Mice	Distribution
Single Blob State	§3.2	k	$\mathbf{x}_{btk} = (\mu_{xtk}, \mu_{ytk}, a_{tk}, b_{tk}, \theta_{tk})^\top$
Single Blob-Contour State	§3	k	$\mathbf{x}_{tk} = (\mathbf{x}_{btk}^\top, \phi_{tk}, c_{tk}, f_{tk})^\top$
Blob State	§3.2	$1:K$	$\mathbf{x}_{bt} = \mathbf{x}_{bt,1:K} = (\mathbf{x}_{bt1}^\top, \dots, \mathbf{x}_{btK}^\top)^\top$
Blob-Contour State	§3.2	$1:K$	$\mathbf{x}_t = \mathbf{x}_{t,1:K} = (\mathbf{x}_{t1}^\top, \dots, \mathbf{x}_{tK}^\top)^\top$
Blob Observation Likelihood	§4.11.1	$1:K$	$p(\mathbf{y}_{bt}\|\mathbf{x}_{bt}) = \prod_g p_g(\mathbf{y}_{btg}\|l_g(\mathbf{x}_{bt}))$
Contour Observation Likelihood	§4.9.1	k	$p(\mathbf{y}_{ctk}\|\mathbf{x}_{tk}) = \prod_m p(\mathbf{y}_{ctkm}\|\mathbf{x}_{tk})$
Contour Observation Likelihood	§4.9.1	$1:K$	$p(\mathbf{y}_{ct}\|\mathbf{x}_t) = \prod_k p(\mathbf{y}_{ctk}\|\mathbf{x}_{tk})$
Blob-Contour Observation Likelihood	§4.14	$1:K$	$p(\mathbf{y}_{bt}, \mathbf{y}_{ct}\|\mathbf{x}_t) = p(\mathbf{y}_{bt}\|\mathbf{x}_{bt})p(\mathbf{y}_{ct}\|\mathbf{x}_t)$
Blob-Contour Transition Distribution	§5	k	$p(\mathbf{x}_{tk}\|\mathbf{x}_{t-1,k})$
Blob-Contour Transition Distribution	§5	$1:K$	$p(\mathbf{x}_t\|\mathbf{x}_{t-1}) = \prod_k p(\mathbf{x}_{tk}\|\mathbf{x}_{t-1,k})$.

$p(\mathbf{x}_{t,1:K}|\mathbf{x}_{t-1,1:K})$, and the importance weight is $p(\mathbf{y}_{bt}, \mathbf{y}_{ct}|\mathbf{x}_{t,1:K})$. Three properties of our problem cause this direct application to fail for any practical number of samples N. First, because the motion of the mice is erratic, the variance of $p(\mathbf{x}_{t,1:K}|\mathbf{x}_{t-1,1:K})$ is high. Second, because the contour feature is sparse, the scores given by $p(\mathbf{y}_{ct}|\mathbf{x}_{t,1:K})$ are only meaningful within a short radius of the optimal fits for all K mice. Third, the dimension of $\mathbf{x}_{t,1:K}$ is proportional to K, thus the search space size is exponential in K. Because of these three properties of the relationship between the importance and true posterior distributions, a huge number of samples must be generated so that enough fall within the short radius of the optimal fits.

We address these problems by performing three sampling steps, instead of just one. Each step incorporates a subset of the observations to gradually hone in on the small subspace truly important in the posterior distribution. The first step performs bootstrap filtering using only the blob observations. This step localizes the search space for the mice by moving the distribution toward the optimal fits and decreasing its variance. The second step performs bootstrap filtering using the contour observations independently for each mouse. This allows the algorithm to find the important regions for each mouse independently, exponentially reducing the search space by preventing a good fit for one mouse from being rejected because it is paired with a bad fit for another (this problem is also addressed in the sensor observation literature [84]). The third step of sampling combines the K sets of particles by sampling independently from each, then weights by the necessary importance weight. These sampling steps are illustrated in Figure 6.3.

One can interpret the first two filtering steps as generating samples from the approximate marginal posteriors $p(\mathbf{y}_t|\mathbf{x}_{tk})$, as in the JPDAF (Section 6.8.1). Then, the importance function in the third sampling step is the product of the posterior marginals:

$$q_3(\mathbf{x}_t|\mathbf{y}_{1:t}) = \prod_{k=1}^{K} p(\mathbf{x}_{tk}|\mathbf{y}_{1:t-1,bt,ctk}). \tag{6.76}$$

We describe these three steps in more detail next. Figure 6.2 shows the algorithm steps.

```
For $t = 1, 2, \ldots$ :
1. Sample from the marginal posteriors:
   For $i = 1, \ldots, N$:
    a.  Choose $\tilde{\mathbf{x}}^{(i)}_{t-1,1:K} \sim \{(\mathbf{x}^{(j)}_{t-1,1:K}, w^{(j)}_{t-1})\}$.
    b.  Generate $\mathbf{b}^{(i)}_{1:K} \sim p(\mathbf{b}_{1:K}|\tilde{\mathbf{x}}^{(i)}_{t-1,1:K})$.
    c.  Compute the weight $w^{(i)}_b \propto p(\mathbf{y}_{bt}|\mathbf{b}^{(i)}_{1:K})$.
    d.  Choose $\tilde{\mathbf{b}}^{(i)}_{1:K} \sim \{\tilde{\mathbf{b}}^{(j)}_{1:K}, w^{(j)}_b)\}$
    e.  For $k = 1, \ldots, K$,
         Generate $\tilde{\mathbf{x}}^{(i)}_{tk} \sim p(\mathbf{x}_{tk}|\tilde{\mathbf{b}}^{(i)}_k)$.
    f.  Compute the weight $w^{(i)}_{tk} = p(\mathbf{y}_{ctk}|\tilde{\mathbf{x}}^{(i)}_{tk})$.
2. Sample from the joint posterior:
   For $i = 1, \ldots, N$:
    a.  For $k = 1, \ldots, K$,
         Choose $\mathbf{x}^{(i)}_{tk} \sim \{(\tilde{\mathbf{x}}^{(j)}_{tk}, w^{(j)}_{tk})\}$.
    b.  Concatenate:  $\mathbf{x}^{(i)}_{t,1:K} = (\mathbf{x}^{(i)}_{t1}, \ldots, \mathbf{x}^{(i)}_{tK})$.
    c.  Compute the importance weight:
         $w^{(i)}_t \propto p(\mathbf{y}_{bt}|\mathbf{x}^{(i)}_{t,1:K}) \sum_{j=1}^{N} w^{(j)}_{t-1} \prod_{k=1}^{K} p(\mathbf{x}^{(i)}_{tk}|\mathbf{x}^{(j)}_{t-1,k})$.
```

Figure 6.2: Blob and Contour Particle Filtering

Step 1: Blob Sampling

In the first sampling step, we perform an iteration of bootstrap filtering (with systematic resampling with replacement) using only the blob observation $\mathbf{y}_{bt}$. Given $\{(\mathbf{x}^{(i)}_{t-1,1:K}, w^{(i)}_{t-1})\}$ representing $p(\mathbf{x}_{t-1,1:K}|\mathbf{y}_{1:t-1})$, we generate $\{\mathbf{b}^{(i)}_{t,1:K}\}_{i=1}^{N}$ from

$$q_1(\mathbf{b}_t|\mathbf{y}_{1:t}) = \int p(\mathbf{b}_t|\mathbf{x}_{t-1})p(\mathbf{x}_{t-1}|\mathbf{y}_{1:t-1})d\mathbf{x}_{t-1}, \tag{6.77}$$

where $p(\mathbf{b}_t|\mathbf{x}_{t-1}) = \prod_k p(\mathbf{b}_{tk}|\mathbf{x}_{t-1,k})$ and $p(\mathbf{b}_{tk}|\mathbf{x}_{t-1,k})$ has the same form as $p(\mathbf{x}_{tk}|\mathbf{x}_{t-1,k})$ for all blob parameters, and is a δ function around the previous state for the contour

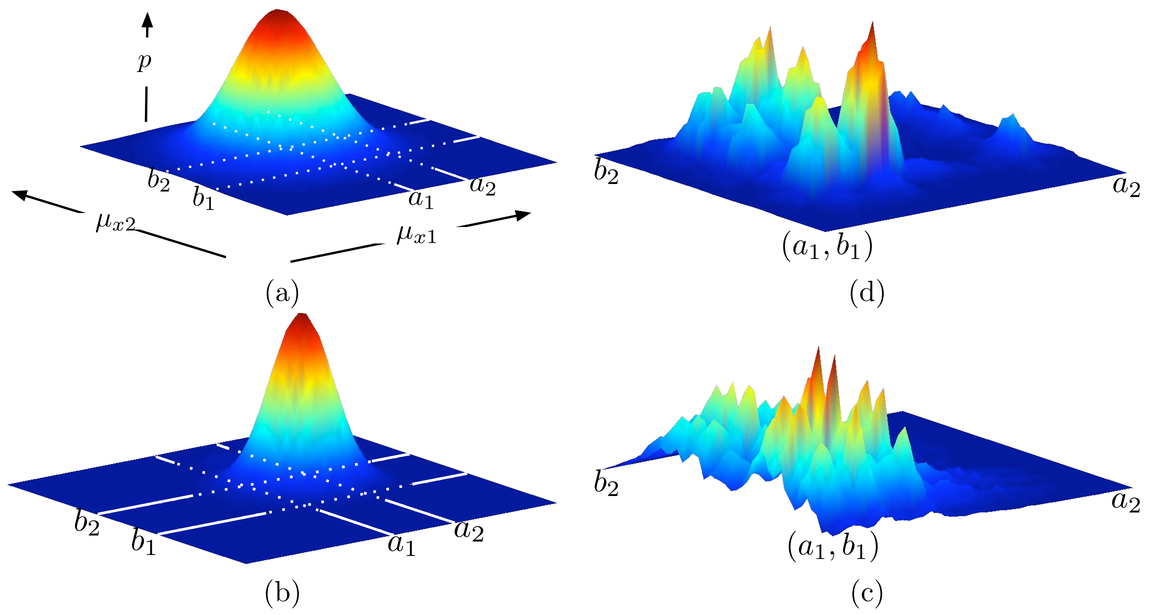


Figure 6.3: Illustration of the importance functions used in each sampling step. The x-axis of each plot corresponds to the x-coordinate of the center of one mouse. The y-axis of each plot corresponds to the x-coordinate of the center of the second mouse. The z-access indicates probability density. The entire axes of (c) and (d) correspond to the small white rectangle $[a_1, a_2] \times [b_1, b_2]$ in (a) and (b). (d) shows the true posterior distribution according to the model, $p(\mathbf{x}_t|\mathbf{y}_{1:t})$. (a) shows the importance function used in bootstrap filtering, the prior distribution $p(\mathbf{x}_t|\mathbf{y}_{1:t-1})$. We see that the distribution is not centered over the right part of the space (the small white rectangle). In addition, it has a high variance. Drawing enough samples from the important part of the space is difficult. (b) shows the results of performing an iteration of bootstrap filtering using only the blob observation to localize the search space. This results in the distribution $p(\mathbf{x}_t|\mathbf{y}_{1:t-1}, \mathbf{y}_{bt})$. Notice that the variance is now smaller and the density is centered over the correct part of the space. (c) shows the product of the marginal posterior distributions $\prod_{k=1}^{K} p(\mathbf{x}_{tk}|\mathbf{x}_{1:t-1}, \mathbf{y}_{bt}, \mathbf{y}_{ctk})$. (c) is very close to (d).

parameters. We then weight by $w_{bt}^{(i)} \propto p(\mathbf{y}_{bt}|\mathbf{b}_t)$ so that

$$p(\mathbf{b}_t|\mathbf{y}_{1:t-1,bt}) \approx \sum_i w_{bt}^{(i)} \delta(\mathbf{b}_t - \mathbf{b}_t^{(i)}). \tag{6.78}$$

The marginal is

$$\begin{aligned} p(\mathbf{b}_{tk}|\mathbf{y}_{1:t-1,bt}) &= \int p(\mathbf{b}_t|\mathbf{y}_{1:t-1,bt}) d\mathbf{b}_{t1:K\backslash k} \\ &\approx \int \sum_i w_{bt}^{(i)} \delta(\mathbf{b}_t - \mathbf{b}_t^{(i)}) d\mathbf{b}_{t,1:K\backslash k} = \sum_i w_{bt}^{(i)} \delta(\mathbf{b}_{tk} - \mathbf{b}_{tk}^{(i)}). \end{aligned} \tag{6.79}$$

This sampling step is illustrated in Figure 6.3(a-b).

Step 2: Independent Contour Sampling

In the second sampling step, we perform an iteration of bootstrap filtering using the contour observation $\mathbf{y}_{ctk}$ for each mouse k independently. Given $\{(\mathbf{b}_{tk}^{(i)}, w_{bt}^{(i)})\}$ representing $p(\mathbf{b}_{tk}|\mathbf{y}_{1:t-1,bt})$, we generate $\{\tilde{\mathbf{x}}_{tk}^{(i)}\}$ from

$$q_2(\mathbf{x}_{tk}|\mathbf{y}_{1:t}) = \int p(\mathbf{x}_{tk}|\mathbf{b}_{tk}) p(\mathbf{b}_{tk}|\mathbf{y}_{1:t-1,bt}) d\mathbf{b}_{tk}, \tag{6.80}$$

where $p(\mathbf{x}_{tk}|\mathbf{b}_{tk})$ is the same as $p(\mathbf{x}_{tk}|\mathbf{x}_{t-1,k})$ for the contour parameters and Gaussian (with a different, smaller variance) around $\mathbf{b}_{tk}$ for the blob parameters. We then weight by $w_{tk}^{(i)} \propto p(\mathbf{y}_{ctk}|\tilde{\mathbf{x}}_{tk})$ so that

$$p(\mathbf{x}_{tk}|\mathbf{y}_{1:t-1,bt,ctk}) \approx \sum_i w_{tk}^{(i)} \delta(\mathbf{x}_{tk} - \tilde{\mathbf{x}}_{tk}^{(i)}). \tag{6.81}$$

This sampling step is illustrated in Figure 6.3(c-d).

Step 3: Combining the Marginals

In the final sampling step, we perform an iteration of sampling to combine the individual mouse marginals. Given $\{(\tilde{\mathbf{x}}_{tk}^{(i)}, w_{tk}^{(i)})\}$ representing $p(\mathbf{x}_{tk}|\mathbf{y}_{1:t-1,bt,ctk})$ for $k = 1, \ldots, K$, we generate $\{\mathbf{x}_t^{(i)}\}$ from

$$q_3(\mathbf{x}_{tk}|\mathbf{y}_{1:t}) = \prod_{k=1}^{K} p(\mathbf{x}_{tk}|\mathbf{y}_{1:t-1,bt,ctk}) \tag{6.82}$$

by independently sampling from $\{(\tilde{\mathbf{x}}_{tk}^{(i)}, w_{tk}^{(i)})\}$ for $k = 1, \ldots, K$ and concatenating. We then weight by

$$w_t^{(i)} \propto \frac{p(\mathbf{x}_t^{(i)}|\mathbf{y}_{1:t})}{q_3(\mathbf{x}_t|\mathbf{y}_{1:t})} \tag{6.83}$$

$$\propto \frac{p(\mathbf{y}_{bt}|\mathbf{x}_t^{(i)}) \prod_k p(\mathbf{y}_{ctk}|\mathbf{x}_{tk}^{(i)}) \int p(\mathbf{x}_t^{(i)}|\mathbf{x}_{t-1}) p(\mathbf{x}_{t-1}|\mathbf{y}_{1:t-1}) d\mathbf{x}_{t-1}}{\prod_k p(\mathbf{x}_{tk}|\mathbf{y}_{1:t-1,bt,ctk})} \tag{6.84}$$

$$\approx \frac{p(\mathbf{y}_{bt}|\mathbf{x}_t^{(i)}) \prod_k p(\mathbf{y}_{ctk}|\mathbf{x}_{tk}^{(i)}) \int p(\mathbf{x}_t^{(i)}|\mathbf{x}_{t-1}) \sum_j w_{t-1}^{(j)} \delta(\mathbf{x}_{t-1} - \mathbf{x}_{t-1}^{(j)}) d\mathbf{x}_{t-1}}{\prod_k \sum_j w_{tk}^{(j)} \delta(\mathbf{x}_{tk}^{(i)} - \tilde{\mathbf{x}}_{tk}^{(j)})} \tag{6.85}$$

$$= \frac{p(\mathbf{y}_{bt}|\mathbf{x}_t^{(i)}) \prod_k p(\mathbf{y}_{ctk}|\mathbf{x}_{tk}^{(i)}) \sum_j w_{t-1}^{(j)} p(\mathbf{x}_t^{(i)}|\mathbf{x}_{t-1}^{(j)})}{\prod_k p(\mathbf{y}_{ctk}|\mathbf{x}_{tk}^{(i)})} \tag{6.86}$$

$$= p(\mathbf{y}_{bt}|\mathbf{x}_t^{(i)}) \sum_j w_{t-1}^{(j)} p(\mathbf{x}_t^{(i)}|\mathbf{x}_{t-1}^{(j)}) \tag{6.87}$$

This sampling step is illustrated in Figure 6.3(d-e).

Our sampling algorithm is consistent because it is a sequence of three consistent steps. The first two steps are standard applications of bootstrap filtering, and are thus consistent. So, each of the K particle sets input in step 3 in the limit as $N \to \infty$ converge pointwise to the true marginal posterior. Step 3 generates samples from the product of the marginal posteriors. If any of the marginals has zero density, then the joint density is also zero. Finally, we weight by the ratio of the joint posterior to the particle approximations of the marginal posteriors. As these particle approximations are consistent, the ratio is also consistent.

6.9.3 Experiments

We evaluated our blob and contour tracking algorithm on a video sequence of three identical mice exploring a cage. This sequence contained 11 occlusions of varying difficulty. The model parameters were chosen by hand using a separate video sequence. Many of the parameters were set using our knowledge about the problem. These include the variance of the transition models and the constraints on the state. Other parameters, including the damping and autoregressive constants, were set to values used in [16]. The contour likelihood parameters were set so that the ranking of the probability of each vector of observed edge detections seemed reasonable. Some of the parameters were chosen somewhat arbitrarily and never varied – these include the number of observation lines and the parameters used in the BraMBLe likelihood. The number of samples was chosen to be $N = 2000$. While we had qualitatively similar performance with $N = 1000$ samples, the results returned by particle filtering varied quite a bit. We thus chose to present results with $N = 2000$ samples, for which the output of our particle filtering algorithm was stable. This number of samples compares favorably to the 4000 samples used to track a pair of leaves in [79] and the 1000 samples used to track two people/blobs in [57].

Summary still frames are shown in Figures 6.4, 6.5, and 6.6. These results demonstrate the following strengths of our algorithm:

- Our contour tracking algorithm is robust to erratic mouse behavior – we never lose a mouse. For instance, we follow mice that jump, drop from the ceiling, and make quick turns and accelerations that are not fit by our simple dynamics model (see Figure 6.4(a-c)).
- Two contours never fit the same mouse.
- Our algorithm is rarely distracted by background clutter. This implies that our feature extraction methods and the blob and contour combination provide robust observation likelihoods. The only exceptions are when *both* algorithms make mistakes: when the blob tracker mistakes shaded bedding for foreground and the contour tracker fits to the edge of a tail (see Figure 6.4(d) for an example).
- Perhaps the most impressive result is that our algorithm accurately tracks the mice through 7 out of 11 occlusions and partway through the other 4. This is because of the detailed fit provided by the contour tracking algorithm and its ability to use features available during occlusions. Example successful frames are shown in Figure 6.5(a-c).
- In general, our algorithm usually found very good contour fits outside of occlusions, much better than those obtained using contour tracking alone.

Our algorithm has a couple of failure modes which we plan on addressing in future work. First, it occasionally gets stuck in local optima in which the contour fit was facing the wrong direction (see Figure 6.4(d)). We plan to address this problem with a better model of the probability of the direction changing. Second, our algorithm swaps identity labels in four occlusions (see Figure 6.6). The reason for this is that the fit of our algorithm is heavily biased by the fit of the BraMBLe algorithm. For occlusions in which the contour observation signals are weak, this bias from BraMBLe can dominate. We propose a solution to this in Section 6.9.4.

For comparison, we also implemented a combined blob-contour tracking algorithm which performed two steps of sampling instead of three, resulting in the final importance function

$$q'(\mathbf{x}_t|\mathbf{y}_{1:t}) = \int p(\mathbf{x}_t|\mathbf{b}_t)p(\mathbf{b}_t|\mathbf{y}_{1:t-1,bt})d\mathbf{b}_t. \tag{6.88}$$

This algorithm has the disadvantage that samples of all k mice are weighted by the product of the contour likelihood for each mouse. Thus, the number of samples fitting blobs is the same, but the effective number of samples used when fitting contours is much smaller. We tested this algorithm on 500 frames with 6 occlusions (our algorithm works on 4). The results fit our theory. While this algorithm was resistant to drift (blob tracking is the same in both algorithms), the contour fits found were less satisfactory. In general, they were less fine-tuned and more variable from frame to frame, suggesting that more samples are needed. This is particularly evident during occlusions. The fits during occlusions are less fine-tuned to the contour data, and therefore more influenced by the blob tracking results. This causes worse fits in every occlusion in the sequence. This algorithm swapped identities twice more than the algorithm proposed in this paper.

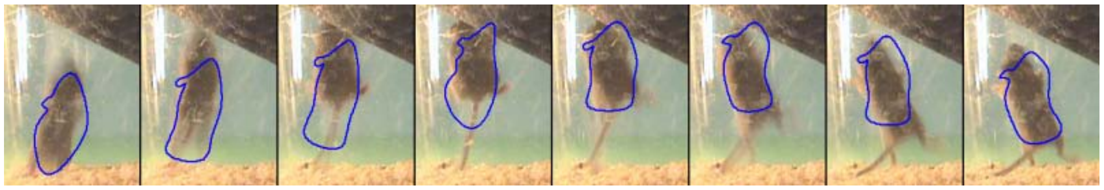
6.9.4 Conclusions and Future Work

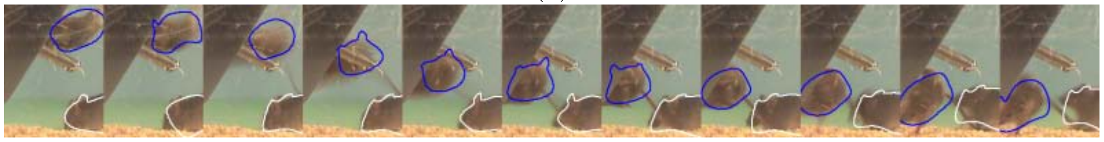
Our algorithm combines the BraMBLe likelihood [57] with the "generic contour likelihood" [78] to utilize a robust set of features necessary for our noisy, real world mouse video. Our algorithm breaks each iteration of particle filtering into three steps, each incorporating a subset of the observations to gradually hone in on the small space in which most mass of the posterior density lies. This can also be seen as modifying the importance function of the particle filtering algorithm to be the product of the posterior marginal densities, thus incorporating the current observation in the importance function. This dramatically reduces the number of samples necessary for accurate tracking.

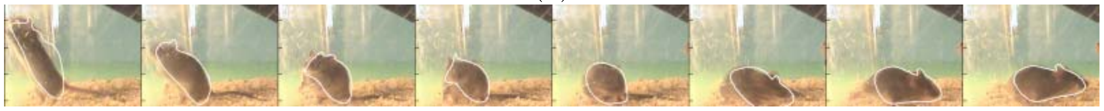
In chapter 7, we discuss possible modifications to this algorithm that use information from both the past and the future to determine the positions of the mice during an occlusion. We hope that this will solve the main failure of the algorithm proposed in this algorithm by making BraMBLe return a more global fit.

6.10 Acknowledgements

Portions of this chapter are based on “Tracking Multiple Mouse Contours (without Too Many Samples)” by K. Branson and S. Belongie [17]. I developed the algorithm, performed the experiments, and wrote the paper.


(a)


(b)


(c)

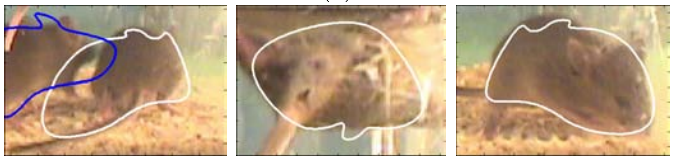
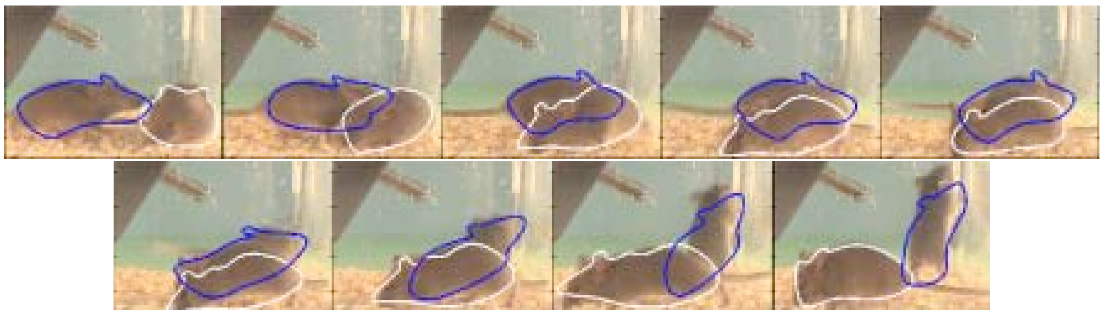
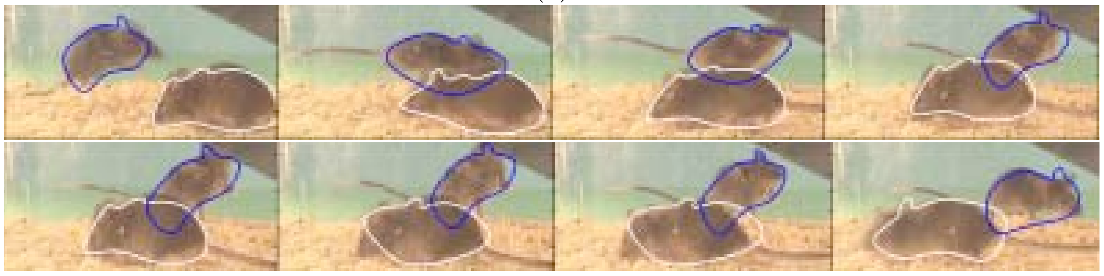

(d)

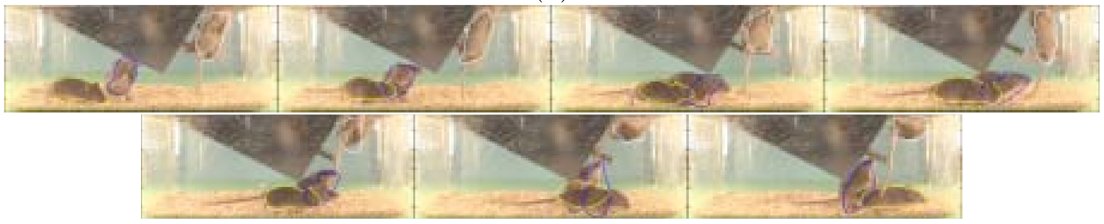
Figure 6.4: (a) Tracking results for a mouse jumping. (b) Tracking results for a mouse falling from the ceiling. (c) Tracking results for a mouse turning quickly. (d) The contour is fit to a tail and the blob is fit to a shadow; the tracker is robust to scratches on the cage; the contour is flipped.



(a)

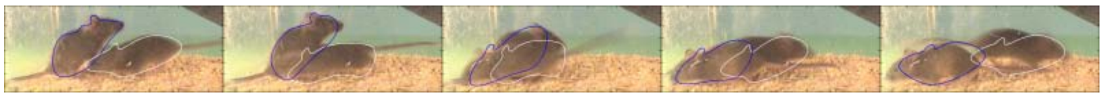


(b)

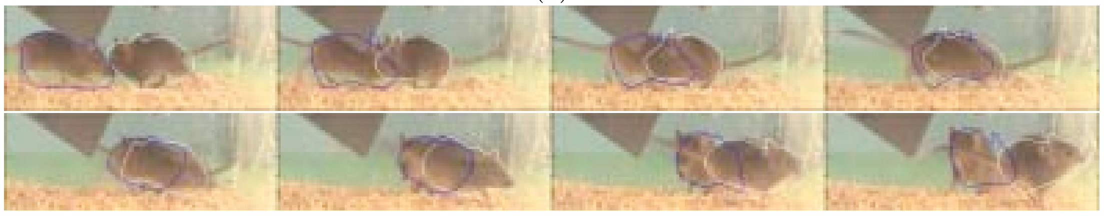


(c)

Figure 6.5: The first three occlusion sequences in which our tracking algorithm performs well.



(a)



(b)

Figure 6.6: The first two occlusion sequences on which our algorithm swaps mouse identities.

7

Acausal Inference Algorithms

In the last chapter, we discussed algorithms that infer statistics of the current state given the previous and current video frames. In many applications, one requires real-time responses to detected activities in the video, thus online inference algorithms are necessary. In applications such automatic health monitoring, it is sufficient to receive results after a few minutes delay. However, because monitoring must be done 24 hours a day, the processing speed must be real-time or near real-time.

In our application, one can wait until the end of difficult occlusions to determine the positions of the targets during the occlusion. Thus, one would like to estimate statistics of the posterior density $p(\mathbf{x}_t|\mathbf{y}_{1:T})$, where t is the current frame and T is the last frame of the occlusion. This is beneficial because it is easy to determine the positions of the mice before the occlusion begins and after the occlusion ends.

In this chapter, we discuss algorithms for acausal inference.

7.1 Affine Flow Plus a Hint

Monte Carlo smoothing algorithms failed because the forward sampling step does not incorporate the future observations. We developed an algorithm that directly incorporates a hint of the future positions of the targets into the forward pass of the algorithm. As discussed previously, the subproblems of foreground/background classification and tracking for separated mice (mice that are neither occluded or occluding) are fairly simple. The subproblem remaining is to track the mouse identities while they are occluding one another. Given the foreground/background labeling, this subproblem reduces to assigning membership of each foreground pixel to the mouse identities.

As segmenting the individual mice is more difficult when a frame is viewed out of context of its neighboring frames, we incorporate a cue from the surrounding sequence of frames. Using a depth ordering heuristic, we associate the front mouse at the start of an occlusion event with the front mouse at the end of an occlusion event. This correspondence holds much of the time because the targets cannot pass through each other. While one mouse occludes another, their depth ordering usually cannot change, as observed in [79]. We predict mouse identity labels for each frame sequentially. However, when labeling a frame during an occlusion event, we incorporate the hint of the future location of the mice in addition to the predicted labels of the previous frames.

Because this correspondence guess is necessary in our occlusion reasoning, images can be processed at frame-rate, but tracking results for occluded or occluding mice are delayed until the end of the occlusion event.

Our approach breaks the tracking task into the subproblems of background/foreground classification, tracking separated mice and tracking through occlusion. First, the pixels in each frame are classified as either foreground or background. Next, the membership of the foreground pixels in every frame is assigned using a simple tracking algorithm without occlusion reasoning. The mouse models computed in this step are used to detect the starts and ends of occlusion events. At the end of an occlusion event, the membership of the foreground pixels in the frames of the occlusion event are reassigned using our occlusion reasoning algorithm. The modules used to solve each of these subproblems are described below.

7.1.1 Background/Foreground Labeling

In our implementation, we used a slightly different background/foreground classification algorithm than that described in Section 4.2. This algorithm did not work as well as the previously described algorithm. However, results were sufficient for this pilot experiment. Next, we describe the simple background/foreground classification we used. This algorithm was sub-optimal, but did not corrupt our results.

To classify pixels in every frame as background or foreground, we model the background using a modified temporal median. The absolute difference of the current frame and the current background estimate is thresholded. Background pixels are those below the threshold and foreground pixels are those above the threshold.

Standard background estimation using a temporal median estimates the background intensity of each pixel as the median of the previous n_B intensities of that pixel.

This technique fails when more of the previous n_B intensities of a pixel correspond to foreground than background. As it often occurs that a mouse (e.g. a sleeping mouse) is still for many frames at a time, we include an additional component depending on the color of the pixel. Each pixel is classified as either mouse color or not mouse color. The intensity of each pixel classified as mouse color is only added to the pixel's intensity history if that pixel has always been classified as mouse color.

To classify the pixels in a frame as mouse color or not, the pixels are segmented into a set number of clusters (we used 3) based on color using k-means. In order to determine which color cluster(s) correspond to mouse color (we assumed only one cluster was mouse color), we use the previous background estimate to classify pixels as background and foreground. The mouse color cluster(s) are the mode color cluster(s) of the foreground pixels.

7.1.2 Separated Target Tracking

In all frames, whether or not there is occlusion, the distribution of the pixel locations of each of the k mice identities is modeled as a bivariate Gaussian. In a frame in which the mice are not occluding each other, the problem of assigning mouse membership to each of the foreground pixels is that of fitting a mixture of k Gaussians to the locations of the foreground pixels. We use the EM algorithm to estimate the mean vectors and (full) covariance matrices of GMM [51]. As the inter-frame motion is small, only a few iterations of EM are necessary when we use the parameters of the GMM at frame $t-1$ to initialize the EM fit at frame t.

7.1.3 Detection of Occlusion Events

The goal of this module is to determine the starts and ends of occlusion events, as well as which mice and foreground pixels are involved. Occlusion events are detected using the GMM parameters computed using EM.

The Fisher distance in the x direction between each pair of target distributions is thresholded to determine when one mouse of the pair is occluding the other. We use only the x distance because all mice are resting on the floor and the x location estimates of our GMM are more stable. The Fisher distance in x between a pair of distributions is defined as

$$J_F[\mu_{x1}, \sigma_{x1}^2, \mu_{x2}, \sigma_{x2}^2] = \frac{(\mu_{x1} - \mu_{x2})^2}{(\sigma_{x1}^2 + \sigma_{x2}^2)/2}, \tag{7.1}$$

where μ_{x1} and μ_{x2} are the x-coordinates of the distributions' means and σ_{x1}^2 and σ_{x2}^2 are the variance in x of the distributions [51]. Because the units of σ_x and μ_x are the same, the Fisher distance is unitless and a constant threshold (we chose 8.0) is used.

7.1.4 Tracking through Occlusion

When an occlusion event is detected, the membership of each foreground pixel in the occlusion must be assigned. The front mouse is tracked through the occlusion by combining several cues: the pixel intensities, the foreground classification, a depth-order heuristic, and the assumed bivariate Gaussian distribution of the pixels owned by a mouse (Section 7.1.2). Given an estimate of the pixels belonging to each mouse in frame t, we compute the "best" affine transformation describing the motion of that mouse from frame t to $t+1$. Given these affine motion estimates, the pixels belonging to each mouse identity in frame $t+1$ are estimated.

In Section 7.1.4 and 7.1.4, we will describe our criterion for the "best" affine motion and how it is optimized. In Section 7.1.4, we will describe how the pixel memberships are estimated.

Optical Flow Computation

Our algorithm for tracking through occlusion is based on optical flow estimation using multiple affine models. Consider the set of pixels belonging to one mouse. We assume that the Horn-Schunck brightness constancy condition (see Section 4.8 holds within this set of pixels, so that

$$I_x u + I_y v + I_t = 0 \tag{7.2}$$

Here, $I(x, y, t)$ denotes the intensity at location $(x, y)^\top$ and time t, the subscript denotes partial differentiation and u and v are the x and y components of the flow at (x, y). As in [54], we use an affine model for the flow of the form

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} a_1 + a_2 x + a_3 y \\ a_4 + a_5 x + a_6 y \end{pmatrix}. \tag{7.3}$$

As discussed in Section 4.8.3, in the least-squares sense, the best $\mathbf{a}$ given only the optical flow cue minimizes

$$H_0[\mathbf{a}] = \sum_{(x,y)\in\mathcal{M}} w(x,y)(\mathbf{z}^\top \mathbf{a} + I_t)^2, \tag{7.4}$$

where $\mathcal{M}$ is the set of pixels belonging to the mouse, the vectors $\mathbf{z}$ and $\mathbf{a}$ are defined as

$$\begin{aligned}\mathbf{z} &= (I_x, I_x x, I_x y, I_y, I_y x, I_y y)^\top,\\ \mathbf{a} &= (a_1, ..., a_6)^\top\end{aligned}$$

and $w(x,y)$ is a measure of the certainty that pixel (x,y) at time t is in $\mathcal{M}$.

Because of the high amount of occlusion and the lack of features on the targets, the optical flow cue alone is not enough to get an accurate motion estimate. We thus add a hint of the future mouse locations in the form of a quadratic regularization term, which nudges the estimate $\mathbf{a}$ toward the prior affine motion estimate $\hat{\mathbf{a}}$, to be discussed in Section 7.1.4. We use the term "prior" because of the close relation of this regularization term to an assumed prior distribution on $\mathbf{a}$ [51]. The strength of this nudge, for each component of $\mathbf{a}$, is defined by the 6×6 matrix $\lambda\Sigma_a^{-1}$. The scalar λ sets the weight of the regularization penalty relative to the optical flow estimate. We use $\lambda = 0.0001$. The matrix Σ_a is a measure of the relative weights of the regularization for each of the individual entries of $\mathbf{a}$. We take Σ_a to be diagonal. Each entry corresponds to our guess of the amount of variance in the corresponding entry of $\mathbf{a}$. With this regularization term, our new criterion is

$$H[\mathbf{a}] = \sum_{(x,y)\in\mathcal{M}} w(x,y)(\mathbf{z}^\top \mathbf{a} + I_t)^2 + \lambda(\mathbf{a} - \hat{\mathbf{a}})^\top \Sigma_a^{-1}(\mathbf{a} - \hat{\mathbf{a}}). \tag{7.5}$$

Taking the partial derivative of $H[\mathbf{a}]$ with respect to $\mathbf{a}$, setting it to zero, and solving for $\mathbf{a}$, we find that

$$\mathbf{a} = (Z^\top W Z + \lambda\Sigma_a^{-1})^{-1}(-Z^\top W \mathbf{I}_t + \lambda\Sigma_a^{-1}\hat{\mathbf{a}}), \tag{7.6}$$

where Z is the $|\mathcal{M}| \times 6$ matrix with rows $\mathbf{z}^\top$, W is a $|\mathcal{M}| \times |\mathcal{M}|$ diagonal matrix of the weights w, and $\mathbf{I}_t$ is a length $|\mathcal{M}|$ vector of the I_t.

Note the following special cases:

$$\mathbf{a} = -(Z^\top W Z)^{-1} Z^\top W \mathbf{I}_t \qquad \text{as} \quad \lambda \to 0 \tag{7.7}$$

which optimizes $H_0[\mathbf{a}]$, and

$$\mathbf{a} = \hat{\mathbf{a}} \qquad \text{as} \quad \lambda \to \infty \tag{7.8}$$

in which the $\mathbf{a}$ is chosen without regard to the optical flow computation.

Note that the affine motion is estimated only for pixels labeled as unoccluded, while the mean and variance of the location of the mouse correspond to all pixel locations, occluded and unoccluded. It is only safe to assume that the affine transformation for the visible part of the mouse equals the affine transformation for the entire mouse if a significant portion of the mouse is observable. While the weight of the optical flow term in the form above is proportional to the number of unoccluded pixels, we found that this weight does not degrade fast enough. We thus ignore the optical flow estimate if more than some fraction (we chose 0.7) of the mouse is occluded, and rely only on the affine motion prior $\hat{\mathbf{a}}$.

Prior Estimation

Next, we discuss the choice of $\hat{\mathbf{a}}$ for each mouse in each frame of the occlusion. As mentioned before, $\hat{\mathbf{a}}$ can be interpreted as the mean of the prior distribution on $\mathbf{a}$. To estimate $\hat{\mathbf{a}}$, we use only the depth ordering cue, though other cues could be incorporated. The depth ordering cue is an estimate of which blob is in front at the start of the occlusion and which blob is in front at the end of the occlusion. As these blobs must correspond to the same mouse, we reason that during the occlusion event, the succession of frame to frame motions must transform the initial front mouse to the final front mouse. We cannot assume that the back mice do not change depth ordering with respect to each other during the occlusion. Instead, we assume that the blob of all the back mice at the start of the occlusion corresponds to the blob of all the back mice at the end of the occlusion. We set the prior estimates for each of the back mice to be all the same.

While many more sophisticated interpolations exist, we found that linearly interpolating the affine motion worked well. To describe the interpolation, we will break the affine motion into two parts,

$$A = \begin{pmatrix} a_2 & a_3 \\ a_5 & a_6 \end{pmatrix}, \qquad \mathbf{t} = (a_1, a_4)^\top \tag{7.9}$$

If the displacement (u, v) is computed in the coordinate system centered on $\boldsymbol{\mu}$ and the pixel locations $\mathbf{p}$ belonging to a mouse follow the normal distribution $\mathcal{N}(\boldsymbol{\mu}, \Sigma)$, then the

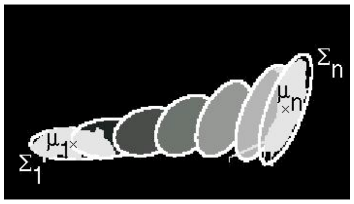


Figure 7.1: An example showing how the mean and covariance of the mouse on the left is linearly interpolated into the mean and covariance of the mouse on the right, using our algorithm for linear interpolation. The leftmost ellipse corresponds to $(\boldsymbol{\mu}_1, \Sigma_1)$ and the rightmost ellipse corresponds to $(\boldsymbol{\mu}_n, \Sigma_n)$. The affine prior $\hat{\mathbf{a}}$ transforms any of these ellipses to the ellipse on its right.

transformed locations $\mathbf{p}' = \mathbf{p} + (u, v)^\top$ follow the normal distribution $\mathcal{N}(\boldsymbol{\mu}', \Sigma')$, where

$$\boldsymbol{\mu}' = \boldsymbol{\mu} + \mathbf{t}, \qquad \Sigma' = A\Sigma A^\top. \tag{7.10}$$

We first compute the transformation $(A_{1:n}, \mathbf{t}_{1:n})$ that transforms the mouse in the first frame of the occlusion event, described by $\mathbf{p}_1 \sim \mathcal{N}(\boldsymbol{\mu}_1, \Sigma_1)$, to the mouse in the last frame of the occlusion event, described by $\mathbf{p}_n \sim \mathcal{N}(\boldsymbol{\mu}_n, \Sigma_n)$, where n is the number of frames in the occlusion event. Any pair in the family

$$\mathbf{t}_{1:n} = \boldsymbol{\mu}_n - \boldsymbol{\mu}_1$$
$$A_{1:n} = \Sigma_n^{1/2} O^\top \Sigma_1^{-1/2}$$

where O is an arbitrary orthogonal matrix will perform the desired transformation [43]. We set the matrix O equal to the identity because the next step requires $A_{1:n}$ to be positive semidefinite. We estimate the prior transformation relating each pair of adjacent frames by

$$\hat{\mathbf{t}} = \frac{\mathbf{t}_{1:n}}{n-1}, \qquad \hat{A} = A_{1:n}^{1/(n-1)}. \tag{7.11}$$

Thus, $\mathcal{N}(\boldsymbol{\mu}_n, \Sigma_n)$ is the result of incrementally applying the transformation $(\hat{A}, \hat{\mathbf{t}})$ to $\mathcal{N}(\boldsymbol{\mu}_1, \Sigma_1)$ $n-1$ times. Figure 7.1 shows an example linear interpolation of the affine parameters.

There are many other ways to estimate $\hat{\mathbf{a}}$; this method was chosen for its simplicity of implementation. Other linear interpolations exist because there are other ways of parameterizing the Gaussian distribution. For example, we could instead search for the transformation $\hat{\mathbf{a}}$ that contains as little scaling as possible. Instead of fitting a line to the mouse parameters at the start and end of the occlusion event, we could fit a spline.

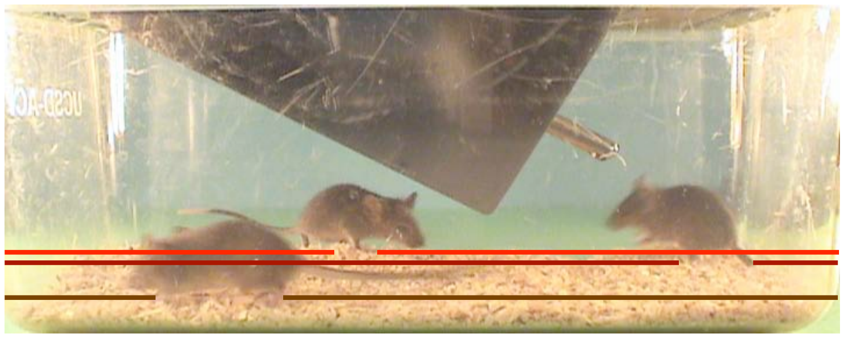


Figure 7.2: The depth-ordering of the mice is evident based on the lowest pixel belonging to each mouse.

This spline would be influenced by heuristics that estimate the likelihood of each parameterization at each frame in the occlusion event. We would then estimate $\hat{\mathbf{a}}_{t:t+1}$ as the transformation that takes the model along the spline at frame t to the model along the spline at frame $t+1$. We plan on exploring alternatives to our linear interpolation in future work.

Estimating the correspondence between mice at the start and end of an occlusion event relies on a very simple heuristic to compute the depth ordering. The front mouse is the mouse that owns the pixel with the lowest (largest) y-coordinate. This is true in any unoccluded environment in which the floor is visible and the camera is above the floor. This depth cue is evident in the example in Figure 7.2. Because this estimate relies heavily on the noisy foreground classification, we used the lowest (largest) y-coordinate in the past 5 frames as a depth estimate.

Pixel Membership Estimation

Given the estimates of the affine motions transforming the mice at frame t to the mice at frame $t+1$, the foreground pixels belonging to each mouse in frame $t+1$ are estimated (see Section 4.8.5). The pixels belonging to one mouse should be similar in both motion and location, as motivated in [121]. In order to incorporate both motion and location, we assign membership based on the weighted sum of proximity and motion similarity. In future work, we plan to incorporate more complex models such as [60]. However, this simple approach worked sufficiently well to justify the incorporation of the affine hint.

We estimate the mean and variance of the location of a mouse at frame $t+1$ by applying the computed affine transformation to the estimated mean and variance of the location of the mouse at frame t. The proximity criterion for a pixel at location $\mathbf{p}$ is

$$J_l[\mathbf{p}] = (\mathbf{p} - \boldsymbol{\mu}_{t+1})^\top \Sigma_{t+1}^{-1} (\mathbf{p} - \boldsymbol{\mu}_{t+1}). \tag{7.12}$$

We also compute the local optical flow for each foreground pixel. For this, we use Lucas-Kanade with a Gaussian window with standard deviation 2.0. Note that this re-uses the spatiotemporal derivatives used in the affine flow estimation. The optical flow of each pixel in a mouse should be similar to the regional optical flow of the entire mouse. The motion similarity term for a pixel $\mathbf{p}$ with motion $(u_{local}, v_{local})^\top$ is

$$J_m[\mathbf{p}] = \lambda_{local}[(u_{local} - a_1)^2 + (v_{local} - a_4)^2]. \tag{7.13}$$

Our total cost function is therefore

$$J[\mathbf{p}] = J_l[\mathbf{p}] + J_m[\mathbf{p}]. \tag{7.14}$$

Each relevant foreground pixel is assigned to the mouse with the lowest summed location and motion similarity terms.

Tracking the Back Mice

To track the back mice, we reapply the algorithm to just the back mice. Because the back mice might be occluded by the front mouse at the start or end of their occlusions, the depth ordering heuristic is much less reliable. We do not use this heuristic if any of the back mice in the occlusion are significantly occluded by the front mouse. Instead, we use $\hat{\mathbf{a}} = \mathbf{0}$, thus the regularization term shrinks the optical flow estimates.

7.1.5 Parameter Sensitivity and Computational Considerations

We have mentioned the parameter settings we used in our experiments throughout this section. These parameters weight the different terms in our optimization. The parameter λ, the weight of the prior term in the flow estimation, was set to 10^{-4}. However, the algorithm is not particularly sensitive to this parameter. Values in the range 10^{-5} to 5×10^{-4} produced similar results. This insensitivity and the small size of effective λ settings is due to the ability of the affine flow estimation stage to rely solely on the prior estimate when too much of the mouse is occluded. In future work, we will experiment with a dynamically computed λ based on an estimate of the reliability of the optical flow cue.

The parameter λ_{local} is the weight of the motion criterion in the mask estimation. It was set to 1.0, but values between 0.25 and 2 produced similar results. As λ_{local} corresponds to the inverse of the variance of u_{local} and v_{local}, dynamic estimation may also work for this parameter.

The matrix Σ_a was set to diag$\{1, 0.1, 0.1, 1, 0.1, 0.1\}$. As Σ_a corresponds to the relative variance of **a**, we plan to fit Σ_a from actual data.

The running time of our occlusion reasoning module is linear in the total number of foreground pixels in the frames of the occlusion event. While our current implementation is in Matlab$^{\text{TM}}$, we believe that a more efficient implementation of this module will run in real time. Currently, the occlusion reasoning takes about 0.642 seconds per frame in our experiments with three mice on a 2.4 MHz Pentium 4. Approximately 0.172 seconds of this time is involved in affine flow estimation, 0.0529 seconds of this time is involved in mask estimation, and the rest of the time is overhead from parameter passing in functions.

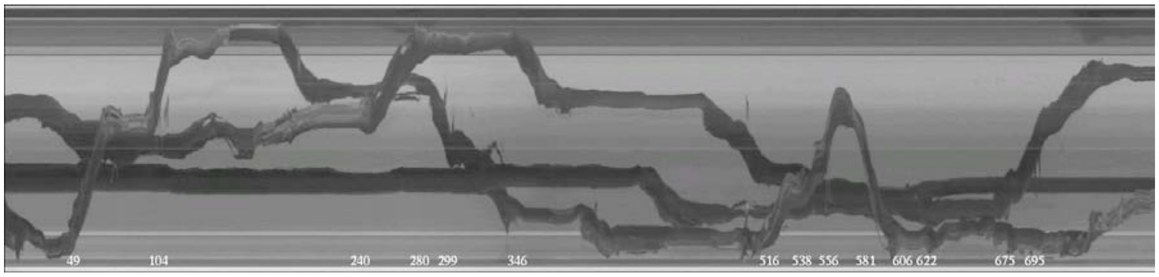
7.1.6 Experiments

We report the initial success of our algorithm in tracking three mice in a cage.

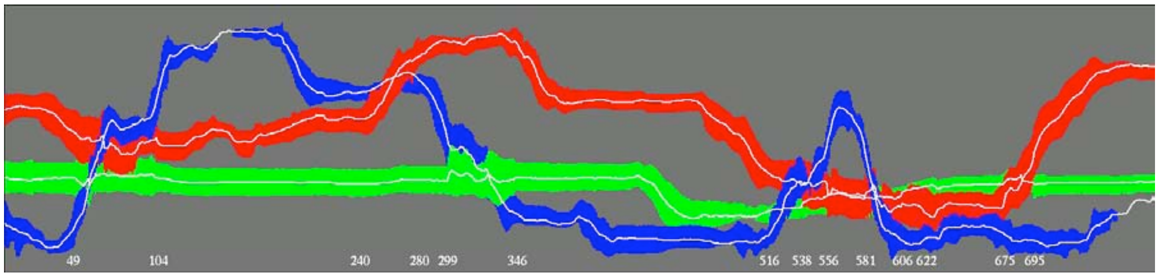
We tested our algorithm on a 1000-frame video sequence taken from the side of a Static Micro-Isolator$^{\text{TM}}$cage containing three adolescent mice. The first 200 frames were used for background initialization and tracking was started at frame 225, in which there was no occlusion. Figure 7.2 shows an example frame from this sequence. The cage is made of a translucent plastic and is approximately the size of a shoe box. At the top of the cage is a metal container with food pellets in one half and a water bottle in the other half. The camera was positioned slightly above the level of the mouse floor. Because of reflections in the table and the top of the cage, we cropped each initially 240×360 pixel frame at the top grate (row 60) and the bottom of the cage (row 193), resulting in 134×360 pixel frames. The video was recorded at 30 frames/second and compressed into Windows Media Video (wmv) format.

Our results are summarized in Figures 7.3 and 7.4. For purposes of visualization, we show in Figure 7.3(a) a single scanline of the image (row 96 of the cropped image) at every frame of the video sequence. Row 96 was chosen as it passes through the middle of the mice. The x-axis of this image is time and the y-axis is actually the x-axis of an original frame. Each dark path from left to right in this (t, x) image is the path a single mouse takes through the sequence; notice there are three paths.

Let us call the mouse that starts at the top of this (t, x) image mouse 1, the mouse



(a) (t, x) raw image data (360×776 pixels): a single scanline of the image at every frame.



(b) (t, x) predicted image (360×776 pixels): membership of points in a scanline of the image at every frame.

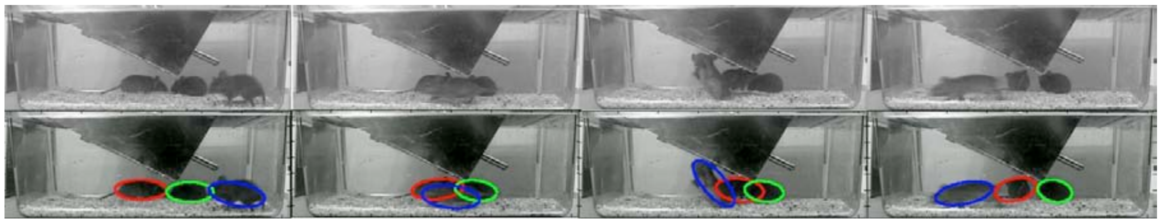
Figure 7.3: **Tracking results** (t, x) plot of results. The x-axis in these images is time and the y-axis is the x-axis of the original frame. Each column corresponds to the same scanline of a different frame.

that starts in the middle mouse 2, and the mouse that starts at the bottom mouse 3. In this sequence, the mice begin unoccluded, then in frame 49 they all move together and mouse 3 passes in front of the other two. In frame 240, mouse 1 passes in front of mouse 3. In frame 299, mouse 3 passes behind mouse 2. In frame 516, all the mice again come together and mouse 3 passes in front, then turns around and again passes in front of them in frame 581. Meanwhile, in frame 538 mouse 1 passes in front of mouse 2. Finally, in frame 675, mouse 1 passes in front of mouse 2.

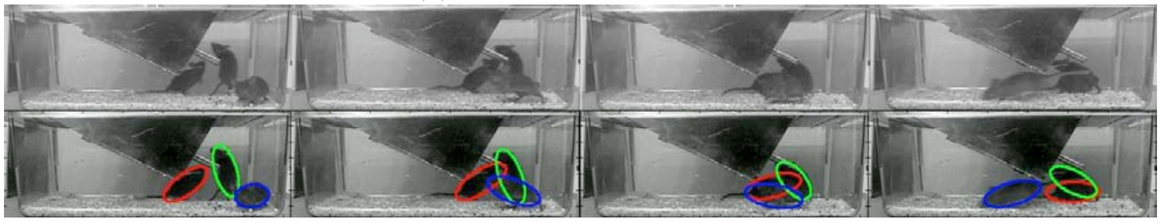
In Figure 7.3(b), we show the labels estimated by our algorithm in this (t, x) format. At each frame, each point along the scanline (row 96) within two standard deviations of a mouse is plotted with a color corresponding to that mouse's label. That is, we plot point (t, x) with a color corresponding to mouse m if

$$((x, r)^\top - \boldsymbol{\mu}_{mt})^\top \Sigma_{mt}^{-1}((x, r)^\top - \boldsymbol{\mu}_{mt}) \leq 2, \tag{7.15}$$

where $\boldsymbol{\mu}_{mt}$ and Σ_{mt} are the estimated parameters of mouse m at frame t and r is the scanline row, 96. Thus, each path of one color is the estimated path of a mouse. There are two breaks in the path of mouse 3; these occur because two standard deviations of



(a) Frames 49, 64, 80, 104



(b) Frames 516, 525, 539, 556.

Figure 7.4: Example frames showing the raw image frames from an occlusion event in the top row and the Gaussian parameters estimated by our algorithm. The ellipses correspond to 2 standard deviations of the Gaussians.

the estimated Gaussian does not intersect the chosen scanline (because of errors in the foreground classification). We also plot the x-component of the centers of the estimated Gaussians in white. For points predicted to belong to multiple mice, we plot the color of the mouse predicted to be in front.

In this 776 frame sequence, the identities of the mice are never switched. In fact, the estimated mouse paths match the actual mouse paths very closely. In Figure 7.4, we show some example image frames and the mouse parameters estimated by our algorithm for two occlusion events.

7.1.7 Discussion and Conclusion

We have presented a collection of modules that combine many cues to track identical, non-rigid, featureless objects through severe occlusions. The novel module is the occlusion tracking module. This module uses a depth ordering heuristic to match up the front mouse at the start of an occlusion with the front mouse at the end of the occlusion. This correspondence is used as a hint that is combined with the optical flow cue. Because of the nature of our targets and the high amount of occlusion, a single frame out of context can have multiple fits that seem equally good. The “premonition” of the final location of the mouse gives our algorithm a way of deciding between these equally good fits. This module is thus a step in the direction of an algorithm that reasons both forward and backward in time.

Mouse identity was tracked without error in all 776 frames, despite noise in the foreground classification, occlusion detection, and depth estimation. We have thus presented an approach for assigning unique identity labels to mice through occlusions that can work with suboptimal modules.

In our experiments, these imperfections did not corrupt the occlusion reasoning module because the true depth ordering of the mice does not change rapidly. Our algorithm combines numerous cues, thus in our experiments one module's failure does not cause the entire algorithm fail. For example, in one case the foreground classification was poor (it missed the feet of a mouse), which led to an incorrect estimate of the depth ordering in an occlusion event. As this occlusion was not complete, there were sufficient intensity cues to overcome the incorrect depth correspondence. The algorithm succeeded despite the incorrect depth estimate.

Our algorithm did not fail on the sequence we tested because all the modules never failed simultaneously. There are cases in which our algorithm would fail. One can imagine a scenario in which an occlusion is detected too early when in fact the mice are just next to one another. The mice can then change depth ordering. Or, as the mice are flexible, one mouse may bend around another mouse (for instance if one mouse climbs over another mouse). Their depth ordering will therefore change during an occlusions. The latter case is rare. For these reasons, in future work we plan to incorporate more acausal cues, e.g. behavioral and appearance cues, to make the estimate more robust. This algorithm can also fail if the depth ordering is not estimated correctly. We would also to develop a more robust depth estimation heuristic that does not rely so heavily on foreground classification. This can include a smoothed estimate that uses the assumption that depth changes slowly as well as occlusion junction detection [82] when our depth ordering heuristic fails (e.g. for back mice).

In conclusion, the major contribution of this work is a method for tracking indistinguishable, featureless targets through occlusion events by combining multiple cues in an acausal fashion throughout the duration of each occlusion event.

7.2 Monte Carlo Smoothing

Monte Carlo smoothing is a sampling-based method for estimating the distribution of the hidden states $\mathbf{x}_{0:T} = (\mathbf{x}_0, ..., \mathbf{x}_T)$ of a Markov chain, given the observations, $\mathbf{y}_{1:T} =$

$(\mathbf{y}_1, ..., \mathbf{y}_T)$. There are two goals addressed in the literature:

- Estimating the distribution of all the hidden states, $p(\mathbf{x}_{0:T}|\mathbf{y}_{1:T}$.
- Estimating the marginal distribution of each state, $p(\mathbf{x}_t|\mathbf{y}_{0:T})$.

To do this, existing algorithms rely on filtering algorithms which estimate $p(\mathbf{x}_t|\mathbf{y}_{1:t})$. Smoothing algorithms take advantage of the chain structure of the distribution and factor the distribution into parts it can estimate using filtering. In this section, we review a number of Monte Carlo smoothing algorithms.

7.2.1 Smoothing by Storing the State Vector

The goal in this section is to estimate the distribution $p(\mathbf{x}_{0:T}|\mathbf{y}_{1:T})$ by a set of particles $\{\mathbf{x}^{(j)}_{0:T|T}\}_{j=1}^N$. This set of particles is found recursively. While we are only interested in estimating $p(\mathbf{x}_{0:T}|\mathbf{y}_{1:T})$, we estimate $p(\mathbf{x}_{0:t}|\mathbf{y}_{1:t})$ for all $t = 0, 1, ..., T$. We use a recursive decomposition of $p(\mathbf{x}_{0:t}|\mathbf{y}_{1:t})$ that involves $p(\mathbf{x}_{0:t-1}|\mathbf{y}_{1:t-1})$:

$$p(\mathbf{x}_{0:t}|\mathbf{y}_{1:t}) = \frac{p(\mathbf{y}_t|\mathbf{x}_{0:t}, \mathbf{y}_{1:t-1})p(\mathbf{x}_{0:t}|\mathbf{y}_{1:t-1})}{p(\mathbf{y}_t|\mathbf{y}_{1:t-1})} \tag{7.16}$$

$$= \frac{p(\mathbf{y}_t|\mathbf{x}_t)p(\mathbf{x}_{0:t}|\mathbf{y}_{1:t-1})}{p(\mathbf{y}_t|\mathbf{y}_{1:t-1})} \tag{7.17}$$

$$= \frac{p(\mathbf{y}_t|\mathbf{x}_t)p(\mathbf{x}_t|\mathbf{x}_{0:t-1}, \mathbf{y}_{1:t-1})p(\mathbf{x}_{0:t-1}|\mathbf{y}_{1:t-1})}{p(\mathbf{y}_t|\mathbf{y}_{1:t-1})} \tag{7.18}$$

$$= \frac{p(\mathbf{y}_t|\mathbf{x}_t)p(\mathbf{x}_t|\mathbf{x}_{t-1})p(\mathbf{x}_{0:t-1}|\mathbf{y}_{1:t-1})}{p(\mathbf{y}_t|\mathbf{y}_{1:t-1})}. \tag{7.19}$$

To implement this recursive decomposition in terms of particle set estimates, we assume that $p(\mathbf{x}_t|\mathbf{x}_{t-1})$ is easy to sample from, that we can evaluate $p(\mathbf{y}_t|\mathbf{y}_{1:t-1})$, and that we have an unweighted particle set $\{\mathbf{x}^{(j)}_{0:t-1|t-1}\}_{j=1}^N$ that estimates the distribution $p(\mathbf{x}_{0:t-1}|\mathbf{y}_{1:t-1})$. For each particle $j = 1, ..., N$, we generate a new particle $\mathbf{x}^{(j)}_{t|t-1}$ according to

$$p(\mathbf{x}_t|\mathbf{x}_{t-1} = \mathbf{x}^{(j)}_{t-1|t-1}). \tag{7.20}$$

We concatenate this new particle to the old particles,

$$\mathbf{x}^{(j)}_{0:t|t-1} = (\mathbf{x}^{(j)}_{0:t-1|t-1}, \mathbf{x}^{(j)}_{t|t-1}). \tag{7.21}$$

The particle set $\{\mathbf{x}^{(j)}_{0:t|t-1}\}_{j=1}^N$ approximates

$$p(\mathbf{x}_{0:t}|\mathbf{y}_{1:t-1}) = p(\mathbf{x}_t|\mathbf{x}_{t-1})p(\mathbf{x}_{0:t-1}|\mathbf{y}_{1:t-1}). \tag{7.22}$$

This is because $p(\mathbf{x}_{0:t-1} = \mathbf{x}^{(j)}_{0:t-1|t-1}|\mathbf{y}_{1:t-1})$ is constant for each j, as is $p(\mathbf{x}_t = \mathbf{x}^{(j)}_{t|t-1}|\mathbf{x}_{t-1} = \mathbf{x}^{(j)}_{t-1})$. To compute a particle set approximating $p(\mathbf{x}_{0:t}|\mathbf{y}_{1:t})$, we weight our particle set $\{\mathbf{x}^{(j)}_{0:t|t-1}\}_{j=1}^{N}$. This requires weighting each particle j by

$$w_t^{(j)} = p(\mathbf{y}_t|\mathbf{x}_t = \mathbf{x}^{(j)}_{t|t-1}), \tag{7.23}$$

resulting in a weighted particle set $\{(\mathbf{x}^{(j)}_{0:t|t-1}, w_t^{(j)})\}_{j=1}^{N}$. In order to have an unweighted particle set, we resample $\{(\mathbf{x}^{(j)}_{0:t|t-1}, w_t^{(j)})\}_{j=1}^{N}$ according to the weights, resulting in an unweighted particle set $\{\mathbf{x}^{(j)}_{0:t|t}\}_{j=1}^{N}$.

The major drawback of this algorithm is the amount of resampling that is performed. Notice that here we are resampling entire sequences. After t_2 iterations, the particles generated in iteration t_1 have been resampled $t_2 - t_1$ times. Experiments in [66] found that for reasonable values of T ($T = 100, 500$), there was a huge amount of degeneracy in the particle set. In his experiment, the set $\{\mathbf{x}^{(j)}_{1|500}\}$ contained only one unique particle.

7.2.2 Smoothing by the Two-Filter Formula

[66] also presents another smoothing formulation, which estimates the marginal $p(\mathbf{x}_t|\mathbf{y}_{1:T})$ for all $t = 1, ..., T$. This algorithm is based on the factorization of $p(\mathbf{x}_t|\mathbf{y}_{1:T})$ into a forward and backward filter:

$$\begin{aligned} p(\mathbf{x}_t|\mathbf{y}_{1:T}) &= p(\mathbf{x}_t|\mathbf{y}_{1:t}, \mathbf{y}_{t+1:T}) && (7.24)\\ &= p(\mathbf{y}_{t+1:T}|\mathbf{x}_t, \mathbf{y}_{1:t-1})p(\mathbf{x}_t|\mathbf{y}_{1:t})/p(\mathbf{y}_{t+1:T}|\mathbf{y}_{1:t}) && (7.25)\\ &\propto p(\mathbf{y}_{t+1:T}|\mathbf{x}_t)p(\mathbf{x}_t|\mathbf{y}_{1:t}) && (7.26) \end{aligned}$$

We call $p(\mathbf{y}_{t+1:T}|\mathbf{x}_t)$ the backward filter and $p(\mathbf{x}_t|\mathbf{y}_{1:t})$ the forward filter. We use any particle filtering algorithm to generate a set of particles $\{(\mathbf{x}^{(j)}_{t|1:t}, w^{(j)}_{t|1:t}\}$ according to $p(\mathbf{x}_t|\mathbf{y}_{1:t})$. We then reweight each particle $(\mathbf{x}^{(j)}_{t|1:t}, w^{(j)}_{t|1:t}$ by $p(\mathbf{y}_{t+1:T}|\mathbf{x}_t = \mathbf{x}^{(j)}_{t|1:t})$. Thus, the support of the smoothing distribution is the same as the support of the forward predicting distribution. To estimate $p(\mathbf{y}_{t:T}|\mathbf{x}_t = \mathbf{x}^{(j)}_{t|1:t})$, we use the following recursive decomposition:

$$\begin{aligned} p(\mathbf{y}_{t+1:T}|\mathbf{x}^{(j)}_{t|1:t}) &= \int p(\mathbf{y}_{t+1:T}, \mathbf{x}_{t+1}|\mathbf{x}^{(j)}_{t|1:t})d\mathbf{x}_{t+1} && (7.27)\\ &= \int p(\mathbf{y}_{t+1:T}|\mathbf{x}_{t+1})p(\mathbf{x}_{t+1}|\mathbf{x}^{(j)}_{t|1:t})d\mathbf{x}_{t+1} && (7.28)\\ &= \int p(\mathbf{y}_{t+2:T}|\mathbf{x}_{t+1})p(\mathbf{y}_{t+1}|\mathbf{x}_{t+1})p(\mathbf{x}_{t+1}|\mathbf{x}^{(j)}_{t|1:t})d\mathbf{x}_{t+1} && (7.29) \end{aligned}$$

To estimate this integral for each $j = 1, ..., N$, we assume that most of the weight of the integral is on the set of particles distributed according to $p(\mathbf{x}_{t+1}|\mathbf{y}_{1:t+1})$, which is $\{\mathbf{x}^{(i)}_{t+1|1:t+1}\}^N_{i=1}$. Then, we approximate the integral by the sum over these particles. As the first term in Equation (7.29) is the weight for the next frame, this is a recursive definition and the weights can be estimated in a backward pass starting at $t = T$ and ending at $t = 1$.

The advantage of this algorithm over the previous algorithm is that it does not suffer from the same particle degeneracy described for Kitagawa's first algorithm. The particles are generated using filtering, and reweighted to estimate the smoothing distribution. The main disadvantage of this algorithm is that samples are proposed using standard particle filtering and then reweighted. If, as is often the case, particle filtering loses modes of the distribution because of the greedy decisions required by the fixed sample size, these modes can never be recovered. Thus, both algorithms suffer if the support of the filtering distribution is very different from the support of the smoothing distribution, and both will work if the support is very similar.

7.2.3 Chen and Lai, 2003

The smoothing algorithm discussed in [21] uses the decomposition

$$p(\mathbf{x}_t|\mathbf{y}_{1:T}) = \frac{p(\mathbf{x}_t|\mathbf{y}_{1:t})p(\mathbf{x}_t|\mathbf{y}_{t+1:T})}{p(\mathbf{x}_t)} \frac{p(\mathbf{y}_{1:t})p(\mathbf{y}_{t+1:T})}{p(\mathbf{y}_{1:T})} \tag{7.30}$$

$$\propto \frac{p(\mathbf{x}_t|\mathbf{y}_{1:t})p(\mathbf{x}_t|\mathbf{y}_{t+1:T})}{p(\mathbf{x}_t)}. \tag{7.31}$$

Using any particle filtering method, we can create a set of samples $\{(\mathbf{x}^{(i)}_{t|1:t}, w^{(i)}_{t|1:t})\}$ to approximate the forward distribution:

$$p(\mathbf{x}_t|\mathbf{y}_{1:t}) \approx \sum_{i=1}^{N} w^{(i)}_{t|1:t}\delta(\mathbf{x}_t, \mathbf{x}^{(i)}_{t|1:t}). \tag{7.32}$$

Assuming we can draw samples from $p(\mathbf{x}_{t+1}|x_t)$, we can draw weighted samples $\{(\mathbf{x}^{(j)}_{t+1|t+1:T}, w^{(j)}_{t+1|t+1:T}\}$ from $p(\mathbf{x}_{t+1}|y_{t+1:T})$ using standard filtering as well:

$$p(\mathbf{x}_{t+1}|\mathbf{y}_{t+1:T}) \approx \sum_{j=1}^{N} w^{(j)}_{t+1|t+1:T}\delta(\mathbf{x}_{t+1}, \mathbf{x}^{(j)}_{t+1|t+2:T}). \tag{7.33}$$

This backward recursion begins with the set of weighted samples $\{\mathbf{x}^{(j)}_{T|T}, w^{(j)}_{T|T}\}$, where $\mathbf{x}_{T|T}^{(j)} = \mathbf{x}_{T|1:T}^{(j)}$ and

$$w_{T|T}^{(j)} \propto \frac{p(\mathbf{y}_T|\mathbf{x}_{T|T}^{(j)})p(\mathbf{x}_{T|T}^{(j)})}{w_{T|1:T}^{(j)}}. \tag{7.34}$$

We must now combine these two filter approximations. Combining the approximations of the terms in equation 7.31, we get the approximation:

$$p(\mathbf{x}_t|\mathbf{y}_{1:T}) \propto \sum_{i=1,j=1}^{N} w_{t|1:t}^{(i)} w_{t+1|t+1:T}^{(j)} \delta(\mathbf{x}_t - \mathbf{x}_t^{(i)}) p(\mathbf{x}_{t|1:t}^{(i)}|\mathbf{x}_{t+1|t+1:T}^{(j)})/p(\mathbf{x}_t^{(i)}). \tag{7.35}$$

This algorithm takes running time $O(TN^2)$ and space $O(TN)$. As in Section 7.2.2, this amounts to a reweighting of the samples from the filtering density, $p(\mathbf{x}_t|\mathbf{y}_{1:t})$, and thus will only be efficient if the smoothing and filtering distributions have similar supports.

7.2.4 Godsill et al., 2002

[49] uses a recursive definition of $p(\mathbf{x}_{t:T}|\mathbf{y}_{1:T})$. For $t = 1$, this is the desired smoothing distribution. Using particle filtering, we can find a particle set representing this distribution for $t = T$, $p(\mathbf{x}_T|\mathbf{y}_{1:T})$. Then, for $t = T, ..., 1$, we use the following recursive decomposition to generate a particle set representing the smoothing density:

$$\begin{aligned} p(\mathbf{x}_{t:T}|\mathbf{y}_{1:T}) &= p(\mathbf{x}_t|\mathbf{x}_{t+1:T}, \mathbf{y}_{1:T})p(\mathbf{x}_{t+1:T}|\mathbf{y}_{1:T}) & (7.36)\\ &= p(\mathbf{x}_t|\mathbf{x}_{t+1}, \mathbf{y}_{1:t})p(\mathbf{x}_{t+1:T}|\mathbf{y}_{1:T}) & (7.37) \end{aligned}$$

We can decompose the first term in equation 7.37:

$$\begin{aligned} p(\mathbf{x}_t|\mathbf{x}_{t+1}, \mathbf{y}_{1:t}) &= \frac{p(\mathbf{x}_{t+1}|\mathbf{x}_t)p(\mathbf{x}_t|\mathbf{y}_{1:t})p(\mathbf{y}_{1:t})}{p(\mathbf{x}_{t+1}, \mathbf{y}_{1:t})} & (7.38)\\ &= \frac{p(\mathbf{x}_{t+1}|\mathbf{x}_t)p(\mathbf{x}_t|\mathbf{y}_{1:t})}{p(\mathbf{x}_{t+1}|\mathbf{y}_{1:t})} & (7.39)\\ &\propto p(\mathbf{x}_{t+1}|\mathbf{x}_t)p(\mathbf{x}_t|\mathbf{y}_{1:t}) & (7.40) \end{aligned}$$

Suppose we have a set of particles $\{\tilde{\mathbf{x}}_{t+1:T}^{(j)}\}$ representing $p(\mathbf{x}_{t+1:T}|\mathbf{y}_{1:T})$, and a set of particles $\{(\mathbf{x}_t^{(j)}, w_t^{(j)})\}$ representing $p(\mathbf{x}_t|\mathbf{y}_{1:t})$. We use the recursive definition in equation 7.40 to generate samples from $p(\mathbf{x}_{t:T}|\mathbf{y}_{1:T})$. To do this, we first generate a particle set $\{(\tilde{\mathbf{x}}_t^{(ij)}, \tilde{w}_t^{(ij)})\}_{i=1}^N$ representing $p(\mathbf{x}_t|\mathbf{x}_{t+1} = \tilde{\mathbf{x}}_{t+1}^{(j)}, \mathbf{y}_{1:t})$. This can be done by reweighting the samples $\{(\mathbf{x}_t^{(j)}\}$ by

$$\tilde{w}_t^{(ij)} = w_t^{(i)} p(\mathbf{x}_{t+1} = \tilde{\mathbf{x}}_{t+1}^{(j)}|\mathbf{x}_t = \mathbf{x}_t^{(i)}).$$

Next, we draw one sample $\tilde{\mathbf{x}}_{t+1:T}^{(j)}$ from $\{\tilde{\mathbf{x}}_{t+1:T}^{(j)}\}$ and one sample $\mathbf{x}_t^{(ij)}$ from $\{(\mathbf{x}_t^{(i)}, \tilde{w}_t^{(ij)})\}_{i=1}^N$. Since $\tilde{\mathbf{x}}_{t+1:T}^{(j)}$ is distributed according to $p(\mathbf{x}_{t+1:T}|\mathbf{y}_{1:T})$ and $\mathbf{x}_t^{(ij)}$ is distributed according to $p(\mathbf{x}_t|\mathbf{x}_{t+1} = \tilde{\mathbf{x}}_{t+1}^{(j)}, \mathbf{y}_{1:t})$, the concatenation of these two samples, $\tilde{\mathbf{x}}_{t:T}^{(j)} = (\tilde{\mathbf{x}}_t^{(ij)}, \tilde{\mathbf{x}}_{t+1:T}^{(j)})$ is distributed according to the product, $p(\mathbf{x}_{t:T}|\mathbf{y}_{1:T})$.

This algorithm takes time $O(N^2T)$ and space $O(TN)$. The authors claim that this algorithm is superior because it generates sample trajectories from the smoothing density without excessive resampling. Like all the other algorithms presented, it has the fault that it relies on the smoothing density being similar to the filtering density. This is because the new sample concatenated is chosen from $\{(\mathbf{x}_t^{(i)}, \tilde{w}_t^{(ij)})\}_{i=1}^N$, which is the filtering samples reweighted. Thus, if the samples in the filtering distribution are not concentrated in the right space, this algorithm will fail.

7.2.5 Monte Carlo Smoothing for Mouse Tracking

We experimented with a variation of the Monte Carlo smoothing algorithms presented in [49] to track the mice through occlusions. Our approach used the blob and contour tracking framework described in Section 6.9.

Empirically, we found that particle filtering did lose modes of the distribution when a mistake was made. Generating samples in a forward pass and reweighting them in a backward pass was therefore not effective.

We then modified [49] to take into account the positions of the targets at the end of the occlusion. Instead of relying on the depth ordering cue, we assumed the states of the targets were known up to a permutation of identity labels. Our approach was to maintain a fixed number of particles for each possible permutation of identity labelings between the mice in the first and last frames. For the three mice, there were $3! = 6$ permutations. If we fix the identities of the mice in the first frame to $(1, 2, 3)$ and we fix the identities of the mice in the last frame to (a, b, c), then the matchings are $(1, 2, 3) = (a, b, c)$, $(1, 2, 3) = (a, c, b)$, ... , $(1, 2, 3) = (c, b, a)$.

Thus, we broke up the posterior into $K! = 6$ hypotheses:

$$p(\mathbf{x}_t|\mathbf{y}_{1:t}) = \sum_{m=1}^{K!} P(\mathbf{x}_{T+1}^m|\mathbf{y}_{1:t}) p(\mathbf{x}_t|\mathbf{y}_{1:t}, \mathbf{x}_{T+1}^m) \tag{7.41}$$

$$p(\mathbf{x}_{t:T}|\mathbf{y}_{1:T}) = \sum_{m=1}^{K!} P(\mathbf{x}_{T+1}^m|\mathbf{y}_{1:T}) p(\mathbf{x}_{t:T}|\mathbf{y}_{1:T}, \mathbf{x}_{T+1}^m). \tag{7.42}$$

Each hypothesis m corresponds to a different permutation of the targets' identities in the final state. We restricted our sampling algorithm to draw $N' = N/K!$ samples for each hypothesis. We then performed the smoothing algorithm of [49] in this stratified sample space. The forward filtering recursion given the hypothesized final state $\mathbf{x}_T^m$ is very similar to standard forward filtering:

$$p(\mathbf{x}_t|\mathbf{y}_{1:t}) \propto \sum_m P(\mathbf{x}_{T+1}^m|\mathbf{y}_{1:t})p(\mathbf{y}_t|\mathbf{x}_t)\int p(\mathbf{x}_t|\mathbf{x}_{t-1},\mathbf{x}_{T+1}^m)p(\mathbf{x}_{t-1}|\mathbf{y}_{1:t-1},\mathbf{x}_{T+1}^m)d\mathbf{x}_{t-1} \quad (7.43)$$

The only differences are (1) the weight of the hypothesis $P(\mathbf{x}_{T+1}^m|\mathbf{y}_{1:t})$ and that the motion model $p(\mathbf{x}_t|\mathbf{x}_{t-1})$ has been replaced by the clamped motion model $p(\mathbf{x}_t|\mathbf{x}_{t-1},\mathbf{x}_{T+1}^m)$. We compute the hypothesis weight as

$$P(\mathbf{x}_{T+1}^m|\mathbf{y}_{1:t}) = \frac{p(\mathbf{y}_t,\mathbf{x}_{T+1}^m|\mathbf{y}_{1:t-1})}{\sum_{m=1}^M p(\mathbf{y}_t,\mathbf{x}_{T+1}^m|\mathbf{y}_{1:t-1})} \quad (7.44)$$

where

$$\begin{aligned} p(\mathbf{y}_t,\mathbf{x}_{T+1}^m|\mathbf{y}_{1:t-1}) &= P(\mathbf{x}_{T+1}^m|\mathbf{y}_{1:t-1})p(\mathbf{y}_t|\mathbf{y}_{1:t-1},\mathbf{x}_{T+1}^m) & (7.45)\\ &= P(\mathbf{x}_{T+1}^m|\mathbf{y}_{1:t-1})\int p(\mathbf{x}_t,\mathbf{y}_t|\mathbf{y}_{1:t-1},\mathbf{x}_{T+1}^m)d\mathbf{x}_t & (7.46)\\ &= P(\mathbf{x}_{T+1}^m|\mathbf{y}_{1:t-1})\int p(\mathbf{y}_t|\mathbf{x}_t)p(\mathbf{x}_t|\mathbf{y}_{1:t-1},\mathbf{x}_{T+1}^m)d\mathbf{x}_t & (7.47)\end{aligned}$$

The backward recursion decomposes similarly. Particle approximations to these recursions are described in Sections 6.9 and 7.2.4.

The main difficulty with this approach was setting the clamped motion model $p(\mathbf{x}_t|\mathbf{x}_{t-1},\mathbf{x}_{T+1})$. If this is a linear Gaussian model, then the effect of the final frame decreases quadratically as t decreases. Thus, for most of the filtering, the effect of the final position is not noticed. At the very end of the occlusion, if tracking has been performed incorrectly, the targets will be pulled to the correct position. We tried a number of different weighting functions varying from exponential to linear. Linear weighting functions had the opposite effect. In cases in which the mouse was still for most of the occlusion, then at the end ran to the other side of the cage, tracking was performed incorrectly. We also attempted algorithms that began filtering at the end and at the beginning with a prior that forced the paths to meet in the middle with the same effect.

The reasons for these difficulties was a lack of accurate estimate of uncertainty. The approach in Section 7 was effective because in frames in which there was enough information for affine flow to be effective, the prior had little effect, while in frames where there was a lack of information, the prior was mainly used. With the appearance likelihoods used, we found no good estimate of the confidence. In future research, we plan to focus on learning confidence estimates from labeled data. In addition, we are also exploring an algorithm that clusters the particles based on their distances to the different proposed final states, and keeps a similar number of particles for each cluster.

7.3 Acknowledgements

Portions of this chapter are based on “Three Brown Mice: See How They Run” by K. Branson, V. Rabaud, and S. Belongie. With insights from S. Belongie, I was responsible for developing the algorithm, performing the experiments, and writing the paper.

8

Conclusions and Future Work

In this thesis, I have addressed the challenging problem of tracking multiple identical mice through severe occlusions. This is a novel problem, not previously addressed by the computer vision community. I have proposed and explored numerous approaches for each part of the tracking problem: choosing a hidden state representation, appearance model, and motion model, and performing inference using the chosen model. I have made significant progress on one of the hardest tracking problems I have encountered. We hope that the approaches described in this thesis are not only useful for mouse tracking, but also for other emerging, novel tracking applications.

My current research focuses on using machine learning techniques to extend the ideas in this thesis.

First, as mentioned in Section 5, we are working on learning a model of the probability of switching from one contour to the next. Our approach involves first learning a tree-structured set of rules for predicting the template class given the continuous parameters of the mouse parameter. This is combined with another tree-structured set of rules for predicting the current class template given the previous class template. Both of these trees take advantage of the hierarchical model learned.

Second, we are exploring methods for learning to combine all the appearance features discussed in Section 4 in an optimal method. Recall that the BraMBLe likelihood, for example, is very peaked. We are exploring methods that smooth this likelihood in an optimal manner, according to manually labeled training data. In more detail, our approach learns a conditional random field [69] in which the feature functions correspond to each of the appearance features discussed in Section 4. We plan to use a variant of gradient tree boosting to learn sets of rules such as, if the BraMBLe likelihood is greater than z, then add z' to the probability.

Finally, we are exploring methods of extending the sampling approach discussed in Section 6.9. One can view this method as a three-tiered resolution pyramid. The lowest tier corresponds to the blob-sampling step, the second lowest tier corresponds to the marginal-contour-sampling step, and the third tier corresponds to the combined observations. We plan to learn many conditional random fields at many different resolutions, where resolution refers to a specific amount of subsampling of each dimension, as well as which features to evaluate and include. We then plan to learn how to combine these conditional random fields in the most efficient manner. This can be done using a shortest-paths dynamic programming algorithm.

Bibliography

[1] E. H. Adelson. Layered representations for image coding. Vision and Modeling Technical Report 181, MIT Media Lab, 1991.

[2] Allentown Inc. Allentown micro-vent system caging. `http://www.allentowninc.com/Plastic_Caging2.htm`.

[3] Y. Altunbasak, P. Eren, and A. Tekalp. Region-Based Parametric Motion Segmentation Using Color Information. *Graphical Models and Image Processing*, 60(1):13–23, 1998.

[4] C. Andrieu, A. Doucet, and E. Punskaya. Sequential Monte Carlo methods for optimal filtering. In A. Doucet, N. de Freitas, and N. Gordon, editors, *Sequential Monte Carlo Methods in Practice*, Statistics for Engineering and Information Science, chapter 4, pages 77–96. Springer, 2001.

[5] S. Avidan. Ensemble tracking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 494–501, 2005.

[6] S. Avidan. SpatialBoost: Adding Spatial Reasoning to AdaBoost. In *Proceedings of the European Conference on Computer Vision*, volume IV, page 386. Springer-Verlag, 2006.

[7] S. Ayer and H. Sawhney. Layered representation of motion video using robust maximum-likelihood estimation of mixture models and mdl encoding. In *Proceedings of the IEEE International Conference on Computer Vision*, page 777, Los Alamitos, CA, USA, 1995. IEEE Computer Society.

[8] T. Bando, T. Shibata, and S. Doya, K. Ishii. Switching particle filters for efficient real-time visual tracking. In *Proceedings of the International Conference on Pattern Recognition*, volume 2, pages 720–723, August 2004.

[9] B. Bascle and R. Deriche. Region tracking through image sequences. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 302–307, 1995.

[10] S. Belongie, J. Malik, and J. Puzicha. Shape context: A new descriptor for shape matching and object recognition. In *Advances in Neural Information Processing Systems*, 2000.

[11] N. Bergman. Posterior cramer-rao bounds for sequential estimation. In A. Doucet, N. de Freitas, and N. Gordon, editors, *Sequential Monte Carlo Methods in Practice*, Statistics for Engineering and Information Science, chapter 15, pages 321–338. Springer, 2001.

[12] C. Berzuini and W. Gilks. RESAMPLE-MOVE filtering with cross-model jumps. In A. Doucet, N. de Freitas, and N. Gordon, editors, *Sequential Monte Carlo Methods in Practice*, Statistics for Engineering and Information Science, chapter 6, pages 117–138. Springer, 2001.

[13] S. Birchfield. Elliptical head tracking using intensity gradients and colorhistograms. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 232–237, 1998.

[14] S. Birchfield. KLT: An implementation of the Kanade-Lucas-Tomasi feature tracker. `http://www.ces/clemson.edu/stb/klt/index.html`, March 2006.

[15] M. Black and P. Anandan. The robust estimation of multiple motions: parametric and piecewise-smooth flow fields. *Computer Vision and Image Understanding*, 63(1):75–104, 1996.

[16] A. Blake and M. Isard. *Active Contours*. Springer, Great Britain, 2000.

[17] K. Branson and S. Belongie. Tracking multiple mouse contours (without too many samples). In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2005.

[18] K. Branson, V. Rabaud, and S. Belongie. Three brown mice: See how they run. In *Proceedings of the IEEE Workshop on Visual Surveillance and Performance Evaluation of Tracking and Surveillance at the International Conference on Computer Vision*, 2003.

[19] C. Bregler and S. Omohundro. Surface learning with applications to lipreading. In *Advances in Neural Information Processing Systems*, volume 6, 1994.

[20] T.-J. Cham and J. M. Rehg. A multiple hypothesis approach to figure tracking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 239–245, 1999.

[21] Y. Chen and T. L. Lai. Sequential Monte Carlo methods for filtering and smoothing in hidden Markov models. Technical Report 2003-18, Duke University, August 2003.

[22] Y. Chen, Y. Rui, and T. Huang. Multicue HMM-UKF for Real-Time Contour Tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(9):1525–1529, 2006.

[23] R. Chin and C. Dyer. Model-based recognition in robot vision. *ACM Computing Surveys (CSUR)*, 18(1):67–108, 1986.

[24] D. Comaniciu, V. Ramesh, and P. Meer. Real-time tracking of non-rigid objects using mean shift. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, 2000.

[25] D. Comaniciu, V. Ramesh, and P. Meer. Kernel-based object tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(5):564–575, 2003.

[26] T. Cootes, C. Taylor, D. Cooper, and J. Graham. Active Shape Models - Their tracking and applications. *Computer Vision and Image Understanding*, 61(2), 1995.

[27] T. F. Cootes, G. J. Edwards, and C. J. Taylor. Active appearance models. In *Proceedings of the European Conference on Computer Vision*, volume 1407, pages 484–??, 1998.

[28] I. Cox and S. Hingorani. An efficient implementation and evaluation of reid's multiple hypothesis tracking algorithm for visual tracking. In *Proceedings of the International Conference on Pattern Recognition*, volume A, pages 437–442, 1994.

[29] I. J. Cox and S. L. Hingorani. An efficient implementation of reid's multiple hypothesis tracking algorithm and its evaluation for the purpose of visual tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(2):138–150, 1996.

[30] D. Crisan. Particle filters - a theoretical perspective. In A. Doucet, N. de Freitas, and N. Gordon, editors, *Sequential Monte Carlo Methods in Practice*, Statistics for Engineering and Information Science, chapter 2, pages 15–42. Springer, 2001.

[31] D. Crisan and A. Doucet. A survey of convergence results on particle filtering methods forpractitioners. *IEEE Transactions on Signal Processing*, 50(3):736–746, 2002.

[32] R. Cucchiara, C. Grana, M. Piccardi, and A. Prati. Detecting moving objects, ghosts, and shadows in video streams. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(10):1337–1342, 2003.

[33] T. Darrell and A. Pentland. Robust estimation of a multi-layered motion representation. In *Proceedings of the IEEE Workshop on Visual Motion*, pages 173–178, 1991.

[34] J. de Freitas, M. Niranjan, A. Gee, and A. Doucet. Sequential Monte Carlo Methods to Train Neural Network Models. *Neural Computation*, 12(4):955–993, April 2000.

[35] J. Deutscher, A. Blake, and I. Reid. Articulated body motion capture by annealed particle filtering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 126–133, 2000.

[36] P. Dollár, Z. Tu, and S. Belongie. Supervised learning of edges and object boundaries. In *CVPR*, June 2006.

[37] A. Doucet. On sequential simulation-based methods for bayesian filtering. Technical Report CUED/F-INFENG/TR. 310, Cambridge University Department of Engineering, 1998.

[38] A. Elgammal, R. Duraiswami, D. Harwood, and L. S. Davis. Background and foreground modeling using nonparametric kernel density for visual surveillance. In *Proceedings of the IEEE*, volume 90(7), pages 1151–1163, July 2002.

[39] P. Fieguth and D. Terzopoulos. Color-based tracking of heads and other mobile objects at videoframe rates. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 21–27, 1997.

[40] Foundation for Biomedical Research. Fbr's position on animal research. `http://www.fbresearch.org/about/position.htm`.

[41] C. Fowlkes. The Berkeley segmentation engine. `http://www.cs.berkeley.edu/~fowlkes/BSE/`, May 2005.

[42] N. Friedman and S. Russell. Image segmentation in video sequences: A probabilistic approach. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 1997.

[43] J. Gårding. *Shape from surface markings*. PhD thesis, Royal Institute of Technology, Stockholm, 1991.

[44] D. Gavrila, J. Giebel, and H. Neumann. Learning shape models from examples. In *Proceedings of the Symposium of the German Association for Pattern Recognition (DAGM)*, pages 369–376, 2001.

[45] D. M. Gavrila. Multi-feature hierarchical template matching using distance transforms. In *Proceedings of the International Conference on Pattern Recognition*, pages 439–444, 1998.

[46] P. Getreuer. Matlab central file exchange – color space converter. `http://www.mathworks.com/matlabcentral/fileexchange/loadFile.do?objectId=7744&objectType=FILE`, May 2005.

[47] J. Giebel. *Learning Dynamic Shape Models for Bayesian Tracking*. PhD thesis, Universitat Mannheim, 2005.

[48] J. Giebel and D. Gavrila. Multimodal shape tracking with point distribution models. In *Proceedings of the Symposium of the German Association for Pattern Recognition (DAGM)*, pages 1–8, 2002.

[49] S. Godsill, A. Doucet, and M. West. Monte Carlo Smoothing for Nonlinear Time Series. *Journal of the American Statistical Association*, 99(465):156–169, 2004.

[50] G. Hager, M. Dewan, and C. Stewart. Multiple kernel tracking with SSD. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, 2004.

[51] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer Verlag, Basel, 2001.

[52] T. Heap and D. Hogg. Improving specificity in pdms using a hierarchical approach. In *Proceedings of the British Machine Vision Conference*, 1997.

[53] B. Horn and B. Schunck. Determining Optical Flow. *Artificial Intelligence*, 17(1-3):185–203, 1981.

[54] M. Irani and P. Anandan. All about direct methods. In *Proceedings of the International Workshop on Vision Algorithms: Theory and Practice*. Springer-Verlag, 1999.

[55] M. Irani, B. Rousso, and S. Peleg. Computing occluding and transparent motions. *International Journal of Computer Vision*, 12(1):5–16, 1994.

[56] M. Isard and A. Blake. ICONDENSATION: Unifying low-level and high-level tracking in a stochastic framework. In *Proceedings of the European Conference on Computer Vision*, volume 1406, pages 893–908, 1998.

[57] M. Isard and J. MacCormick. BraMBLe: A Bayesian multiple-blob tracker. In *Proceedings of the IEEE International Conference on Computer Vision*, 2001.

[58] J. Ferryman and S. Maybank, University of Reading, UK. Performance evaluation of tracking and surveillance (PETS) data set, 2001.

[59] S. Jabri, Z. Duric, H. Wechsler, and A. Rosenfeld. Detection and location of people in video images using adaptive fusion of color and edge information. In *Proceedings of the International Conference on Pattern Recognition*, pages 4627–4631, 2000.

[60] N. Jojic and B. J. Frey. Learning flexible sprites in video layers. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 01, page 199, Los Alamitos, CA, USA, 2001. IEEE Computer Society.

[61] E. Jones and S. Soatto. Layered Active Appearance Models. In *Proceedings of the IEEE International Conference on Computer Vision*, volume 2, 2005.

[62] S. Julier, J. Uhlmann, and H. Durrant-Whyte. A new approach for filtering nonlinear systems. In *Proceedings of the American Control Conference*, volume 3, 1995.

[63] P. KaewTraKulPong and R. Bowden. An improved adaptive background mixture model for real-time tracking with shadow detection. In *Proceedings of the European Workshop on Advanced Video-based Surveillance Systems*, 2001.

[64] M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active contour models. *International Journal of Computer Vision*, pages 321–331, 1988.

[65] K. Kim, D. Harwood, and L. Davis. Background updating for visual surveillance. In *Advances in Visual Computing*, pages 337–346. Springer, 2005.

[66] G. Kitigawa and S. Sato. Monte carlo smoothing and self-organising state-space model. In A. Doucet, N. de Freitas, and N. Gordon, editors, *Sequential Monte Carlo Methods in Practice*, Statistics for Engineering and Information Science, chapter 9, pages 177–196. Springer, 2001.

[67] D. Koller, J. Weber, T. Huang, J. Malik, G. Ogasawara, B. Rao, and S. Russell. Towards robust automatic traffic scene analysis in real-time. In *Proceedings of the International Conference on Pattern Recognition*, volume 1, 1994.

[68] F. Kristensen, P. Nilsson, and V. Owall. Background Segmentation Beyond RGB. In *Proceedings of the Asian Conference on Computer Vision*, volume 3852, page 602. Springer-Verlag, 2006.

[69] J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the International Conference on Machine Learning*, pages 282–289, 2001.

[70] D. Lee. Effective Gaussian Mixture Learning for Video Background Subtraction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(5):827–832, 2005.

[71] K.-C. Lee and D. Kriegman. Online learning of probabilistic appearance manifolds for video-based recognition and tracking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 1, pages 852–859, Los Alamitos, CA, USA, 2005. IEEE Computer Society.

[72] J. Lim, D. Ross, R.-S. Lin, and M.-H. Yang. Incremental learning for visual tracking. In *Advances in Neural Information Processing Systems*, volume 18, pages 793–800, 2005.

[73] J. Liu. Metropolized independent sampling with comparisons to rejection sampling and importance sampling. *Statistics and Computing*, 6:113–119, 1996.

[74] J. Liu and R. Chen. Blind Deconvolution Via Sequential Imputations. *Journal of the American Statistical Association*, 90(430), 1995.

[75] J. S. Liu and R. Chen. Sequential Monte Carlo methods for dynamic systems. *Journal of the American Statistical Association*, 93(443):1032–1044, 1998.

[76] B. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings of the International Joint Conferences on Artificial Intelligence*, pages 674–679, 1981.

[77] J. MacCormick. *Probabilistic Modelling and Stochastic Algorithms for Visual Localisation and Tracking*. PhD thesis, Oxford, January 2000.

[78] J. MacCormick. *Stochastic Algorithms for Visual Tracking*. Distinguished Dissertations. Springer, Great Britain, 2002.

[79] J. MacCormick and A. Blake. A probabilistic exclusion principle for tracking multiple objects. *International Journal of Computer Vision*, 39(1):57–71, 2000.

[80] D. Martin, C. Fowlkes, and J. Malik. Learning to detect natural image boundaries using local brightness, color, and texture cues. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(5):530–549, May 2004.

[81] F. Moscheni, S. Bhattacharjee, and M. Kunt. Spatio-temporal segmentation based on region merging. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(9):897–915, 1998.

[82] S. Niyogi. Detecting kinetic occlusion. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1044–1049, 1995.

[83] K. Okuma, A. Taleghani, N. de Freitas, J. Little, and D. Lowe. A boosted particle filter: Multitarget detection and tracking. In *Proceedings of the European Conference on Computer Vision*, 2004.

[84] M. Orton and W. Fitzgerald. A bayesian approach to tracking multiple targets using sensor arrays and particle filters. *IEEE Transactions on Signal Processing*, 2002.

[85] M. Piccardi. Background subtraction techniques: a review. In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics*, volume 4, pages 3099–3104, 2004.

[86] M. Pitt and N. Shephard. Auxiliary variable based particle filters. In A. Doucet, N. D. Freitas, and N. Gordon, editors, *Sequential Monte Carlo Methods in Practice*, Statistics for Engineering and Information Science, chapter 13, pages 273–293. Springer, 2001.

[87] R. Radke, S. Andra, O. Al-Kofahi, and B. Roysam. Image change detection algorithms: a systematic survey. In *Proceedings of the IEEE International Conference on Image Processing*, volume 14 (3), pages 294–307, 2005.

[88] Y. Raja, S. McKenna, and S. Gong. Segmentation and Tracking Using Color Mixture Models. In *Proceedings of the Asian Conference on Computer Vision*, pages 607–614. Springer-Verlag London, UK, 1998.

[89] D. Ramanan. *Tracking People and Recognizing their Activities.* PhD thesis, University of Delaware, 2000.

[90] D. Ramanan and D. Forsyth. Finding and tracking people from the bottom up. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, 2003.

[91] D. Ramanan, D. Forsyth, and A. Zisserman. Tracking people by learning their appearance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2007:65–81, 2001.

[92] C. Rasmussen and G. Hager. Joint probabilistic techniques for tracking multi-part objects. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 16–21, 1998.

[93] C. Rasmussen and G. D. Hager. Probabilistic data association methods for tracking complex visual objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(6):560–576, 2001.

[94] J. Rehg and T. Kanade. Visual tracking of high dof articulated structures: An application to human hand tracking. In *Proceedings of the European Conference on Computer Vision*, volume II, pages 35–46, May 1994.

[95] D. Reid. An algorithm for tracking multiple targets. *IEEE Transactions on Automatic Control*, 24(6):843–854, 1979.

[96] Y. Rui and Y. Chen. Better Proposal Distributions: Object Tracking Using Unscented Particle Filter. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 786–793, 2001.

[97] C. Schmid, R. Mohr, and C. Bauckhage. Evaluation of Interest Point Detectors. *International Journal of Computer Vision*, 37(2):151–172, 2000.

[98] D. Schulz, W. Burgard, D. Fox, and A. Cremers. Tracking multiple moving targets with a mobile robot using particle filters and statistical data association. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 2, 2001.

[99] D. Schulz, W. Burgard, D. Fox, and A. Cremers. People Tracking with Mobile Robots Using Sample-Based Joint Probabilistic Data Association Filters. *The International Journal of Robotics Research*, 22(2):99–116, 2003.

[100] S. Sclaroff and J. Isidoro. Active blobs. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1146–1153, 1998.

[101] B. Selman, H. Kautz, and B. Cohen. Local search strategies for satisfiability testing. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 26:521–532, 1996.

[102] Y. B. Shaalom and T. E. Fortman. *Tracking and Data Association.* Academic Press, Boston, 1988.

[103] Y. Sheikh and M. Shah. Bayesian Modeling of Dynamic Scenes for Object Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(11):1778–1792, 2005.

[104] J. Shi and J. Malik. Motion segmentation and tracking using normalized cuts. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1154–1160, 1998.

[105] J. Shi and C. Tomasi. Good features to track. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 1994.

[106] L. Sigal, S. Bhatia, S. Roth, M. J. Black, and M. Isard. Tracking loose-limbed people. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 01, pages 421–428, Los Alamitos, CA, USA, 2004. IEEE Computer Society.

[107] C. Stauffer and W. Grimson. Adaptive background mixture models for real-time tracking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 1999.

[108] M. B. Stegmann. Active appearance models. `http://www2.imm.dtu.dk/~aam/`, December 2004.

[109] M. B. Stegmann. *Generative Interpretation of Medical Images.* PhD thesis, Informatics and Mathematical Modelling, Technical University of Denmark, DTU, Richard Petersens Plads, Building 321, DK-2800 Kgs. Lyngby, 2004. Awarded the Nordic Award for the Best Ph.D. Thesis in Image Analysis and Pattern Recognition in the years 2003-2004 at SCIA'05.

[110] J. Sullivan. *A Bayesian Framefork for Localisation in Visual Images.* PhD thesis, Oxford, September 2000.

[111] H. Tao, H. Sawhney, and R. Kumar. Object tracking with Bayesian estimation of dynamic layer representations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(1):75–89, 2002.

[112] H. Tao, H. S. Sawhney, and R. Kumar. Dynamic layer representation with applications to tracking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 02, page 2134, Los Alamitos, CA, USA, 2000. IEEE Computer Society.

[113] University of California Center for Animal Alternatives. The mouse in science: Why mice? `http://www.vetmed.ucdavis.edu/Animal_Alternatives/whymice.htm`.

[114] R. van der Merwe, N. de Freitas, A. Doucet, and E. Wan. The unscented particle filter. Technical Report CUED/F-INFENG/TR380, Cambridge University Engineering Department, August, 2000.

[115] R. van der Merwe, N. de Freitas, A. Doucet, and E. Wan. The unscented particel filter. In *Advances in Neural Information Processing Systems*, Nov 2001.

[116] J. Vermaak, S. Godsill, and P. Perez. Monte carlo filtering for multi-target tracking and data association. *Transactions on Aerospace and Electronic Systems*, 41(1):309–332, January 2005.

[117] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2001.

[118] P. Viola and M. Jones. Robust real-time object detection. *International Journal of Computer Vision*, 2002.

[119] E. Wan and R. Van Der Merwe. The unscented Kalman filter for nonlinear estimation. In *Proceedings of the IEEE Adaptive Systems for Signal Processing, Communications, and Control Symposium*, pages 153–158, 2000.

[120] J. Wang and E. Adelson. Representing moving images with layers. In *Proceedings of the IEEE International Conference on Image Processing*, volume 3 (5), pages 625–638, 1994.

[121] Y. Weiss and E. H. Adelson. A unified mixture framework for motion segmentation: Incorporating spatial coherence and estimating the number of models. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 00, page 321, Los Alamitos, CA, USA, 1996. IEEE Computer Society.

[122] G. Welch and G. Bishop. An introduction to the kalman filter. Technical Report TR 95-041, University of North Carolina at Chapel Hill, Chapel Hill, NC 27599-3175, July 2006.

[123] Wikipedia. Lab color space – Wikipedia, the free encyclopedia. `http://en.wikipedia.org/w/index.php?title=Lab_color_space&oldid=943475366`, 2006.

[124] O. Williams, A. Blake, and R. Cipolla. Sparse bayesian learning for efficient visual tracking. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(8):1292–1304, 2005.

[125] C. Wren, A. Azarbayejani, T. Darrell, and A. Pentland. Pfinder: real-time tracking of the human body. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(7):780–785, 1997.

[126] Y. Yang and M. Levine. The background primal sketch: An approach for tracking moving objects. *Machine Vision and Applications*, 5(1):17–34, 1992.