Bug Vision: Experiments in Low Resolution Vision

by

Ali Rahimi

B.S., University of California at Berkeley (1997)

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning
in partial fulfillment of the requirements for the degree of

Master of Science in Media Arts and Sciences

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2001

Author ....
Program in Media Arts and Sciences,
School of Architecture and Planning
August 17, 2001

Certified by ....
Alex (Sandy) Pentland
Professor of Media Arts and Sciences
Thesis Supervisor

Accepted by ....
Dr. Andrew Lippman
Chair, Department Committee on Graduate Studies
Program in Media Arts and Sciences

Bug Vision:
Experiments in Low Resolution Vision

by

Ali Rahimi

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning
on August 17, 2001, in partial fulfillment of the
requirements for the degree of
Master of Science in Media Arts and Sciences

Abstract

Tracking multiple people using cameras is similar to the well-studied problem of tracking multiple radar or sonar echoes. This thesis shows that current camera-based tracking algorithms convert each image in a video sequence to a list of targets through a segmentation step, and pass this target set to a traditional multiple-point-target tracking algorithm. Various tracking vision-based strategies as well as point tracking strategies are discussed.

Bayesian solutions to the point-tracking problem are well understood, because the generative models need describe the dynamics of simple point objects. In addition, the radar tracking problem assumes that measurements are noise corrupted positions, which makes it easy to cast the tracking problem in a Bayesian framework.

Unlike radar, cameras report observations as images. Though point object dynamics can still be used to describe the hidden state of targets, the observation model is an image formation process. As such, the typical solution to tracking in the camera-based tracking community is to reduce each image to a point set, where each point corresponds to a potential target. However, this step introduces uncertainty that is usually not modeled.

This thesis proposes a Bayesian person-tracking algorithm which models the entire process of tracking, from the dynamics of the targets to the formation of easy to compute image transforms. An approximate Bayesian tracking algorithm based on Variational Bayes is developed. All the benefits of a Bayesian framework including modeling of the certainty of the recovered results and model selection are taken advantage of.

The resulting person tracking algorithm can operate on extremely poor quality imagery. In addition, the tracker can compute the number of targets in the scene automatically as a side effect of its Bayesian formulation.

Thesis Supervisor: Alex (Sandy) Pentland
Title: Professor of Media Arts and Sciences

Experiments in Low Resolution Vision

by

Ali Rahimi

The following people served as readers for this thesis:

Thesis Reader ....

Trevor Darrell
Assistant Professor
Electrical Engineering and Computer Science, AI Lab
Massachusetts Institute of Technology

Thesis Reader ....[illegible]

Christoph Bregler
Assistant Professor
Computer Science Department
Stanford University

Acknowledgments

I am foremost indebted to Tom Minka for publishing a wealth of material on machine learning and statistics. His papers have provided a concrete foundation for most of the statistics I've learned in the past year. He's also provided continuous support and has convinced me that it's worth understanding and solving a problem completely and thoroughly.

My advisor, Sandy Pentland, provided the original motivation for performing computer vision with cheaply computed features. The length of chapter 4 and the poor quality of the background subtraction and motion maps in this thesis attest that I'm now a firm believer in compensating for bad data with good statistics. Sandy was wholly responsible for matriculating me into that school of thought.

Sumit Basu, Brian Clarkson, Tony Jebara, Tanzeem Choudhury, Yuri Ivanov, and Yuan Qi have provided me with a research community. I'm very thankful to Sumit, Brian and Tanzeem for the weekly Markovia meetings. I learned more from our reading group than anywhere else. And I'm sorry for constantly barging into Sumit and Brian's office to ask them questions and share observations with them.

Many thanks to Ben Recht for showing me a the trick for computing the Ψ function in chapter 4, and for reminding me that statistics may have corrupted my entire way of thinking and that perhaps I ought to diversify.

Contents

List of Figures

Chapter 1

Introduction

My long-term research goal is to explore how much we can infer about events in the world using a large number of simple, cheap distributed sensors. As an example, I have often sketched out the following scenario:

> Secret agents have been tailing Sam and his cohorts for the past month. They have summarized their pattern of activity as follows:
>
> Whenever Sam is hungry, he goes downtown and has lunch at any one of 3 restaurants. After lunch, he goes to the park for a walk. After an hour or so, he goes home for a nap. When he wakes up, he may either go back downtown or to the park. In fact, all of Sam's associates exhibit this behavior, but they act independently of each other.
>
> The surveillance must go on for another year. Unfortunately, funding has run out and all of the secret agents must be replaced by a single photodetector.
>
> Our task: to figure out where to place the photodetector and what algorithm to use.

The way it's posed, the task seems intractable. Is it really possible to determine the locations of several individuals from a single, fixed, line of sight sensor? It's definitely possible, but the reason why it's possible may be disappointing. To see why, we need to agree on what the job of a sensor is.

A sensor's task is to report some *guess* as to what is happening in the real world. When a scale reports that an object weighs 1kg, we interpret the result as "I'm not sure how much this object weighs, but it's probably close to 1kg. It could be 1.1kg, or 0.9kg, but if I had to bet on it, I would say 1kg." The object has a true weight, but the scale reports its certainty over a distribution of weights. A great sensor is one which assigns high confidence to a small range of possibilities near the true value. A decent sensor is one which is confident in a wider range of possibilities (thus admitting possibilities which are farther away from the truth). A bad sensor is one which assigns high confidence to values which aren't actually reflective of the truth.

When our job is to locate a person using one photodetector, we are actually looking for a sensor which reports certainty over a distribution of possible locations. An accurate tracker would assign high certainty to locations near the targets' true positions and low certainty to locations away from them. A dishonest sensor over-estimates its ability by assigning high certainty to the true value but insignificant certainty to values which may very well be correct as well. Even worse, a bad sensor could assign very high probability to an incorrect location and negligible certainty to every other value. I set out to design a sensor which could honestly report its certainty in the position of people from sparse measurements.

Since the placement and orientation of the photodectors has a large impact on tracking performance, my initial goal was to devise an algorithm which could compute the optimal placement of a set of low resolution sensors. When I began this research, I had assumed the existence of a Bayesian tracking algorithm for finding people using distributed sensors. My contribution would automatically fine-tune the location of the sensors in order to improve tracking performance in a given setting. Unfortunately, I have not been able to find a Bayesian tracker would could report its performance accurately, making it impossible for my optimizing algorithm to honestly claim that tracking performance had been improved. Since my goal depends on such a tracker, this thesis sets out to provide one. My initial goal of optimizing sensor locations will be the subject of another inquiry.

1.1 Scope

Abstractly, this thesis is about estimating the state of the real world. A sensor makes a measurement y and returns its certainty over a set of values θ, where θ represents a quantity of interest in the real world. In the case of a scale, y represents the tension of a spring in the scale, and θ represents the weight of the person standing on the scale. The job of the scale is to make measurements and to return a function which describes its certainty over all θ: $p(\theta|y)$.

Of course, in practice, scales return one number, not a function over all numbers. But well documented scales implicitly describe the mapping between this number and a probability distribution function. For example, whenever the back of a scale mentions the RMS error, or provides a disclaimer claiming that the scale is only accurate up to ± 1 mg, it is in fact telling us how to go from the weight it reports to a probability distribution. Suppose a scale says it has a variance σ. This variance parametrizes the family of Gaussian functions, so the reported weight actually parametrizes:

$$\begin{aligned} p(\theta|y) &= \mathcal{N}(\theta; M(y), \sigma^2) \\ &= \frac{1}{\sqrt{2\pi}\sigma} \exp((\theta - M(y))^2/\sigma^2), \end{aligned}$$

where $M(y)$ is the weight reported by the scale. A good scale is one with a small variance σ. An honest scale is one which accurately reports σ. A bad scale reports a distribution which assigns relatively low probability to the true value.

Tracking people using low resolution sensors has a similar structure: Various sensors placed in the field transduce the state of the world into electrical signals. Individually, each sensor measures the distance to a moving object, the speed of an object, the intensity of the sound the object produces, its brightness, its weight, etc. They each report a probability distribution over these quantities. A tracking algorithm is responsible for using these quantities to report its belief about the location of the target. A good tracker's beliefs are clustered around one value which correctly describes the position of the targets:

$$p([\ \theta^1 \quad \theta^2 \quad \dots \quad \theta^k\ ], k|y)$$

Here, the individual θ^i represent the locations of a person, and k is the number of people. Designing an algorithm to compute such a distribution is the subject of this thesis.

1.2 Intended Audience

This document assumes familiarity with probability theory. It assumes the reader is comfortable with Bayes rule, marginalization of joint densities, the idea of statistical independence, and independence diagrams.

1.3 Outline

This thesis describes how to obtain a probabilistic description of the location of people over time using easy to obtain measurements. The sensor used is a full resolution camera which measures the amount of motion at every pixel.

Chapter 2 describes the state-of-the-art in camera-based people tracking. It begins with the fundamentals of tracking multiple point sources and suggests that most vision-based trackers implicitly assume the targets of interest are point sources. This raises certain issues which have, in the past, been dealt with using heuristics.

Chapter 3 shows that by representing the target not as a point source but as a collection of points, most of the problems of today's vision-based trackers can be resolved. This chapter derives a probabilistic model that can be used to describe the problem in concrete terms. The model is simple to sketch out and captures a lot of intuition about what happens in the real world. However, performing exact inference on this model is intractable.

Chapter 4 describes Variational Bayes (VB), a tool that can be used to perform approximate inference on the model layed out in chapter 3. This chapter presents an easy to understand derivation of Variational Bayes and relates the derivation to the traditional way of looking at VB.

VB can then be combined with the traditional Kalman Filter to build a multi-person tracking system. This merger is documented in chapter 5.

Chapter 6 concludes with logical extensions of the work.

Chapter 2

Multi-Person Tracking

Tracking the location of people over time is critical to applications such as intelligent rooms[8, 14, 18], automatic reporting of suspicious activities[26], interactive environments[30], or modeling of people's interactions[2]. The most common sensor used in these applications are high resolution cameras. Figure 2-1 shows an example view from a camera installed in a smart room. We would like each person in the field of view to be assigned a unique identifier and tracker as long as they remain in the scene. The label of a person must not change while the subject remains in the field of view. In addition, the locations and labels should be equipped with a measure of uncertainty described as a probability distribution over all possible labels and locations.

There are many vision-based person tracking systems, but most of them don't provide an accurate measure of certainty. Those which do, do it in forms which do not lend themselves to having their performance optimized. This chapter begins by reviewing some fundamental statistical techniques. Section 2.1 examines the point-source multi-target tracking problem and reviews a variant of Bar-Shalom and Tse's Probabilistic Data Association Filter (PDAF). The PDAF will be viewed in light of the fundamentals established in the first section of this chapter. Once the fundamentals of multi-point tracking are laid out, this chapter reviews some vision-based systems. It will be shown that people trackers reduce each person to a point source to make tracking easier. The next chapter shows how tracking can be improved by removing this reduction step and modeling each person as a cloud of

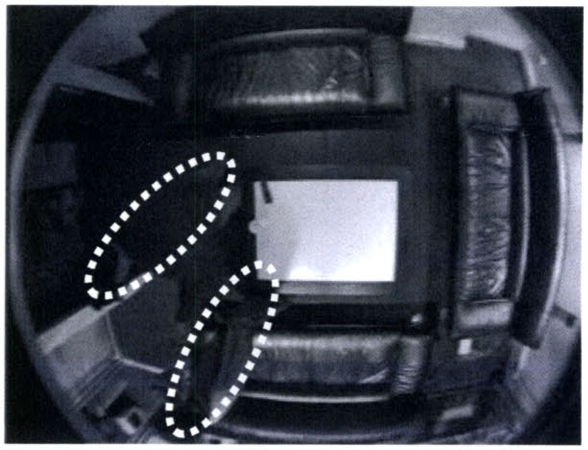

Figure 2-1: Finding people in a room. People are tracked over time. They are identified with an oval which stays on them as they move around the room.

points instead of a single point.

2.1 Point-based MTT

In the traditional Multi-Target-Tracking (MTT) problem, a radar provides noisy measurements of the location of several targets (airplanes, missiles, boats, or countermeasures). The task of the MTT algorithm is to use these noisy location measurements to update a set of tracks which it has been maintaining and to report the uncertainty in their recovered position[1]. Figure 2-2 depicts the problem that Multi-Target-Tracking addresses. At each time step, a Bayesian MTT must use all available data to report a distribution over all possible states of the targets:

$$p(\theta_t^1, \theta_t^2, \ldots, \theta_t^3 | y_0 \ldots y_t)$$

To provide an example of a proper Bayesian tracker, the following section describes the one object tracking algorithm, which is solved using a Kalman Filter.

2.1.1 Kalman Filtering

Kalman Filtering addresses a simplest tracking scenario, where one target produces one measurement at each time step. The job of the Kalman Filter is to maintain a track which is consistent with the known dynamics of the target. It is presented here to show the minimum effort required in tracking objects in a Bayesian framework.

The Kalman Filter assumes that an object moves according to Markovian dynamics. That is, at each time step, the density $p(\theta_{t+1}|\theta_t)$ completely defines the evolution of the target's state, independent of older *theta*'s. Linear Markovian dynamics are often used in tracking. When the state θ is a vector of numbers, the dynamics are specified in terms of a recurrence relation:

$$\theta_{t+1} = A\theta_t + w_t, \tag{2.1}$$

where w_t is zero mean white Gaussian noise and A is called the state update matrix. Equation (2.1) describes a system whose evolution depends only on its current state, with some added perturbation which is independent of all other quantities in the system. To concretize the example, if θ_t represents the position, and velocity of an object, one could describe the motion of that object in 1 dimension as:

$$\theta_{t+1} = \begin{bmatrix} x_{t+1} \\ \dot{x}_{t+1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x_t \\ \dot{x}_t \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} w_t$$

The state update matrix here adds the position and the velocity of the object at time t to object the new position of the object, and updates the velocity by adding some noise to it. This is one example of Markovian dynamics. Contrary to popular belief, the development of the Kalman Filter however does not depend on this linearity. As will be shown later, the only requirement on the dynamics of the object is that a density $p(\theta_{t+1}|z)$ be Gaussian an computable from $p(\theta_t|z)$ (where z is some random variable). In the case of linear dynamics, these conditions hold. But one could employ any heuristic for performing the update.

In addition to a dynamics model, the Kalman Filter requires a measurement model. It assumes that each state is observed under a linear transformation and additive white Gaussian noise. So the measurements are distributed according to $p(y_t|\theta_t) = N(y_t; C\theta_t, \Sigma_{y|\theta})$. Traditionally, this is written as

$$y_t = C\theta_t + v_t, \tag{2.2}$$

where v_t has covariance $\Sigma y|\theta$. Going back to 1D motion of a physical object, if the sensor observes the position of the object, the observation equation becomes

$$y_t = \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} x_t \\ \dot{x}_t \end{bmatrix} + v_t$$

Alternatively, the sensor could observe only the velocity, in which case C would become $\begin{bmatrix} 0 & 1 \end{bmatrix}$, or the sensor could observe both quantities, in which case C is the identity matrix and y_t is a 2D vector. In either case, the observation is a Gaussian with a mean described by a linear function of the state and given covariance.

In summary, the graphical model assumed by the Kalman Filter is a hidden Markov model and is represented in figure 2-3. It is assumed that the object changes state according to Markovian dynamics and the state is observed under noisy conditions governed by (2.2).

The aim of the Kalman Filter is to estimate the posterior distribution $p(\theta_t|y_0..y_t)$ at each time step. That is, it uses all past observations to compute a distribution over possible states. One could imagine a contrasting sub-optimal algorithm computes a distribution over θ_t without taking all past observations into account (instead using only y_t, for example). In this sense, the Kalman Filter is the optimal causal filter for θ.

One of the useful independence relationships that can be gleaned from the independence diagram is that if the state is known, past observations don't provide any new information for predicting the output:

$$p(y_t|\theta_t, y_0..y_t) = p(y_t|\theta_t)$$

Using this property, the posterior state distribution can be factorized:

$$p(\theta_t|y_0..y_t) = \frac{1}{Z}p(y_t|\theta_t, y_0..y_{t-1})p(\theta_t|y_0..y_{t-1}) = \frac{1}{Z}p(y_t|\theta_t)p(\theta_t|y_0..y_{t-1}), \tag{2.3}$$

where Z is used to normalize the distribution so that it is conditioned on y_t. The term

$p(\theta_t|y_0..y_{t-1})$ is the one step state predictor: given all measurements up to time y_{t-1}, it provides a prediction over the possible states at the next time step. Computing it is a matter of propagating the estimate $p(\theta_{t-1}|y_0..y_{t-1})$ one step forward in time using the dynamics model of the target. If $p(\theta_t|\theta_{t-1})$ is known, this involves an integration, but it is usually simpler to think of this step as "applying the dynamics and diffusing." Since $p(\theta_{t-1}|y_0..y_{t-1})$ was the outpout of the filter at hte previous time step, this algorithm is recursively reusing its output. The algorithm reuses its old answer to avoid having to explicitly reuse past observations. Instead, past observations are summarized in the one step state predictor.

To see why, let's put equation (2.3) under a different light. Consider an observation y generated from a linear transformation on a Gaussian random variable μ (as in equation (2.2)). Assuming prior information that μ is Gaussian with mean μ_0 and covariance Σ_0, the posterior distribution of μ is:

$$p(u|y) = \frac{p(y|u)p(u)}{\int_u p(y|u)p(u)} = \frac{N(y; Cu, \Sigma_{y|u})N(u; u_0, \Sigma_0)}{\int_u N(y; u, \Sigma_{y|u})N(u; u_0, \Sigma_0)} \tag{2.4}$$

Note the similarity between this problem and the Kalman Filter. This equation underlines the role of the one step state predictor $p(\theta_t|y_0..y_{t-1})$ as a prior on θ_t in equation (2.3. It's possible to shows that $p(u|y)$ is a Gaussian with the following mean and covariance:

$$\begin{aligned} E[\mu] &= u_0 + K(y - Cu_0) && (2.5)\\ \Sigma_{u|y} &= \Sigma_0 - K(C\Sigma_0 C^T + \Sigma_{y|u})K^T = (\Sigma_0^{-1} - C^T\Sigma_{y|u}^{-1})^{-1} && (2.6)\\ K &= \Sigma_0 C^T(C\Sigma_0 C^T + \Sigma_{y|u})^{-1} && (2.7) \end{aligned}$$

In equation (2.3), the one step predictor $p(\theta_t|y_0..y_{t-1})$ plays the role of $p(u)$ in equation (2.4), and $\frac{1}{Z}$ is $p(y)$. If the one step state predictor is Gaussian, equations (2.5)-(2.7) compute the posterior θ given all past measurements. This is how the Kalman Filter incorporates new measurements.

2.1.2 Graphical Model for MTT

In single-target single-measurement tracking, the correspondence between the point and the track is trivial. However, in MTT, each target generates an unlabeled measurement, and the tracking algorithm must determine the origin of each measurement. Traditionally, MTT has been split into two steps: data association and filtering. Data association determines the correspondence between targets and tracks. Once this correspondence is established, the filtering step updated each track using its assigned target.

One way to think of MTT is that N targets move independently of each other according to Markovian dynamics. The state of the kth target, $\theta^k(t)$, might be updated according to linear dynamics as in equation (2.1), for example. At each time step, track k emits a noisy position in the form of a linear function of $\theta^k(t)$, corrupted by white Gaussian noise of known covariance. Each emission k is then mapped to mth observation $y_m(t)$, so that there are a total of N measurements.

If the mapping between m and k were known, the tracks could be individually updated using N separate Kalman Filters. However, measurements are unlabeled, and their assignments to their tracks is unknown, so the tracking algorithm must determine this labeling before performing any updates.

Figure 2-4 shows an independence diagram for one time step of the MTT problem. The emission of every track is probabilistically shuffled by a switch π. Track k is mapped to measurement m with probability $p(\pi_m^k)$. Therefore, given the true state of the target, the distribution over $y_m(t)$ is:

$$p(y_m(t)|\theta(t)) = \sum_{k=1}^{N} N(y_m(t); C\theta^k(t), \Sigma_{y|\theta})p(\pi_m^k)$$

Note that this representation is very general and can allow situations where a measurement is assigned to two different tracks, or where a track generates two measurements. For example, if $p(\pi_1^2)$ and $p(\pi_1^3)$ are non-zero, track 1 would be mapped to measurements 2 and 3. Similarly, if $p(\pi_2^1)$ and $p(\pi_3^1)$ are non-zero, tracks 2 and 3 will have confounded measurements, both observed as measurement 1. Therefore, $p(\pi)$ must be properly parametrized in order to avoid undesired possibilities. Figure 2-5 shows the full independence diagram where the state of each track evolves over time, and the emissions of the tracks are shuffled at each time step.

2.1.3 The Probabilistic Data Association Filter

The Probabilistic Data Association Filter (PDAF) of Bar-Shalom and Tse[1] is an algorithm for tracking multiple point objects which exhibit linear Markov dynamics. It processes each track separately, first computing assignments between measurements and the track, then updating each track.

The PDAF performs data association by computing $p(\pi^k_m|y(0)..y(t))$. This posterior mapping is a fuzzy assignment between measurements and tracks. If the mappings were known exactly, each track could be updated using its measurement using a Kalman Filter. Since the PDAF estimates fuzzy assignments, all measurements are combined to update each track. Measurements which are more likely to have originated from a track are weighed more heavily when updating that track. Figure 2-6 shows that all points are used to update all tracks. Because the PDAF updates all tracks using all points, it is performing both data association and filtering.

Like the Kalman Filter, the PDAF is a recursive algorithm, and the computation of $p(\pi^k_m|y(0)..y(t))$ uses all available quantities computed so far:

$$p(\pi^k_m|y(0)..y(t)) = \frac{p(y_m(t)|y(0)..y(t-1), \pi^k_m)p(\pi^k)}{\sum_{i=1}^N p(y_m(t)|y(0)..y(t-1), \pi^k_i)p(\pi^i)} \tag{2.8}$$

where $p(\pi_k)$ is the prior probability that any measurement came from track k. $p_k(y_m(t)|y(0)..y(t-1))$ is the one step observation predictor of the kth track, as in the Kalman Filter formulation. This quantity represents the probability that track k will emit a measurement $y_m(t)$ given all past data. The formula above is a straightforward application of Bayes rule on the quantities involved.

During the filtering step, each track is updated independently of the other tracks using these fuzzy assignments. According to the Kalman Filter, if a measurement $y_m(t)$ is assigned to track k with full certainty, the update computes a new Gaussian distribution

$p(\theta^k(t)|y(0)..y(t-1)), y_m(t), \pi_m^k)$ with mean

$$E[\theta^k(t|t)] = E[\theta^k(t|t-1)] + K_k(y_m(t) - C_k E[\theta^k(t|t-1)]),$$

where $\theta^k(t|t-1)$ is the one step state predictor for the track. If, however, there is uncertainty as to which measurement should be used to update the track, the PDAF suggests using a mixture of the possible posteriors as the posterior:

$$p(\theta^k(t)|y(0)..y(t)) = \sum_{m=1}^{N} p(\theta^k(t)|y(0)..y(t-1), y_m(t), \pi_m^k) p(\pi_m^k|y(0)..y(t)) \tag{2.9}$$

The first factor is the predictor output of the Kalman Filter, and second factor is computed from equation (2.8). The resulting distribution is a mixture of Gaussians and cannot be represented by a single Gaussian. In addition, if this mixture is used as the one step state predictor at the next time step, the number of mixture components to represent the density grows to N^N. After T iterations, there will be $(N^N)^T$ mixtures to maintain. Instead, the PDAF approximates the mixture model with a single Gaussian. The mean of the Gaussian is set to the mean of the mixture:

$$\begin{aligned} E[\theta_k(t|t)] &= \sum_{m=1}^{N} p(\pi_m^k|y(0)..y(t))(E[\theta_k(t|t-1)] + K_k(y_m(t) - C_k E[\theta_k(t|t-1)])) \\ &= E[\theta_k(t|t-1)] + K_k(\sum_{m=1}^{N} p(\pi_m^k|y(0)..y(t))(y_m(t) - C_k E[\theta_k(t|t-1)]) \end{aligned}$$

which effectively uses the weighted average of the innovations to update the one step predictor. The variance of the Gaussian is set accordingly. Approximating the mixture of Gaussian with a single Gaussian at each step curbs the explosion of modes caused by considering multiple assignments. This provides the filtering step with a Gaussian one step state predictor, which allows the algorithm to use the Kalman update.

2.1.4 Alternatives to PDAF

Collapsing a mixture of Gaussians into a single Gaussian is one possible simplification for representing the uncertainty in tracking. Many other techniques have been proposed for curbing the exponential growth required in representing the posterior distribution of target states.

The Multiple Hypothesis Tracker[11] can retain a fixed number of possibilities simultaneously, and prune off those which are extremely unlikely. Equation (2.9) shows how uncertainty in association results in uncertainty in the estimate. A probability is associated with each of the components of this mixture model. Whenever the probability of one of these components becomes small, it can be pruned away.

Using dynamic programming to prune unlikely paths has also been suggested. At each time step, an association is hypothesized. Tracing through the trellis of associations over time defines an association path. A given path defines the association decisions made by the tracker over time. It is common practice to use the Viterbi algorithm to choose the most likely path through this trellis and to throw away all others. Using dynamic programming algorithms such as Viterbi helps identify the most likely path without ever maintaining a large number of hypothesis. See [6] for a comprehensive review of variants of this technique.

Instead of maintaing one Gaussian for each possible set of associations, particle filtering techniques can be used to perform estimation. In[15], the association and the state of each Gaussian is sampled from the likelihood function. As a result, each particle represents not only one instantiation of the association, but also represents one instantiation of the target state. A more efficient method would be to have each particle represent one instantiation of the associations, but represent the uncertainty in the state variables as the posterior mean and covariance. A variant of this approach is used in [24] who also endows each particle with target occlusion information.

2.2 Vision-Based MTT

The exposition of point-MTT belies the complexity of the camera-based MTT tracking problem. In the version of MTT presented in the previous section, there are as many targets as there are measurements[1]. In camera-based MTT, however, the input at each time step is an image, so there is no clear mapping between measurements and tracks. Since the MTT problem is well posed, most vision-based trackers first identify a set of targets in the image. Once the image is reduced to this point-target representation is obtained, traditional MTT can be applied.

2.2.1 Adapting MTT to Cameras

The vision community has used various standard approaches to transform an image into a target set appropriate for MTT. The most common approaches involve background subtraction somewhere in the process. Background subtraction works by building up an appearance model of the background over time[26, 27, 13]. Once a good model has been built, an incoming image is compared against the background and points which do not have background appearance according to the background model are marked as foreground. The foreground image indicates pixels which are not part of the background, and are in some sense interesting. Presumably, they identify pixels corresponding to people. Figure 2-7 shows an example of a foreground image.

.

Once it has been computed, the foreground image is segmented into individual targets, often by a low-pass filter, followed by a connected components pass[14, 8]. However, segmentation on a still image is an ill-defined problem: no matter how clever the algorithm, there is always the possibility of over-segmenting the image (identifying a single target as two targets) or under-segmenting it (identifying two targets as the same target).

Many other algorithms exist for converting an image into a set of targets. Rasmussen[24] uses snakes[16] to track the contour of targets. Wren[31] uses skin tones to identify targets,

[1]The original PDAF actually does handle spurious measurements by modeling a number of false targets and identifying them during the association step. It then omits these false targets from the update step.

which are stipulated to be two hands and a face. Beymer[3] tracks small features, combining them into targets if they move coherently over time. One of the strong points of the last two techniques is the strong influence knowledge about the whereabouts of the target has in selecting features. This thesis takes the trend one step further and completely merges segmentation and tracking into one process, where they are computed jointly.

2.2.2 Clustering

Ideally, each pixel should be fuzzily assigned to each target. An improvement over hard segmentation such as snakes or connected components is to use clustering to build a probabilistic association between pixels and targets. Then the PDAF algorithm will hypothesizes an association between these compound targets and tracks. Segmentation with clustering of nearby pixels doesn't suffer from connected component's over-segmentation, and it can easily incorporate known information about the location of the characteristics of the targets. Since exact boundaries don't need to be computed either, the clustering segmenter doesn't need to suffer from the complexity of snakes.

The intuition behind the clustering segmenter is that a background-subtracted image can be thought of as a mixture of point-cloud generating processes. The clustering algorithm could then identify the locations of these blobs. Once the blobs have been located, any filtering and data association algorithm can be used to update tracks.

The results shown in chapter 5 show how the expectation maximization (EM) algorithm [9, 19] or the K-means algorithm[10] can be used to identify these clusters. EM is given knowledge about the shape of people through a conjugate prior on the covariance matrix of each Gaussian.

Keep in mind the two layers of uncertainty in this algorithm. Each pixel is probabilistically assigned to a blob. Then each blob is probabilistically assigned to each track. One of the features of the camera-based MTT algorithm in this thesis combines these two layers together.

2.3 Summary of Issues

Background subtraction is in itself a difficult problem. If the background subtraction fails, segmentation fails. Incorrect segmentation confuses trackers by introducing spurious targets or failing to report real targets. These failures typically result in dropped tracks (because the individual blobs are too small and are ignored by the tracker), or spurious tracks (because an object is split in two). Clustering addresses this issue to a degree. Under-segmentation is also common when targets approach each other. This typically results in tracks being lost or incorrectly merged and consistently reduces the resolution of the tracker in close quarters.

All the algorithms described so far must be told beforehand the number of targets in the scene. If an object leaves or enters the scene, the tracker performs unpredictably, or must resort to employing some heuristic.

Most of these problems have been addressed using heuristics, at the cost of requiring the user to specify a large number of thresholds which are situation-dependent. In addition, heuristics are brittle, their performance cannot be well quantified, and contribute little to our understanding of a problem.

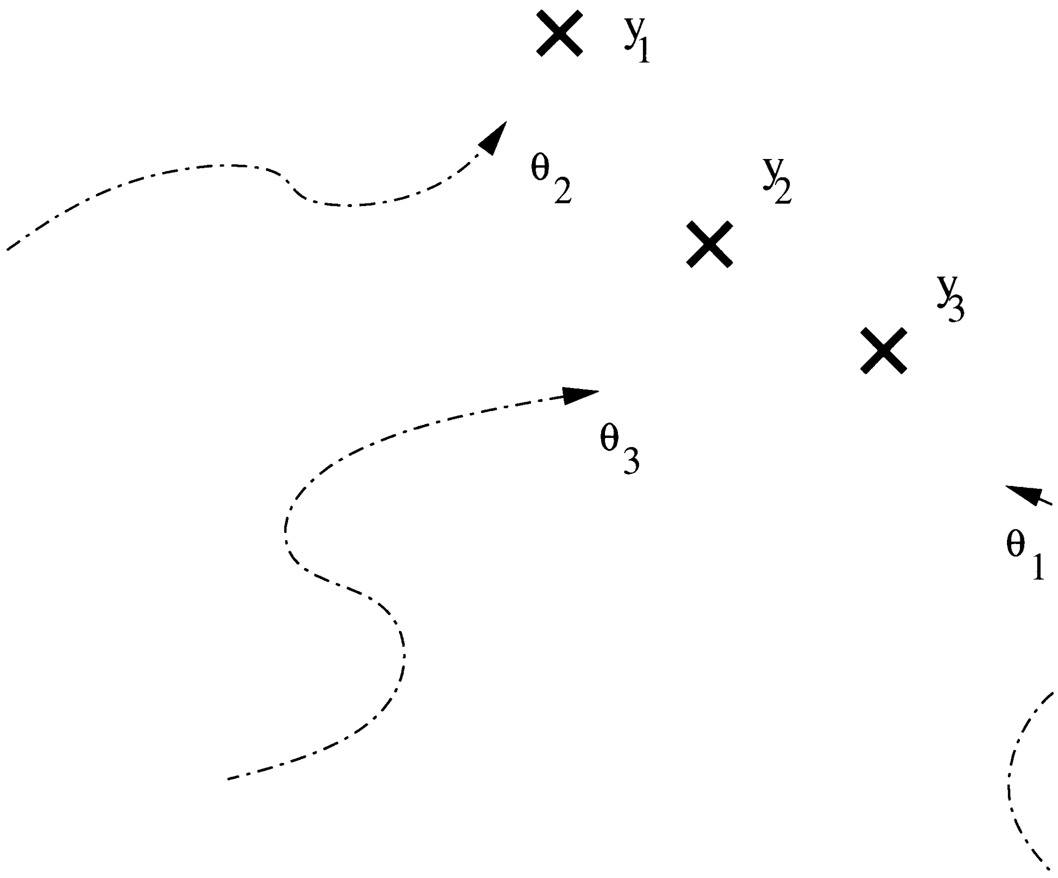

Figure 2-2: A three target tracker. The tracker maintains three tracks. Dashed curved lines represent the track maintained for each target up to time $t-1$. At time t, the location of three objects is measured, but the identity of the objects is unkonwn, so the correspondance between tracks and measurements is unknown. The MTT algorithm must guess at this correspondance and update the three tracks using this new set of measurements. Each track is represented as a probability distribution over locations, as a function of time (not shown).

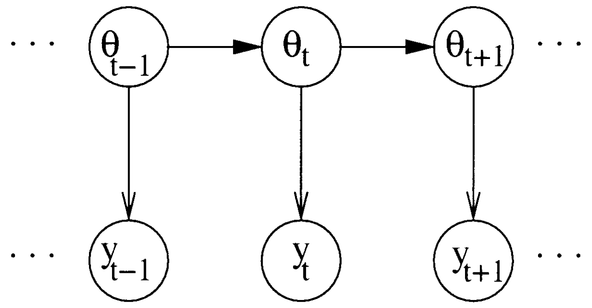

Figure 2-3: Graphical model assumed by the Kalman Filter. The hidden variables θ evolve according to linear Markovian dynamics. The observations y are a linear function of the state, corrupted by white Gaussian noise.

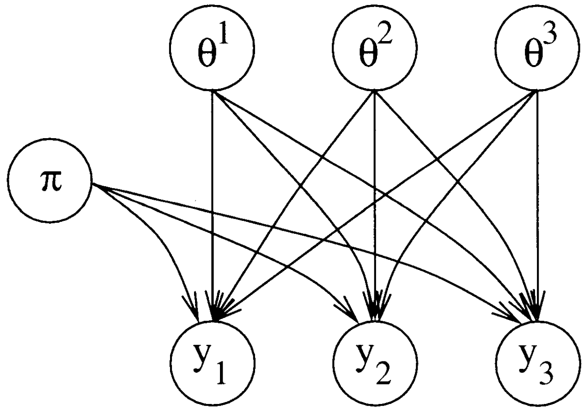

Figure 2-4: Graphical model for MTT. At each time step, each track emits an observation using equation (2.2). These observations are shuffled and measured. The switch π tells each y_m node which θ^k to observe and corrupt.

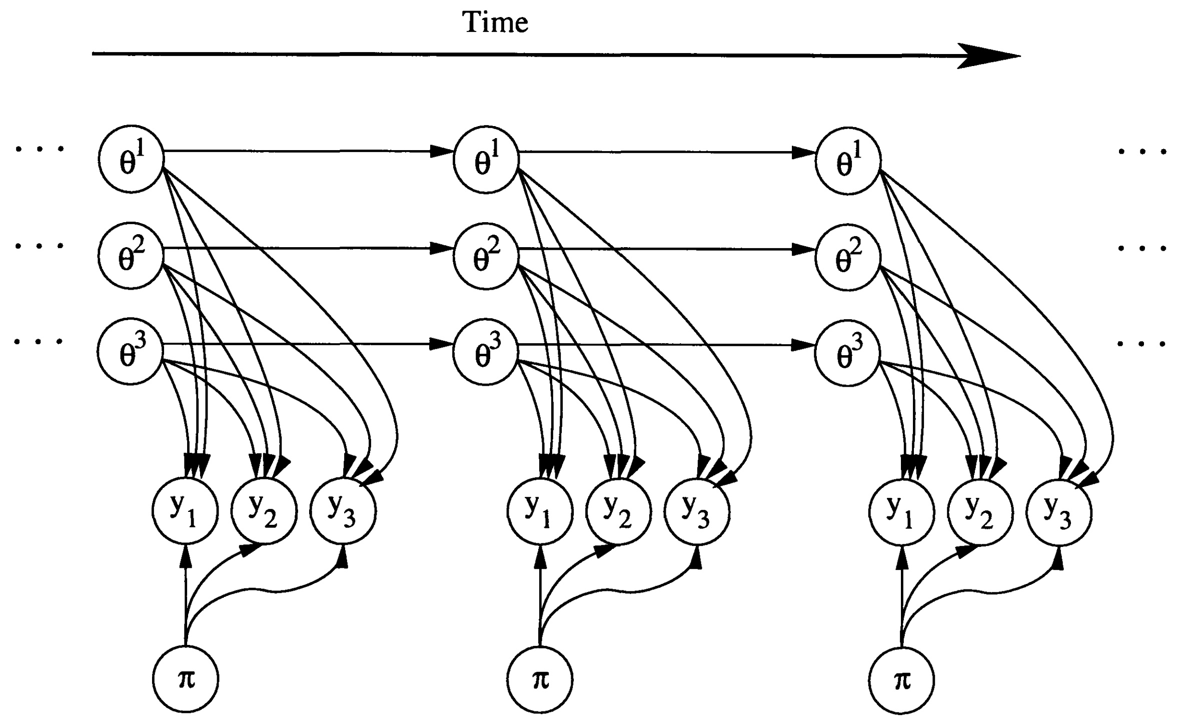

Figure 2-5: The complete graphical model for the MTT problem. This model uses the model of figure 2-4 to emit measurements at each time step. After each emission, the track parameters θ^k are updated according to Markovian dynamics.

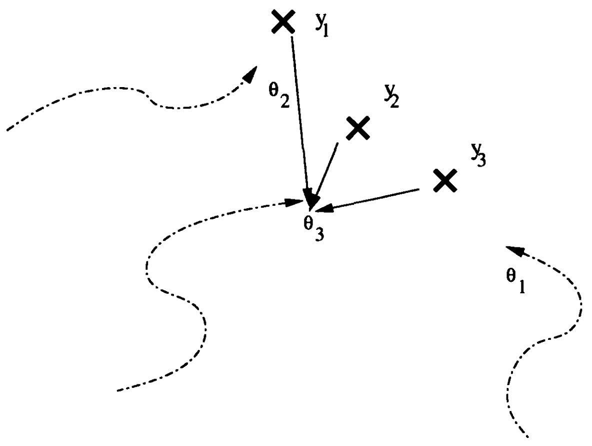

Figure 2-6: PDAF Schematic. The PDAF updates each track independently. For each track, the probability that each measurement came from that track is computed. This probability is used to weight the influence on the track in an update equation.

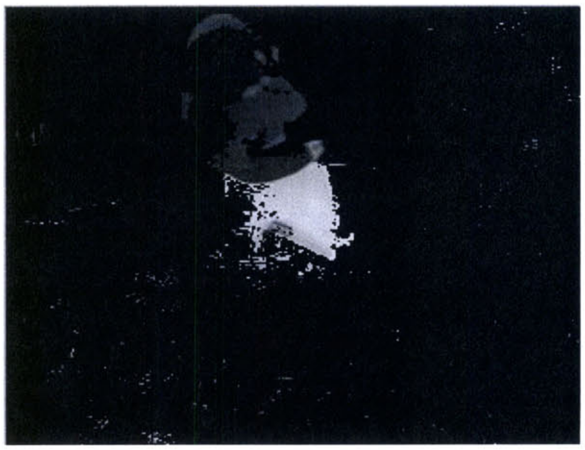

Figure 2-7: Background subtraction result. The background is colored white. Foreground pixels are retained, as they were. The background is similar to that of figure 2-1)

Chapter 3

A Model for Vision-based MTT

Regardless of the point tracking algorithm, all the vision-based algorithms discussed so far assume that the targets tracked are points. At each time step, they perform an initial image segmentation pass where image pixels are assigned to targets and a location is assigned to each target. This segmentation often involves background subtraction [14, 29, 22], connected components[14], snakes[24], region-based segmentation[22, 28] or some kind of clustering, such as EM ([23] or the EM tracker described in section 2.2.2) . This simplification step often leaves the underlying MTT algorithm with false positives and missed targets, which result in poor tracking. These problems stem from the inevitable over- and under- segmentation caused by an independent segmentation step which is uninformed about the tracking context.

To address the issue of poor segmentation, this chapter presents two image formation models based on mixtures of Gaussians. The first model still depends on background subtraction, but performs segmentation and tracking jointly. The second model describes the formation of motion maps and tracks changes in the motion map. Both of these models express the multi-person tracking problem in terms of tracking the evolution of mixtures of Gaussians over time to fit the appearance in the input image.

The first algorithm represents each foreground image as a mixture of Gaussians. As the appearance of the foreground image changes, the mixture model is updated to reflect the location of people.

To avoid background segmentation altogether, this chapter proposes an alternative to the traditional segmentation-for-tracking paradigm. By running a high-pass filter at each pixel, pixels which undergo sudden change over time are highlighted. Since these changes are typically caused by motion, regions with high intensity indicate regions where large motion has occurred. The mixture of Gaussians image formation model is modified to model these motion images. The tracking algorithm can then track the evolution of a mixture of Gaussians in a motion image.

As will be shown in chapter 5, this abstraction, coupled with the the Bayesian inference algorithm presented in chapter 4, helps overcome most of the issues of multi-person tracking.

3.1 People as Clouds of Points

The Bayesian approach advocates the use of a generative model for performing inference. Chapter 2 provided a generative model for the point-MTT problem where the locations of the targets are shuffled by a switch and corrupted by Gaussian noise. It then formulated tracking as estimating the hidden location of the targets given the shuffled, noisy measurements. This section provides a generative model for the appearance of a foreground image. The proposed MTT algorithm tracks people by performing inference on this model.

This generative model represents groups of pixels by a Gaussian, so that the foreground image can be represented by a mixture of Gaussians. The following sketch describes how one might generate two clumps of pixels:

> We are told that a painting is drawn by two collaborating painters. The canvas starts off completely white. At each time step, an arbiter tosses a coin and one of the painters is allowed to paint. To paint, each painter samples an (x,y) location from his 2D Gaussian, and plots that point on the canvas with dark paint. If there is already paint in that location, he applies more dark paint to that location.

Figure 3-1 shows an example of an image formed by this process. The picture is reminiscent of foreground imagery (see figure 2-7). So then one way to characterize a foreground

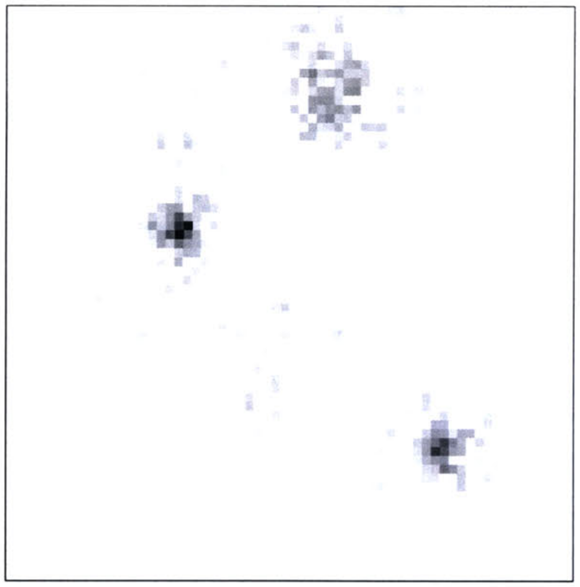

Figure 3-1: Example of mixture of Gaussians for representing images. When a Gaussian emits a location that is already occupied, the value at the location is incremented. This image is generated from 4 Gaussians, one of which has a large covariance for modeling background noise.

image is by describing it in terms of a mixture of Gaussians and mixture weights. The mixture weights are symbolized by the bias of the coin, and the parameters of the Gaussians govern the shape of the blob drawn by each painter. When in section 2.2.2 a classifier was employed to segment the image, it was in fact computing a characterization of the image in terms of these parameters. This chapter explains how to add tracking information to improve the segmentation and how to model the uncertainty in the segmentation on tracking.

As people move, the foreground image changes, and the tracking algorithm must update its mixture of Gaussians to represent the observed foreground image. Locating people from a single foreground image involves computing a posterior distribution over the mixture weights and the parameters of the Gaussians. Chapter 4 will show how to compute this posterior.

Figure 3-2 shows an independence diagram for a mixture of Gaussians. y_i is the 2D location of a point plotted at each turn. μ_k and Σ_k are the means an variances of the

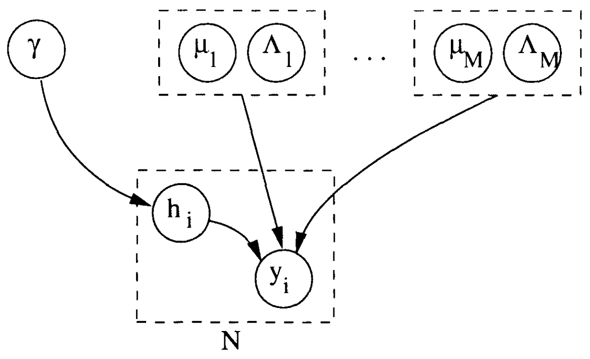

Figure 3-2: Independence diagram for a mixture of Gaussians. Unlike the popular alternative representation, this diagram makes it clear that the component parameters are *not* independent given observations.

Gaussian, and γ represents the mixing weights. The variables h_i are switch variables which determine which μ and Σ are used to parametrize the Gaussian from which the observation y_i is emitted. The plate around h_i and y_i indicates that the experiment is repeated N times, so that a total of N points are produced. Each experiment uses the same instantiation of μ, Σ, and γ, and generates a new h_i and y_i.

3.2 People as Evolving Gaussians

Having modeled how a single frame is generated, it's easy to model the evolution of frames over time. Taking inspiration from the state-space model of the Kalman Filter, the mean of each Gaussian is allowed to evolve using Markovian dynamics. The variance is given a strong prior for retaining the dimensions of a human, and the weights are given a prior for being evenly distributed.

Figure 3-3 shows a graphical model for the Markovian evolution of a mixture of Gaussian. It is similar to the grpahical model model used for the PDAF (figure 2-4), except the observation nodes and the switches emit many points instead of just one.

Interpreting the graph as an evolving mixture of Gaussians provides an intuition for how the inference algorithm is to operate. Clustering algorithms such as K-means[10] or EM for mixture models[9] iterate between estimating the mean of each cluster and assigning points to clusters. Computing these assignments can be thought of as segmenting and identifying targets, and updating the mean of the clusters can be thought of as tracking.

One can see how the two step tracking algorithms of the previous chapter are in fact approximating the correct behavior: they first segment the image, then update tracks. But the proposed image formation model suggests that iterating between tracking and segmentation is more appropriate.

3.3 Avoiding Background Subtraction: Motion Maps

Background subtraction is a notoriously difficult task. A naive implementation of it requires that the system build a background model without people present in the field of view. Even after a background model has been acquired, the system is still susceptible to long term changes in the appearance of the background. There exist robust background subtraction algorithms which are resilient to various calamities of real-life scenes, but they do so at the cost of added CPU cycles[26, 27, 13]. In short, if background subtraction can be avoided, it's usually worth avoiding it.

A simpler way to find targets is to identify areas exhibiting signs of motion. A motion map is a retinotopic image of the scene where the magnitude of each pixel represents the amount of motion at the corresponding location in the world[5]. Using a temporal band pass filter at each input pixel $I(x, y)$ is a simple way to obtain a motion map:

$$I_t^*(x, y) = \alpha(I_t(x, y) - I_{t-1}(x, y) + I_{t-1}^*(x, y))$$

This filter accentuates temporal changes in the value of each pixel. If a pixel does not change much, I^* is close to zero. If there is an abrupt change in the pixel's value, I^* will reflect the change for a brief time. Notice that no filtering is being performed spatially, in the directions of x and y. Filtering is only happening over time, each pixel applying its filter independently of adjacent filters. The motion map I^* can be used instead of the foreground image, after making some modifications to the graphical model. Figure 3-4 shows a typical output of this filter.

Computing motion maps is similar to background subtraction in the sense that the background is allowed to adapt very quickly. As a result, if a target is motionless for even a short time, it will disappear completely from I^*. This notion can be incorporated into the foreground image formation model by allowing the velocity of the Gaussians to influence their weights. Augmenting the parameter set of each target with a speed component and allowing this speed to influence the mixture weight establishes the appropriate conditional relationships. Figure 3-5 enhances the foreground image formation model over time to show the resulting structure. Chapter 5 will provide more intuition about the behavior of motion maps and how the MTT copes with the issues it raises.

3.4 Inference on Markov Chains

Having described two generative models, the question remains: given a sequence of images generated by these models, how can the parameters of the model be tracked over time? The Kalman Filter solves this problem in the case of one track. A filter with an identical structure to the Kalman Filter can be used to perform the tracking, assuming that the density $p(\theta_t|y_t)$ can be computed, where θ_t represents the parameter set of the mixture model at time t and y_t is the image at time t.

To incorporate a new measurement, the Kalman Filter first propagates the estimate of θ obtained from observations up to time t to the next time step. That is, the density $p(\theta_{t+1}|y_0..y_t)$ is predicted from $p(\theta_t|y_0..y_t)$. This predictor then serves as a prior distribution for applying Bayes rule on $p(y_{t+1}|\theta_{t+1})$ to obtain $p(\theta_{t+1}|y_0 \dots y_{t+1})$. This view of the one step predictor as a prior immediately leads to a general method for updating hidden variables in arbitrary Markov Chains. See section 2.1.1 for a more detailed explanation.

This process is very general, since its development made very few assumptions. If the dynamics are Markovian and the observations depend only on the current state, this framework can be used to track hidden variables.

Updating the state estimate with a new observation involves propagating the current estimate of θ to obtain the one step predictor (ie, applying the dynamic model). The difficult step is usually applying Bayes rule to to invert the observation model $p(y|\theta)$ into

$p(\theta|y)$. In the Gaussian case, this step involves computing the Kalman gain matrix. In the case of a mixture of Gaussians, it involves evaluating an intractable integral.

Chapter 4 shows how Variational Bayes can be used to perform this second task approximately. Chapter 5 incorporates the results of chapter 4 into the general tracking framework developed here.

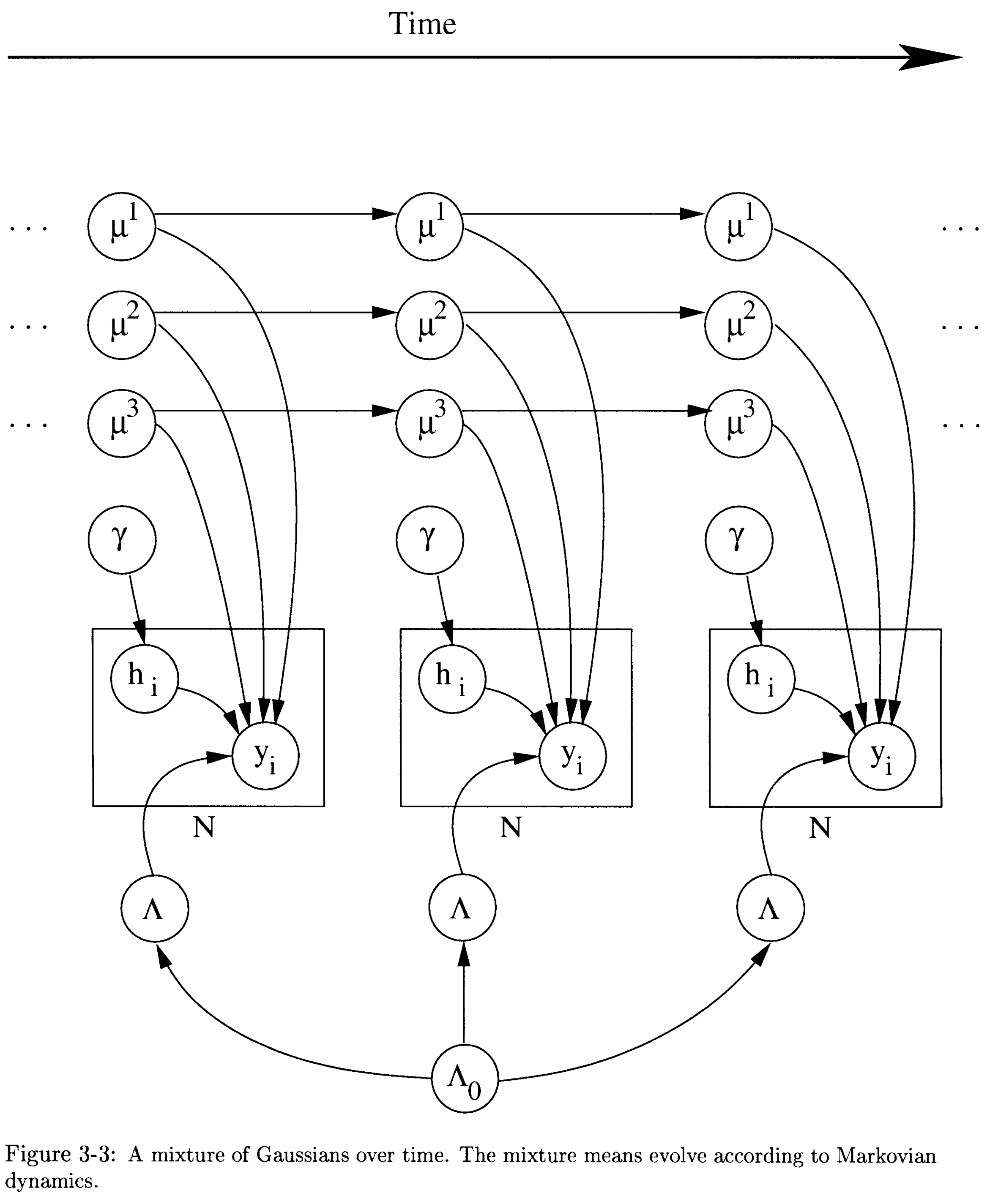

Figure 3-3: A mixture of Gaussians over time. The mixture means evolve according to Markovian dynamics.

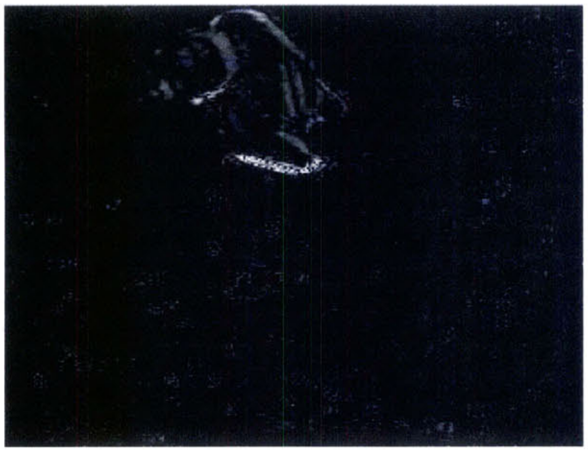

Figure 3-4: A motion map. Generated from a similar sequence as figure 2-7.

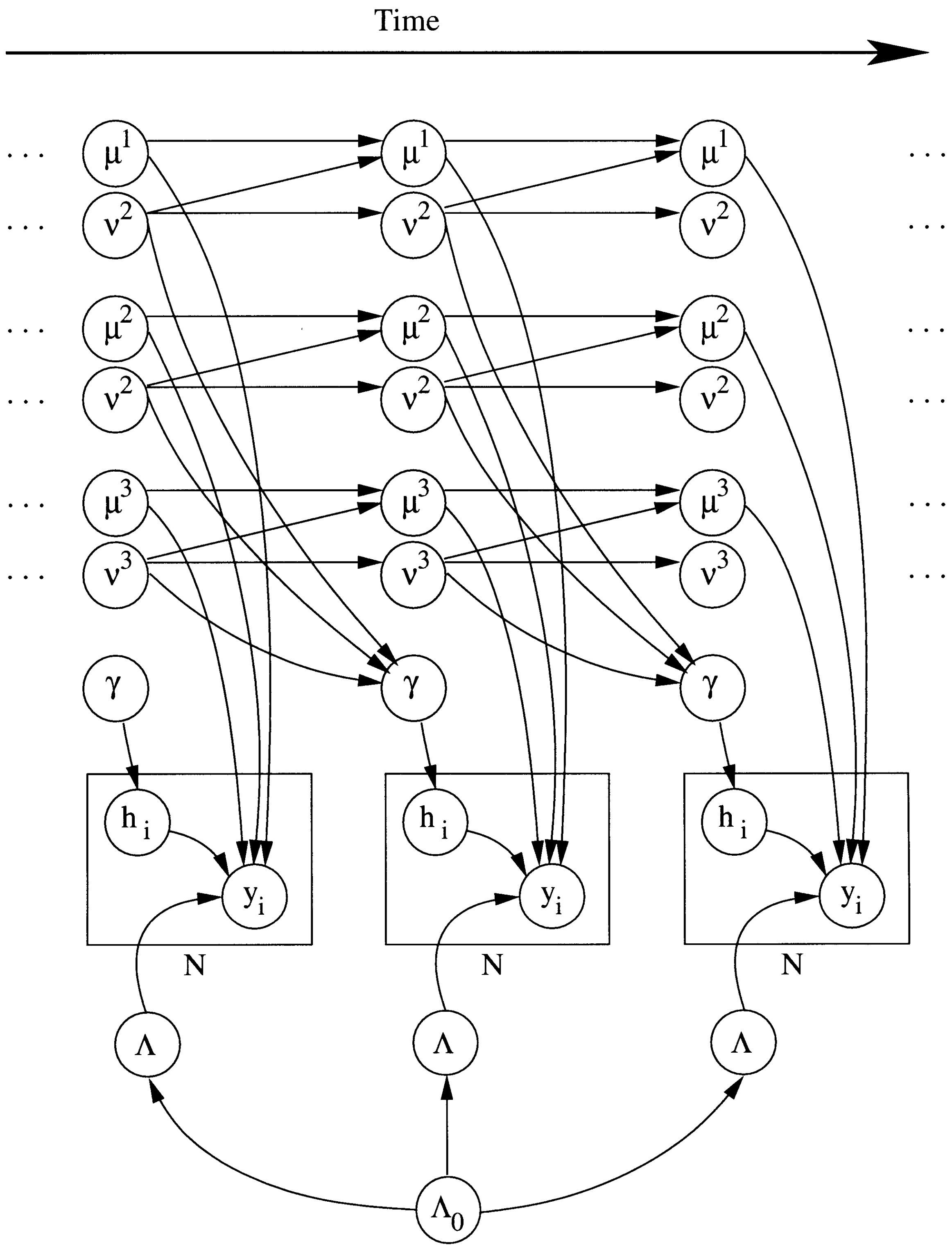

Figure 3-5: Independence diagram for generating a motion map. The mixture weights are determined by the velocity of each person.

Chapter 4

Variational Bayes

Chapter 3 introduced the mixture of Gaussian as a way to represent foreground images and motion maps. It proposed a tracking framework which could be used to track the means of the mixture model over time. This framework incorporates all measurements up to some time t to obtain a distribution over the variables of interest at time t:

$$p(\theta_t|y_0..y_t) \tag{4.1}$$

The framework involved two steps: 1) Computing the one step predictor $p(\theta_t|y_0..y_{t-1})$, which involves simulating the dynamics from the recovered state at time $t-1$, and 2) Inverting $p(y_t|\theta_t)$ to obtain $p(\theta_t|y_t)$ using the one step state predictor as a prior. The last chapter showed that performing these steps integrates all past data to obtain the posterior (4.1).

This chapter explains how to perform step 2 for a mixture of Gaussians. It provides an approximate analytical form for the posterior distribution $p(\theta|Y)$.

4.1 Anatomy of Mixture Models

The graphical model for mixture models is replicated in figure 4-1. μ is the set of mixture component parameters. Each μ^h parametrizes its corresponding component distribution. h itself is drawn from a distribution over the components, and selects the component from

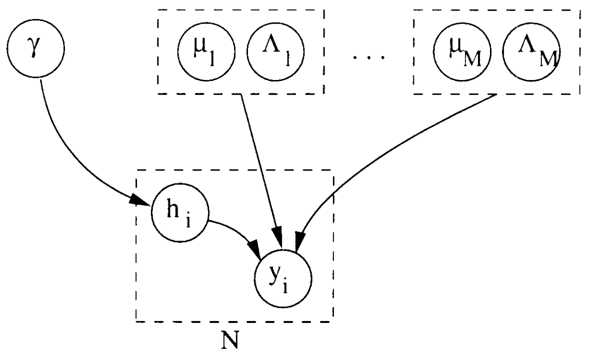

Figure 4-1: Graphical model for a mixture of Gaussian. Replica of figure 3-2.

which observation y_i is drawn. The distribution over components is parametrized by γ so that γ_h is the probability that the hth component is chosen, so γ are the mixing proportions. Note that μ and γ do not change from sample to sample.

The joint distribution over all the elements of a mixture model can be read off the independence diagram:

$$p(y_1..y_N, h_1..h_N, \mu^1..\mu^M, \gamma) = \prod_{i=1}^{N} p(y_i|h_i, \mu)p(h_i|\gamma) \prod_{j=1}^{M} p(\mu^j)p(\gamma)$$

In the case of a mixture of Gaussians, the joint becomes

$$p(\{y_i\}, \{h_i\}, \mu, \gamma) = \prod_i^N \sum_h^M \mathcal{N}(y_i; \mu_u^h, \mu_\Sigma^h)\gamma_h, \tag{4.2}$$

and $\mu^h = [\ u^h \quad \Sigma^h\ ]$. The prior over the mixture means μ_u^h is assumed to be Gaussian:

$$p(\mu_u^h) = \mathcal{N}(\mu_u^h; u_0^h, \Sigma_0^h) = |2\pi\Sigma_0^h|^{-\frac{1}{2}} \exp\left(-\frac{1}{2}(\mu_u^h - u_0^h)^T(\Sigma_0^h)^{-1}(\mu_u^h - u_0^h)\right)$$

In addition, $p(h|\gamma)$ is a multinomial distribution over h with parameter γ:

$$p(h|\gamma) = \gamma_h,$$

and the prior over γ is assumed to be Dirichlet:

$$p(\gamma) = \mathcal{D}(\gamma; \alpha_0^1..\alpha_0^M) = \frac{\Gamma(\sum_h \alpha_0^h)}{\prod_h \Gamma(\alpha_0^h)} \prod_h^M \gamma_h^{\alpha_0^h - 1}$$

Typically, the only observed quantities from a mixture model are the outputs $Y = \{y_1..y_N\}$. This chapter addresses the problem of recovering a distribution over the parameters γ and μ given Y. As will be shown, calculating this distribution is intractable, and approximate methods must be employed. Although $p(\gamma, \mu|Y)$ is intractable, the generative model for the observations is easy to write:

$$p(Y|\mu, \gamma) = \prod_i^N \sum_h^M p(y_i|\mu, h)p(h|\gamma) \tag{4.3}$$

Bayes rule relates this distribution to the desired distribution over parameters:

$$p(\mu, \gamma|Y) = \frac{p(Y|\mu, \gamma)p(\mu, \gamma)}{p(Y)}, \tag{4.4}$$

where $p(Y|\mu, \gamma)$ is defined by equation (4.2). The quantity $p(\mu, \gamma)$ is a prior over the parameters. It represents what we know apriori about these parameters, before observing any data Y. As described in section 3.4, tracking will supply the one step state predictor $p(\mu_t, \gamma_t|y_0..y_{t-1})$ as a prior. Looking at figure 4-1, the marginal distribution of γ and μ are decoupled, so the prior can be written as

$$p(\mu, \gamma) = p(\mu)p(\gamma) = \prod_{h=1}^M \mathcal{N}(\mu_u^h; u_0^h, \Sigma_0^h)\mathcal{D}(\gamma; \alpha_0^1 \ldots \alpha_0^M)$$

It will be shown in chapter 5 that μ and γ can become coupled when modeling motion maps. This coupling happens because the velocity of a blob determines its weight in the mixture model. However, chapter 5 deals with this situation by eventually factorizing the distribution, so it's safe to proceed with the assumption that the priors are independent.

The denominator in equation (4.4) is known as the evidence for the model. In equation (4.4), its job is to normalize the numerator so that it integrates to 1 with respect to μ and

γ. However, it is an extremely useful quantity in itself. To see where it gets its name, consider a mixture model $\mathcal{M}_3$ with 3 mixture components and a mixture model $\mathcal{M}_4$ with 4 mixture components. Suppose we want to know which model to use. Maximum likelihood hypothesis testing suggests a test:

$$p(Y|\mathcal{M}_3) \underset{\mathcal{M}_4}{\overset{\mathcal{M}_3}{\gtrless}} p(Y|\mathcal{M}_4)$$

If the likelihood of $\mathcal{M}_3$ is greater than that of $\mathcal{M}_4$, hypothesis testing suggests picking $\mathcal{M}_3$. Compute $p(Y)$ comes with a well grounded framework for model selection. In fact, as is proposed below, the key to finding the posterior over the parameters turns out to be the model evidence $p(Y)$.

4.2 Use of the Evidence in Computing the Posterior

Equation (2.4) showed how to compute the posterior mean $p(u|Y)$ of a single Gaussian analytically. Is it possible to compute the posterior parameter distribution analytically for a mixture of Gaussians as well? One way to approach this is to evaluate each term in equation (4.4) analytically and to carry out the division analytically.

If the form of the numerator is recognizable as a scaled version of a well known distribution, this computation is simplified, because $p(Y)$ is the scalar which normalizes the numerator, so its value can be assumed. To see how this can be done, rewrite the posterior, and treat $p(Y)$ as a normalizing constant:

$$p(\theta|Y) \propto p(Y|\theta)p(\theta)$$

This equation reveals the functional form of $p(\theta|Y)$ up to a scale factor. If this form is recognizable, for example, if it's a scaled Gaussian or scaled Dirichlet, $p(Y)$ must be the scaling factor which normalizes the integral over θ to 1. Unfortunately, in the general mixture case where there are many observations, the functional form is a sum of M^N Gaussians. Althought one Gaussian is easy to normalize, normalizing M^N Gaussians is intractable, and so in general, $p(\theta|Y)$ is elusive.

One way to see this explosion in complexity is to compute each term in equation (4.4) and to carry out the division analytically. The prior term and the likelihood term were dealt with in the previous section. $p(Y)$ can be obtained by integrating out the parameter θ:

$$p(Y) = \int_\theta p(Y,\theta) = \int_\theta p(Y|\theta)p(\theta) = \int_\mu\int_\gamma p(\mu)p(\gamma)\prod_i^N\sum_h^M p(y_i|\mu,h_i)p(h_i|\gamma) \tag{4.5}$$

This integral is difficult to solve for. It has the form:

$$p(Y) = \int_\theta f_1(\theta)f_2(\theta)f_3(\theta)\ldots f_{M^N}(\theta),$$

so swapping the order of the integration and the product is infeasible. Integration by parts may seem promising but it too is a dead end because it still involves the same number of terms as the integrand. One possible approach is to carry out the product algebraically over the summation over h. Each term corresponds to an instantiation of N assignments dictated by $\{h_i\}$:

$$\begin{aligned}
\prod_i^N\sum_h^M p(y_i|\mu,h)p(h|\gamma) &= (p(y_1,h_1=1|\theta)+p(y_1,h_1=2|\theta)+\ldots+p(y_1,h_1=M|\theta)) \\
&\quad (p(y_2,h_2=1|\theta)+p(y_2,h_2=2|\theta)+\ldots+p(y_2,h_2=M|\theta)) \\
&\quad \ldots \\
&\quad (p(y_N,h_N=1|\theta)+p(y_N,h_N=2|\theta)+\ldots+p(y_N,h_N=M|\theta)) \\
&= p(y_1,h_1=1|\theta)p(y_2,h_2=1|\theta)\ldots p(y_N,h_N=1|\theta) \\
&\quad +p(y_2,h_2=1|\theta)p(y_2,h_2=1|\theta)\ldots p(y_N,h_N=2|\theta) \\
&\quad +p(y_2,h_2=1|\theta)p(y_2,h_2=1|\theta)\ldots p(y_N,h_N=3|\theta) \\
&\quad \ldots \qquad (4.6)\\
&\quad +p(y_2,h_2=M|\theta)p(y_2,h_2=M|\theta)\ldots p(y_N,h_N=M|\theta) \qquad (4.7)
\end{aligned}$$

If there are M components and N points, this results in M^N terms, each of which has to be individually integrated analytically. For this reason, normalizing a mixture of Gaussians is intractable.

Because of the explosion in the number of terms involved in calculating $p(Y)$, we resort to an approximate technique. Various approaches exist for computing the integral of equation (4.5) approximately. Of those that provide an analytical solution, Minka[21] proposes Laplace's method and lower bound integration (a concept which underlies Variational Bayes). Laplace's method fits a scaled Gaussian approximation to the integrand in equation (4.5). However, the Gaussian fit is based on the local behavior of the second derivative of the integrand at the peak, and so tends to not reflect the true $p(Y, \theta)$.

This thesis employs Variational Bayes for computing an approximation to the evidence. As a byproduct of computing an approximation to $p(Y)$, VB computes $q(\theta)$, an approximation to the true posterior $p(\theta|Y)$.

4.3 Approximating p(Y)

This derivation of the approximant $q(\theta)$ is based on Minka's tutorial "Using Lower Bounds to Approximate Integrals"[21]. The tutorial is somewhat abstruse, so this chapter aims to provide a simplified exposition.

As mentioned in the previous section, the object of interest, $q(\theta)$ is an intermediate result obtained when computing an approximation to $p(Y)$. Since we are interested in both entities ($p(Y)$ for determining the number of mixtures and $q(\theta)$ for filtering), this development is particularly well suited.

The goal is to compute an analytic form for

$$p(Y) = \int_\gamma \int_\mu p(Y, \gamma, \mu)$$

Since calculating the integrand was shown to be impractical, we approximate it by a function $g(\gamma, \mu; \phi)$. To simplify notation, the combination of γ and μ will be referred to as θ. The family of functions g is parametrized by ϕ and is chosen because it is easy to integrate:

$$g(\theta, \phi) \approx p(Y, \theta) \tag{4.8}$$

A sensible reaction might be to choose ϕ to make g as close as possible to $p(Y, \theta)$. But what is meant by "close" is important here, since the ultimate goal is to use g as the integrand:

$$p(Y) \approx \rho(Y; \phi) = \int_\theta g(\theta, \phi) \tag{4.9}$$

So for a set of observations Y, ϕ must be chosen to make $\rho(Y; \phi)$ as close as possible to $p(Y)$. It is in this sense that $g(\theta, \phi)$ must resemble $p(Y, \theta)$.

An important subtelty here is that ϕ may depend on Y. One might pick a unique ϕ independent Y to approximate $p(Y)$, but in general, the approximation will be much better if ϕ is allowed to depend on Y. This might sound unintuitive, but keep in mind that g, a function of θ, is already parametrized by Y. ϕ is just an additional parameter, and it's reasonable to believe that allowing ϕ to be a function of Y provides more freedom for the bound.

Approximating $p(Y)$ by $\rho(Y; \phi)$ is only useful if the best ϕ is easy to find. The problem is that a criterion for the best ϕ might look like

$$\phi^*(Y) = \arg\min_\phi |p(Y) - \rho(Y; \phi)|, \tag{4.10}$$

for any given Y. This optimization in general requires that $p(Y)$ be known, but since $p(Y)$ is the unknown entity to be approximated in the first place, it is difficult to solve for ϕ^* without more strenuous assumptions.

4.3.1 Approximating the Integrand

Equation (4.8) proposes finding $g(\theta; \phi)$ which can approximate $p(Y, \theta)$. It will be convenient to pick an approximant which is also a lower bound:

$$g(\theta, \phi) \leq p(Y, \theta) \qquad \forall \theta, \phi,$$

so the approximation (4.9) to the integral becomes:

$$p(Y) \geq \rho(Y;\phi) = \int_\theta g(\theta,\phi)$$

Before, the best ϕ was the one which made ρ as close as possible to $p(Y)$. Now, it can be thought of as the one that *maximizes* ρ:

$$\phi^* = \arg\max_\phi \rho(Y;\phi) = \arg\max_\phi \int_\theta g(\theta,\phi) \tag{4.11}$$

Picking g to be a lower bound on the integrand converts the integral approximation problem of equation (4.10) to a well-posed optimization. Unlike the general formulation of (4.10), this optimization doesn't require a comparison against $p(Y)$.

Jensen's inequality[7] provides one form of the lower bound $g(\theta;\phi)$ for $p(Y,\theta)$. For a summation, it provides the lower bound:

$$\begin{aligned} \ln \sum_i a_i b_i &\geq \sum_i b_i \ln a_i, \qquad \text{provided} \qquad \sum_i b_i = 1 \\ &\text{or,} \\ \sum_i a_i b_i &\geq \prod_i a_i^{b_i} \end{aligned}$$

Since $p(Y,\theta)$ is a summation (see (4.3)), Jensen's inequality is well suited. Applying it verbatim to each of the factors of $p(Y,\mu,\gamma)$ separately gives

$$\begin{aligned} p(Y,\mu,\gamma) &= p(\mu)p(\gamma)\prod_i^N \sum_h^M p(y_i,\mu|h)p(h|\gamma) \\ &\geq g(\mu,\gamma;\phi) = p(\mu)p(\gamma)\prod_i^N \prod_h^M p(y_i,\mu|h)^{p(h|\gamma)} \end{aligned}$$

One of the benefits of this form is that it is easy to integrate: there is no longer an explosion of summed terms. However, instead of providing a family of functions with a tunable parameter, Jensen's inequality has provided a fixed function which does not use the parameter ϕ. In other words, this equation commits $g(\theta)$ to be a fixed lower bound which, but it is not the highest lower bound obtainable through Jensen's. The bound can be tightened by introducing the function ϕ_i for each data point:

$$\begin{aligned} p(Y,\mu,\gamma) &= p(\mu)p(\gamma)\prod_i^N \sum_h^M \frac{p(y_i,h,\mu,\gamma)}{\phi_i(h)}\phi_i(h) \\ &\geq g(\mu,\gamma;\phi) = p(\mu)p(\gamma)\prod_i^N \prod_h^M \left[\frac{p(y_i,h,\mu,\gamma)}{\phi_i(h)}\right]^{\phi_i(h)}, \qquad \sum_h \phi_i(h) = 1 \end{aligned}$$

Note that since Jensen's inequality is being applied to each summation under the product, each data point is equipped with its own ϕ_i. Keep in mind that h is discrete, so ϕ can be fully represented as an M by N matrix. To see that the introduction of ϕ improves the bound, consider the case with one data point:

$$\sum_h^M p(y,h,\mu,\gamma) \geq \prod_h^M \left[\frac{p(y,h,\mu,\gamma)}{\phi(h)}\right]^{\phi(h)} \tag{4.12}$$

vs.

$$\sum_h^M p(y,h,\mu,\gamma) \geq \prod_h^M p(y,\mu|h)^{p(h,\gamma)} \tag{4.13}$$

The value of the bound (4.13) is fixed, but the bound (4.12) can varry with ϕ. The use of ϕ is advantageous because it allows the bound freedom to move around. It worst, the addition of ϕ can't hurt. If $\phi(h) = p(h,\gamma)$,

$$\prod_h^M \left[\frac{p(y,h,\mu,\gamma)}{p(h,\gamma)}\right]^{p(h,\gamma)} = \prod_h^M p(y|h,\mu)^{p(h,\gamma)},$$

which is equal to the bound without ϕ. So having ϕ can't hurt.In fact, the addition of ϕ has a pleasant consequence: a ϕ can always be chosen to to make the lower bound touch $p(Y,\gamma,\theta)$ at any given μ and γ. To prove this claim, take the log of equation (4.12):

$$\ln\sum_h^M p(y,h,\mu,\gamma) \geq \sum_h^M \ln\phi(h)\ln\frac{p(y,h,\mu,\gamma)}{\phi(h)}$$

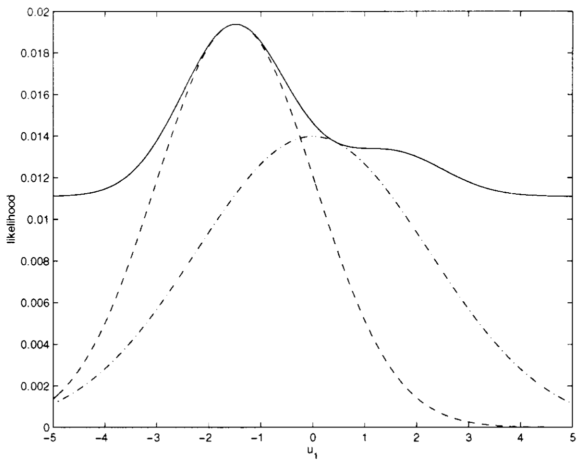

Figure 4-2: Comparison of Jensen's bound on a 2 component mixture. The solid curve is a cross section of $p(Y|\mu,\gamma)$ over one of the means. This distribution is representative of the posterior in that it represents a cross section of likelihood function which ultimately dictates the posterior over means. The dash-dotted curve is the sloppy bound of equation (4.13). The dashed curve is Jensen's bound with the introduction of ϕ. The μ^* used to compute ϕ is the maximum-likelihood μ (ie, the one at the peak of the distribution), so the lower bound was guaranteed to touch the peak of the likelihood function.

Picking $\phi^*(h) = p(h|y,\mu^0,\gamma^0)$ yields

$$\sum_h^M p(h|y,\mu^0,\gamma^0)\ln\frac{p(y,h,\mu^0,\gamma^0)}{p(h|y,\mu^0,\gamma^0)} = \ln p(y,\mu^0,\gamma^0) = \ln\sum_h^M p(y,h,\mu^0,\gamma^0) \tag{4.14}$$

meaning that for a particular Y, ϕ can be chosen to make the bound $g(\mu,\gamma;\phi)$ touch $p(Y,\mu,\gamma)$ at arbitrary μ^0 and γ^0. This is much better than what the bound without ρ can do, since in general, it will not touch $p(Y,\mu,\gamma)$ anywhere. See appendix A for a more detailed description.

When , ϕ is in fact maximizing the bound for a particular Y and μ.

Note that although picking $\phi = p(h|Y,\mu^0,\gamma^0)$ makes $g(\mu;\gamma)$ touch the true integrand, it does not mean that this particular choice of ϕ also maximizes the $\rho(Y;\phi)$. After all, the choice of μ^0 and γ^0 is arbitrary, so there is no reason to believe that the resulting ϕ is the best ϕ. The objective is still to maximize $\rho(Y;\phi)$, and not the integrand. This exposition only served to show that it is beneficial to introduce ϕ because it can be used provide a higher quality bound.

Another important consequence of introducing ϕ is that the integrand lower bound has become decoupled:

$$\begin{aligned} g(\mu, \gamma; \phi) &= p(\mu)p(\gamma) \prod_{i}^{N} \prod_{h}^{M} \left[\frac{p(y_i, h, \mu, \gamma)}{\phi_i(h)}\right]^{\phi_i(h)} \\ &= \left[p(\mu) \prod_{ih} p(y_i|h, \mu)^{\phi_i(h)}\right] \left[p(\gamma) \prod_{ih} p(h|\gamma)^{\phi_i(h)}\right] \prod_{ih} \phi_i(h)^{-\phi_i(h)} \\ &= g(\mu, \phi)g(\gamma, \phi) \end{aligned} \tag{4.15}$$

Thus $g(\mu; \phi)g(\gamma; \phi)$ approximates $p(Y, \mu, \gamma)$ not only by lower bounding it, but by decoupling the influence of μ and γ. Furthermore, $g(\mu; \phi)$ is itself decoupled with respect to the individual μ's:

$$g(\mu, \gamma; \phi) = \prod_{h}^{M} g(\mu^h; \phi)g(\gamma; \phi),$$

so that $\rho(\phi)$ is a product of decoupled integrals:

$$\rho(\phi) = \int_\mu \int_\gamma g(\mu, \gamma; \phi) = \int_\mu \int_\gamma \prod_{h}^{M} g(\mu^h; \phi)g(\gamma; \phi) = \left[\int_\gamma g(\gamma; \phi)\right] \prod_{h}^{M} \left[\int_{\mu^h} g(\mu^h; \phi)\right] \tag{4.16}$$

Figure 4-3 represents the decoupling induced by Jensen's bound. In the typical formulation of variational Bayes, the designer choses a decoupling first and introduces it into the system[17, 12]. In the case of mixtures of Gaussians, Jensen's inequality is what induces the decoupling. The particular decoupling of equation (4.16) models the observations M independent data sets emitted by M Gaussians with different, independent parameters.

This section showed how to design a lower bound on the integrand. The following section describes a general way to tune its parameter ϕ to bring its integral ρ closer to the true evidence.

4.3.2 Optimizing the Integral

There are many ways to optimize $\rho(\phi)$ for ϕ, but the Variational Bayes community often favors Expectation Maximization (EM)[9, 19] as an optimization technique for its robustness

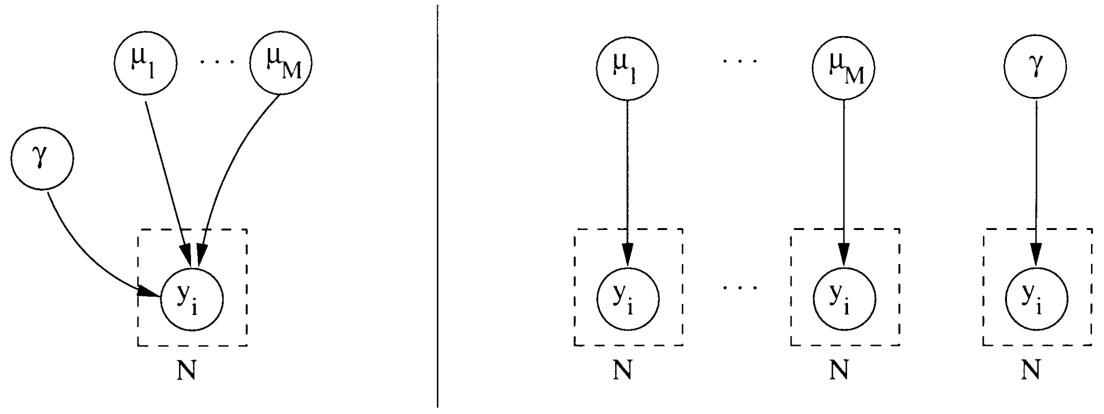

Figure 4-3: The decoupling induced by Jensen's bound. The diagram on the left represents the independence relationship between the observations and the parameters of interest in the true $p(Y, \gamma, \mu)$. The component membership h has been integrated out. The diagram on the right shows the independence relationship induced by Jensen's inequality. The diagram replicates the observations, as if many independent sets of observations were generated.

and guarnatee to find a local minimum. To optimize a function $\rho(\phi)$, EM defines an auxiliary lower bound $D(\phi, q)$:

$$D(\phi, q) \leq \rho(\phi) \qquad \forall \phi, q, \tag{4.17}$$

where q is a free parameter chosen by EM. A q^0 is found to make

$$D(\phi^0, q^0) = \rho(\phi^0) \tag{4.18}$$

for some initial ϕ^0. $D(\phi, q^0)$ still satisfies (4.17), but touches ρ at ϕ^0. As a result, finding a ϕ^1 which increases D cannot result in a decrease in $\rho(\phi)$:

$$\begin{aligned} \phi^1 &= \arg\max_{\phi} D(\phi, q^0) \\ D(\phi^1, q^0) &\geq D(\phi^0, q^0), \qquad \text{so} \qquad \rho(\phi^1) \geq \rho(\phi^0) \end{aligned} \tag{4.19}$$

Thus, ρ can be increased by finding a new ϕ. However, D is no longer touching $\rho(\phi)$ at the new ϕ^1, so the next step of EM involves finding a new q^1 so that equation (4.18) holds for the newly computed ϕ^1. This involves maximizing $D(\phi_1, q)$ over q:

$$\begin{aligned} q^1 &= \arg\max_q D(\phi^1, q) && (4.20) \\ D(\phi^1, q^1) &\geq D(\phi^1, q^0), \qquad \text{and} \qquad D(\phi^1, q^1) = \rho(\phi^1) \end{aligned}$$

To increase $\rho(\phi)$ again, ϕ^2 is computed using equation (4.19) (known as the M-step), followed by equation (4.20), (called the E-step). The process continues until ϕ settles to a fixed value. signifying a local maximum of $D(\phi, q)$. Since at its maximum, $D(\phi)$ touches $\rho(\phi)$, reaching a maximum of D is the same as reaching a maximum of ρ.

To summarize so far, $p(Y)$ is lower bounded by $\rho(Y;\phi)$ because the integrand of $p(Y)$ was lower bounded by g. The ϕ^* which maximizes $\rho(Y;\phi)$ also brings $\rho(Y;\phi)$ closest to $p(Y)$. Finding ϕ^* using EM requires a function $D(\phi, q)$, a lower bound on $\rho(Y;\phi)$.

Jensen's inequality can be used to obtain lower bound D on ρ. In the case of a continuous integral, Jensen's bound is:

$$\ln \int_\theta p(\theta) f(\theta) \geq \int_\theta p(\theta) \ln f(\theta), \qquad \text{provided} \qquad \int_\theta p(\theta) = 1$$

But as before, this bound is tighter if a hidden variable is introduced:

$$\ln \int_\theta p(\theta) f(\theta) \frac{q(\theta)}{q(\theta)} \geq \int_\theta q(\theta) \ln \frac{p(\theta) f(\theta)}{q(\theta)}$$

Keep in mind that in the case of a mixture model, $\theta = [\mu^1, \ldots, \mu^M, \gamma]$, so $\rho(\phi)$ is actually an integral over M μ's and one γ. Jensen's bound can be applied twice to obtain:

$$\begin{aligned} \ln \int_\mu \int_\gamma g(\mu;\phi) g(\gamma;\phi) &\geq \int_\mu q(\mu) \ln \frac{\int_\gamma g(\mu;\phi) g(\gamma;\phi)}{q(\mu)} \\ &= \int_\mu \int_\gamma q(\mu) q(\gamma) \ln \frac{\frac{g(\mu;\phi) g(\gamma;\phi)}{q(\mu)}}{q(\gamma)} \\ &\geq \int_\mu \int_\gamma q(\mu) q(\gamma) \ln \frac{g(\mu;\phi) g(\gamma;\phi)}{q(\mu) q(\gamma)} && (4.21) \\ &= \ln D(\phi, q_\mu, q_\gamma) \end{aligned}$$

Substituting for g results in:

$$\ln D(\phi, q_\mu, q_\gamma) = \int_\mu \int_\gamma q(\mu)q(\gamma) \left[\sum_{ih} \phi_i(h) \ln p(y_i|h, \mu) + \phi_i(h) \ln p(h|\gamma) - \phi_i(h) \ln \phi_i(h)\right] \\ - \int_\mu q(\mu) \ln \frac{p(\mu)}{q(\mu)} - \int_\gamma q(\gamma) \ln \frac{p(\gamma)}{q(\gamma)} \qquad (4.22)$$

EM can now be applied to $\rho(\phi)$ using $D(\phi, q_\mu, q_\gamma)$ as the lower bound. In the E-Step, D is maximized over q_μ and q_γ. In the M-Step, the optimization is over ϕ. The next section describes these steps in detail.

4.3.3 Maximizing over $q(\mu)$

The E-Step finds $q(\mu)$ and $q(\gamma)$ which maximize equation (4.21). The separability of $g(\mu, \gamma; \rho)$ in terms of μ and γ allows us to perform the optimizations independently. Maximizing over $q(\mu)$ requires finding:

$$q^*(\mu) = \arg\max_{q_\mu} \int_\mu q(\mu) \ln \frac{g(\mu; \phi)}{q(\mu)}$$

It's easy to show by substitution that

$$q^*(\mu) = \frac{g(\mu; \phi)}{\int_\mu g(\mu; \phi)} \qquad (4.23)$$

makes D touch ρ, maximizing it (see the analogous development of equation (4.14)). Another way to think about (4.23) is that $q^*(\mu)$ is just $g(\mu; \phi)$, normalized over μ, so that $\int_\mu q^*(\mu) = 1$. The relevant term to normalize in equation (4.15) is

$$g(\mu; \phi) = p(\mu) \prod_h \prod_i p(y_i|h, \mu)^{\phi_i(h)}$$

When $p(\mu)$ is Gaussian, this function turns out to be a scaled Gaussian. Normalizing $g(\mu; \phi)$ then simply involves finding the mean and covariance of the Gaussian. To see this, expand

$g(\mu; \phi)$:

$$\begin{aligned} g(\mu; \phi) &= \prod_h g(\mu_u^h; \phi) \\ &= \prod_h^M p(\mu_u^h) \left|2\pi\mu_\Sigma^h\right|^{-\frac{N}{2}} \exp\left[-\frac{1}{2}\sum_i^N \phi_i(h)(y_i - \mu_u^h)^T(\mu_\Sigma^h)^{-1}(y_i - \mu_u^h)\right] \end{aligned} \qquad (4.24)$$

Since the $g(\mu; \phi)$ is decoupled with respect to individual μ's, each $g(\mu^h; \phi)$ can be normalized individually. Define the following terms:

$$\begin{aligned} K &= \sum_i \phi_i(h) \\ \bar{y} &= \frac{1}{K}\sum_i \phi_i(h) y_i \end{aligned}$$

Using these definitions, and ignoring terms in (4.24) which do not depend on μ,

$$\begin{aligned} g(\mu_u^h; \phi) &\propto p(\mu_u^h) \exp\left[-\frac{1}{2}\sum_i^N \phi_i(h)(y_i - \mu_u^h)^T(\mu_\Sigma^h)^{-1}(y_i - \mu_u^h)\right] \\ &\propto p(\mu_u^h) \exp\left[c_1 - \frac{1}{2}K\bar{y}^T(\mu_\Sigma^h)^{-1}\bar{y} - \mu_u^h) - \frac{1}{2}K(\mu_u^h)^T(\mu_\Sigma^h)^{-1}(\bar{y} - u)\right] \\ &\propto p(\mu_u^h) \exp\left[c_2 - \frac{1}{2}K(\bar{y} - \mu_u^h)(\mu_\Sigma^h)^{-1}(\bar{y} - \mu_u^h)\right] \\ &\propto p(\mu_h^h)\mathcal{N}(\mu_u^h; \bar{y}, \mu_\Sigma^h/K) \end{aligned}$$

Since $p(\mu_u^h)$ is Gaussian, normalizing the final product is identical to normalizing the joint of data and mean in equation (2.4). Applying equations (2.5)-(2.7) yields:

$$\begin{aligned} G &= \Sigma_0(\Sigma_0 + \mu_\Sigma^h/K)^{-1} \\ C_q^h &= (\Sigma_0^{-1} - (\mu_\Sigma^h/K)^{-1})^{-1} \\ m_q^h &= u_0 + G(\bar{y} - u_0) \\ q^*(\mu_u^h; \phi) &= \mathcal{N}(\mu_u^h; m_q^h, C_q^h), \end{aligned}$$

where m_0 and Σ_0 are the mean and covariance of the prior on μ_u^h.

4.3.4 Maximizing over $q(\gamma)$

Just as with $q(\mu)$, finding the maximal $q(\gamma)$ is a matter of normalizing $g(\gamma;\phi)$ over γ:

$$q^*(\gamma) = \frac{g(\gamma;\phi)}{\int_\gamma g(\gamma;\phi)}, \tag{4.25}$$

where the numerator can be suggestively expanded to reveal:

$$g(\gamma;\phi) = p(\gamma)\prod_{ih} p(h|\gamma)^{\phi_i(h)} \propto \prod_h \gamma_h^{\alpha_0^h - 1} \prod_h \gamma_h^{\sum_i^N \phi_i(h)},$$

which has the form of a scaled Dirichlet, since $p(\gamma)$ is a Dirichlet. Hence the normalized version must be a Dirichlet:

$$q^*(\gamma;\phi) = \mathcal{D}(\gamma; \alpha_0^1 + \sum_i \phi_i(1), \alpha_0^2 + \sum_i \phi_i(2), \ldots, \alpha_0^M + \sum_i \phi_i(M)),$$

where, α_0 are the Dirichlet coefficients of the prior $p(\gamma)$.

4.3.5 Maximizing over ϕ

The M-Step must maximize D over ϕ, subject to the constraint that $\phi_i(h)$ is a proper probability distribution over h:

$$\begin{aligned} \phi_i^* &= \arg\max_{\phi_i(h)} D(\phi, q_\mu, q_\gamma) \\ &= \arg\max_{\phi_i(h)} \ln D(\phi, q_\mu, q_\gamma), \qquad \text{s.t.} \qquad \phi_i(h) > 0, \text{ and } \sum_h \phi_i(h) = 1 \end{aligned}$$

If ϕ_i is parametrized in such a way that these conditions are always guaranteed, the optimization can be performed by setting differentials with respect to the parameters to zero, and solving for the parameters.

The domain of $\phi_i(h)$ is $h \in 1..M$. Therefore M real numbers can be used to parametrize

ϕ_i[1]. One way to parametrize $\phi_i(h)$ using the the numbers $\beta^i_1..\beta^i_M$ is described in[4]:

$$\phi_i(h) = \frac{e^{\beta^i_h}}{\sum_{j=1}^{M} e^{\beta^i_j}}$$

For all choices of β_j, the above guarantees that $\phi_i(h)$ is positive, because $e^{\beta}_j > 0$, and that it sums to 1 over h, because the denominator normalizes the distribution. The partial derivative of $\phi_i(h)$ with respect to β^i_j is

$$\frac{\partial}{\partial \beta_j}\phi(h) = \begin{cases} \phi(j) - \phi^2(j), & \text{if} \quad h = j \\ -\phi(j)\phi(h) & \text{if} \quad h \neq j \end{cases}$$

To optimize D with respect to ϕ_i, its derivative with respect to β^i_h is set to 0:

$$\begin{aligned} 0 &= \frac{\partial}{\partial \beta^i_j} \ln D(\phi, q_\mu, q_\gamma) \\ &= \frac{\partial}{\partial \beta^i_j} \int_\mu \int_\gamma q(\mu) q(\gamma) \left[\sum_h \phi_i(h) \ln p(y_i|h,\mu) + \phi_i(h) \ln p(h|\gamma) - \phi_i(h) \ln \phi_i(h) \right] \\ &= \int_\mu \int_\gamma q(\mu) q(\gamma) \left[-\sum_h \phi_i(j)\phi_i(h) \left(\ln p(y_i|h,\mu) + \ln p(h|\gamma) - 1 - \ln \phi_i(h) \right) \right] \\ &\quad + \int_\mu \int_\gamma q(\mu) q(\gamma) \phi_i(j) \left[\ln p(y_i|h=j,\mu) + \ln p(h=j|\gamma) - 1 - \ln \phi_i(h=j) \right] \end{aligned}$$

For a particular observation y_i, the first integral is a constant multiple of $\phi_i(j)$. In addition, $-1 - \ln \phi_i(h)$ is not a function of γ or μ, so it can be pulled out of the integral:

$$\begin{aligned} 0 &= c_i \phi_i(j) - \phi_i(j)(1 + \ln \phi_i(j)) + \phi_i(j) \int_\mu \int_\gamma q(\mu) q(\gamma) \left[\ln p(y_i|j,\mu) + \ln p(j|\gamma) \right] \\ \phi_i(h) &\propto \exp\left(\int_\mu q(\mu) \ln p(y_i|h,\mu) + \int_\gamma q(\gamma) \ln p(h|\gamma) \right) \qquad (4.26) \end{aligned}$$

Equation (4.26) shows how to update ϕ for each data point. This ϕ increases the bound D, making it touch the objective $\rho(\phi)$. Then when D is maximized over ϕ, $\rho(\phi)$ is increased

[1]In fact, only $M - 1$ parameters are necessary, due to the additional constraint that $\phi_i(h)$ must sum up to 1.

as well.

$\phi_i(h)$ can be evaluated in two parts involving μ and γ. The first term of the exponent is an expectation over a quadratic form:

$$\begin{aligned} E_{q(\mu)} & \left[-\frac{1}{2}(y_i - \mu_u^h)^T(\mu_\Sigma^h)^{-1}(y_i - \mu_u^h)\right] \\ & = -\frac{1}{2}(y_i - m_q)^T(\mu_\Sigma^h)^{-1}(y_i - m_q) - \frac{1}{2}\mathrm{tr}((\mu_\Sigma^h)^{-1}C_q), \end{aligned} \tag{4.27}$$

where m_q and C_q are the means and covariance of $q(\mu)^2$. The result follows because the expectation of a quadratic form under a Gaussian is a another quadratic form[25].

The second term of the exponent in (4.26) is:

$$E_{q_\gamma}[\ln p(h|\gamma)] = E_{q_\gamma}[\ln \gamma_h] = \Psi(\alpha_q^h) - \Psi(\sum_j^M \alpha_q^j), \tag{4.28}$$

where $q_{\alpha M}^h$ are the parameters of the Dirichlet distribution $q(\gamma)$, and where the equality shown in [20] was used to obtain the final result. The function Ψ is defined as

$$\Psi(x) = \frac{\Gamma'(x)}{\Gamma(x)} = (\ln \Gamma(x))'$$

where Γ is the Gamma function, the continuous analogue of the factorial function.

Substituting (4.27) and (4.28) into equation (4.26), gives the update for $\phi_i(h)$:

$$\begin{aligned} \phi_i(h) & \propto \exp\left(\int_\mu q(\mu)\ln p(y_i|h,\mu) + \int_\gamma q(\gamma)\ln p(h|\gamma)\right) \\ & \propto \exp\left(-\frac{1}{2}(y_i - m_q)^T(\mu_\Sigma^h)^{-1}(y_i - m_q) - \frac{1}{2}\mathrm{tr}((\mu_\Sigma^h)^{-1}C_q) + \Psi(\alpha_q^h)\right), \end{aligned}$$

To summarize, m_q^h and C_q^h are the parameters of the Gaussian $q(\mu^h)$. α_q are the parameters of the Dirichlet $q(\gamma)$. μ_Σ^h is the covariance of the hth Gaussian in the mixture model.

[2]That $q(\mu)$ is Gaussian is shown in section 4.3.3.

4.3.6 Evaluating the Integral

The algorithm so far computes ϕ^*, $q^*(\mu)$ and $q^*(\gamma)$ for maximizing $\rho(\phi)$. It remains to compute $\rho(\phi^*)$ to obtain the lower bound on $p(Y)$, if that is the ultimate entity of interest.

When EM converges, the lower bound $D(\phi, q_\mu, q_\gamma)$ touches $\rho(\phi)$ at ϕ^*. Therefore, one way to evaluate $\rho(\phi^*)$ is to evaluate equation (4.22) using ϕ^*, q_μ^*, and q_γ^*. But in fact, the previous sections have already done most of the work of (4.22). $\rho(\phi)$ can be computed by reusing these values.

In general, if $p(Y,\theta)$ and $p(Y|\theta)$ are both known, then $p(Y)$ is easy compute:

$$p(y) = \int_\theta p(y,\theta) = \frac{p(y,\theta)}{p(\theta|y)} \tag{4.29}$$

Now recall that $\rho(\phi^*)$ is an integral of $g(\mu;\phi^*)g(\gamma;\phi^*)$ over μ and γ, and that $q^*(\mu)q^*(\gamma)$ are in normalized versions of the g's (see (4.23), (4.25), and (4.16)):

$$\begin{aligned}\rho(\phi^*) &= \int_\mu\int_\gamma g(\mu,\gamma;\phi^*) = \prod_h^M \int_{\mu^h} g(\mu^h;\phi^*) \int_\gamma g(\gamma;\phi^*) \\ &= \frac{g(\gamma;\phi^*)}{q^*(\gamma)} \prod_h^M \frac{g(\mu^h;\phi^*)}{q^*(\mu^h)} \end{aligned} \tag{4.30}$$

Equation (4.30) holds for any μ and γ (just like any value can be used for θ in (4.29). So $\rho(\phi^*)$ can now be evaluated by choosing convenient values for μ and γ and evaluating g using (4.15). Furthermore, the calculation can be easily performed in the log domain, since the g terms involve products and exponents.

4.4 Summary

This chapter has described how to compute the posterior distribution over the parameters of a Gaussian given an image and prior information on the distribution of the parameters. The resulting distributions $q(\mu)$ and $q(\gamma)$ are approximations to $p(\theta,\gamma|Y)$, in that they assume the joint $p(\theta,\gamma,Y)$ factors into $g(\theta)g(\gamma)$. The approximant g is brought as close as possible to the true joint in the sense of equation (4.9), and the q's are computed from this

g.

The algorithm involved a step where parameters of g were first computed:

$$\phi_i(h) \propto \exp\left(-\frac{1}{2}(y_i - m_q)^T(\mu_\Sigma^h)^{-1}(y_i - m_q) - \frac{1}{2}\mathrm{tr}((\mu_\Sigma^h)^{-1}C_q) + \Psi(\alpha_q^h)\right),$$

followed by a step where the q's are computed:

$$q^*(\gamma;\phi) = \mathcal{D}(\gamma; \alpha_0^1 + \sum_i \phi_i(1), \alpha_0^2 + \sum_i \phi_i(2), \ldots, \alpha_0^M + \sum_i \phi_i(M)) \quad (4.31)$$

$$q^*(\mu_u^h;\phi) = \mathcal{N}(\mu_u^h; m_q^h, C_q^h), \quad (4.32)$$

where m_q^h and C_q^h were defined in equation (4.25). The following chapter shows how the q's can be plugged into the Markov chain filtering framework of the previous chapter to track point clouds.

These equations are simple to interpret.

$\phi_i(h)$ is a membership function: it's the probability that the ith pixel emanated from the hth Gaussian. It depends on three factors: it's Mahalanobis distance do the mean of the Gaussian, the certainty in the estimate of the mean of the Gaussian, and the mixture weight of that Gaussian.

$q(\gamma)$ is the distribution over mixture weights. It returns the probability that a given γ is the correct set of mixture weights. It is Dirichlet and depends on the prior probabilities on the mixture weights (the α_0's), and the number of pixels assigned to each Gaussian. If a Gaussian has many pixels assigned to it, its Dirichlet coefficient is higher, giving it more weight.

Finally, $q(\mu)$ is a distribution over means. Its covariance is a function of the size of the blob, divided by the number of pixels assigned to this Gaussian. The more pixels are assigned to this Gaussian, the more certain its mean becomes. Its mean is the average location of every pixel, weighted by their membership function. Priors on the mean and covariance of this distribution are then allowed to modify these values.

Chapter 5

Results

The VB algorithm of the previous chapter was coupled with the Bayesian Markov chain estimation algorithm of chapter 4. The result is a multi-person tracker which can locate, track and count the number of people at about 14 frames per second on an 800 Mhz Pentium III.

The following section reviews the features of the tracker which stem directly from the Bayesian framework. The subsequent section provides sample images from the tracking and discusses the performance of the tracker.

5.1 Features

Sample motion maps and foreground maps were shown in figures 2-7 and 3-4. Since this kind of imagery is well represented by a mixture of Gaussians, the VB mixture estimation algorithm of chapter 4 is well suited for working with this data. Running connected components on these images shows how difficult tracking would be without the benefit of a mixture model's fuzzy clustering. Figure 5-1 shows the result of running connected components on figures 2-7 and 3-4. The Bayesian framework allows the input imagery to be of extremely poor quality.

Another benefit of the Bayesian framework is that complexity is automatically penalized. When fitting a mixture of Gaussians with maximum likelihood (ML) EM (ignoring the

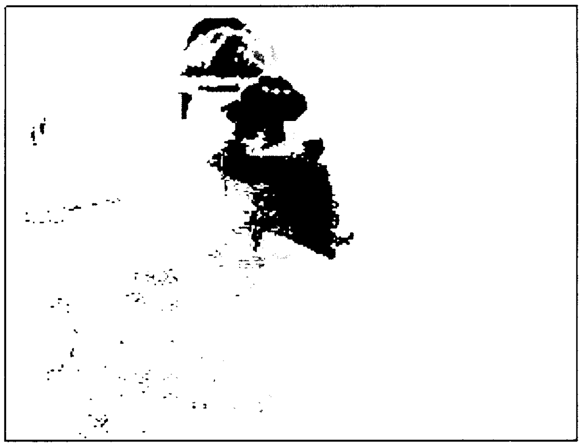

Figure 5-1: Connected components. Running connected components on figure 2-7. The intensity of each pixel is its label. Note the severe over-segmentation.

pathological case where the likelihood goes to infinity[4]), all components are utilized. The more parameters is allowed to tune, the better it fits the data, so that the ML mixture weights are almost never 0. On the other hand, when performing model selection with Bayes rule, more complex models are penalized because the integration of many parameters ultimately lowers the evidence of the model. See appendix C for an explanation. As a result, the number of people in the scene is counted automatically!

The other benefit of the Bayesian approach is that the uncertainty in segmentation is carried through to the tracking. The update step for $q(\mu)$ shows that the degree of certainty in the estimate of the mean of each blob is a function of how many data points contribute to the blob (due to the denominator K in the variance of q_μ). As such, when updating tracks, the segmentation uncertainty plays a role in the uncertainty of the position.

5.2 Tracking Performance

The Bayesian tracker is compared against theclustering tracker described in section 2.2.2. The clustering tracker is a maximum aposteriori (MAP) tracker which does not take advantage of temporal coherency during its segmentation step.

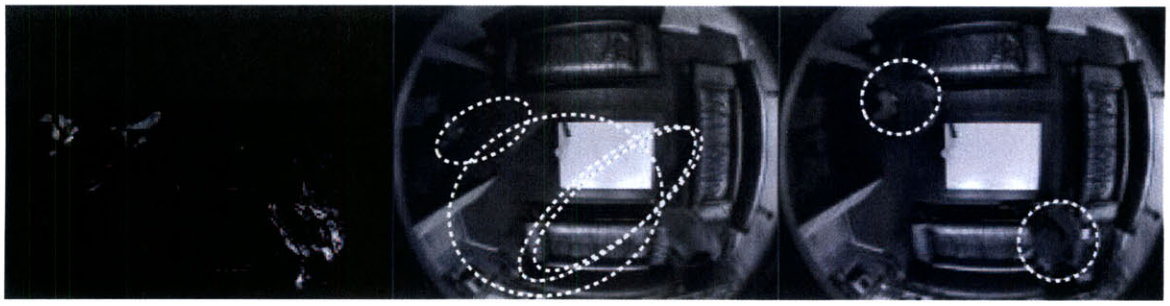

Figure 5-2: Tracking with motion maps. *Left:* Foreground image. *Middle:* MAP clustering tracker. *Right:* VB tracker. Ellipses were hand-drawn on top of the ellipses rendered by the tracker for added emphasis.

The most important shortcoming of the clustering tracker is that it cannot dynamically adjust the number of blobs to track. In these experiments, this number was set to a convenient value. The Bayesian tracker incremented its blob count by 1 every 5 frames, and was allowed to prune mixture components automatically at each iteration of the VB algorithm.

In addition, the performance of the two trackers is compared when the input consists of motion images instead of foreground images. When input images are motion maps, the Bayesian tracker uses the graphical model of figure 3-5. When the input frames are foreground images, the graphical model of figure 3-3 is used. The clustering tracker does not distinguish between the two types of imagery.

Figure 5-2 uses motion maps to compare the two trackers when the targets are far apart. For the clustering tracker, the background blob in the center tries to lock on to the subject on the left but must also compensate for the background noise. The elongated blob between the two targets is stuck in a local minimum. The VB tracker however has correctly identified that there are only two people in the scene and has located them correctly.

Figure 5-3 compares the algorithms under similar circumstances, except with foreground imagery. The VB tracker has incorrectly counted the number of blobs due to severe noise in the foreground image. The targets are correctly tracked otherwise. As an extension to this work, it would be interesting to determine what changes need to be made to handle

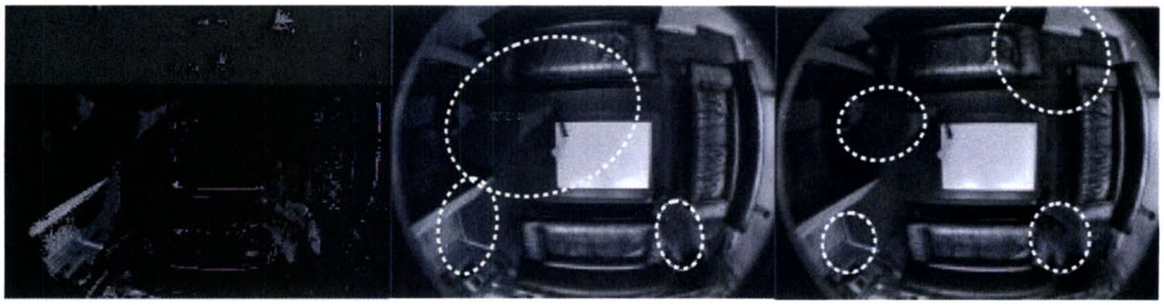

Figure 5-3: Tracking with background subtraction. *Left:* Foreground image. *Middle:* MAP clustering tracker. *Right:* VB tracker. Both trackers have false positives because the background subtraction is consistently sloppy.

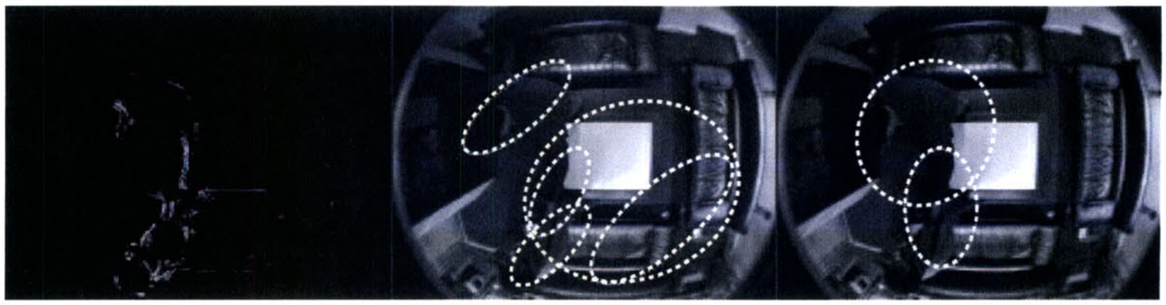

Figure 5-4: Tracking connected targets. *Left:* Motion image. *Middle:* MAP clustering tracker. *Right:* VB tracker.

stationary noise blobs correctly.

Finally, figure5-4 compares the two algorithms in close quarters using motion images. VB again correctly identifies two targets, even though they are connected. The clustering tracker has over-fitted one of the Gaussians to the top subject's arm.

Chapter 6

Conclusions and Extensions

This thesis presented a multi-target tracker which tracks people by modeling image formations. The tracker assumes that the location of a person undergoes Markovian dynamics and that at each frame, a person emits a Gaussian which represents his appearance. This justifies using a mixture of Gaussians as a generative model for images.

Motion maps are also suggested as an alternative to foreground images. The Bayesian framework in which the generative model is derived can easily accommodate motion maps by allowing a person's velocity to influence the mixture weights of the corresponding blob.

Because the algorithm is presented in a Bayesian framework and the inference algorithm is approximately Bayesian, model order selection is a fortuitous byproduct. The tracker can automatically determine the number of people in the scene because the Bayesian framework automatically penalizes complexity in generative models.

Finally, the performance of the tracker is compared to that of a maximum likelihood tracker which does not 1) perform automatic model selection and 2) allow tracking to influence segmentation.

The current tracker does not completely take advantage of the coupling between motion and the intensity of motion maps. The reported results do not focus on this issue because I still find the nature of this coupling elusive. The next step is to examine this coupling in more detail. Once the tracker performs well and reports accurate error information, it will form the first step towards going lower in resolution.

Bibliography

[1] Y. Bar-Shalom and Fortmann T. E. *Tracking and Data Association.* Academic Press, Orlando, Florida, 1998.

[2] S. Basu, T. Choudhury, B. Clarkson, and A Pentland. Learning human interactions with the influence model. Technical report, MIT Media Lab, June 2001. Submitted.

[3] David Beymer and Jitendra Malik. Tracking vehicles in congested traffic. In *SPIE*, volume 2902, Transportation Sensors and Controls: Collision Avoidance, Traffic Management, and ITS, pages 8–18, 1996.

[4] C. Bishop. *Neural Networks and Pattern Recognition.* Oxford University Press, 1995.

[5] A. Bobick and J. Davis. The representation and recognition of action using temporal templates. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(3):257–267, 2001.

[6] K. Buckley, A. Vaddiraju, , and R. Perry. A new pruning/merging algorithm for mht multitarget tracking. In *Radar*, May 2000.

[7] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory.* Wiley, August 1991.

[8] Trevor Darrell, David Demirdjian, Neal Checka, and Pedro Felzenszwalb. Plan-view trajectory estimation with dense stereo background models. In *International Conference on Computer Vision*, 2001.

[9] A.P. Dempster, N.M. Laird, and D.B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society*, 39(1):1–38, 1977.

[10] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis.* Wiley, New York, 1973.

[11] T.E. Fortmann, Y. Bar-Shalom, and M. Scheffe. Sonar tracking of multiple targets using joint probabilistic data association. In *IEEE Journal of Oceanic Engineering*, volume 8, pages 173–184, 1983.

[12] Z. Ghahramani and G.E Hinton. Variational learning for switching state-space models. *Neural Computation*, 12(4):963–996, 1998.

[13] Daniel Gutchess, Miroslav Trajkovic, Eric Cohen-Solal, and Anil Jain Damian Lyons. A background model initialization algorithm for video surveillance. In *International Conference on Computer Vision*, 2001.

[14] Michael Harville, Gaile Gordon, and John Woodfill. Foreground segmentation using adaptive mixture models in color and depth. In *IEEE Workshop on Detection and Recognition of Events in Video*, 2001.

[15] C. Hue, J-P. Le Cadre, and P. Peréz. A particle filter to track multiple objects. In *IEEE Workshop on Multi-Object Tracking*, pages 61–68, 2001.

[16] M. Isard and A. Blake. Contour tracking by stochastic propagation of conditional density. In *European Conference on Computer Vision*, pages 343–356, 1996.

[17] T. Jaakkola. *Tutorial on variational approximation methods.* MIT Press, 2000.

[18] J. Krumm, S. Harris, B. Meyers, B. Brumitt, M. Hale, and S. Shafer. Multi-camera multi-person tracking for easyliving. In *IEEE Workshop on Visual Surveillance*, July 2000.

[19] T.P. Minka. Expectation-maximization as lower bound maximization. Technical report, MIT Media Lab, http://vismod.www.media.mit.edu/˜tpminka/papers/em.html, 1998.

[20] T.P. Minka. Bayesian inference, entropy, and the multinomial distribution. Technical report, MIT Media Lab, http://vismod.www.media.mit.edu/˜tpminka/papers/multinomial.html, 2000.

[21] T.P. Minka. Using lower bounds to approximate integrals. Technical report, MIT Media Lab, http://www.media.mit.edu/˜tpminka/papers/rem.html, 2001.

[22] A. Mittal and L. Davis. Unified multi-camera detection and tracking using region-matching. In *IEEE Workshop on Multi-Object Tracking*, pages 3–10, 2001.

[23] N. Oliver, A. Pentland, and F. Berard. Lafter: Lips and face real time tracker. In *Computer Vision and Patt. Recog.*, 1997.

[24] C. Rasmussen. Joint likelihood methods for mitigating visual tracking disturbances. In *IEEE Workshop on Multi-Object Tracking*, pages 69–76, 2001.

[25] Sam Roweis. Gaussian identities. Technical report, Gatsby Computational Neuroscience Unit, http://www.gatsby.ucl.ac.uk/˜roweis/notes.html, july 1999.

[26] Chris Stauffer and W.E.L Grimson. Adaptive background mixture models for real-time tracking. In *Computer Vision and Pattern Recognition*, June 1999.

[27] Bjoern Stenger, Visvanathan Ramesh, Nikos Paragios, Frans Coetzee, and Joachim Buhmann. Topology free hidden markov models: Application to background modeling. In *International Conference on Computer Vision*, 2001.

[28] Chang T-H and Shaogang G. Tracking multiple people with a multi-camera system. In *IEEE Workshop on Multi-Object Tracking*, pages 19–26, 2001.

[29] K. Teknomo, Y. Takeyama, and H. Inamura. Frame-based tracing of multiple objects. In *IEEE Workshop on Multi-Object Tracking*, pages 11–18, 2001.

[30] C. Wren, A. Azarbayejani, T. Darrell, and A. Pentland. Pfinder: Real-time tracking of the human body. *SPIE Conference on Integration Issues in Large Commercial Media Delivery Systems*, 1995.

[31] C. Wren and A. Pentland. Dynamic models of human motion. In *Proceedings of Face and Gestures*, 1998.

Appendix A

Jensen's Bound on a Simple Mixture

The development of Variational Bayes for mixtures of Gaussians relied on Jensen's inequality to provide a lower bound on the observation model:

$$p(Y|\mu,\gamma) = \prod_{i}^{N} \sum_{h}^{M} p(y_i,\mu|h)p(h|\gamma)$$

In a 1D two mixture component problem, this function is plotted as a function of μ in figure A-1. The observations are

$$Y = [-1.5, -1, 0.5, 1.5]$$

The aim of chapter 5 was to normalize this likelihood function, weighted by priors on μ and γ so that it sums to 1 when integrated over μ and γ.

Figure A-2 shows a 1D cross-section of $p(Y;\mu,\gamma)$. Two bounds are compared: Jensen's bound without tunable parameter ϕ:

$$\prod_{i=1}^{4} \prod_{h=1}^{2} p(y_i|h,\mu)^{p(h|\gamma)}, \tag{A.1}$$

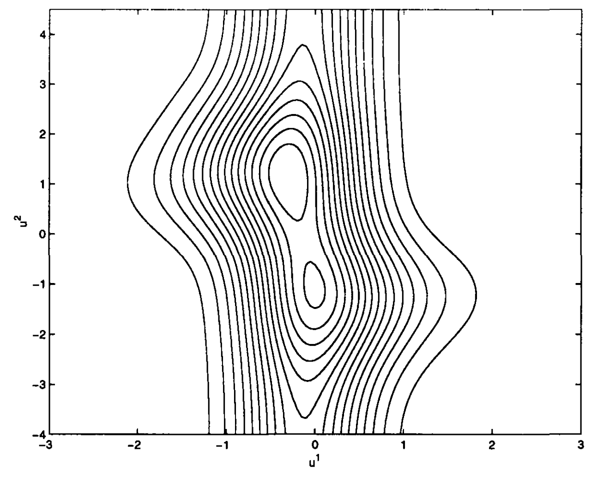

Figure A-1: Contour plot of the likelihood function $p(Y|\mu,\gamma)$. γ is held fixed at $\gamma = [0.9, 0.1]$.

and the bound with ϕ:

$$\prod_{i=1}^{4}\prod_{h=1}^{2}\left[\frac{p(y_i,h|\mu,\gamma)}{p(h|y_i,\hat{\mu}^h)}\right]^{p(h|y_i,\hat{\mu}^h)} \tag{A.2}$$

The simple bound of equation (A.1) seems like a fine bound, but as shown in figure A-3, it can become very flat in certain locations compared to the bound of equation (A.2). The graphs where generated by plugging in successive values of μ and evaluating the corresponding equation.

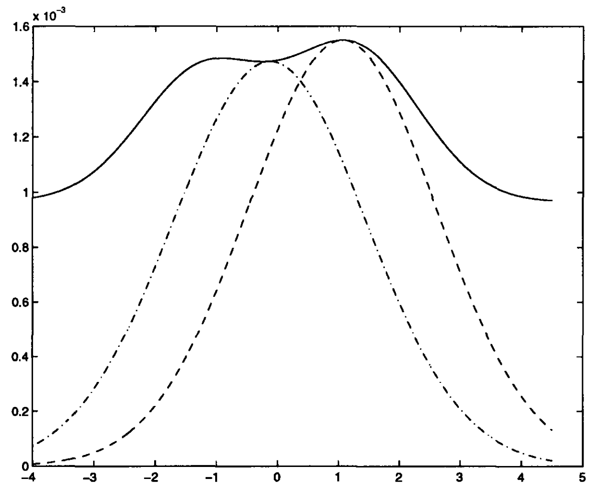

Figure A-2: Cross section of likelihood function with bounds.. The solid curve is a 1D cross section of the likelihood curve of figure A-1 through $\mu^1 = -0.15$. The dash-dotted line is a lower bound obtained with the simple lower bound of equation (4.13). The dotted line is a lower bound fitted using (4.12), with $\hat{\mu}$ set to the maximum-likelihood spot in the cross section.

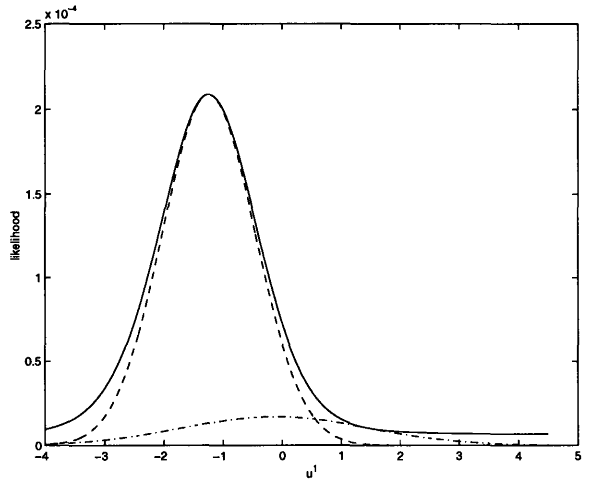

Figure A-3: Cross section at $\mu^1 = 1.45$. The parameterless lower bound is very flat, whereas the ϕ bound can be adjusted to fit the peak.

Appendix B

Source Code for 1D Gaussian Mixture

The following is MATLAB 6 source code for computing the posterior parameter estimate of a 1D mixture of Gaussians.

```
D = 1;
M = 5;
x = [randn(D,30), randn(D,70)+6]*1e-2;
N = length(x);

% Priors on the mixture mean.
V_priors = 10000*ones(1,M); % Covariance of the mean
m_priors = zeros(1,M);      % Mean of the mean.
V = 1e-4*ones(1,M);         % The covariance of each blob. Fixed.

q_ij = rand(M,N);           % \phi.
q_ij = q_ij ./ repmat(sum(q_ij,1),M,1);

while 1
  K = sum(q_ij,2);
  if any(K == 0)            % If one of the components has coefficient
       kil = find(K==0);    % of 0, get rid of the component.
       q_ij(kil,:) = [];
       V_priors(kil) = [];
       m_priors(kil) = [];
       V(kil) = [];
       K(kil) = [];
       q_mean_v(kil) = [];
       q_mean_m(kil) = [];
       M = length(K);
  end

  average = (q_ij * x')./K;

  psis_1 = (gammaln(1+sum(q_ij,2)+1e-5)-gammaln(1+sum(q_ij,2)))*1e5;
  psi_2 = (gammaln(2+length(x)+1e-5)-gammaln(M+length(x)))*1e5;

  for i=1:M
    q_mean_v(i) = inv(inv(V(i)/K(i)) + inv(V_priors(i)));
    q_mean_m(i) = q_mean_v(i)*(inv(V(i)/K(i))*average(i) + inv(V_priors(i))*m_priors(i));

    q_ij(i,:) = exp(-0.5*log(det(2*pi*V(i))) ...
                -0.5*((x-q_mean_m(i)).*(x-q_mean_m(i)))*inv(V(i)) ...
                -0.5*trace(q_mean_v(i)*inv(V(i))) + ...
                (psis_1(i)-psi_2));
  end

  q_ij = q_ij ./ repmat(sum(q_ij,1),M,1);

  fprintf('---');
  q_mean_m
  sum(q_ij,2)'
end
```

Appendix C

Why Bayes Rules

In the model selection problem, we are given a data set and are asked to identify the generative model whence the data came.

Consider the model selection task where a data set may have been either generated either independently of x or according to x:

$$\begin{aligned}\mathcal{M}_1: &\quad p(Y|x,\theta) = p(Y|\theta)\\ \mathcal{M}_2: &\quad p(Y|x,\theta) \neq p(Y|\theta),\end{aligned}$$

For example,

$$\begin{aligned}\mathcal{M}_1: &\quad y_i = [\mu, 0] + [1,1]w\\ \mathcal{M}_2: &\quad y_i = [\mu, x] + [1,1]w,\end{aligned}$$

where w is zero mean and known variance.

The data set Y is given to someone who performs model selection using maximum likelihood. They differentiate the likelihood function with respect to θ and x, find the most likely parameters $\theta^{1,2}$ and $x^{1,2}$ in both models, and notice that under $\mathcal{M}_2$, the likelihood is greater. One way to write this is:

$$
\begin{aligned}
\mathcal{L}[\mathcal{M}_1] &= \int_x \int_\theta p(Y|\theta)\delta(\theta - \theta^1) \\
\mathcal{L}[\mathcal{M}_2] &= \int_x \int_\theta p(Y|\theta, x)\delta(\theta - \theta^2)\delta(x - x^2) \\
\mathcal{L}[\mathcal{M}_1] &\gtrless^{\mathcal{M}_1}_{\mathcal{M}_2} \mathcal{L}[\mathcal{M}_2]
\end{aligned}
$$

Of course, this happens because the data will by chance happen to not be zero mean in the second dimension even if $\mathcal{M}_1$ did generate the data, and so they will use x to compensate for the offset in the mean. Bad move!

The Bayesian however, performs an integration over x and θ (assuming uniform priors) to marginalize out the parameters and to obtain the evidence $p(Y)$ under both models, then reports the model under which the evidence is greatest:

$$
\begin{aligned}
p(Y|\mathcal{M}_1) &= \int_x \int_\theta p(Y|\theta)p(\theta) && \text{(C.1)} \\
p(Y|\mathcal{M}_2) &= \int_x \int_\theta p(Y|x, \theta)p(x)p(\theta) && \text{(C.2)} \\
p(Y|\mathcal{M}_1) &\gtrless^{\mathcal{M}_1}_{\mathcal{M}_2} p(Y|\mathcal{M}_2)
\end{aligned}
$$

If the data originated from $\mathcal{M}_1$, integrating over x in (C.2) causes the likelihood function to be sampled at sub-optimal locations, not just at 0. However, not integrating over x in equation (C.1) keeps x at 0, and so obtains a much higher likelihood. So the Bayesian is likely to identify the correct model, whereas the fellow using maximum likelihood will always favor models with more parameters because more parameters allow him to express the peculiarities of the particular instantiation of the dataset.

[illegible]