14

Recursive Estimation-Maximization (R.E-M.)

An algorithm for segmentation

by

Panayiotis Peter (Pan) Kamvysselis

Submitted to the Department of Electrical Engineering and Computer Science

in Partial Fulfillment of the Requirements for the Degrees of

Bachelor of Science in Electrical Engineering and Computer Science

and Master of Engineering in Electrical Engineering and Computer Science

at the Massachusetts Institute of Technology

May 22, 1998

Author___

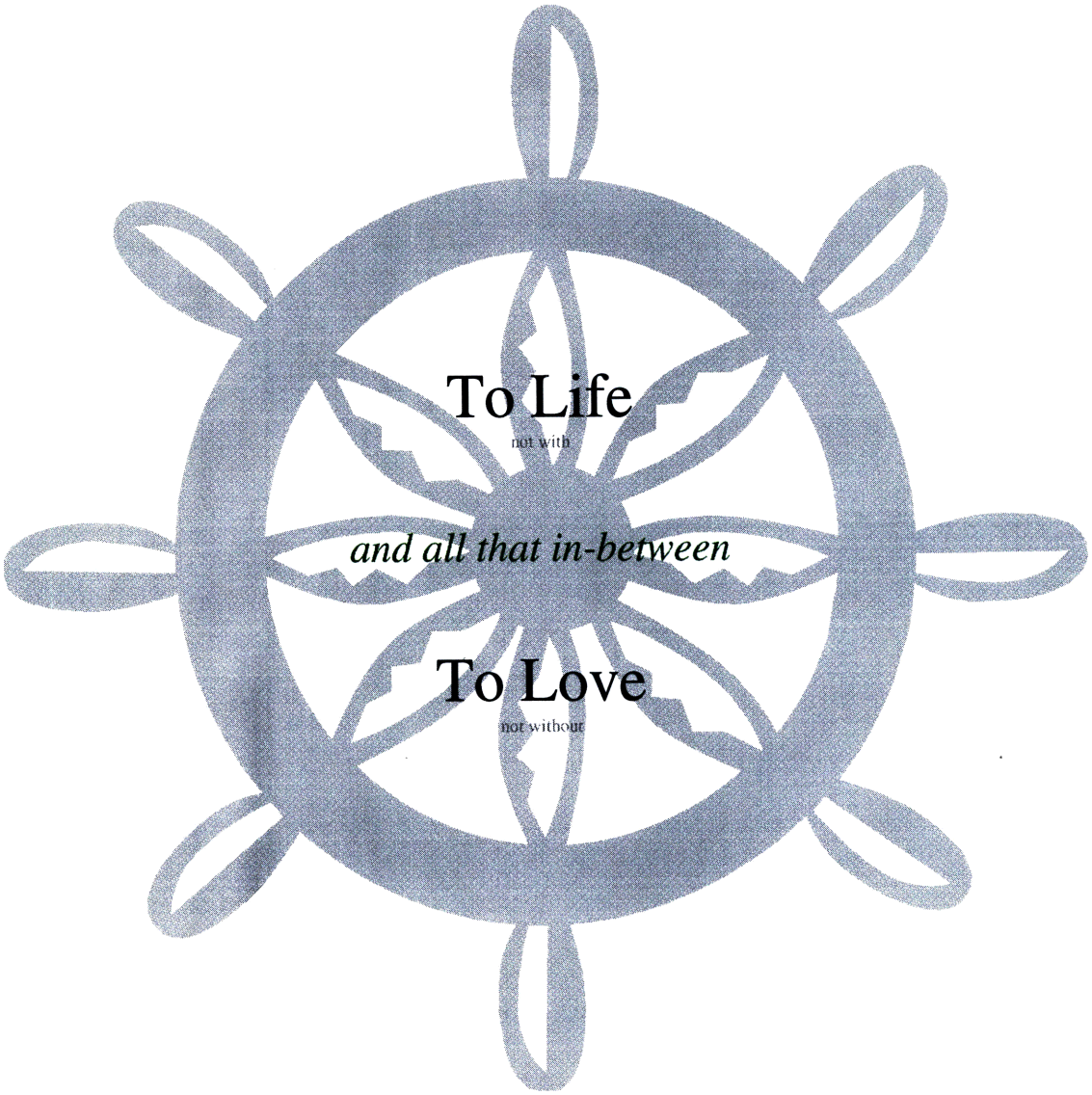
Department of Electrical Engineering and Computer Science
May 22, 1998

Certified by_____

Randall Davis
Thesis Supervisor

Accepted by_____

Arthur C. Smith
Chairman, Department Committee on Graduate Theses



To Life

not with

and all that in-between

To Love

nor without

Recursive Estimation-Maximization (R.E-M)
An algorithm for segmentation

by
Panayiotis Peter (Pan) Kamvysselis

Submitted to the
Department of Electrical Engineering and Computer Science

May 22, 1998

In Partial Fulfillment of the Requirements for the Degree of
Bachelor of Science in Electrical Engineering and Computer Science
and Master of Engineering in Electrical Engineering and Computer Science

ABSTRACT

We present a model-based segmentation algorithm, R.E-M for recursive estimation-maximization. The algorithm iterates between three steps: the R.R-step for recursion; the R.E-step for estimation; and the R.M-step for maximization.

During the R.E-step, data points are assigned to models by measuring deviation from prediction. During the R.M-step, model parameters are estimated using the assignment of the data points. During the R.R-step, a new model is added to improve the prediction of the models. The algorithm iterates between the three steps until a global cost function reaches zero.

We demonstrate the algorithm on one-dimensional Gaussian mixtures and grade distributions from real exams. We also use R.E-M to tackle the problem of motion segmentation. In particular, we present an approach to segmentation using optical flow information in the case of occluded, rigid bodies moving about fixed centers of rotation. We present segmentation results for simulated flow and two-image sequences of the wheels of working clocks.

The algorithm is guaranteed to terminate, is deterministic, and requires no prior knowledge of the number of classification clusters. R.E-M converges rapidly and outperforms E-M for Gaussian mixtures in most cases. We discuss the correspondence of the global cost function's zero and the optimal number of classification clusters. We also present other cases of segmentation for which R.E-M might be useful.

Thesis Supervisor: Randall Davis
Title: Professor Electrical Engineering and Computer Science

Table of Contents

List of figures

List of equations

Equation (2-17) $$z_i = \begin{cases} 1 & \text{if x comes from model i} \\ 0 & \text{otherwise} \end{cases}$$ 37

Equation (2-18) $$P(\mathbf{x}^n, x) = \prod_{i=1}^{K} \left[\pi_i \cdot P_i(\mathbf{x}^n \mid \theta_i)\right]^{z_i}$$ 38

Equation (2-19) $$l(\mathbf{X}) = \sum_n \sum_i z_i^n \log\left[\pi_i \cdot P_i(\mathbf{x}^n \mid \theta_i)\right]$$ 38

Equation (2-20) $$Q(\mathbf{X}) = \sum_n \sum_i h_i^n \log\left[\pi_i \cdot P_i(\mathbf{x}^n \mid \theta_i)\right]$$ 38

Equation (2-21) $$h_i^n = \frac{\pi_i P(x^n \mid \theta_i)}{\sum_{j=1}^{K} \pi_j P(x^n \mid \theta_j)}$$ 38

Equation (2-22) $$h_i^n = \frac{\pi_i P(x^n \mid \theta_i)}{\sum_{j=1}^{K} \pi_j P(x^n \mid \theta_j)} \quad \text{E-step}$$ 38

Equation (2-23) $$\hat{\Theta} = \arg\max_{\Theta} \sum_n \sum_i h_i^n \log\left[\pi_i \cdot P_i(\mathbf{x}^n \mid \theta_i)\right] \text{ M-step}$$ 38

Equation (2-24) $E(x+udt, y+vdt, t+dt) = E(x,y,t)$ 39

Equation (2-25) $(Ex,Ey)*(u,v) = -Et.$ 39

Equation (2-26) $$\frac{E_t}{\sqrt{E_x^2 + E_y^2}}$$ 39

Equation (2-27) $$E_x \approx \left[\begin{array}{l} +\frac{1}{4\delta x}\left(E_{i+1,j,k} + E_{i+1,j,k+1} + E_{i+1,j+1,k} + E_{i+1,j+1,k+1}\right) \\ -\frac{1}{4\delta x}\left(E_{i,j,k} + E_{i,j,k+1} + E_{i,j+1,k} + E_{i,j+1,k+1}\right) \end{array}\right]$$ 39

Equation (2-28) $$E_y \approx \left[\begin{array}{l} +\frac{1}{4\delta x}\left(E_{i,j+1,k} + E_{i,j+1,k+1} + E_{i+1,j+1,k} + E_{i+1,j+1,k+1}\right) \\ -\frac{1}{4\delta x}\left(E_{i,j,k} + E_{i,j,k+1} + E_{i+1,j,k} + E_{i+1,j,k+1}\right) \end{array}\right]$$ 40

Equation (2-29) $$E_t \approx \left[\begin{array}{l} +\frac{1}{4\delta x}\left(E_{i,j,k+1} + E_{i,j+1,k+1} + E_{i+1,j,k+1} + E_{i+1,j+1,k+1}\right) \\ -\frac{1}{4\delta x}\left(E_{i,j,k} + E_{i,j+1,k} + E_{i+11,j,k} + E_{i+1,j+1,k}\right) \end{array}\right]$$ 40

Equation (3-30) $\theta_{EM} = \{(\pi, \mu, \sigma)\}$ for all π, μ, σ with $0<\pi<1$, $0<\mu<1$ and $0<\sigma$. 45

Equation (3-31) $$\tilde{\boldsymbol{\theta}}_{EM} = \{\theta_i = (\pi_i, \mu_i, \sigma_i)\}_{i=1}^{L} \text{ where } \pi_{\theta_i} = \pi_i, \sigma_{\theta_i} = \sigma_i, \sum_{i=1}^{L} \pi_i = 1 \text{ and } \theta_i \in \boldsymbol{\theta}$$ 45

Equation (3-32) $$h_i^n = \frac{\pi_{\theta_i} \cdot G\left(\mathbf{x}^n \mid \boldsymbol{\mu}_{\theta_i}, \boldsymbol{\Sigma}_{\theta_i}\right)}{\sum_{j=1}^{L} \pi_{\theta_j} \cdot G\left(\mathbf{x}^n \mid \boldsymbol{\mu}_{\theta_j}, \boldsymbol{\Sigma}_{\theta_j}\right)}$$ 46

Equation (3-33) $$\pi_i = \frac{\sum_{n=1}^{N} h_i^n}{N} \quad \mu_{\theta_i} = \frac{\sum_{n=1}^{N} x^n h_i^n}{\sum_{n=1}^{N} h_i^n} \quad \Sigma_{\theta_i} = \frac{\sum_{n=1}^{N} h_i^n \cdot \left(x^n - \mu_{\theta_i}\right)^2}{\sum_{n=1}^{N} h_i^n}$$ 46

Equation (3-34) $$x = \left\{ x_i = \left(\frac{i}{N}, h\left(\frac{i}{N} \right) \right) \right\}_{i=0}^{N}$$ 47

Equation (3-35) $$D(\theta_i, \mathbf{x}) = N \cdot \log(\pi_i) + \sum_{n=1}^{N} \mathbf{x}_2^n \cdot \log\left(G\left(\mathbf{x}_1^n \mid \boldsymbol{\mu}_{\theta_i}, \boldsymbol{\Sigma}_{\theta_i}\right)\right)$$ 49

Equation (3-36) $$D_{G_{abs}}(\theta_i, x) = \sum_{n=1}^{N} \text{abs}\left(\pi_{\theta_i} \cdot G\left(x_1^n \mid \mu_{\theta_i}, \Sigma_{\theta_i}\right) - \left(x_2^n - \sum_{j \neq i} \pi_{\theta_j} \cdot G\left(x_1^n \mid \mu_{\theta_j}, \Sigma_{\theta_j}\right) \right) \right)$$ 50

Equation (3-37) $$D_{tot}(\boldsymbol{\theta}_k, \mathbf{r}) = \left\| \mathbf{M}_k(\mathbf{r}) - \mathbf{M}(\mathbf{r}) \right\|$$ 52

Equation (3-38) $$X_k' = \frac{\sum_{x,y} h_k(x,y) \cdot \left(x - \frac{M_v(x,y)}{\omega_k} \right)}{\sum_{x,y} h_k(x,y)}$$ 53

Equation (3-39) $$Y_k' = \frac{\sum_{x,y} h_k(x,y) \cdot \left(y + \frac{M_u(x,y)}{\omega_k} \right)}{\sum_{x,y} g_k(x,y)}$$ 53

Equation (3-40) $$\omega_k' = \frac{\sum_{x,y} h_k \cdot \left(\left(x - X_k\right) \cdot M_v - \left(y - Y_k\right) \cdot M_u \right)}{\sum_{x,y} h_k \cdot \left(\left(x - X_k\right)^2 + \left(y - Y_k\right)^2 \right)}$$ 53

List of algorithms

1 Introduction

This work provides a unified treatment of Recursive Estimation-Maximization, R.E-M, a class of recursive algorithms for segmentation.

Chapter 2 introduces neural network theory as a way of explaining Estimation-Maximization algorithms, which are a subset of R.E-M, and builds towards the R.E-M framework, which opens chapter 3. Chapter 3 also contains a R.E-M formulation of the classical E-M algorithm for Gaussian mixtures, a direct R.E-M formulation of the mixture problem and a R.E-M formulation of motion segmentation for occluded, rigid bodies rotating about fixed centers of motion. Chapter 4 presents segmentation results for simulated Gaussian mixtures, exam grade distributions, simulated flow of rigid wheels and segmentation of a two-image sequence of the wheels of a clock. Chapter 5 provides a thorough discussion of the algorithm including termination, correctness and convergence rates. Chapter 6 concludes this work while chapter suggests further research in segmentation.

1.1. Purpose

We start this introductory chapter with a simple example that motivates this work.

1.1.1 Motivation

Professor D.V. from M.I.T.'s electrical engineering and computer science department has a problem. The end of the semester is approaching and he has to assign grades to the 500 students in his class, but he does not know the best way to do it and he doesn't know if he should give grades below a C letter grade. Professor D.V. is fair and he strongly believes that students with similar performance on the final exam should get similar grades. Being a statistics professor for too many years, he believes that the students belong to categories that resemble Gaussians with different means and variances. By these assumptions, the grade distribution, shown in Figure (1-1) comes from the sum of these Gaussians that model students of various abilities.

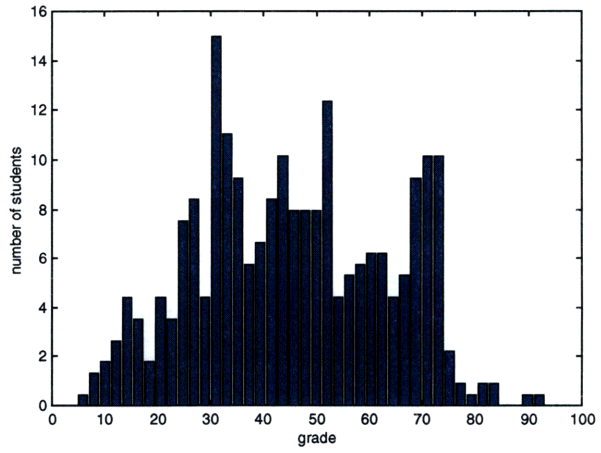
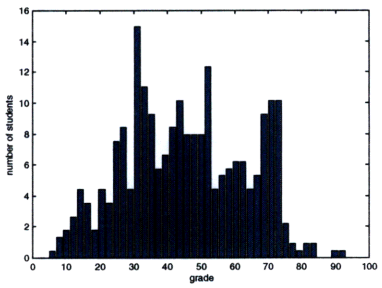


Figure (1-1) Final exam grade distribution for Professor D.V.'s statistics class

Professor D.V. would like to find the underlying Gaussians that gave rise to this distribution using a computer program. In the best case, the program will help him decide how many grades to assign and how to assign them. In the worse case, it will show that his decision to model student performance using Gaussians was not of the best taste.

1.1.2 Goal

The purpose of this work is to develop and describe a segmentation algorithm, R.E-M, which segments linearly separable data and in particular Gaussian mixtures. The algorithm performs very well on simple problems like that of professor D.V. which involve Gaussian distributions as shown in Figure (1-2).

a. 

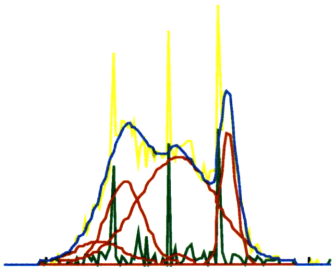
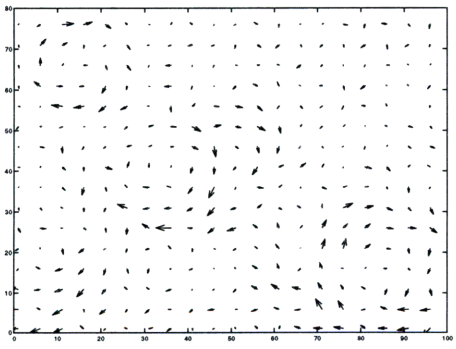
b. 

Figure (1-2) Segmentation of Professor D.V.'s final exam grade distribution

R.E-M can also segment data that are more complicated. Figure (1-3)-a shows simulated motion flow for six occluded rigid wheels spinning about fixed centers of motion. The algorithm finds the underlying number of models and their parameters, and segments the image as shown Figure (1-3)-b.



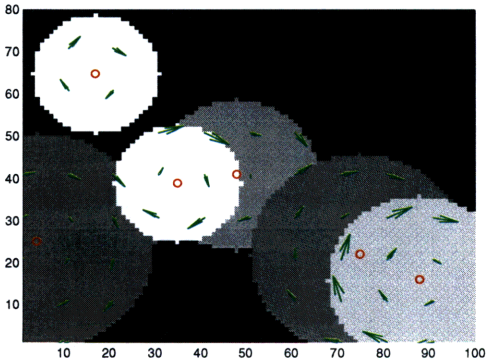


Figure (1-3) Segmentation of simulated motion flow

Given two consecutive images from a captured video sequence showing a mechanical clock's wheels moving as the clock works, R.E-M automatically finds the number of wheels and their centers and rotational velocities as shown in Figure (1-4).

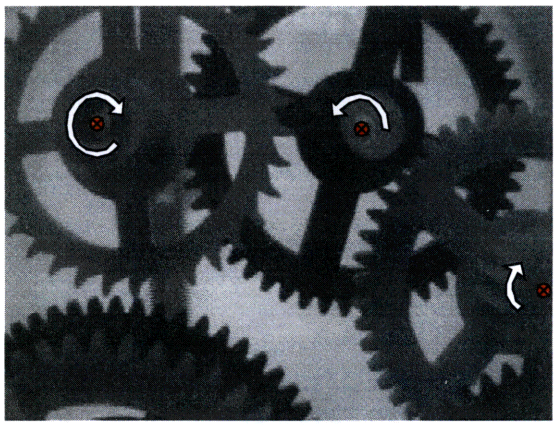


Figure (1-4) Segmentation of image from clock sequence

1.2. Segmentation

Professor D.V.'s grade distribution problem and the clock problem have a common ground, *segmentation*, but the first seems to be hard for humans to solve, while the second might appear hard for machines to solve. We distinguish between data that are easy for humans to segment and data that are easily segmented automatically because we present segmentation results from both categories.

1.2.1 Human-separable data

It is very easy for people to separate some things into categories. Our visual system, in particular, is very apt in distinguishing different objects in our field of view. When we see a new rigid, planar mechanism working, we usually cannot identify the individual components as objects we have seen before. We can however easily tell the number of pieces and describe their shapes and motion. We are performing blind separation because we are putting things into unknown categories. We assign portions of our field of view to unknown objects with specific characteristics. Arguably, the human brain might be receiving information from memory and logic that is impossible to process by a vision algorithm, but there is a notion of which data can be easily separated by humans and which cannot. Computers, on the other hand, might be better on tasks such finding underlying structure to a grade distribution. People fail to separate complicated

distributions, like that of Figure (1-1), mainly because we are trying to solve the problem using our visual system, which is not used to separating things into Gaussians.

1.2.2 Computer-separable data

Computers are good with Gaussians (and so are statisticians) thus it makes sense to develop automated solutions to such problems where humans fail. Given limited information, such as just the motion flow of an image, we might even fail to segment visual data that computers may be able to segment. The simulated flow at each pixel of Figure (1-5) comes from one of three underlying wheels, at random. There exists no obvious representation of this problem for which a person could separate the image. R.E-M reliably separates such data. This is probably because our visual system performs local area averaging, which in this case of non-locally-constrained flow comes out as white noise.

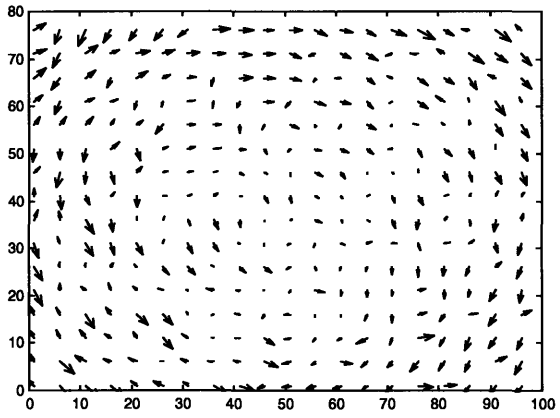


Figure (1-5) A motion flow field hard to segment visually.

Similar arguments may account for the fact that computers cannot separate most images: we have probably not provided them with a proper representation yet. Motion-flow alone a relevant representation for motion segmentation of rigid, moving objects.

Therefor, we test our algorithm on flow data and on the Gaussian data of professor D.V.'s distribution.

1.3. Unsupervised learning

We will adopt a statistical approach to the problem of *blind separation*, in particular *unsupervised learning* for neural networks. Estimation-Maximization algorithms are model-based algorithms, which have been developed for mixtures of experts and hidden Markov models, among other applications. Model-based approaches to blind separation are termed "clustering" because they separate data into clusters. On the other end of the spectrum, there are "subspace methods" which include principle component analysis, Hopfield networks and Boltzman machines. These methods try to find common features in the data. We will adopt a clustering approach to segmentation, strongly tinted with E-M ideas and will not further mention subspace methods in this work.

1.3.1 E-M algorithms

Estimation-Maximization algorithms [Dempster et al., 1977] have been used extensively for unsupervised learning. These algorithms try to maximize to probability of a data set by adjusting model parameters accordingly. They have been favored over other learning algorithms because they are guaranteed to converge. They usually are orders of magnitude faster than stochastic gradient descent algorithms, and tend to perform better in the presence of local minima. EM algorithms have a few shortcomings:

- They depend on initial conditions (non-determinism)
- They depend on initial knowledge of the right number of models (model-dependence)

Various methods of making EM algorithms perform better include deterministic annealing and robust estimation. Applications of the algorithm include image segmentation, sound separation and other clustering problems. We are particularly interested in motion segmentation.

1.3.2 Motion segmentation

Recent work on image segmentation using a mixture of experts architecture has been fruitful. Segmentation techniques have been developed for segmentation based on motion [Weiss and Adelson, 1996], color and shape [Jepson and Black, 1996], etc. Recursive approaches to segmentation [Black and Anandan, 1993, Irani and Peleg, 1992, Bergen et al., 1990] have been proposed for the limited context of motion segmentation. This work builds on similar ideas to provide a unified treatment of Recursive Estimation-Maximization.

1.4. Recursive Estimation Maximization

Recursive estimation maximization is a superset of traditional estimation-maximization, which attempts to correct some of the drawbacks of E-M. In particular, it is deterministic and does not depend on initial knowledge of the number of models. It shares some of E-M's qualities, in that it is guaranteed to terminate and it shows high convergence rates. Unlike E-M, it works with general model descriptions, sets that may have discreet or continuous parameters.

The thesis of this work is that given its advantages, it might make sense to use R.E-M instead of E-M for model-based segmentation problems.

1.5. The segmentation challenge

This section shows some of the challenges every segmentation algorithm has to face. Chapter 6 answers the question of how well R.E-M satisfies these challenges.

1.5.1 Correctness

To be successful, segmentation algorithms need to find segmentations of data that are correct by some measure. This is easier to assert for simulation data than for real data, because we know how the segmentation "should" look. Even in the case of simulations, however, the data may be convoluted enough so that an explanation that differs from the underlying cause of the data is more plausible. In such cases we cannot expect segmentation algorithms to be correct. Thus, when we talk about "correctness" of segmentation, it will always be in the light of the training data and never in the light of the underlying cause of the data. The error measures for our results represent deviation of the prediction from the observed data and not from the underlying models generating the data. We have observed some cases where the prediction of R.E-M explains some data better than the "ground truth" models.

1.5.2 Convergence rates

Although correctness is the primary standard by which to judge a segmentation algorithm, the convergence rate, or number of iterations, before the algorithm terminates is also an important

measure. Algorithms that try all possible combinations of all possible models for some training set exist, however there are usually enough combinations to prevent such algorithms from terminating in time for the results to be useful. E-M is an example of an algorithm that has no upper bound on the number of iterations it requires. It can be run for as long as one desires, and the longer it runs, presumably, the better the results produced. This having been said, the fewer number of iterations a segmentation algorithm requires to arrive to the same segmentation, the better.

1.5.3 Determinism

If, given the same data, an algorithm produces the same result every time it is called deterministic. If the answer changes every time the algorithm is called, the algorithm is non-deterministic. Presumably there is one correct answer for any given segmentation problem and many "almost" correct ones. Algorithms that occasionally find the correct segmentation and often find an "almost" correct segmentation are useful. Algorithms that find the correct segmentation every time are of course preferable, but a consistent "near correct" segmentation might be preferable to many different "near correct" answers. We therefore believe determinism to be a good thing in segmentation.

1.5.4 Generality

Finally, a segmentation algorithm is successful if it can be applied to many segmentation problems. Case-specific algorithms that perform better for any given problem are of course more suitable to the problem, but generality is a virtue by itself, especially since there is no specialized algorithm for every segmentation problem.

2 Theory

This chapter provides a brief overview of theoretical concepts necessary to understand this work, but assumes a basic knowledge of calculus, linear algebra and probability theory. We begin with an introduction to neural network theory and the way neural networks can be used for learning. After a brief overview of supervised learning, we introduce unsupervised learning and the E-M algorithm for mixtures of experts on which R.E-M is built. The chapter concludes with some basic notions of machine vision and optical flow that will be useful for segmenting sequences of images and a list of useful background reading. The reader familiar with these concepts may skip to the following chapter without any loss of continuity.

2.1. Neural networks

For many generations people have tried to understand intelligence. The entire field of Artificial Intelligence deals with precisely that question: "How does the mind work?" Researchers use a variety of approaches to the problem of understanding the functioning of the mind. While artificial intelligence fans take a top-down approach and argue that dealing with the mind should be at a high level of abstraction, just below reasoning, brain and cognitive science fans like bottom-up thinking. They argue that since the brain, sacred sanctuary of intelligence, is composed of neurons, it makes sense to also start with neuron-like computational abstractions and see how to do something intelligent with them. We will adopt the later point of view for our work that is closer to low-level function of the brain than reasoning.

2.1.1 Introduction to neural networks

Neural networks are computational elements whose structure was inspired by real neurons. They are made up of simple units, neurons that have inputs that we represent as x_1, x_2..., one output that we denote as y, and a transfer function F from inputs to output. We typically associate a weight w_i with each input x_i and we include an additional (bias) weight w_0. The output of the unit is a function of the inputs and can be written as $y = w_1 \cdot x_1 + w_2 \cdot x_2 + ... + w_0$. Figure (2-6) bellow shows a simple, single element neural network.

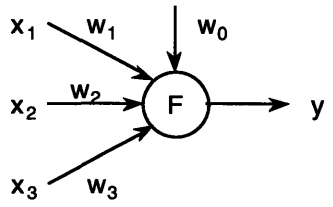


Figure (2-6) Basic unit of a neural network

To describe the behavior of the unit we construct a column matrix **w** with the weights [w_0, w_1, w_2, ...], and a column matrix **x** with the inputs [1, x_1, x_2, x_3, ...] (by convention the first element of the input matrix **x** is always the constant 1). With this convention $\mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \dots \end{bmatrix}$, $\mathbf{x} = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \dots \end{bmatrix}$ and we can

write the unit's function as in Equation (2-1) where $\mathbf{w}^T$ denotes the matrix transpose of the column matrix **w**.

Equation (2-1) $y = F(\mathbf{w}^T \mathbf{x})$

We show how such a network can learn by showing *linear regression*, which provides the best intuitive understanding of learning techniques.

2.1.2 Linear regression

Suppose we had a simple function of two inputs x_1 and x_2: $y = a \cdot x_1 + b \cdot x_2$ but we did not know the constants a and b. We have however a set of N pairs $\left\{\left(\mathbf{x}_i, y_i^*\right)\right\}_{i=1}^N$ satisfying the equation. We call this set a *training set* and y* is called the *target*. We would like to use a neural network to learn the constants a and b using our training set. For this we will construct the linear neural net of Figure (2-7).

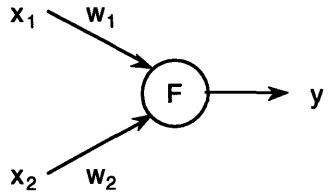


Figure (2-7) A two-input neural network for linear regression

Given the i^{th} pair of the training set, the output of the net is $y = w_1 \cdot x_1 + w_2 \cdot x_2 + ... + w_0$. For regression, we set the transfer function of this simple net to:

Equation (2-2) $F(\mathbf{w}^T \mathbf{x}) = y = w_1 \cdot x_1 + w_2 \cdot x_2 + ... + w_0$

We would like the output y of the net to match the target y_i^* for all i. We thus need to modify w_1 and w_2 b to make y (the output of the unit) match y* (the target of the training pair) for all training pairs. Presumably, w_1 and w_2 will be equal to a and b respectively when we have matched the output of the net to the target for all training pairs.

To do this use the following rule: for every training pair (**x**, y*), sequentially, update the weights by adding some small increment dw_1 and dw_2 according to the following *learning rule*: $dw_1 = \mu (y^* - y)x_1$ and $dw_2 = \mu (y^* - y)x_2$. In vector notation, we can write this *perceptron rule* as:

Equation (2-3) $$\Delta \mathbf{w} = \mu \cdot (y^* - y) \cdot \mathbf{x}$$

μ is called the *learning rate* of the neural net. It is of crucial importance to keep μ small enough to converge to a solution and large enough to converge fast.. We repeat this update sequentially for our entire data set, until we decide[1] to stop.

To understand why the learning rule of Equation (2-3) works, consider the graph of Figure (2-8).

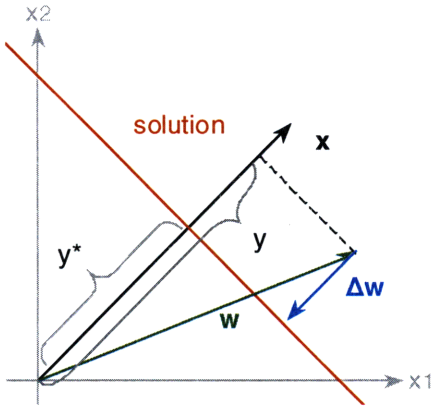


Figure (2-8) Graphical representation of learning in linear regression

Here we have plotted the input of a given training pair as a vector **x** in the input space (x1, x2). We have also represented the weights w_1 and w_2 as the weight vector **w** in the same space. Assuming for simplicity that $\|\mathbf{w}\|=1$, the length of the projection of **w** onto **x** is exactly y. To see this recall that the length of the projection is $proj_x \mathbf{w} = \frac{\mathbf{w}^T \mathbf{x}}{\|\mathbf{w}\|}$

We have also indicated the target y* as a length along the **x** direction. The line perpendicular to **x** that's labeled "solution" indicates all vectors **w'** whose projection on **x** is y*. We would like to change **w** so that it's on the solution line by adding some vector $\Delta \mathbf{w}$. One such vector is

[1] Deciding when to stop turns out to be quite challenging. We address the issue further as a point of discussion.

$\Delta \mathbf{w} = \frac{1}{\|x\|}(y^* - y)\cdot \mathbf{x}$ which is our training rule. If we relax the ||w||=1 constraint, the learning rule becomes $\Delta \mathbf{w} = \frac{1}{\|x\|^2}(y^* - y)\cdot \mathbf{x}$ for a simple two-input neural network.

Similar arguments apply to more complex networks, and the learning rule very often turns out to be very similar to Equation Equation (2-3).

2.2. Supervised learning in neural networks

Linear and other types of regression are useful for fitting functions to data. Suppose however that we are faced with a slightly different problem, one in which we drop coins on a table from two different fixed locations. After we dropped enough coins to get an idea of what the coin distribution looks like, we drop a new coin without watching from which location it came from and would like to predict where it came from. This is a problem of supervised learning. We have a training set (coins) on a table with labels (the location they came from) and we would like to assign a probability on each point in space for a coin at that point coming for one of the two dropping locations. There are two main approaches to classification: the discriminant based and the density based approach.

2.2.1 Supervised learning

Suppose we have k classes with *class labels* $\{\theta_1, \theta_2, \ldots, \theta_k\}$ which might corresponds to dropping locations for coins, and d-dimensional *Gaussian* class-conditional density functions with mean μ_i and variance Σ_i which might correspond to the probability of a coin landing a some point if it came from a given location. Gaussians are important probability classes because independent trials coming from any distribution always form Gaussians after enough trials. Gaussians are functions with the general equation:

Equation (2-4) $$G(\mathbf{x} \mid \boldsymbol{\mu}, \boldsymbol{\Sigma}) = \frac{1}{(2\pi)^{\frac{d}{2}} \cdot |\boldsymbol{\Sigma}|^{d}} e^{-\frac{1}{2}\cdot(\mathbf{x}-\boldsymbol{\mu})^T \boldsymbol{\Sigma}^{-1}(\mathbf{x}-\boldsymbol{\mu})}$$

d is the number of dimensions of the Gaussian. Equation (2-9) shows one-dimensional and two-dimensional Gaussians.

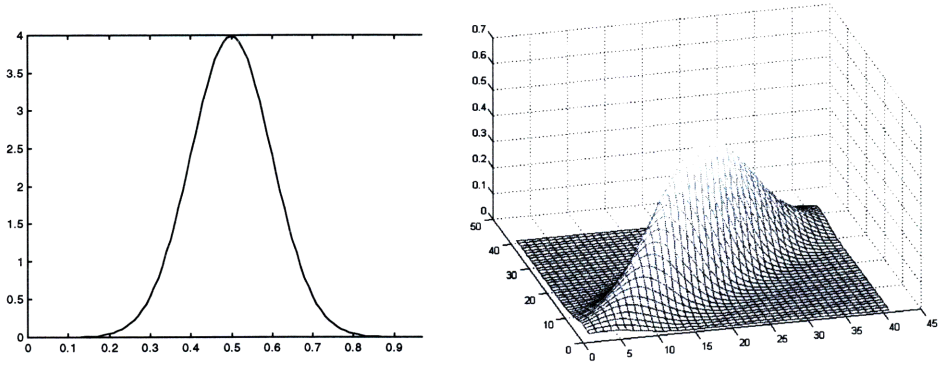


Figure (2-9) One-dimensional and two-dimensional Gaussians

Figure (2-10) shows training points of two classes generated by two independent Gaussian processes with means μ1 and μ2 and variances Σ1 and Σ2. We can think of those as the coins that fall from two different locations, after thy land on a table. Coins labeled "o" come from the first location while coins labeled "x" come from the second location.

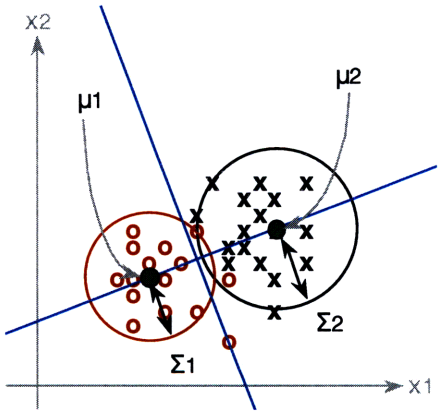


Figure (2-10) Training set for supervised learning

We will present two approaches for classifying the coins so that we may predict where future coins came from: discriminant-based classification and density-based classification.

2.2.2 Discriminant-based classification

In discriminant-based classification, we discriminate between labeled data points using probabilities. In the coin example, we would like to find the probability that a coin came from location 1 versus location 2 for each point on the table. We wish to construct a neural network that given a sample x will give us the probabilities P ($\theta_i|\mathbf{x}$) of that sample coming from class i. Such a network is shown in Figure (2-11).

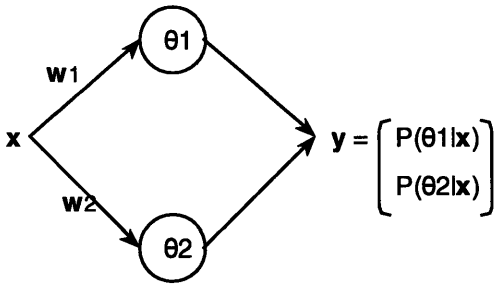


Figure (2-11) Single-layer network for binary classification

The probability $P(\theta_i|\mathbf{x})$ is the *posterior.*, the probability for a data point coming from a given class. However, we don't yet know this probability. Assume that we know $P(\theta_i)$, the *prior* or the a-priori probability of the class with label θ_i. This is how likely is a coin to come from location 1 versus location 2, regardless of where it landed. We also know $P(\mathbf{x}|\theta_i)$, the likelihood given by Equation (2-4) of a coin landing on location **x** given that it came from location 1 or location 2. A fundamental probability rule, *Baye's rule,* allows us to express the posterior in terms of the priors and the likelihood as follows:

Equation (2-5)
$$P(\theta_i \mid \mathbf{x}) = \frac{P(\mathbf{x} \mid \theta_i) \cdot P(\theta_i)}{\sum_{m=1}^{k} P(\mathbf{x} \mid \theta_m) \cdot P(\theta_m)}$$

Plugging Equation (2-4) into Equation (2-5) and simplifying yields the following transfer function for the classification network:

Equation (2-6)
$$y_i = \frac{1}{1+e^{-\mathbf{w}_i^T \mathbf{x}}}$$

The function of Equation (2-6) is called the *logistic sigmoid function*, or the logistic discriminant, or simply *sigmoid.* Given the output of the network and the target y* of a training pair, we would

like to minimize some distance measure between the two, to make them match for all training pairs. It turns out that a good distance metric, or a *cost function* is *cross-entropy*:

Equation (2-7) $$\text{cost} = J(\mathbf{w}) = -\sum_{i=1}^{N}\left[y_i^* \cdot \log(y_i) + \left(1 - y_i^*\right) \cdot \log(1 - y_i)\right]$$

The reason why this is a "good" cost function is that if we set Δw in the cross-entropy equation above proportional to the negative of the gradient and we solve, we get the *logistic discriminant learning rule* which looks very similar to the perceptron rule:

Equation (2-8) $$\Delta \mathbf{w} = \mu\left(y^* - y\right) \cdot \mathbf{x}$$

This means that the only thing we changed from linear regression to discriminant-based classification is the transfer function of the unit. Instead of using the identity function, we are now using the logistic discriminant function. Discriminant-based methods find the posterior probabilities for all models and all points in the parameter space of the problem. In the case of the coins they assign probabilities at each point of the table that a coin that landed there came from location 1 and location 2. Density-based methods on the other hand don't compute posterior probabilities. They try to assign model parameters that maximize the likelihood of the data.

2.2.3 Density-based classification[2]

Density-based techniques provide direct methods for estimating the model parameters of a multi-way classification problem. To understand how this is possible, we first need to introduce the *likelihood principle*. This intuitive statement dates back to the work of Fisher in the early 1920's and can be stated as follows:

> *A model is more plausible or likely than another model in the light only of the given observed data, if it makes those data more probable.*

Thus, the likelihood of a model is the probability of the observed data given that model:

Equation (2-9) $$L(\theta; X) = P(X|W)$$

[2] Adapted from *Maximum Likelihood Estimation and the Exponential Family,* an informal paper by Thomas Hofmann, 9/16/1997.

L is the likelihood of a model θ given observed data X, while P is the probability of the data X given model θ. The likelihood function is utilized instead of the probability to stress the fact that L (θ; X) is considered to be a function of θ for given observations. More often we are interested in the *log-likelihood*:

Equation (2-10) $l(\theta; X) = \log (L (\theta; X))$

The *maximum likelihood principle* states that the best explanation within θ for the observed data is provided by:

Equation (2-11) $\hat{\theta} = \arg\max_{\theta}\{l(\theta; X)\}$

Suppose x is a set of n *independent identically distributed* (i.i.d.) samples distributed according to Equation (2-4). Then the log-likelihood of the entire data set is:

Equation (2-12) $$l(\theta; X) = -\frac{1}{2}\sum_{i=1}^{n}(x_i - \mu)^T \Sigma^{-1}(x_i - \mu) - \frac{1}{2}\log|\Sigma| - \frac{d}{2}\log 2\pi$$

From the maximum likelihood principle, the best estimates for the Gaussian model parameters μ and Σ can be shown to be:

Equation (2-13) $$\begin{cases} \mu = \frac{1}{n}\sum_{i=1}^{n} x_i \\ \Sigma = \frac{1}{n}\sum_{i=1}^{n}(x_i - \mu)(x_i - \mu)^T \end{cases}$$

2.3. Unsupervised learning in neural networks

Unsupervised learning is what this work is about, thus this section is very important in understanding subsequent work. We treat in detail Gaussian mixture models and Estimation-Maximization for Gaussian mixtures.

2.3.1 Introduction to unsupervised learning

In unsupervised learning, we don't have the luxury of having a class label associated with each data point of the training set. Thus we can always turn a supervised learning training set into an unsupervised learning training set by throwing away the class labels, as shown in Figure (2-12).

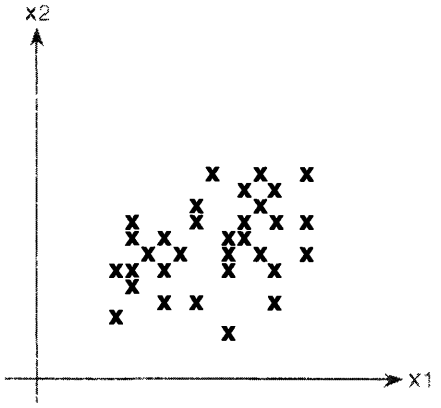


Figure (2-12) Training set for unsupervised learning

This unrealistically simple two-dimensional training set might have been generated from two underlying Gaussians, in which case we would try to fit two Gaussian models to it. In principle, we need to have some underlying knowledge of how the data came about in order to be able to perform unsupervised learning. As in supervised learning, we have N data points and we wish to fit K models to the data points. Finding the correct number of models for a general unsupervised learning training set is an ill-posed problem, but we will provide further discussion in the context of our thesis later on.

2.3.2 Gaussian mixture models

Suppose we wish to fit K Gaussians to a d-dimensional training set of N points. Each Gaussian has a mean μ, a covariance matrix Σ and a prior π. Thus the model parameters are $\theta_i=\{\mu_i, \Sigma_i, \pi_i\}$. As before, the class-conditional probabilities are given by:

Equation (2-14)
$$P_i(x \mid \theta_i)=\frac{1}{(2\pi)^{\frac{d}{2}}|\Sigma_i|^{\frac{1}{2}}} e^{-\frac{1}{2}(x-\mu_i)^T \Sigma_i^{-1}(x-\mu_i)}$$

For the training set of Figure (2-12) and two random initial Gaussian distributions, we would like to obtain a fit as shown in Figure (2-13).

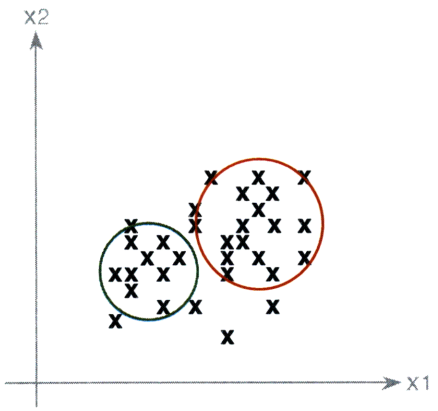


Figure (2-13) Plot of the unsupervised learning training set and models after learning

The probability of all data points **x** in light of the models is given by the sum over all ways of obtaining the data points:

Equation (2-15) $$P(\mathbf{x}) = \sum_i \pi_i \cdot P_i(\mathbf{x} \mid \theta_i)$$

2.3.3 Estimation-Maximization for Gaussian mixtures

The log likelihood of a data point with probability as in Equation (2-15) is:

Equation (2-16) $$l(\mathbf{X}; \Theta) = \sum_x \log \sum_i \pi_i \cdot P_i(\mathbf{x} \mid \theta_i)$$

To find model parameters that maximize this likelihood we would need to perform *gradient descent* on the parameter space. This is equivalent to finding the partial gradient of the likelihood of Equation (2-16) with respect to the model parameters. Finding the partial gradient of the log of a sum is not obvious. We use a technique that will simplify the likelihood function. The technique consists of introducing *indicator variables* $\mathbf{z}_i$ for each data point **x** as follows:

Equation (2-17) $$z_i = \begin{cases} 1 & \text{if x comes from model i} \\ 0 & \text{otherwise} \end{cases}$$

The probability of a data point x^n simplifies to:

Equation (2-18) $$P(\mathbf{x}^n, x) = \prod_{i=1}^{K} \left[\pi_i \cdot P_i\left(\mathbf{x}^n \mid \theta_i\right)\right]^{z_i}$$

The likelihood of the entire data set becomes:

Equation (2-19) $$l(\mathbf{X}) = \sum_n \sum_i z_i^n \log\left[\pi_i \cdot P_i\left(\mathbf{x}^n \mid \theta_i\right)\right]$$

We estimate the unknown indicator variables $\mathbf{z}_i$ by $\mathbf{h}_i$, an estimate of their expected value $h_i^n = E\left[z_i^n \mid x^n\right]$. This is readily solved with Baye's rule if we realize that it is just the probability of $\mathbf{z}_i$ given the data point. To summarize:

Equation (2-20) $$Q(\mathbf{X}) = \sum_n \sum_i h_i^n \log\left[\pi_i \cdot P_i\left(\mathbf{x}^n \mid \theta_i\right)\right]$$

Equation (2-21) $$h_i^n = \frac{\pi_i P\left(\mathbf{x}^n \mid \theta_i\right)}{\sum_{j=1}^{K} \pi_j P\left(\mathbf{x}^n \mid \theta_j\right)}$$

The EM algorithm iterates between estimating model parameters by maximizing the likelihood of Equation (2-20) and solving for the estimation of Equation (2-21). The two steps of the algorithm are summarized below:

Equation (2-22) $$h_i^n = \frac{\pi_i P\left(\mathbf{x}^n \mid \theta_i\right)}{\sum_{j=1}^{K} \pi_j P\left(\mathbf{x}^n \mid \theta_j\right)} \qquad \text{E-step}$$

Equation (2-23) $$\hat{\Theta} = \arg\max_{\Theta} \sum_n \sum_i h_i^n \log\left[\pi_i \cdot P_i\left(\mathbf{x}^n \mid \theta_i\right)\right] \quad \text{M-step}$$

These two steps are repeated iteratively for as many iterations as needed. The problem of initial conditions is usually addressed by offering a random guess. E-M relies on knowing the correct number of models for a unsupervised learning problem before we can begin the estimation and maximization steps. This work addresses these shortcomings of E-M.

2.4. Optical flow[3]

This brief section will be useful in understanding how to compute optical flow from real images in the clock segmentation sequence.

2.4.1 Optical flow

Optical flow is the apparent motion of the brightness pattern in an image, which moves as the objects that give rise to them move. The optical flow is related to the motion field, which is the motion of the objects that give rise to an image; we will rely on this correspondence to estimate relative motion.

If we let E(x,y,t) be the irradiance (or intensity) at time t at the image point (x,y), then if u(x,y) and v(x,y) are the x and y components of the optical flow vector at that point, we expect that the irradiance will be the same at time t=dt at the point (x+dx, y=dy) where dx=udt and dy=vdy. That is,

Equation (2-24) $E(x+udt, y+vdt, t+dt) = E(x,y,t)$

We can rewrite this equation as:

Equation (2-25) $(Ex,Ey)*(u,v) = -Et.$

The component of optical flow in the direction of the brightness gradient $(Ex,Ey)^T$ is thus:

Equation (2-26) $$\frac{E_t}{\sqrt{E_x^2 + E_y^2}}$$

where Et is the partial derivative of E(x,y,t) with respect to t, and Ex and Ey are the partial derivatives of E(x,y,t) with respect to x and y respectively. Ex, Ey and Et are given by:

Equation (2-27) $$E_x \approx \left[\begin{array}{l} +\frac{1}{4\delta x}\left(E_{i+1,j,k} + E_{i+1,j,k+1} + E_{i+1,j+1,k} + E_{i+1,j+1,k+1}\right) \\ -\frac{1}{4\delta x}\left(E_{i,j,k} + E_{i,j,k+1} + E_{i,j+1,k} + E_{i,j+1,k+1}\right) \end{array} \right]$$

[3] Adapted from Robot Vision, chapter 12. Berthold Horn, MIT Press, 1994

Equation (2-28)
$$E_y \approx \left[\begin{array}{l} +\frac{1}{4\delta x}\left(E_{i,j+1,k} + E_{i,j+1,k+1} + E_{i+1,j+1,k} + E_{i+1,j+1,k+1}\right) \\ -\frac{1}{4\delta x}\left(E_{i,j,k} + E_{i,j,k+1} + E_{i+1,j,k} + E_{i+1,j,k+1}\right) \end{array} \right]$$

Equation (2-29)
$$E_t \approx \left[\begin{array}{l} +\frac{1}{4\delta x}\left(E_{i,j,k+1} + E_{i,j+1,k+1} + E_{i+1,j,k+1} + E_{i+1,j+1,k+1}\right) \\ -\frac{1}{4\delta x}\left(E_{i,j,k} + E_{i,j+1,k} + E_{i+11,j,k} + E_{i+1,j+1,k}\right) \end{array} \right]$$

2.4.2 Aperture problem

The optical flow in the direction of the brightness gradient is given by Equation (2-26). However nothing is known for the optical flow at right angles to this direction, that is, along the isobrightness contour. This ambiguity is also known as the aperture problem.

2.5. Useful readings

- Alvin W. Drake. Fundamentals of applied probability theory. McGraw-Hill, 1988. This text provides an excellent introduction to probability theory and statistics.
- Thomas H. Cormen, Charles E. Leiserson and Ronald L. Rivest. Introduction ot Aglorithms. The MIT Press, 1997. The algorithmic notation used in this work parallels the standard set by this other excellent text.
- Berthold K. P. Horn. Robot Vision. The MIT Press, 1994. This book lays the foundations of machine vision and provides a nice mathematical treatment of low-order vision.
- Christopher M. Bishop. Neural Networks for Pattern Recognition. Oxford University Press, 1995. This text provides a treatment of neural networks.
- John Hertz, Anders Krogh and Richard G. Palmer. Introduction to the theory of neural computation. Addison-Wesley, 1991. This book expands on ideas on statistical mechanics and their applications to neural networks.

3 Recursive estimation maximization

This is the main chapter of this thesis. We present the R.E-M algorithm and derive equations for Gaussian mixtures, motion-flow-based classification and recognition of partially occluded objects in cluttered scenes. In Chapter 4 we present detailed results that we obtained using the algorithms presented here. Chapter 5 provides a thorough discussion of R.E-M including termination, correctness, and convergence rate.

3.1. The algorithm

As we saw in section 2.3.3, estimation-maximization algorithms work in two tightly coupled steps:

1. **Segmentation**: data points are assigned to models by measuring deviation from prediction
2. **Modeling**: Model parameters are estimated using the assignment of the data points.

From this description, there are two observations to be made: 1. There is no notion of where to start. 2. There is no notion of when to stop. We attempt to impose constraints by proposing starting and ending conditions for R.E-M and ways of automatically finding the number of models. For notation conventions, the reader is advised to read chapter 2 on Theory. For a discussion of these conditions refer to chapter 3.

3.1.1 Setup

Suppose we have a training set $\mathbf{x} = \{x_i\}_{i=1}^{N}$ of data we wish to segment and a model set $\boldsymbol{\theta} = \{\theta_i\}_{i=1}^{K}$ with fixed models that we wish to fit to the data. We seek the working set $\tilde{\boldsymbol{\theta}} = \{\theta_i \in \boldsymbol{\theta}\}_{i=1}^{L}$ of L models which minimize some error function $E(\boldsymbol{\theta},\mathbf{x})$ for all θ in $\tilde{\boldsymbol{\theta}}$. We also provide a cost function $D(\theta,x)$ which measures deviations of model θ from $\mathbf{x}$.

3.1.2 Overview

We now describe R.E-M in its more general form. We provide a plain-text explanation of the algorithm as well as detailed description of the major steps subsequently.

Algorithm (3-1) R.E-M($\mathbf{x}$, $\boldsymbol{\theta}$, $\tilde{\boldsymbol{\theta}}$, E, D)

1 **do** error = E($\tilde{\boldsymbol{\theta}}$,$\mathbf{x}$)

2 $\tilde{\boldsymbol{\theta}}_{save} = \tilde{\boldsymbol{\theta}} = \left\{ \tilde{\boldsymbol{\theta}}, \underset{\theta}{\operatorname{argmin}} E\left(\left\{\tilde{\boldsymbol{\theta}}, \theta\right\}, \mathbf{x}\right) \right\}$ % R.R-step

3 **do** set = E($\tilde{\boldsymbol{\theta}}$,x)

4 **foreach** θ_i in $\tilde{\boldsymbol{\theta}}$

5 $\mathbf{h}_i = \dfrac{\pi_{\theta_i} \cdot e^{-\frac{D(\mathbf{x},\theta_i)}{\sigma_{\theta_i}^2}}}{\sum_{j=1}^{K} \pi_{\theta_j} \cdot e^{-\frac{D(\mathbf{x},\theta_j)}{\sigma_{\theta_j}^2}}}$ % R.E-step

6 **end**

7 $\tilde{\boldsymbol{\theta}} = \arg\max_{\tilde{\boldsymbol{\theta}}} \sum_i \mathbf{h}_i D(\mathbf{x} | \theta_i)$ % R.M-step

8 **end**

9 **until** E($\tilde{\boldsymbol{\theta}}$,$\mathbf{x}$) - set < ε

10 **until** E($\tilde{\boldsymbol{\theta}}$,$\mathbf{x}$) - error < 0

11 **return** $\tilde{\boldsymbol{\theta}}_{save}$

3.2. Description

The algorithm works in three tightly coupled steps. We provide here a description of the three steps

3.2.1 R.R-Step

In the R.R-step, we add a new model to our working set to minimize some global measure of deviation. This is the real novelty of R.E-M. It allows new models to be added at locations given by the error function. For some given data, there are much better ways to add a model rather than randomly as E-M does. The model should be added where it minimizes the most error of some case-specific function. This function should compute the total deviation of the working set from the observations of the training set, which may or may not be the sum of the deviations from observation of the individual models. This dependence of the models on the data as a whole accounts for the robustness of R.E-M.

The R.R-step has to perform a minimization of function D with respect to the current working set. A Newton-Rhapson algorithm could be used for this step, but case-specific R.R-steps that are more efficient may be preferable for different D functions.

3.2.2 R.E-step

In this step we estimate the credits of each model in our working set for each data point of our training set. This step can be thought of as putting springs between the models and the data points. The springs that are shorter pull with greater force than the longer springs, thus the models that explain some data better are given more credit for that data.

3.2.3 R.M-step

In this step we change the parameters of the models to obtain a maximum fit for the data, but we weight our fit by the credits computed beforehand. The R.M-step can be thought of as doing the actual modeling or segmentation. In this step we again perform a maximization. Efficient case-specific maximization can be developed for different D functions.

3.2.4 Termination

Refer to the discussion section for a treatment of termination. We note here that the errors measured by lines 9 and 10 can be averaged over a few iterations ,which allows the algorithm to momentarily increase the error to perhaps discover a better segmentation. Examples of this are shown in the results section.

3.2.5 Summary of R.E-M

Given:

- Training set: $\mathbf{x} = \{x_i\}_{i=1}^{N}$
- Model set: $\boldsymbol{\theta} = \{\theta_i\}_{i=1}^{K}$
- Working set: $\tilde{\boldsymbol{\theta}} = \{\theta_i \in \boldsymbol{\theta}\}_{i=1}^{L}$
- Cost function: D (θ,**x**)
- Error function E ($\boldsymbol{\theta}$,**x**)

R.E-M($\mathbf{x}$, θ, $\tilde{\theta}$, E, D) returns a working set $\tilde{\theta} = \{\theta_i \in \theta\}_{i=1}^{M}$ (with M>=N), satisfying

$\tilde{\theta} = \arg\max_{\tilde{\theta}} \sum_i \mathbf{h}_i D(\mathbf{x} | \theta_i)$ and for which E (θ,$\mathbf{x}$) is at a minimum.

3.3. R.E-M formulation of E-M for Gaussian mixtures

We first implement E-M to show R.E-M's expressive power. R.E-M is a superset of E-M.

3.3.1 Parameters

Our training set is simply the set of data that we wish to segment $x = \{x^n\}_{n=1}^{N}$. We will use the standard E-M models for Gaussian mixtures for our model set. This is the infinite set of all Gaussians.

Equation (3-30) $\theta_{EM} = \{(\pi, \mu, \sigma)\}$ for all π, μ, σ with $0<\pi<1$, $0<\mu<1$ and $0<\sigma$.

The initial guess for our working set is

Equation (3-31) $\tilde{\theta}_{EM} = \{\theta_i = (\pi_i, \mu_i, \sigma_i)\}_{i=1}^{L}$ where $\pi_{\theta_i} = \pi_i, \sigma_{\theta_i} = \sigma_i, \sum_{i=1}^{L} \pi_i = 1$ and $\theta_i \in \theta$

For all i, θ_i is randomly selected from θ subject to the above constraints. We compute the cost function with Algorithm (3-2) which simply computes the log-likelihood of Equation (2-19). G($\mathbf{x}|\mu,\sigma$) is the Gaussian distribution function of Equation (2-4).

Algorithm (3-2) D_{EM} (θ_i,$\mathbf{x}$)

1 return $\sum_{n=1}^{N} \log\left[\pi_{\theta_i} G(\mathbf{x}^n | \mu, \Sigma_{\theta_i})\right]$

We may therefor use the M step of E-M in this problem, which solves the optimization problem of the M-step directly. The equations for the updates of the parameters of the models for the M-step of E--M are reproduced below.

Equation (3-32)
$$h_i^n = \frac{\pi_{\theta_i} \cdot G\left(\mathbf{x}^n \mid \boldsymbol{\mu}_{\theta_i}, \boldsymbol{\Sigma}_{\theta_i}\right)}{\sum_{j=1}^{L} \pi_{\theta_j} \cdot G\left(\mathbf{x}^n \mid \boldsymbol{\mu}_{\theta_j}, \boldsymbol{\Sigma}_{\theta_j}\right)}$$

Equation (3-33)
$$\pi_i = \frac{\sum_{n=1}^{N} h_i^n}{N} \qquad \mu_{\theta_i} = \frac{\sum_{n=1}^{N} x^n h_i^n}{\sum_{n=1}^{N} h_i^n} \qquad \Sigma_{\theta_i} = \frac{\sum_{n=1}^{N} h_i^n \cdot \left(x^n - \mu_{\theta_i}\right)^2}{\sum_{n=1}^{N} h_i^n}$$

Since E-M is inherently non-recursive with respect to the number of models, we set the error function to a constant which guarantees that the condition on line 10 of Algorithm (3-1) will always evaluate to TRUE and thus R.E-M will never loop.

Algorithm (3-3) E_{EM} ($\boldsymbol{\theta}$,**x**)

```
1  return constant
```

3.3.2 Formulation

The complete R.E-M formulation of E-M for L models is:

Algorithm (3-4) EM(**x**, L)

```
1  return R.E-M(x, θ_EM, θ̃_EM, E_EM, D_EM)
```

3.4. R.E-M for Gaussian mixtures

The R.E-M formulation of E-M simply shows that R.E-M can work exactly like E-M. We provide here a native R.E-M formulation for Gaussian mixtures, which makes no prior assumptions on the number of models necessary, is deterministic and converges faster than traditional E-M as the results of chapter 4 seem to indicate. We show here the form of the training sets, the model set, the

working set and the error and cost functions we used in our experiments. A Matlab implementation of R.E-M for Gaussian mixtures is given as an appendix.

3.4.1 Training set

From a data set of P numbers distributed from 0 to 1, we construct N bins of width 1/N in which we distribute the numbers. This creates a histogram, which is our training set **x.**

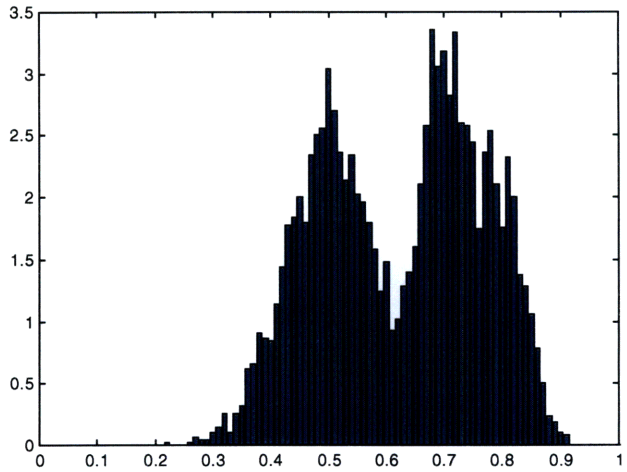


Figure (3-14) Histogram obtained from from a Gaussian mixture distribution

Each training point is a pair with an index from 0 to 1 in 1/N increments and the value h(i/N) of the histogram at that index.

Equation (3-34) $$x = \left\{ x_i = \left(\frac{i}{N}, h\left(\frac{i}{N} \right) \right) \right\}_{i=0}^{N}$$

The width 1/N of the bins is the smoothing parameter of section 5.1.6 and helps decrease the variance of our estimator. R.E-M finds the estimator with the minimum bias for this variance.

3.4.2 Model set

We use a finite model set with fixed-parameter models and not continuous-parameter models for reasons discussed in chapter 5. We need a large enough model set θ_G to describe the Gaussian mixture of the data set to within the desired accuracy. The set we used in our simulations was made up of 250 models for each mean, for 10 means distributed between 0 and 1. There were a

total of 2,500 models our model set which is enough to describe most common situations but still much smaller than an infinite set. The 250 models with zero mean are shown in Figure (3-15). The one-dimensional Gaussians are drawn on parallel planes in three dimensions for easier visualization.

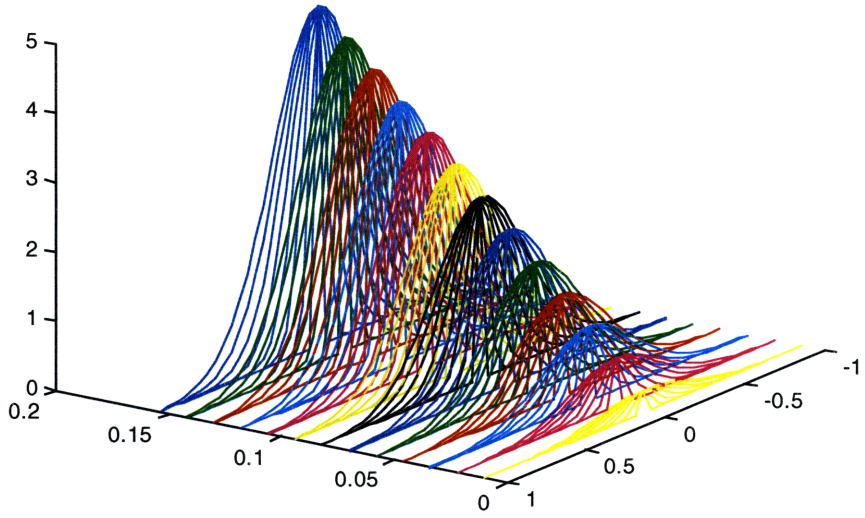


Figure (3-15) A subset of 250 elements from the R.E-M model set for Gaussian mixtures

Alternatively we may scale our model set. We might add more models or substract a few depending on prior knowledge of the underlying mixture. For our simulations for example, we removed the models with variance above 0.2 from our model set because we did not want explanations with such high-variance. We can also call R.E-M with a coarse model set to get an initial estimate and repeat the call with a finely grained set around the solution returned by R.E-M. We use Algorithm (3-5) to create the model set $\boldsymbol{\theta}_{\mathbf{G}}$ of Figure (3-15).

Algorithm (3-5) θ_G

```
1  for mu = 0:.1:1
2      for sg = .01:.04:.3,
3          for pr = .01:.012:.165,
4              pi=80*pr*sg;
5              θG = { θG, (pi, mu ,sg)}
6          end
7      end
8  end
```

3.4.3 Working set

The working set for the algorithm is the empty set, since we want R.E-M to automatically find the correct number of models without initial guesses.

3.4.4 Cost function

There are a variety of ways to approach the question of what is a good cost function for R.E-M for Gaussian mixtures. Many alternatives are possible.

A first approach would be to use the log-likelihood of the data as a cost function. The cost function D would be the same as D_{EM} except for our change of representation of the training set. We account for the change by observing that all data points of any given bin of our histogram have the same probability given some model, but there are more of them, namely as many as the value of the histogram at that point. If we represent by $\mathbf{x}_1^n$ the first element of $\mathbf{x}^n$, which is some number from 0 to 1, and by $\mathbf{x}_2^n$ the second element of $\mathbf{x}^n$, which is the value of the histogram at that number as you may recall, we get:

Equation (3-35) $$D(\theta_i,\mathbf{x}) = N \cdot \log(\pi_i) + \sum_{n=1}^{N} \mathbf{x}_2^n \cdot \log\left(G\left(\mathbf{x}_1^n \mid \boldsymbol{\mu}_{\theta_i}, \boldsymbol{\Sigma}_{\theta_i}\right)\right)$$

Although powerful, the cost function above resembles E-M too closely and does have some problems that E-M has, as we found out in our simulations. It may for example prevent R.E-M from converging to the right solution if it gets trapped in a local minimum. We present a more robust cost function that in practice works better than the log-likelihood.

We adopt a R.M.S. (root mean-squared) error approach. We will try to minimize the difference between the training set and the working set using a cost function of the form:

Equation (3-36) $$D_{G_{abs}}(\theta_i, x) = \sum_{n=1}^{N} abs\left(\pi_{\theta_i} \cdot G\left(x_1^n \mid \mu_{\theta_i}, \Sigma_{\theta_i}\right) - \left(x_2^n - \sum_{j \neq i} \pi_{\theta_j} \cdot G\left(x_1^n \mid \mu_{\theta_j}, \Sigma_{\theta_j}\right)\right)\right)$$

The first term inside the absolute value function is the contribution of model θi and the second is the data minus the contributions of all the other models of the mixture. To make sure that this cost function gives more weight to points near the mean of each model we will add a term to the cost function so that the error away from the mean matters less than the error near the mean. This is in effect a robust error function. Our total cost function becomes:

Algorithm (3-6) $D_G(\theta_i, \mathbf{x})$

1 return $$\sum_{n=1}^{N}\left[D_{G_{abs}} \cdot \left(\frac{G\left(x_1^n \mid \mu_{\theta_i}, \Sigma_{\theta_i}\right) \cdot \left(\sqrt{2\pi \cdot \Sigma_{\theta_i}^{\ 2}}\right) - 1}{1.5} + 1\right)\right]$$

The robust function, which multiplies the standard error from before, is the Gaussian model itself, but scaled appropriately to be 1 at its mean and fall off to about 0.3 at the extremities instead of 0. (This choice is arbitrary, but works well in our examples, however the question of the exact form of these robust functions is subject of further research.) One such Gaussian is shown in Figure (3-16).

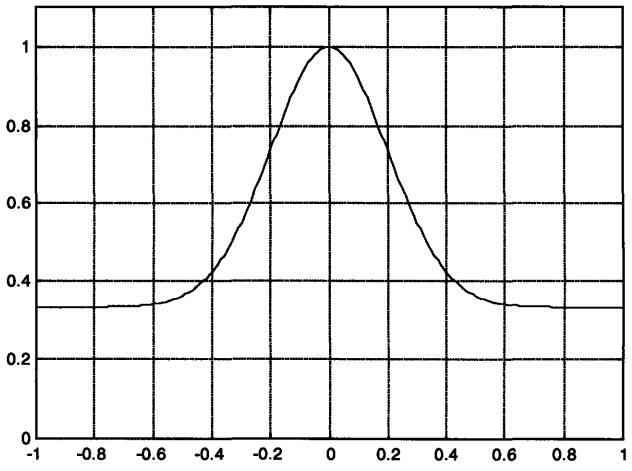


Figure (3-16) The robust term of R.E-M's cost function for Gaussian mixtures

Intuitively, multiplying the error by the scaled model itself is a form of cross-validation. This function has the effect of smoothing out errors far from the model's mean, which should not matter for that model.

3.4.5 Error function

The deviation of a mixture of Gaussians from observation is given by the sum of Equation (3-36) for all the models.

Algorithm (3-7) E_G (θ,**x**)

1 **return** $\sum_{i=1}^{L}\left[D_{G_{abs}}(\theta_i, x)\right]$

In fact, we are finding the dominant peak of the distribution that remains after subtracting off the effects of all the models in our training set. We add the new model where this residual error is best explained.

3.4.6 R.E-M for Gaussian mixtures

The complete R.E-M formulation of the problem of segmentation of Gaussian mixtures is given below:

Algorithm (3-8) R.E-M_G(**x**)

1 **return** R.E-M ($\mathbf{x}_G$, θ_G, {}, E_G, D_G)

3.5. R.E-M for optical flow segmentation

This section presents an R.E-M formulation of the problem of segmenting images of rigid, planar objects moving with pure rotation. (This is general enough for translation too, because translation can be though of as rotation about an infinitely long axis). There is no limit on the number of moving objects nor is there any difficulty posed by occlusion. The main constraint is the quality of the motion flow obtained. We present a cost function for simulated flow which makes use of robust estimation ideas and brings down the optimization of the R.M-step to a two dimensional search problem that can be performed fairly efficiently. The aperture problem of real images brings the number of dimensions of the optimization problem up to 3 again, but one may use other methods to compute the flow of two images before segmentation.

3.5.1 Training set

For simulated flow, a data point is the optic flow vector $\mathbf{M}(\mathbf{r})=\begin{bmatrix} u(\mathbf{r}) \\ v(\mathbf{r}) \end{bmatrix}$ at point $\mathbf{r}=\begin{bmatrix} x \\ y \end{bmatrix}$ from section 2.4.1. For real data, the vector **M** points along the direction of the brightness gradient and the flow along that direction is computed using Equation (2-26). Ex, Ey and Et are computed using Equation (2-27), Equation (2-28)and Equation (2-29).

3.5.2 Model set

Our models have $\theta_i = \{X_i, Y_i, \omega_i\}$ where (X_i,Y_i) is a center of motion and ω_i is the angular velocity. Figure (3-17) shows a sample model of the model set with parameters (50,40,0.2);

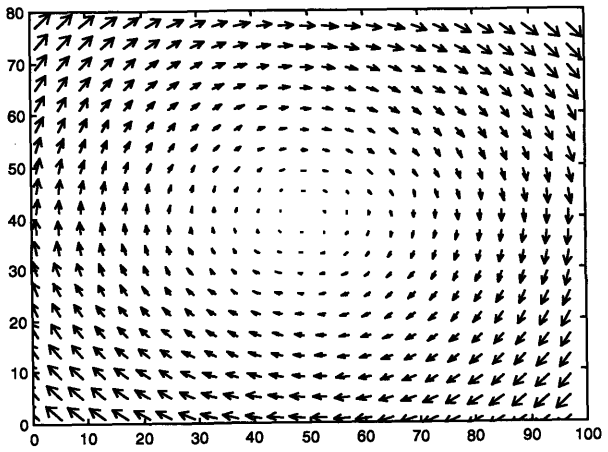


Figure (3-17) Sample model for rigid wheel segmentation

3.5.3 Cost function

The first cost function we consider is the most obvious and powerful one: The norm of the vector difference between observation and prediction:

Equation (3-37) $$D_{tot}(\theta_k,\mathbf{r})=\|\mathbf{M}_k(\mathbf{r})-\mathbf{M}(\mathbf{r})\|$$

where $\mathbf{M}_k(\mathbf{r})$ is the motion flow at point **r** predicted by model k and $\mathbf{M}(\mathbf{r})$ is the optic flow at point **r**. There are unfortunately no closed form solutions to the R.M-step with this choice. We note however that the square of this function gives closed form solutions to the update equations

for the centers of motion and angular velocities. We find these equations by taking the partial derivatives of the square of Equation (3-37) with respect the the model parameters, subject to the constraint that the priors add up to 1. We use a LaGrange multiplier to enforce the constraint.

Equation (3-38) $$X_k' = \frac{\sum_{x,y} h_k(x,y) \cdot \left(x - \frac{M_v(x,y)}{\omega_k} \right)}{\sum_{x,y} h_k(x,y)}$$

Equation (3-39) $$Y_k' = \frac{\sum_{x,y} h_k(x,y) \cdot \left(y + \frac{M_u(x,y)}{\omega_k} \right)}{\sum_{x,y} g_k(x,y)}$$

Equation (3-40) $$\omega_k' = \frac{\sum_{x,y} h_k \cdot \left((x - X_k) \cdot M_v - (y - Y_k) \cdot M_u \right)}{\sum_{x,y} h_k \cdot \left((x - X_k)^2 + (y - Y_k)^2 \right)}$$

Unfortunately these equations do not give adequate results in practice: we need to find the minima of the cost function with respect to the parameters analytically (using for example Newton's method).

Decoupling the motion center estimation from the angular velocity estimation is in this case a good idea, because it reduces the dimensionality of the search space by one and makes the algorithm faster. We propose therefor a cost function that does not depend on the angular velocities but simply on the angular deviation of prediction from observation.

Equation (3-41) $$D_{ang}(\theta_k, \mathbf{r}) = \begin{cases} 1 & \text{if} \quad \|\mathbf{M}_k\| \otimes \|\mathbf{M}\| \\ \sin\left(\frac{|\angle \mathbf{M}_k(\mathbf{r}) - \angle \mathbf{M}(\mathbf{r})|}{2} \right) & \text{o.w.} \end{cases}$$

$\angle$ is the angle of a vector and $\otimes$ is the logical XOR operator. A plot of the absolute value of the sine is shown in Figure (3-18). We use the sine of the absolute value of the angular difference of the predicted flow and the observed flow instead of just the absolute value because of robustness considerations as before. The sine function has the effect of giving more importance to small

errors than large ones. To see this, consider Figure (3-18) which shows a plot of the sine function from $-\pi/2$ to $\pi/2$.

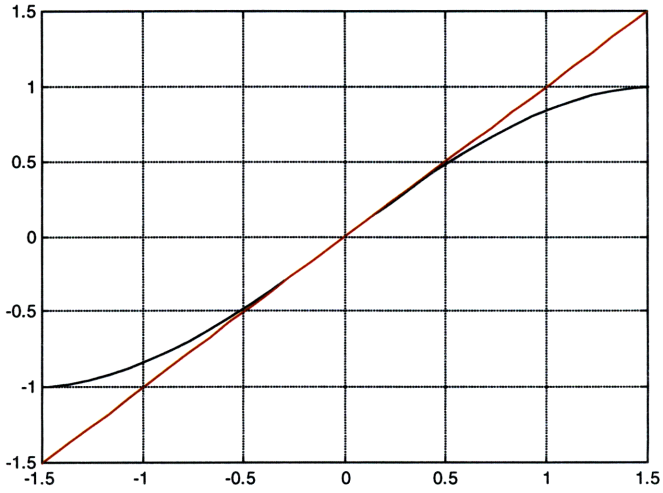


Figure (3-18) The sin(x) function as a robust estimator

As the angular error gets large, towards $\pi/2$, the sin function decreases the effective error, while where the angular error is small, near the origin, the sin function looks similar to the identity function which is the straight line in the figure.

Because of the XOR operator, Equation (3-41) also sets the error to 1 (the maximum error allowed) whenever we compare an angle to that of a zero vector. This convention works well in practice. Once the centers of motion are estimated, the angular velocities are easy to estimate, using for example R.E-M with the cost function of Equation (3-37). [Jepson and Black, 1993] obtained closed form solutions to a similar R.M-step using a cost function similar to the one in Equation (3-41) for the estimation of optical flow using mixture models. [Maclean, Jepson and Frecker, 1992] obtain a similar solution for segmentation of independent object motion and use a Newton-Rhapson algorithm for the minimization of the R.E-M step.

We can use the cost function of Equation (3-41) directly and get adequate results, however we have improved the results by using a variant of renormalization, a relaxation technique used in many physics problems. Renormalizing means reducing a known problem to a simpler equivalent one. In our case, we are using information about the errors at every point in the image to scale the error at any given point. We effectively decouple the error at any given pixel from the total error:

Equation (3-42) $$D'(\vec{r}) = \frac{D(\vec{r})}{\sum_{\vec{r}} D(\vec{r})}$$

This cost function makes the error surfaces of the M step of the algorithm smoother and allows the algorithm to resolve flow fields that cannot be resolved by the previous cost equations alone. Renormalization makes the algorithm much more insensitive to the choice of the variance σ of the models that we can set to the same value for all the models. It also allows us to use the same prior for all models, which allows us to perform and unconstrained minimization at the R.E-M step.

The intuition behind the renormalization idea is that there is no reason why a big total error for a given model should matter more than a smaller total error of another model at the pixel level. Renormalization gives all models equal responsibility at the pixel level.

3.5.4 Deterministic annealing

The motion field on the right of Figure (3-19) below looks very much like the motion field generated by a single spinning wheel. There are, however, three wheels of similar diameters with nearby centers, spinning at the same speed as shown on the left of the figure.

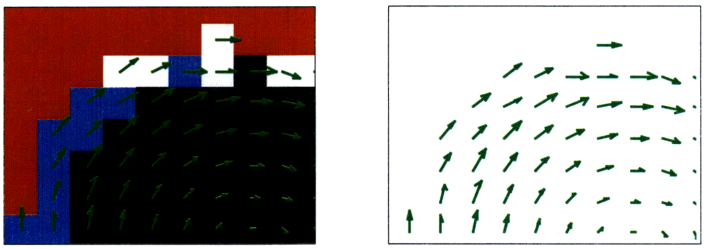


Figure (3-19) A motion field hard to segment

Although with real data small errors would make it even harder to solve this field into three different models, we want an algorithm robust enough in simulation that it correctly resolves such situations. The idea is to start with a large variance σ (that is the same for all our models as we discussed above), and make it smaller as the algorithm progresses. This effectively assigns most pixels to almost all the models to begin with, and as the models adapt to the data, we differentiate more and more between them. This technique is known as deterministic annealing.

The following three figures show the estimated segmentation through three iterations of the inner loop of R.E-M as it converges to the correct solution and stops.

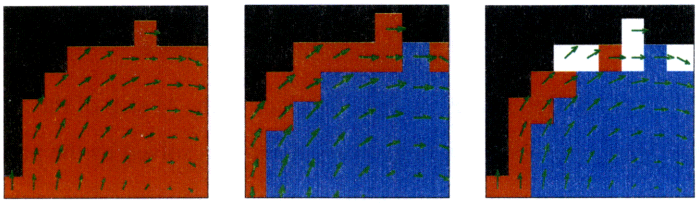


Figure (3-20) R.E-M resolves the field on the left in three wheels using deterministic annealing

3.5.5 Error function

The error of R.E-M for motion segmentation (which is a "hard" version of the log-likelihood) is given by:

Equation (3-43) $$E_F(\boldsymbol{\theta},\mathbf{x}) = \sum_{n=1}^{N} \min_{\boldsymbol{\theta}_i}\left(D_F(\boldsymbol{\theta}_i,\mathbf{x})\right)$$

The algorithm is guaranteed to converge because E is bound below by 0. To see this, recall that:

Equation (3-44) $$\begin{cases} \sum_{n} D_F\left(\boldsymbol{\theta},\mathbf{x}^n\right) = 1 \\ 0 \le D_F(\boldsymbol{\theta},\mathbf{x}) \le 1 \end{cases} \Rightarrow\ 0 \le E_F \le 1$$

3.5.6 R.E-M for motion segmentation

Finally we give the complete R.E-M formulation of the motion segmentation problem:

Algorithm (3-9) R.E-$M_F(\mathbf{x})$

1 **return** R.E-M ($\mathbf{x}_F$, $\boldsymbol{\theta}_F$, {}, E_F, D_F)

4 Results

This chapter presents results obtained by running the R.E-M algorithms developed previously on a variety of data. Data came from one-dimensional Gaussian mixture distributions, real exam grade distributions, simulated optical flow for rotating rigid bodies and pairs of consecutive images of videos of working clock mechanisms. For each case, we present R.E-M's progress, the relevant segmentations obtained and when possible we comparisons with the traditional E-M algorithm.

4.1. Gaussian mixtures

We start with distributions coming from known mixtures because it's easier to test whether the results match the underlying models. In all our examples, we generate a data set of a few thousand points that form an histogram. We separate the histogram data in 100 bins and run the R.E-M Algorithm (3-8) for Gaussian mixtures, using the model set of Algorithm (3-5). In some cases, we

show comparisons with results from E-M. For these cases, we use an implementation of the R.E-M formulation of E-M for Gaussian mixtures given by Algorithm (3-4).

4.1.1 Underlying models with little overlap

The training set for the first segmentation we present comes from a mixture of 4 Gaussians that doesn't show great overlap. The ground truth distribution is shown in Figure (4-21)-d. From this distribution we generated 5,000 data points whose histogram is shown in Figure (4-21)-a. We show the results at the end of each iteration of the algorithm in Figure (4-21)-b. through Figure (4-21)-c and Figure (4-21)-g through Figure (4-21)-j. The rest of the figures show the RMS error between prediction and observation at each iteration of the algorithm. While the different figures correspond to different iterations of the outer loop of R.E-M, the x axis of each figure showing the error corresponds to the number of minimizations of each iteration, which include the R.R-step and the R.M-step of the algorithm.

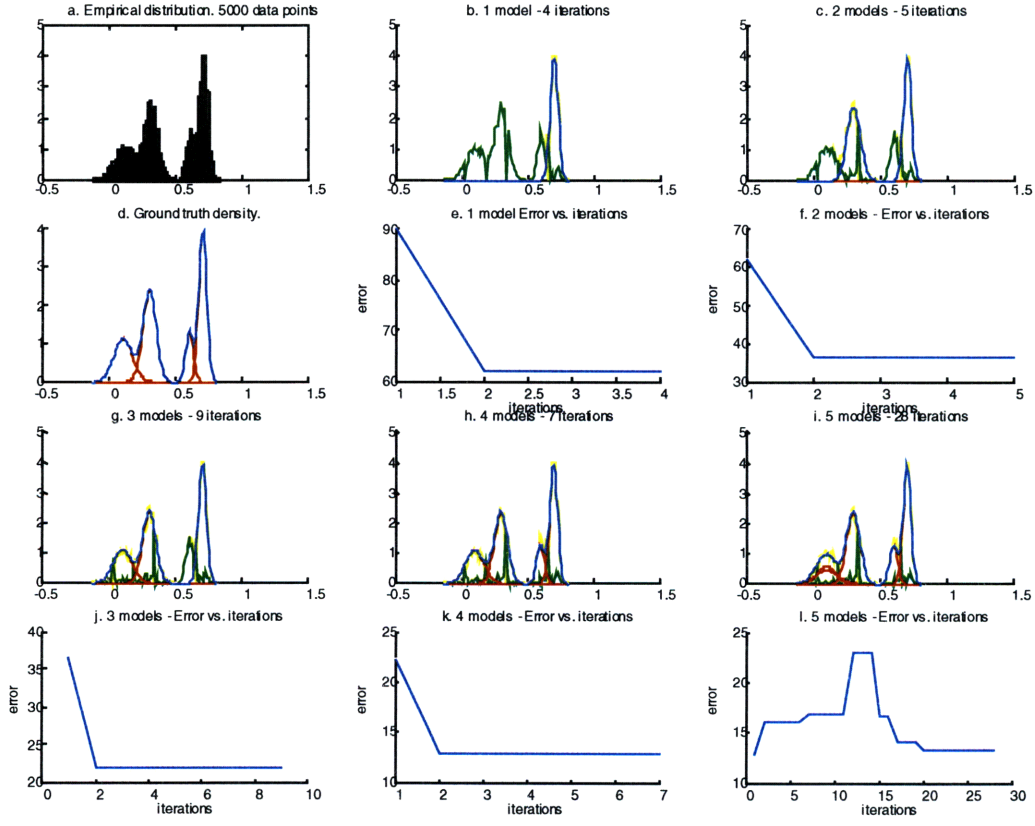


Figure (4-21) Segmentation obtained by running R.E-M for Gaussian mixtures

The algorithm starts by finding the model that minimizes the RMS error between prediction and data. This happens in this case to be the dominant peak of the distribution. This reduces the error from 90 to just over 60. The algorithm runs the R.E-step and R.M-step, which counts as one more iteration, bringing the total to 3 (we counted the zeroth model as the first iteration). Since the working set does not change in the R.E and R.M-steps, the algorithm adds a new model at the second dominant peak as shown in Figure (4-21)-c. This reduces the error from just over 60 to about 35. Again, noticing no change during the R.E and R.M-steps, the algorithm adds a third model. We notice more iterations without any change of the error, but this is because we are counting the minimization for each individual model as one iterations. Since there are 3 models in the working set, the R.M-step takes three iterations, at the end of which a minor change in the working set induces another 3 iterations as shown in Figure (4-21)-j. Adding a fifth model, in Figure (4-21)-i, causes the total error to increase and decrease again, as the algorithm tries to find the best way to fit the data to all the models. After the working set does not change, the algorithm is ready to add a sixth model. It notices however than the total RMS error decrease of Figure (4-21)-i does not justify adding a fifth model, much less a sixth. Therefor it returns with the working set of Figure (4-21)-h, which has 4 models.

The algorithm is deterministic given a training set, but is not guaranteed to follow the same path to the solution. Figure (4-22) shows the algorithm running on data generated from the same underlying distribution as before. In this case however, there happens to be a better minimization of the RMS error by placing an initial model over the second more dominant peak, which happened to receive more data points. This illustrates that we are still dealing with probabilities and nothing is guaranteed. The algorithm however converges to the same solution, which is shown in graph h. of both figures.

In the second case, we allowed models of height less than 0.5 in our model set. As a result, adding a fifth model does not show the behavior of Figure (4-21)-i, but instead a monotonic decrease of the error. However, the decrease is still only by less than 1 (from 17 to 16.2), and the algorithm still correctly stops with 4 models, since this decrease is too small (close to zero for our implementation).

Figure (4-21)-and Figure (4-22) illustrate that a finer grained working set makes the algorithm run smoother but does not increase performance significantly. It may on the other hand slow the

algorithm since the larger the model set, the more time consuming the minimizations of the R.R and R.M-steps.

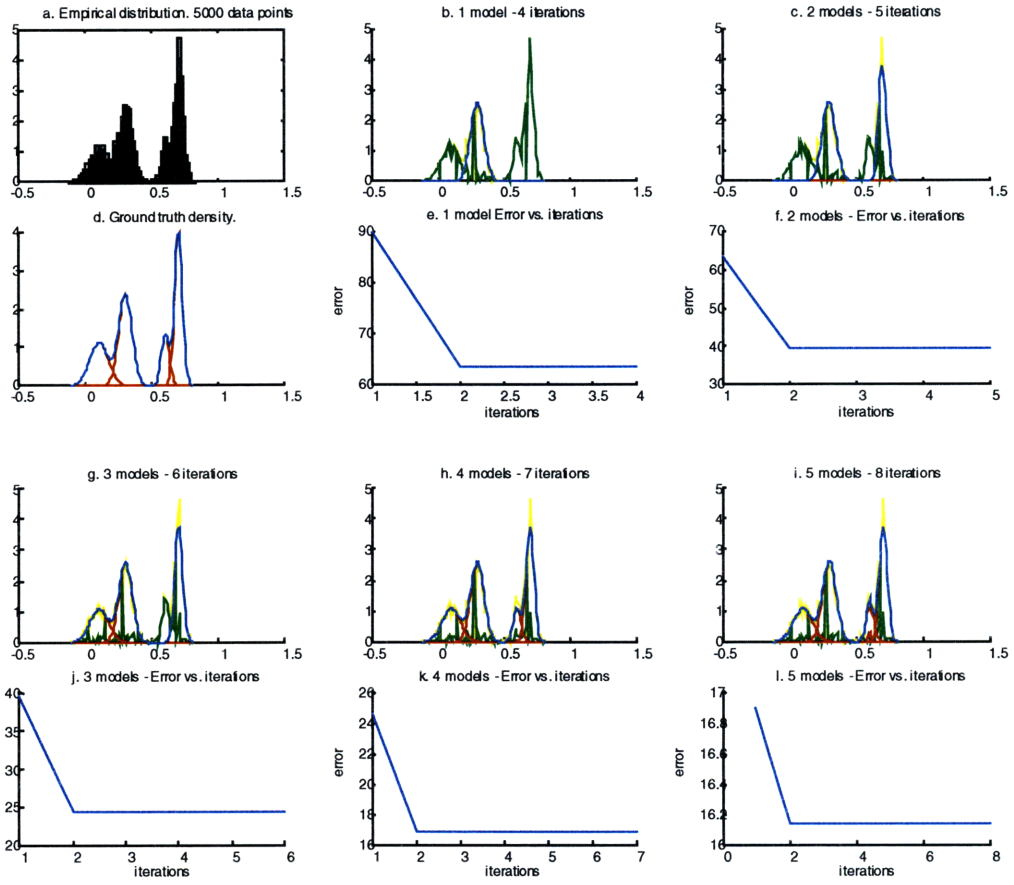


Figure (4-22) A different path to the same solution for different training sets

Figure (4-23) shows, for comparison, the segmentation that E-M obtains after 1,000 iterations. Graph a. shows the empirical distribution, b. shows the ground truth density, c. the E-M answer and d. the error versus the number of iterations in a log-log scale. E-M never converges to the correct solution because it is trapped in a local minimum. Intuitively, the gap between the two peaks in the distribution will never allow any of the models of E-M to cross from the peak on the left to the peak to the right. The left peak incorrectly received 3 models while the right peak only received one. This is a typical situation for E-M.

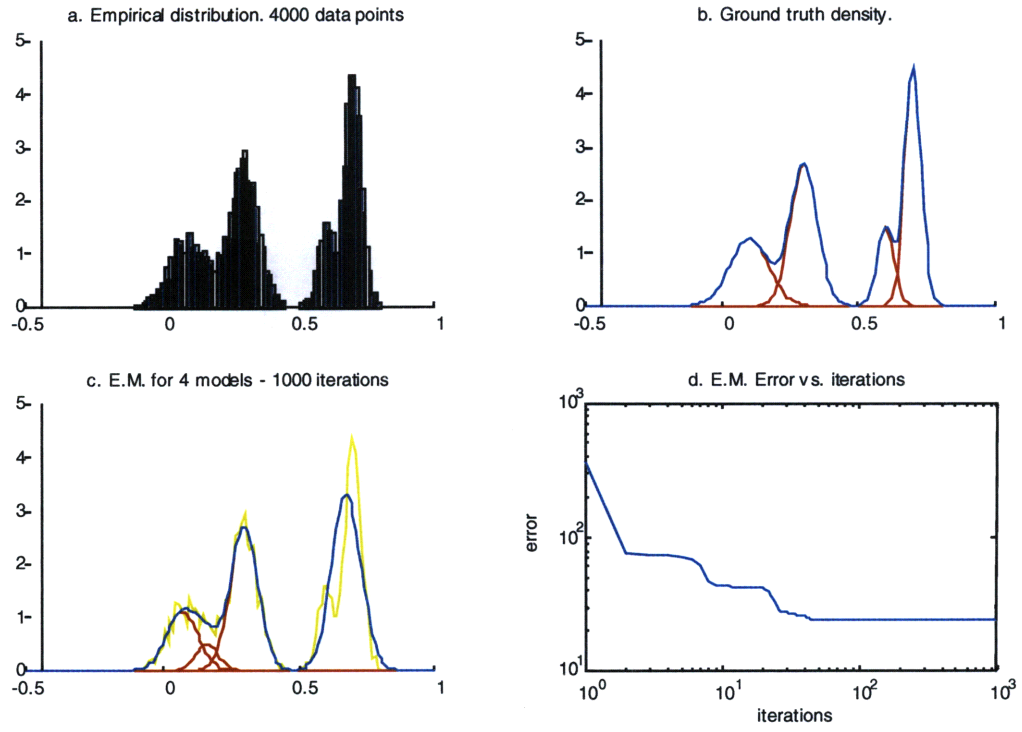


Figure (4-23) Segmentation obtained by running E-M

4.1.2 Underlying models with medium overlap

As our second example, we show a distribution where the underlying Gaussians overlap significantly. Figure (4-24)-a shows the empirical distribution obtained for 5,000 data points from the underlying distribution of Figure (4-24)-b.

Again, we separated the data in 100 bins and run R.E-M for Gaussian mixtures but we also run E-M from Algorithm (3-4). Figure (4-24)-c shows the segmentation E-M found after 1,000 iterations given the knowledge that there were 3 underlying models. Figure (4-24)-d shows the segmentation R.E-M obtained after terminating automatically after 32 iterations (optimizations), without knowledge of the number of underlying models. Figure (4-24)-e shows the RMS error of E-M versus the number of iterations and Figure (4-24)-f the same error for R.E-M. Figure (4-24)-f also has labels at the places where the number of models in R.E-M's working set changed.

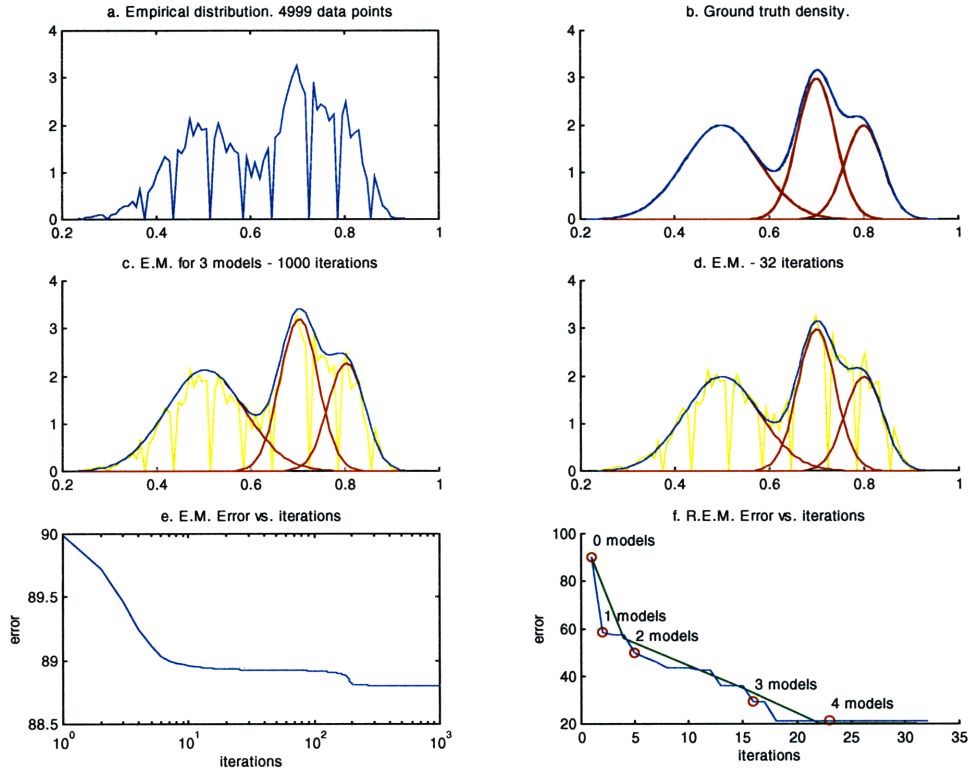


Figure (4-24) Segmentations obtained by running E-M and R.E-M on the same data

The segmentation of Figure (4-24)-c is the correct one, but E-M rarely obtains the correct segmentation for this example as well as many other such examples. About nine times out of ten, E-M converges to a local minimum, one of which is shown in Figure (4-25). Whether or not E-M converges to the right solutions entirely depends solely on initial conditions, which are set randomly.

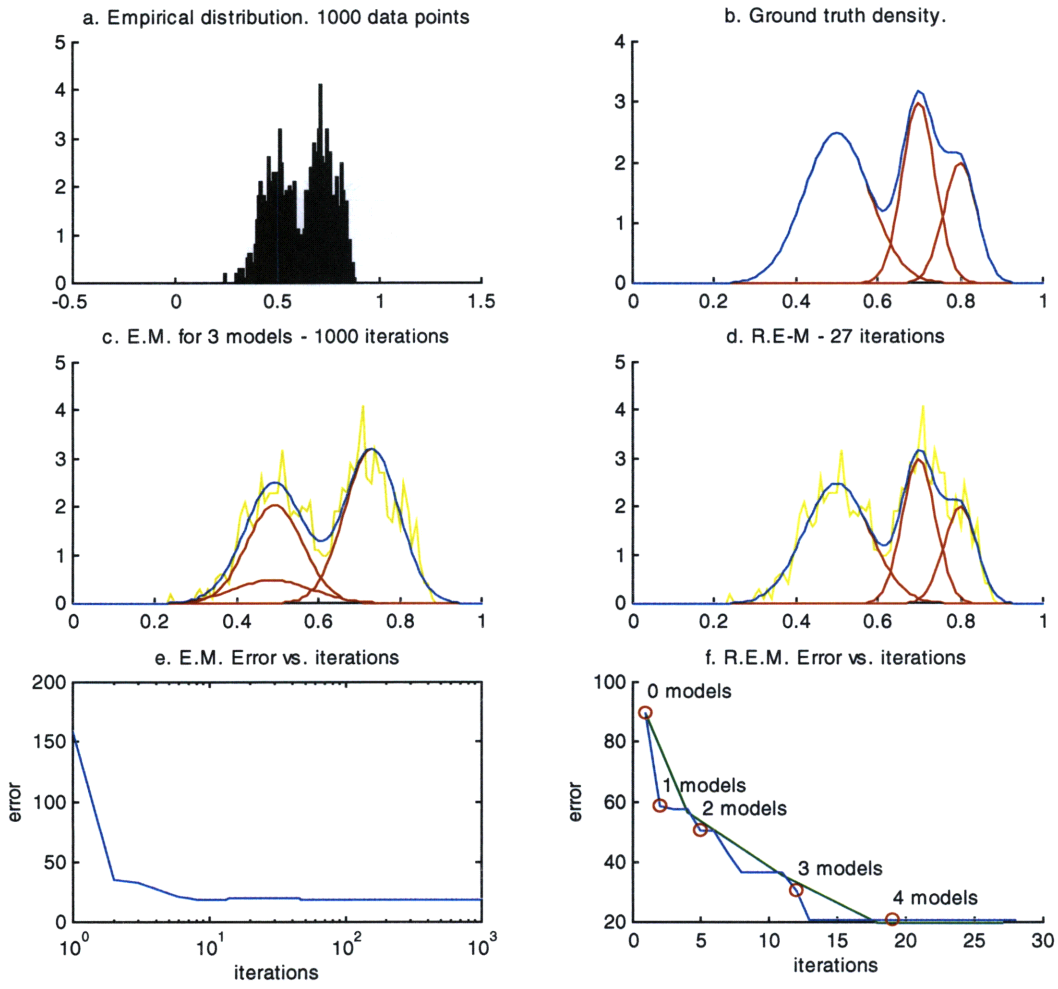


Figure (4-25) A common case for which E-M is trapped to a local minimum

4.1.3 Underlying models with significant overlap --- exam-grade distribution

We now present an example of a distribution complicated enough to represent exam-grades. The distribution is shown in Figure (4-26)-a. It comes from four underlying Gaussians shown in Figure (4-26)-b.

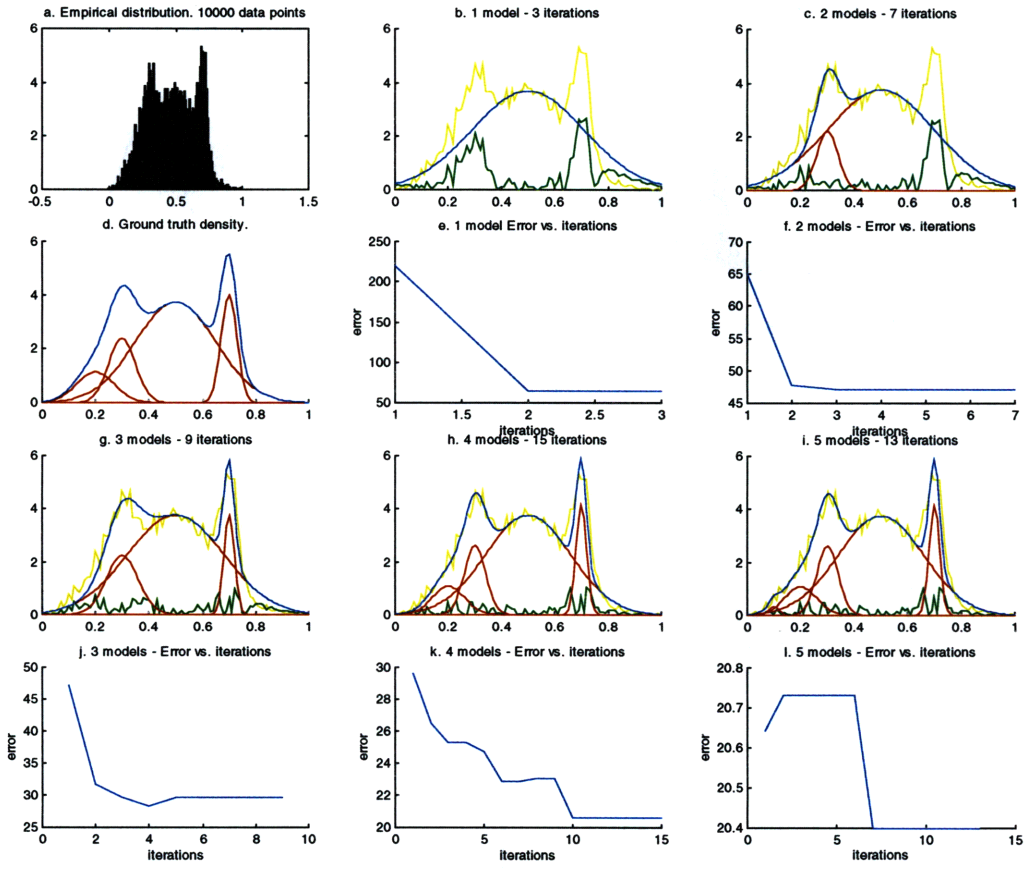


Figure (4-26) R.E-M segmentation of a hard-to-segment distribution that could represent exam grades

The algorithm progresses as shown in Figure (4-26). Figure (4-26)-k is interesting, because it shows how R.E-M can adjust model parameters to obtain a better fit for a given distribution. Figure (4-26)-l shows that adding a 5^{th} model decreases the total error by only by about 0.2. The algorithm stops with the resulting segmentation of Figure (4-26)-h and a total error of a little above 20.

Figure (4-27) shows the E-M algorithm for the same data. There are fewer data points due to software limitations in the implementation of E-M, however this does not affect the results significantly. The results obtained after running E-M for 1,000 iterations are shown in Figure (4-27)-c. The algorithm started with 4 models with random parameters and eventually got trapped

into a local minimum. We tried the same example about 5 times with new random initial conditions and never were able to obtain an answer close to the underlying mixture.

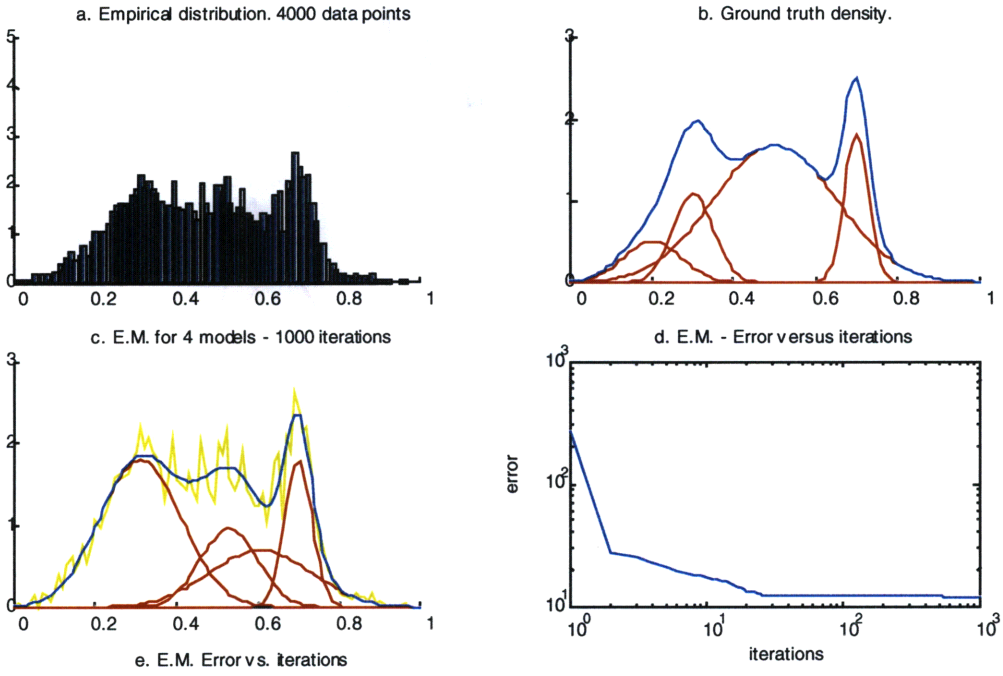


Figure (4-27) E-M segmentation of a hard-to-segment distribution that could represent exam gradess

4.1.4 A singular case

We finally show an example of a singular case in which the peaks of the distribution is flat. The underlying mixture is shown in Figure (4-28)-b and the empirical distribution in Figure (4-28)-a. R.E-M returns with the result of Figure (4-28)-e. The intermediary steps of the algorithm are also shown in Figure (4-28)-c and Figure (4-28)-d.

Figure (4-29) shows E-M on the same distribution. After 1,000 iterations E-M gives the result of Figure (4-29)-c. This matches the underlying models better than R.E-M's result, but we should recall than E-M has prior knowledge of the underlying number of models.

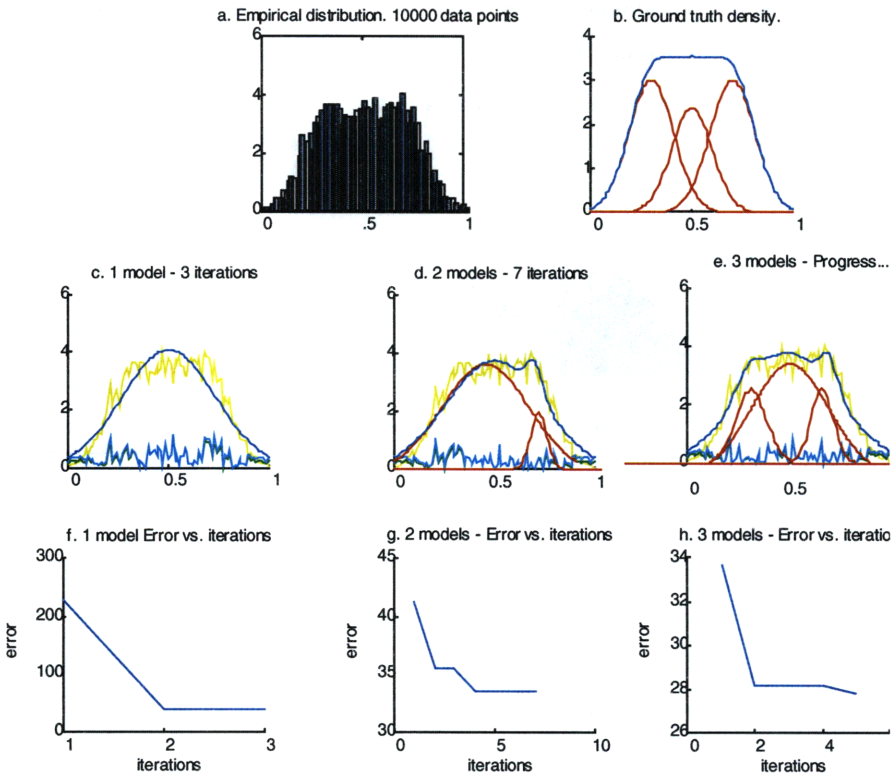


Figure (4-28) R.E-M segmentation of a singular distribution

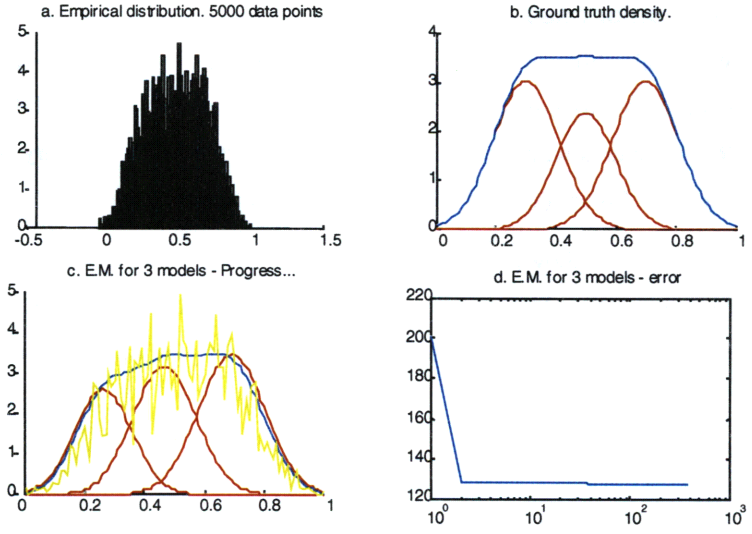


Figure (4-29) E-M segmentation of a singular distribution

4.2. Motion flow

This section presents results of Algorithm (3-9) on simulated motion flow data. We start with low resolution simulated images, show a higher resolution example and finish with segmentation of a video sequence of two images showing a working clock. We do not present comparisons between E-M and R.E-M here. Detailed figures of the progress of the algorithms are in Appendix A

4.2.1 Low resolution simulations

We show the segmentations obtained by running R.E-M on both non-occluded and occluded simulated optic flow data. Figure (4-30) shows simulated flow for three non-occluded spinning wheels. We show for convenience the wheels themselves in the figure, but only the flow is available to R.E-M.

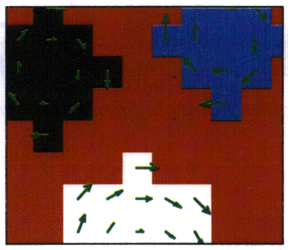


Figure (4-30) Simulated motion flow for non-occluded rotating wheels

Figure (4-31) shows the segmentations obtained at the end of each iteration of R.E-M. The algorithm correctly identifies three centers, finds the correct angular velocities for the wheels and segments the image and stops.

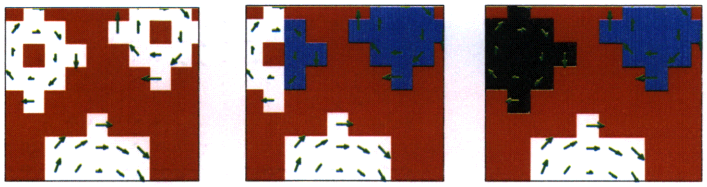


Figure (4-31) Segmentations obtained at the end of each iteration of R.E-M for motion segmentation

Figure (4-32) shows the motion flow and the original segmentation of a typical situation of three occluded wheels spinning at the same angular velocity

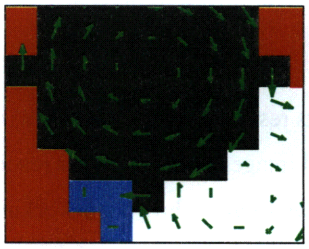


Figure (4-32) Simulated motion flow for occluded rotating wheels

Figure (4-33) shows the segmentations obtained at the end of each iteration of R.E-M. The algorithm correctly identifies three centers, finds the correct angular velocities for the wheels, and segments the image and stops.

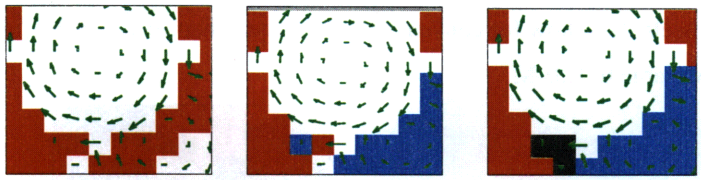


Figure (4-33) Segmentation obtained at the end of each iteration of R.E-M for motion segmentation

4.2.2 High resolution simulations

We present here a complicated example of a simulated motion flow with six underlying spinning wheels. The flow is shown in Figure (4-34).

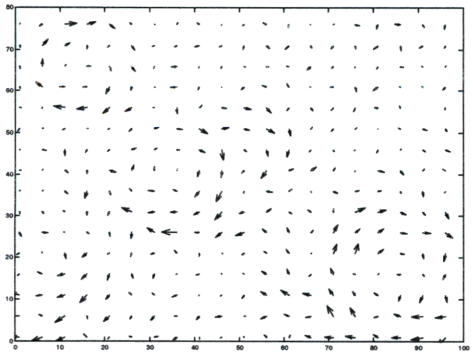


Figure (4-34) Simulated high-resolution flow for 6 underlying rotating wheels

The algorithm finds the number of the underlying wheels, their centers and their velocities and segments the image as shown in Figure (4-35).

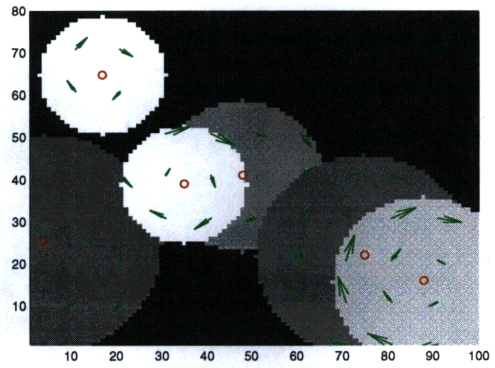


Figure (4-35) Segmentation of simulated motion flow with six underlying wheels

4.3. Clock images

We present the segmentation obtained from a pair of video-images showing a working clock. The first of the two images in shown in Figure (4-36). We obtained the motion flow from the pair of images using the results of section 2.4.1. The vertical bar in the image is there because we wanted to occlude the top wheel as well.

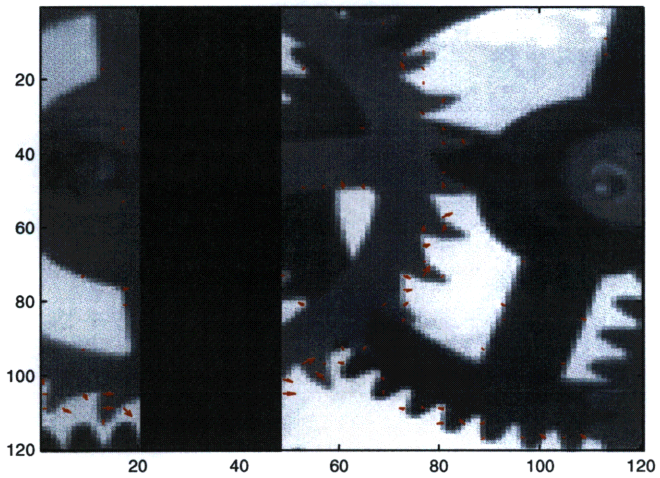


Figure (4-36) First of the two images used for motion segmentation of a clock sequence.

The algorithm correctly finds two wheels in the image and places their centers as shown in Figure (4-37).

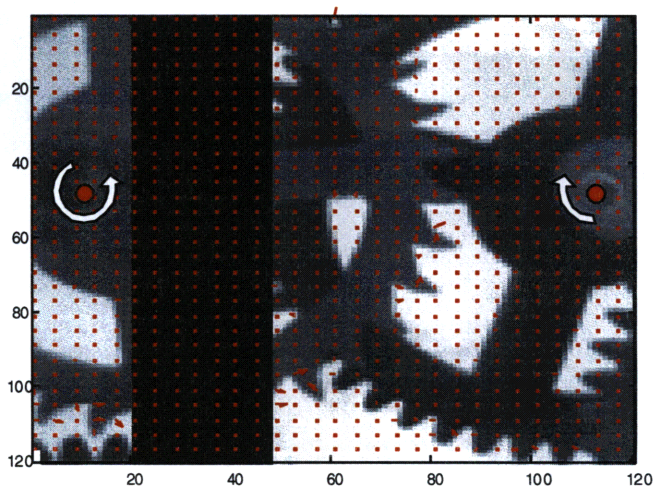


Figure (4-37) Segmentation of clock sequence

5 Discussion

This chapter provides a thorough discussion of the R.E-M algorithm. We cover termination, correctness, convergence rate, determinism, generality, and the bias-variance tradeoff. We also include a brief discussion of the advantages of a finite number of models and the general setup of the algorithm and list some areas that might present difficulties to the algorithm. We conclude this chapter with a comparison of the traditional E-M algorithm and the R.E-M algorithm.

5.1. Properties of the R.E-M algorithm

This section provides a discussion of most important properties of R.E-M in its general form. Case-specific discussion is included when appropriate.

5.1.1 Termination

We begin by proving that R.E-M terminates. This proof necessitates that the error function E of the algorithm be bounded bellow by some finite number:

Equation (5-45) $E(\tilde{\theta}, x) > K$ where K is a finite constant

Then the loop in steps 3-9 terminates for ε positive. If the function "E-set" of line9 is positive, we exit. If it's negative, then we loop, but the value of "set" for the next loop will be diminished. Since "set" is computed using E and E is bounded bellow, loop 3-9 terminates. The same argument holds for loop 1-10. Since all loops of the algorithm terminate, the algorithm terminates. E measures the deviation of observation from prediction. E will usually be bounded bellow by 0, thus guaranteeing termination.

5.1.2 Correctness

We cannot prove formally that the algorithm converges to the "correct" segmentation because finding such segmentations is an ill-posed problem. We argue however that the segmentation R.E-M finds is correct keeping in mind that sounder segmentation might exist of the same data. We will refer the reader to our results section where we present a large variety of segmentation problems that are solved by R.E-M. We have tried most possible mixtures of two, three and four Gaussian models and with the exception of pathological cases such as the one of section 4.1.4 R.E-M finds mixtures very similar to the underlying models. In the case of motion flow, R.E-M segmented correctly about 50 different flows of up to eight random occluded wheels.

5.1.3 Convergence rate

Intuitively, we start with a given maximum error when no models are present in the working set. We decrease this error at each step by at least ε. Since in most cases E is bounded bellow by zero, we stop after at most M steps from Equation (5-46).

Equation (5-46) $$M = \frac{E(\{\}, \mathbf{x})}{\varepsilon} \text{ steps}$$

Because R.E-M add new models as to decrease the error E as much as possible and adjusts model parameters with the same goal, it typically converges much faster than in M iterations. Typical converges rates from our results are less than a few dozen iterations. The more models we add in

the working set the more iterations are performed however, thus M may increase dramatically in the worse case for unlimited models in the working set. Typically, we are only dealing with tens of models.

5.1.4 Determinism

R.E-M is deterministic for a given model set and training set. This means that it will converge to the same solution every time it is run with the same arguments. We have argued in section 1.5.3 that determinism is good. We have also shown in section 4.1.1 that even small changes in the training set might change the behavior of R.E-M. It is easy to see that all steps of the algorithm are deterministic. Therefor the algorithm is itself deterministic.

5.1.5 Generality

The set formulation of R.E-M makes the algorithm very general. By changing our model set and training set, we used the same algorithm to segment Gaussian mixtures and motion flows. R.E-M is much more general than that. Chapter 7 provides a few suggestions of segmentation problems that R.E-M should be capable of addressing.

5.1.6 Low bias and variance

We define the mean squared error of an estimator to be the estimation over all training sets of the squared difference between prediction and observation.

Equation (5-47) $$MSE \equiv E_T\left[\theta - \hat{\theta}\right]^2$$

For every training set, we learn the optimal model parameters $\hat{\theta}$, while the "true" model parameters underlying the data are θ. We can expand the MSE as follows:

Equation (5-48)
$$MSE \equiv E_T\left[\theta - E_T\left[\hat{\theta}\right] + E_T\left[\hat{\theta}\right] - \hat{\theta}\right]$$
$$E_T\left[\left(E_T\left[\hat{\theta}\right] - \theta\right)^2\right] + E_T\left[\left(E_T\left[\hat{\theta}\right] - \hat{\theta}\right)^2\right] + 2 \cdot E_T\left[\left(E_T\left[\hat{\theta}\right] - \theta\right)\left(E_T\left[\hat{\theta}\right] - \hat{\theta}\right)\right]$$
$$\left(E_T\left[\hat{\theta}\right] - \theta\right)^2 + E_T\left[E_T\left[\hat{\theta}\right] - \hat{\theta}\right]^2$$

The first term of this expression is a bias term, since it involves the difference of the expected value over training sets of estimator $\hat{\theta}$ and the true parameters θ. The second term is the variance of our estimator.

Equation (5-49) $\text{MSE} = \text{bias}^2 + \text{variance}$

The error of the prediction from every data point of a training set is $\varepsilon=y^*-y$. The expected value of that error over all data points of all training sets is $E_{T,\varepsilon}\left[y^* - y\right]^2 = E_\varepsilon\left[y^* - y\right]^2 + E_T[\hat{y} - y]^2$ which can be written as follows:

Equation (5-50) $E_{T,\varepsilon}\left[y^* - y\right]^2 = \text{irreducible error} + \text{MSE}$

Therefor MSE should be small, we need thus unbiased estimators with minimum variance.

R.E-M selects new models and optimizes model parameters to minimize MSE at each step. The variance of our estimator is kept minimal because we are using discreet models, thus eliminating too complicated explanations made up of models that are very similar to each other. In other words, R.E-M will not overfit because the set of models with which it can describe an observation is limited to the model set. R.E-M minimizes variance and bias, making it a good estimator.

5.1.7 Robustness

R.E-M uses a cost function to determine how well a model of the working set fits the training set and an error function to determine how well the working set fits the training set. It allows any kind of functions, as long as it is bound bellow by some finite number, in particular robust cost functions. Two robust cost functions have already been presented (in the case of Gaussian mixtures and motion flow segmentations), but the topic requires further research. Robust cost functions have many advantages, the main advantage being, as the name suggests, robustness. Such functions tend to perform well in the presence of noise, incomplete data and complicated observations. The advantages of using robust error functions are made clear in the results section and accounts for a big part of R.E-M's accuracy.

5.2. Potential problems

This section lists a few potential pitfalls of R.E-M. This list is not complete and the subject requires further research.

5.2.1 Symmetry problems

There are cases such as the one presented in 4.1.4 for which R.E-M does not find a good segmentation. These cases usually show high symmetry, just enough to make it equally costly to add any model of the model set to the training set. In such cases, R.E-M will select a model which is not necessarily the best to add some working set. The algorithm might or might not recover from such mistakes. This point brings up another pitfall of R.E-M.

5.2.2 Forward additiveness

Because of the way R.E-M works, it might start with a model that does not lead to a correct segmentation. When this happens, the algorithm might continue adding models with wrong parameters (wrong compared to the underlying models of the training set). The algorithm will eventually correct the model parameters for optimal match, but it might end up with more models in the working set than necessary for a given segmentation. The problem arises because the algorithm is forward additive, which means it adds models to the working set as it goes forward in time but never removes any. To avoid this there should be a way to remove models from a working set, and a forward-backward approach. We are currently working on bringing these ideas into the R.E-M formulation.

5.2.3 Big optimizations

As with every optimization algorithm, R.E-M has to perform big optimizations during the R.R-step and the R.M-step. There may be direct solutions to these optimizations as in the case of the R.E-M formulation for Gaussian mixtures, or even shortcuts to the optimization as in the case of R.E-M for motion flow segmentation. However, in the general form, R.E-M suffers from the optimization problem, although it is an optimization over the discreet space of the model set.

5.3. R.E-M versus E-M

This section provides a short comparison of E-M and R.E-M. The basis of the comparison are Gaussian mixtures.

5.3.1 Segmentation versus estimation

The driving difference between the two algorithms is that R.E-M is a segmentation algorithm, while E-M is a maximum-likelihood estimator. Both algorithms may be used for segmentation, but E-M does not care to find differences in the data, it just tries to match the data with it's current belief of what the data comes from.

R.E-M may result in a working set that makes some data very likely, in fact it does that most of the time. However, it does not get to that set by maximizing likelihood. It rather finds pieces or segments of the data that are best explained with some model and proceeds to find models that best fit the rest of the data. As such, R.E-M is purely a segmentation algorithm, since at no time does it try to maximize some likelihood (unless the cost function corresponds to likelihood). R.E-M does not care for the data that are not well explained by its current working set, which make the algorithm peculiarly quick. Were it to try to explain all the data with fewer models than necessary, R.E-M would take as long as E-M does. As section 4.1 shows however, R.E-M converges with orders of magnitude fewer iterations than E-M.

5.3.2 R.E-M as an E-M preprocessor

If we were to run only the R.R-step of the algorithm, we could use R.E-M as a preprocessor to E-M, which makes intelligent choices for the initial working set to use in E-M. After finding these initial conditions, we could then run E-M to find the maximum likelihood working set. R.E-M thus is not a replacement to good old E-M, rather an enhancement or a superset of the algorithm, geared heavily towards segmentation.

6 Conclusion

Although we have not fully explored the limits and advantages of R.E-M, we believe this work to be successful in that it provides an algorithm that is general and performs well in difficult segmentation problems.

We believe the incremental EM algorithm we developed to be the correct way of segmenting motion flow and a variety of other data. The key to the success of the algorithm is picking the correct initial conditions for the new models that we add incrementally and deciding when we should stop adding new models, which is subject to further research.

7 Future work

Future work for improving R.E-M includes further research on robust error and cost functions, forward-backward passes to the algorithm and robust ways of finding the correct number of models.

Our algorithm currently works for rigid objects that rotate about a fixed axis. It can be extended to handle translating objects trivially, since pure translation is a special case of pure rotation (it's a rotation about an infinitely long axis).

The algorithm can be used for a variety of segmentation problems in addition to those described in this work. In particular R.E-M should be useful for image segmentation based on color, shape, contour, or the combination of all, for segmenting silhouettes in known object contours and solving puzzles.

Acknowledgments

Randall Davis for his careful review, confidence and advice.

Paul Viola for his direction and encouragement.

Ford corporation for the RA.

My parents for being wonderful.

My siblings for being there.

My professors for being inspiring.

MIT, for being a great place.

AI Lab for being the best.

My roommate, for not complaining for the endless nights of typing on the computer.

My friends, for leaving me alone for a while to finish this work.

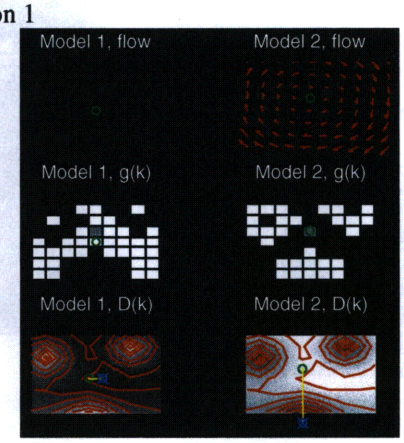
Chris Stauffer and Erik Miller for printing assistance.

Appendix A Results

Low resolution motion flow

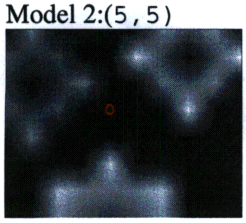
Old models:
Model 1: (6,4,0)

Incremental EM for centers of motion - Iteration 1



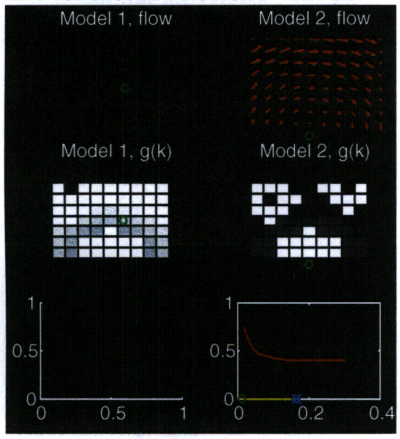
b.

New model:
Model 2:(5 , 5)



a.

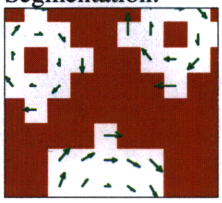
EM for model velocities - Iteration 1



c.

Results:
Model 1: (6,4,0)
Model 2: (5,0, .165)

Segmentation:

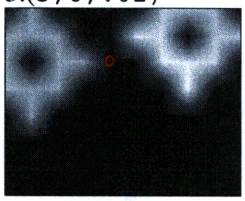


d.

The first iteration of R. E-M. We start with **a.** an model of 0 angular velocity centered at (6,4) and add a model at the centroid of the error surface (5,5). **b.** For each model, we compute $g_k(\vec{r})$ from equation (2) and minimize $\theta_k^{'}$ from equation (3) using D from equation (8) renormalized using equation (9). The green rings show the centers of the models, and the blue rings show the solution to equation (3). Yellow lines show the step taken by the M-step of the algorithm. **c.** Once the centers are estimated we run EM to estimate the angular velocities using the cost function. Finally, we segment the image.

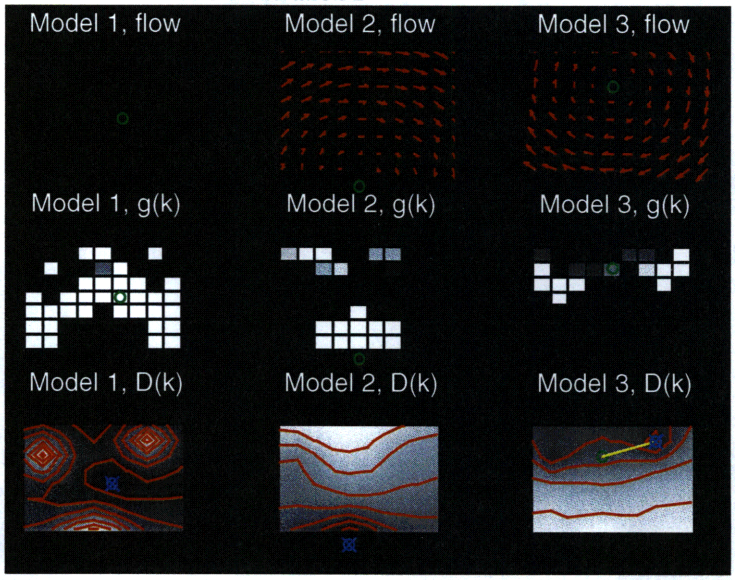
Old models:
Model 1: (6,4,0)
Model 2: (5,0, .165)

New model:
Model
3:(5,6,.01)



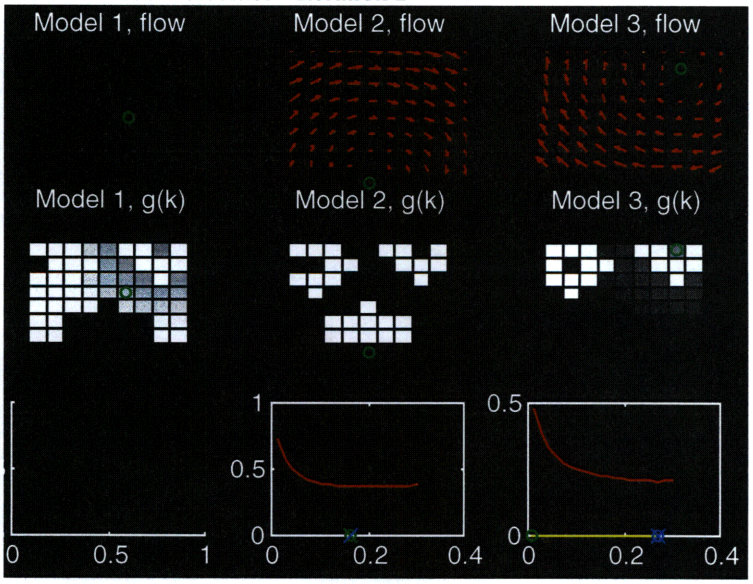
a.

Incremental EM for centers of motion - Iteration 2



b.

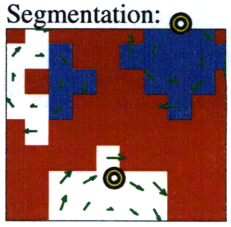
EM for model velocities - Iteration 2



c.

Results:
Model 1: (6,4,0)
Model 2: (5,0, .165)
Model 3: (8, 7, .270)

Segmentation:

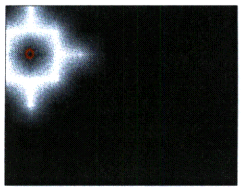


d.

The second iteration

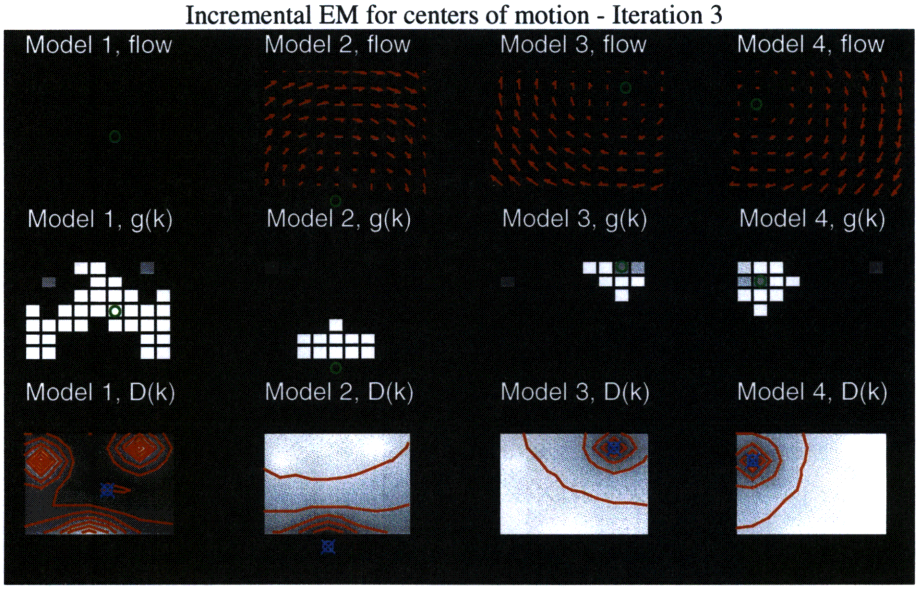
Old models:
Model 1: (6,4,0)
Model 2: (5,0, .165)
Model 3: (8, 7, .270)

New model:
Model
4:(7,7,.01)



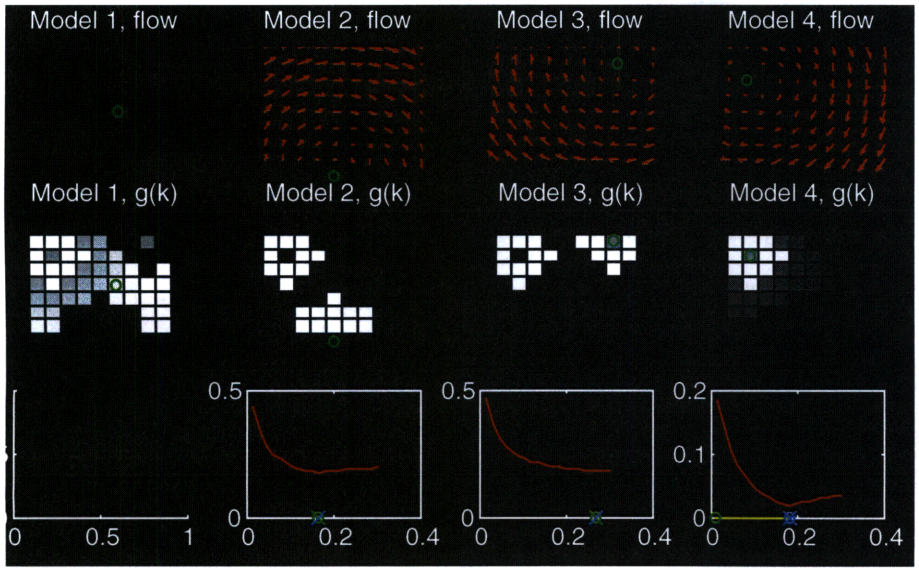
a.

Incremental EM for centers of motion - Iteration 3



b.

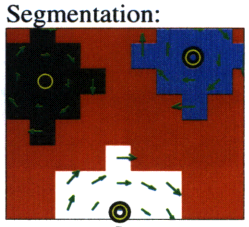
EM for model velocities - Iteration 3



c.

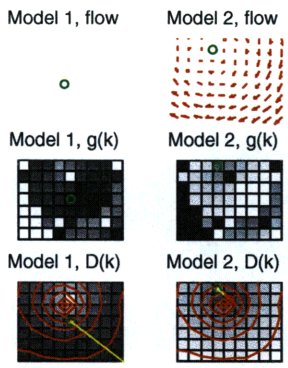
Results:
Model 1: (6,4,0)
Model 2: (5,0, .165)
Model 3: (8, 7, .270)
Model 4: (2,5, .180)

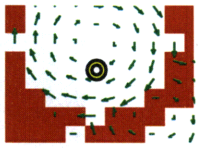
Segmentation:



d.

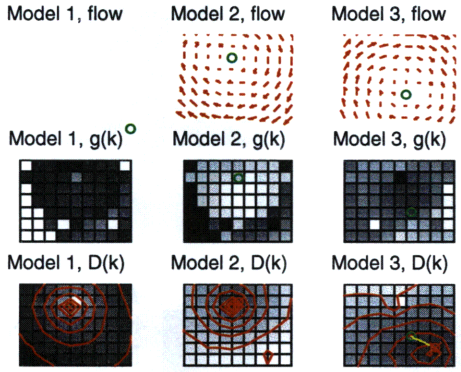
The third iteration. The error does not decrease by adding additional models thus the algorithm stops after this iteration. The segmentation show in **d.** is the correct one, and so are the model centers and angular velocities computer, also shown in **d.**

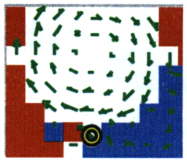




E = 0.2598

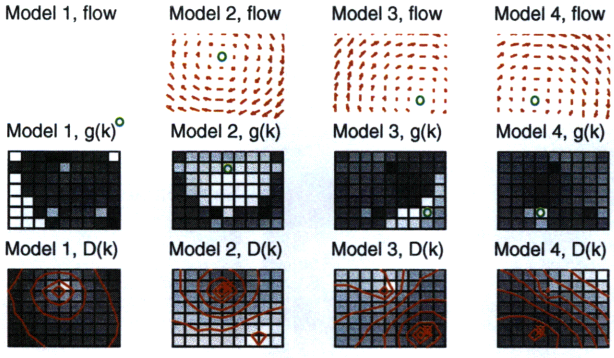
a.

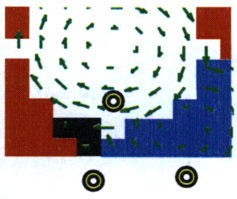




E = 0.003

b.



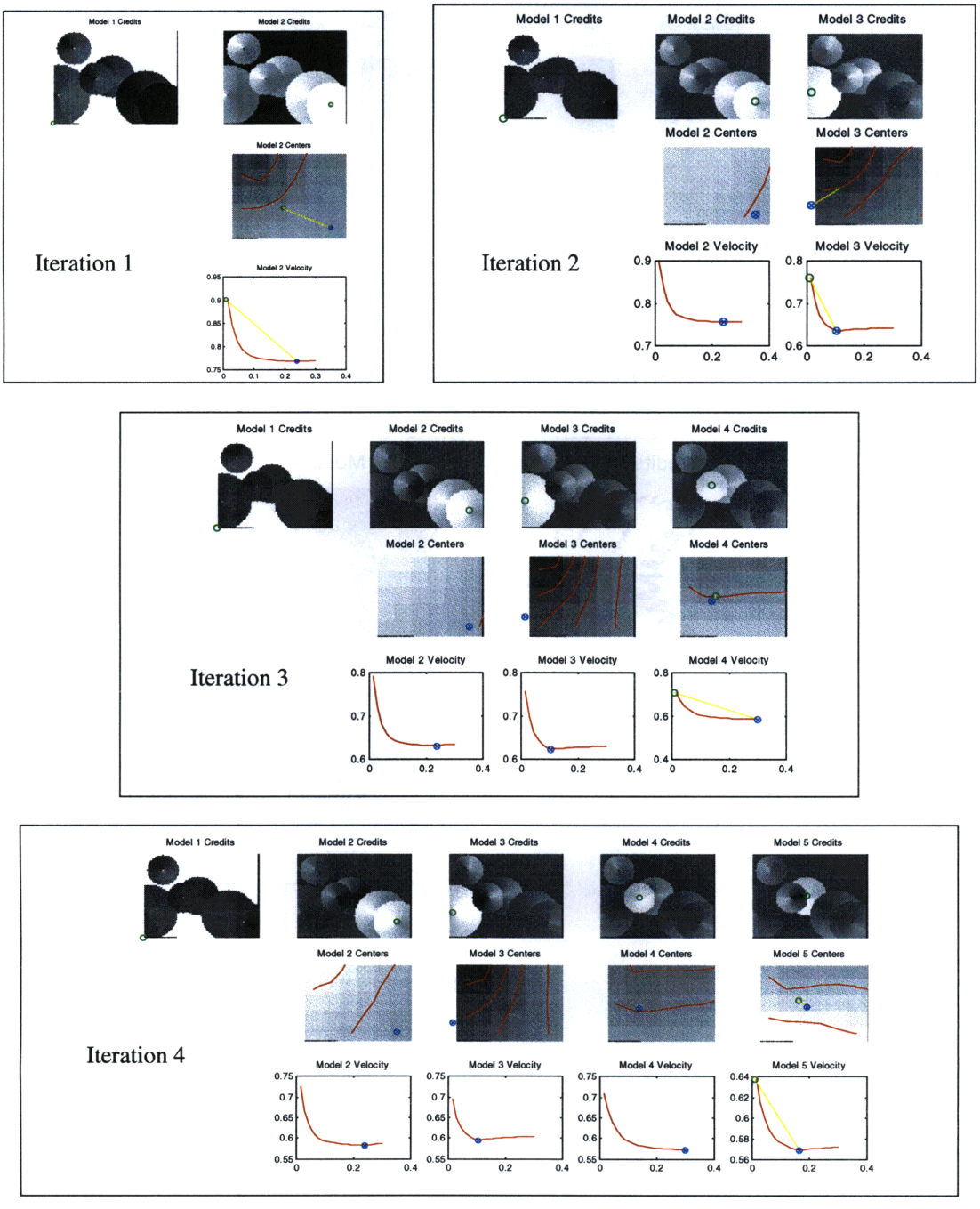


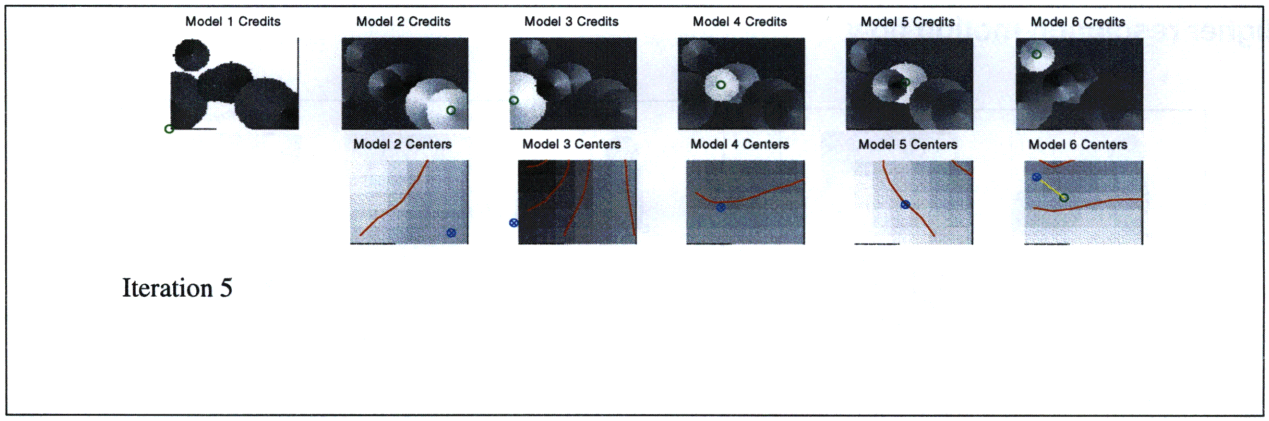
E = 1.1038e-17

c.

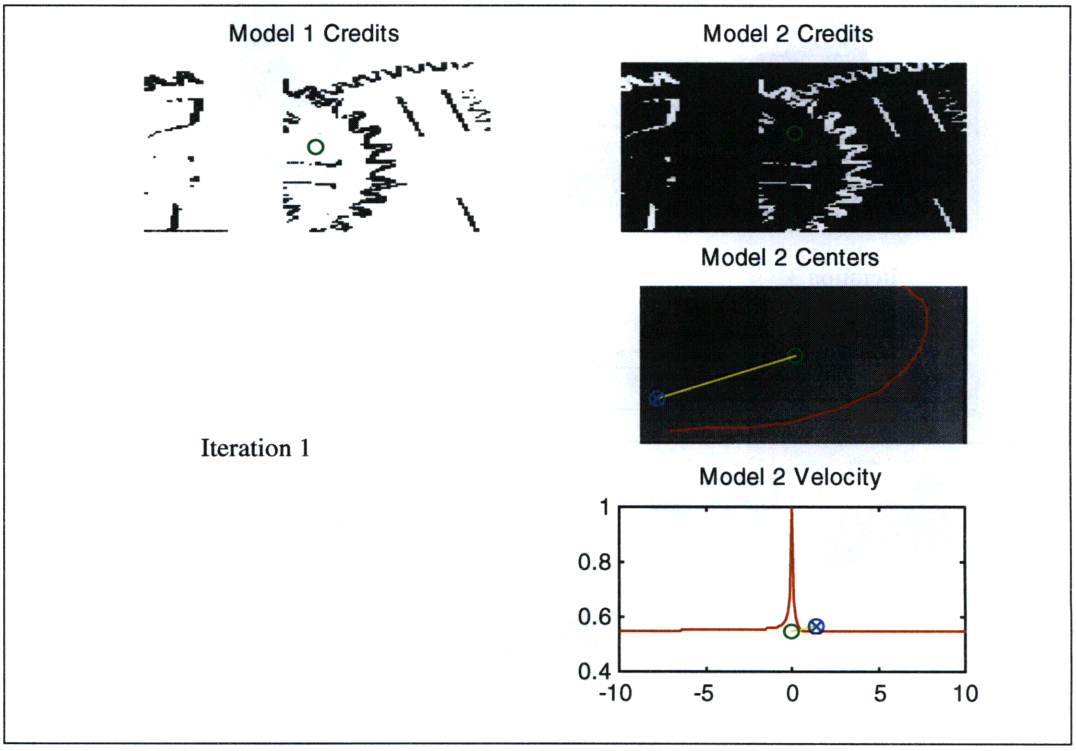
We only show the credits and error surfaces for the center of motion estimation and the segmentation at the end of iterations a. 1, b. 2 and c. 3 of the algorithm as well as the error computed at the end of each iteration. The error of c. does not decrease by adding additional models so the algorithm stops with the correct segmentation at the third iteration.

Higher resolution motion flow





Video-sequence images of a working clock



Appendix B Matlab Code

This section provides Matlab implementations of most of the algorithms described in this work.

None of this souce code is guaranteed to work, and the results presented are not guaranteed to come from this version of the code.

The source code is provided "as is" without embellishments or detailed comments, purely for reference. The most recent versions of this source code can be found at:
http://neverland.mit.edu/thesis/matlab/
and
http://neverland.mit.edu/thesis/tests/

Matlab implementation of E-M for Gaussian mixtures

```
function [pi, mu, sg, iterations] =
fitmix(z,data,x,nmix,iter, h_progress,
h_error,pi,mu,sg)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Initialization
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Set up random initial parameters
if nargin <=7,
   pi = ones(1,nmix);
   mu = rand(1,nmix)*1.0;
   sg = ones(1,nmix)*0.2;
end
num = length(x);
iterations=0;
h = zeros(num, nmix);

% Initialize errors
prediction = mgauss(pi,mu,sg,z);
old_error = sum(abs(data-
prediction/sum(prediction)*sum(data)));
all_errors = old_error;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% EM loop
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for it=1:iter,

      % Plot progress
   if h_progress~= 0,
      subplot(h_progress);
      cla;
      hold on;
      plot(z,data,'y');
      for i=1:size(pi,2),
         plot(z,mgauss(pi(i),mu(i),sg(i),z),
'r');
      end
      y = mgauss(pi,mu,sg,z);
      plot(z,y);
      plot(z,cost(y,data),'r');
      drawnow;
   end

   % Plot errors
   if h_error ~=0,
      prediction = mgauss(pi,mu,sg,z);
      new_error = sum(abs(prediction-data));
      all_errors =
cat(2,all_errors,new_error);
      subplot(h_error);
      cla;
      hold off;

semilogx(all_errors(2:size(all_errors,2)));
      hold on;
      drawnow;
   end

   % Initialization
   prev = cat(2,cat(2,pi',mu'),sg');
   iterations = iterations+1;

   % E-step
   for n=1:nmix,
      h(:,n) = pi(n)*gauss(mu(n),sg(n),x);
```

```
   end
   h = h./(sum(h,2)*ones(1,nmix));

   % M-step
   pi = sum(h,1);
   for n=1:nmix,
      mu(n) = x'*h(:,n)/pi(n);
      sg(n) = sqrt((((x-mu(n)) .* (x-
mu(n)))' * h(:,n)) /pi(n));
   end
   pi=pi/num;


   % Early termination
   if  sum(sum(cat(2,cat(2,pi',mu'),sg') ==
prev)) == size(prev,1)*size(prev,2),
      break;
   end
end

% Plot results
if h_progress~= 0,
   subplot(h_progress);
   cla;
   hold on;
   plot(z,data,'y');
   for i=1:size(pi,2),
      plot(z,mgauss(pi(i),mu(i),sg(i),z),
'r');
   end
   y = mgauss(pi,mu,sg,z);
   plot(z,y);
   drawnow;
end
```

Matlab implementation of R.E-M for Gaussian mixtures

```
function [res_pr, res_mu, res_sg,
iterations] = iem(z,data,h_progress,
h_error,nmix,pr,mu,sg)


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Initialization
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Iterations and variance of REM
iterations = 0;
sigma = 2;
allowable_error = 5; % if the outer loop
fails to improve error by that much, we exit

% Set up outlier model
if nargin >5,
   pr=cat(2,0,pr);
   sg=cat(2,1,sg);
   mu=cat(2,.5,mu);
else
   pr=0;
   sg=1;
   mu=.5;
end

% Initialize error
prediction = mgauss(pr,mu,sg,z);
old_error = sum(abs(data-prediction))
all_errors = old_error;
outer_errors = [old_error;size(mu,2)-1];
outmost_errors = [old_error;1];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% REM outer loop
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
done=0;
while ~done, % break to exit

   % Print stuff
   fprintf('REM outer loop\n');

   % Add new model where it minimizes the
most error
   merror = inf;
   nmodels=size(mu,2)+1;
   rest_data=abs(data-mgauss(pr,mu,sg,z));
   previous_stuff = data-mgauss(pr,mu,sg,z);
   pr(nmodels)=0;
   mu(nmodels)=0;
   sg(nmodels)=0;
   for mmu = .1:.1:.9,
      for ssg = .01:.02:.3,
         for ppr = .02:.012:.165,
            ppi=80*ppr*ssg;
   %    for mmu = .1:.1:.9,
   %       for ssg = .01:.02:.3,
   %          for ppi = .01:.02:.2,
            old_mu=mu(nmodels);
            old_pr=pr(nmodels);
            old_sg=sg(nmodels);
            mu(nmodels)=mmu;
            pr(nmodels)=ppi; %80*ppi*ssg;
            sg(nmodels)=ssg;
            prediction = mgauss(ppi, mmu,
ssg,z);
            err =
cost(prediction,previous_stuff);
            if (sum(err)<merror),
               merror=sum(err);
               mu(nmodels)=mmu;
               pr(nmodels)=ppi;
               sg(nmodels)=ssg;
            else
               mu(nmodels)=old_mu;
               pr(nmodels)=old_pr;
```

```
            sg(nmodels)=old_sg;
          end
        end % for pr
      end % for ssg
      drawnow;
   end % for mmu



   % Print progress
   if(h_progress ~= 0),
      subplot(h_progress);
      cla;
      hold on;
      plot(z,data,'y');
      y = mgauss(pr,mu,sg,z);
      plot(z,cost(y,data),'g');
      for i=2:size(pr,2),
         plot(z,mgauss(pr(i),mu(i),sg(i),z),
'r');
      end
      plot(z,y);
      drawnow;
   end

   % Plot errors
   if h_error~=0,
      subplot(h_error);
      cla;
      hold on;

      plot(outmost_errors(2,:),
outmost_errors(1,:),'g');

      all_errors =
cat(2,all_errors,sum(abs(mgauss(pr,mu,sg,z)-
data)));
      plot(all_errors);

      if nargin < 5,
         outer_errors =
cat(2,outer_errors,[sum(abs(mgauss(pr,mu,sg,
z)-data));size(mu,2)-1]);
         prev_i=-1;
         for i=1:size(outer_errors,2),
            if outer_errors(2,i)~=prev_i,

plot(i,outer_errors(1,i),'ro');
               text(i+.1,
outer_errors(1,i)+7, sprintf('%d models',
outer_errors(2,i)));
               prev_i=outer_errors(2,i);
            end
         end
      end
      drawnow;
   end

   % Save stuff
   res_mu = mu(2:size(mu,2)-1);
   res_sg = sg(2:size(sg,2)-1);
   res_pr = pr(2:size(pr,2)-1);
```

```
   %%%%%%%%%%%%%%%%%%
   % REM inner loop %
   %%%%%%%%%%%%%%%%%%
   em_iterations=0;
   while 1, % break to exit

      if size(mu,2)==2,
         break;
      end

      em_iterations=em_iterations+1;

      % Print stuff
      fprintf('Em inner loop\n');
      prev = cat(2,cat(2,pr',mu'),sg')'

      old_mu=mu;
      old_sg=sg;
      old_pr=pr;

      for n=2:size(mu,2), % adjust model
parameters

         %%%%%%%%%%%%%%%%%%%%%%
         % RE-step
         %%%%%%%%%%%%%%%%%%%%%%
         prediction =
mgauss(old_pr(n),old_mu(n),old_sg(n),z);
%         total = data-mgauss(pr,mu,sg,z);
         G = exp(-(1-
(prediction/max(prediction)))/sigma); %
credits
         s = G;

         s = s + exp(-(abs(data-
mgauss(pr,mu,sg,z))<.4)/sigma); % outliers
         for kk=2:size(mu,2),
            if (kk ~= n),
               prediction =
mgauss(old_pr(kk), old_mu(kk), old_sg(kk),
z);
               s = s + exp(-(1-
(prediction/max(prediction)))/sigma);
            end
         end
         G = G ./ s;

         %%%%%%%%%%%%%
         % RM-step   %
         %%%%%%%%%%%%%

         previous_stuff = data-
(mgauss(pr,mu,sg,z)-
mgauss(pr(n),mu(n),sg(n),z));

         merror=inf;
         for mmu = .1:.1:.8,
            for ssg = .02:.02:.3,
               for ppr = .01:.012:.165,
                  ppi=80*ppr*ssg;
        %for mmu = .1:.1:.9,
         %   for ssg = .01:.02:.2,
```

```
         %       for ppi = .01:.02:.3,
                   o_pr=pr(n);
                   o_mu=mu(n);
                   o_sg=sg(n);
                   mu(n)=mmu;
                   pr(n)=ppi;
                   sg(n)=ssg;
                   prediction = mgauss(ppi,
mmu, ssg, z);
                   err =
G.*cost(prediction,previous_stuff);
                   if (sum(err)<=merror),
                      if 0,
                         plot(z,err);
                         drawnow;
                      end
                      merror=sum(err);
                   else
                      mu(n)=o_mu;
                      pr(n)=o_pr;
                      sg(n)=o_sg;
                   end
                end % for pr
             end % for ssg
             drawnow;
          end % for mmu

          % Print progress
          if(h_progress ~= 0),
             subplot(h_progress);
             cla;
             hold on;
             plot(z,data,'y');
             y = mgauss(pr,mu,sg,z);
             plot(z,cost(y,data),'g');
             for i=2:size(pr,2),

plot(z,mgauss(pr(i),mu(i),sg(i),z), 'r');
             end
             plot(z,y);
             drawnow;
          end

          % Plot errors
          if h_error~=0,
             subplot(h_error);
             cla;
             hold on;

             plot(outmost_errors(2,:),
outmost_errors(1,:),'g');

             all_errors =
cat(2,all_errors,sum(abs(mgauss(pr,mu,sg,z)-
data)));
             plot(all_errors);

             if nargin <5,
                outer_errors =
cat(2,outer_errors,[sum(abs(mgauss(pr,mu,sg,
z)-data));size(mu,2)]-1);
                prev_i=-1;
                for i=1:size(outer_errors,2),
                   if
outer_errors(2,i)~=prev_i,

plot(i,outer_errors(1,i),'ro');
                      text(i+.1,
outer_errors(1,i)+7, sprintf('%d models',
outer_errors(2,i)));

prev_i=outer_errors(2,i);
                   end
                end
             end
             drawnow;
          end
       end % for n

% Termination condition of REM inner loop
       new_models =
cat(2,cat(2,pr',mu'),sg')';
       if sum(sum(prev==new_models)) ==
size(prev,1)*size(prev,2),
          break
       end

    end % while - REM inner loop ends

% Termination condition of REM outer loop
    new_error = sum(abs(mgauss(pr,mu,sg,z)-
data))
    outmost_errors =
cat(2,outmost_errors,[outer_errors(1,size(ou
ter_errors,2));size(outer_errors,2)]);
    if nargin < 5,
       if (old_error - new_error >
allowable_error),
          old_error = new_error;
       else
          done=1;
       end
    else
       if size(mu,2)-1 >= nmix,
          res_mu = mu(2:size(mu,2));
          res_sg = sg(2:size(sg,2));
          res_pr = pr(2:size(pr,2));
          done=1;
       end
    end

end % while - REM outer loop ends

% Plot result
if h_progress~= 0,
    y = mgauss(res_pr,res_mu,res_sg,z);
    subplot(h_progress);
    cla;
    hold on;
    plot(z,data,'y');
    plot(z,cost(y,data),'g');
    for i=1:size(res_pr,2),

plot(z,mgauss(res_pr(i),res_mu(i),res_sg(i),
z), 'r');
    end
```

```
   plot(z,y);
   drawnow;
end

% Plot errors
if h_error~=0,
   subplot(h_error);
   cla;
   hold on;

   plot(outmost_errors(2,:),
outmost_errors(1,:),'g');

   all_errors =
cat(2,all_errors,sum(abs(mgauss(pr,mu,sg,z)-
data)));
   plot(all_errors);

   if nargin < 5,
      outer_errors =
cat(2,outer_errors,[sum(abs(mgauss(pr,mu,sg,
z)-data));size(mu,2)]-1);
      prev_i=-1;
      for i=1:size(outer_errors,2),
         if outer_errors(2,i)~=prev_i,
            plot(i,outer_errors(1,i),'ro');
            text(i+.1, outer_errors(1,i)+7,
sprintf('%d models', outer_errors(2,i)));
            prev_i=outer_errors(2,i);
         end
      end
   end
   drawnow;
end

iterations = size(all_errors,2);

function er = cost(prediction, motion)

er=abs(prediction-
motion).*h(prediction,motion);


function m = h(prediction,observation)

m = ((prediction/max(prediction)-1)/1.5+1);
```

Matlab sample test code for Gaussian mixtures

```
% This test compares EM's and REM's error
versus number of iterations

pi = [.5  .2 .3   ];
mu = [.5  .8 .7   ];
sg = [.08 .04 .04 ];
datapoints=5001;
iter = 1000;

z=0:.01:1;
mixture =
createGaussData(mu,sg,pi/sum(pi),z,datapoint
s);
y = mgauss(pi,mu,sg,z);
x = mixture';

 [v,h] = hist(x,z);
v=v/sum(v)*sum(y);
z = h;
data=v;

figure(1);
subplot (3,2,1);
cla;
hold on;
bar(h,v);%/sum(v)*20);
title(sprintf('a. Empirical distribution. %d
data points', size(x,1)));
drawnow;

subplot(3,2,2);
cla;
hold on;
%axis([-.5 1.5 0 1.0]);
y = mgauss(pi,mu,sg,z);
%data = y;
title(sprintf('b. Ground truth density.',
sum(y.*log(y+0.001))));
hold on;
for i=1:size(pi,2),
   plot(z,mgauss(pi(i),mu(i),sg(i),z), 'r');
end
plot(z,y);
drawnow;

% Set up axes
subplot(3,2,6);
cla;
title('f. R.E.M. Error vs. iterations');
xlabel('iterations');
ylabel('error');
hold on;
h_error = gca;
subplot(3,2,4);
cla;
title (sprintf('c. R.E.M. - Progress...'));
hold on;
h_progress = gca;
% Call REM
[pi, mu2, sg, it] =
iem(z,data,h_progress,h_error);
subplot(h_progress);
title (sprintf('d. R.E.M. - %d iterations',
it));

% Set up axes
subplot(3,2,5);
cla;
```

```
title('e. E.M. Error vs. iterations');
xlabel('iterations');
ylabel('error');
hold on;
h_error = gca;
subplot(3,2,3);
cla;
title (sprintf('c. E.M. for %d models -
Progress...', size(mu,2)));
hold on;
h_progress = gca;
% Call EM
[pi, mu, sg, it] =
fitmix(z,data,x,size(mu,2),iter,h_progress,h
_error);
subplot(h_progress);
title (sprintf('c. E.M. for %d models - %d
iterations', size(mu,2), it));
subplot(h_error);
hold on;
title('e. E.M. Error vs. iterations');
xlabel('iterations');
ylabel('error');
```

```
function
d=createGaussData(mu,sg,pi,z,numPoints)

res = mgauss(pi,mu,sg,z);
res = res/max(res); % make it probabilities
n=1;

while n<numPoints,

   p = max(1,round(rand*size(z,2)));
   if(res(p)>rand)
      d(n)=z(p);
      n=n+1;
   end
end
function res = gauss(mu, sg, x)

res=exp(-.5*(x-mu).*(x-
mu)*inv(sg^2))/((2*pi)^(size(x,2)/2)*sg^size
(x,2));
%res=exp(-.5*(x-mu)'*inv(sg^2)*(x-
mu))/((2*pi)^(size(x,1)/2)*norm(sg)^size(x,1
));
```

Matlab implementation of R.E-M for motion segmentation

```
function main(mode)
% MAIN  Image segmentation code

global modelbase_u modelbase_v;
global X Y N wheels;

%load variables.mat;
warning off;

if (nargin < 1)
  mode = 1;
end

if (mode == 0)    % Simulated wheels

% Constants
N = 7;  % number of models
X=100;
Y=80;

% Generate random data
  wheels = randomwheels(N,X,Y);
 [motion data] = generatewheels(wheels,X,Y);

  % Plot wheels
  if 1,
    figure(1);
    hold off;
    plotfield(motion,'g');
    axis([0 X 0 Y]);
    axis('off');
    hold on;
    imagesc(data);
    colormap('gray');
    plotfield(motion,'g');
   end
else % Real wheels

  % Read in images % center is at 34,176
  a = double(imread('smal_1.tif'));
  b = double(imread('smal_2.tif'));
  data = a;

  [u v] = flow(a,b);
  motion = u+i*v;
  [Y X] = size(motion);

  % Plot stuff
  figure(1);
  hold off;
  plotfield(u+i*v);
  hold on;
  clf
  imagesc(data);
  colormap('bone');
  hold on;
  plotfield(u+i*v,'b');

end


% Initialize models - incremental EM
```

```
newmodels = [floor(X/2); floor(Y/2); 0; 0];
[data1 errors] = segment(motion, newmodels);
newmodels(1:2,2) = centroid(errors);
newmodels(3,2) = .01;


%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Augmented EM algorithm
%%%%%%%%%%%%%%%%%%%%%%%%%%%
done = 0;
iter = 0;
err = Inf;
while ~done,

   %Initialization stuff
   models=newmodels
   iter = iter +1;
   figure; clf; colormap('bone');

   %Print some stuff
   wheels
   models
   fprintf('\nIncremental EM iteration %d',
iter);

   % Find the centers of motion
   newmodels = centers(motion, models,.001);

   % Find model velocities
   newmodels = velocities(motion, newmodels,
.001);

   % Segment the data
   [data1 errors] = segment(motion,
newmodels);

   % Continuation condition
   if sum(sum(errors)) < err;
      err=sum(sum(errors))
      s=size(newmodels);
      newmodels(1:2,s(2)+1) =
centroid(errors);
      newmodels(3,s(2)+1) = .01;
   else
      s=size(newmodels);
      newmodels = newmodels(:,1:s(2)-1);
      done = 1;
   end

end

wheels
newmodels
err



function newmodels = centers(motion, models,
sg)
% CENTERS Find centers of motion for motion
fields
%   findcenters(motion,N) fits N motion
models to the motion field in
%   motion and returns a matrix with the
models as the columns.
%
%   See also: RANDOMWHELLS, GENERATEWHEELS,
GENERATEMODEL, COST


%%%%%%%%%%%%%%%%%%%%%%%%%
% Initialization
%%%%%%%%%%%%%%%%%%%%%%%%%

sg = 5;

% Initialization
iter = 1;      % number of iteration
[Y X] = size(motion);
N = size(models,2);
done = 0;
it = 0;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% E-M algorithm
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

while ~done,

   it = it +1;
   newmodels = models;

   fprintf('\nEM for centers of motion.
Iteration: %d Variance:%g\n', it, sg);
   fprintf('Paused...');
   pause
   fprintf(' ok\n');


   for k=1:N,  % for all models

      prediction =
generatemodel(models(:,k),X,Y);

      %%%%%%%%%%%%%%%%%%%%
      % E-step
      %%%%%%%%%%%%%%%%%%%%

      % Compute the credits g(k) for all
data points
      D = cost(prediction, motion);
      G = exp(-D/sg); % priors
      s = G;
      for kk=1:N, % idiot, inneficient loop
         if kk ~= k,
            pred =
generatemodel(models(:,kk),X,Y);
            s = s + exp(-cost(pred,
motion)/sg);
         end
      end
      G = G ./ s;

      % Plot credits
```

```
      subplot(3,N,k);
      cla;
      plot(2,2);
      axis([0 X 0 Y]);
      axis 'off';
      hold on;
      imagesc(G);
      plot(models(1,k), models(2,k), 'go');
      title(sprintf('Model %d Credits',k));
      drawnow;
   end


   for k=1:N,  % for all models

      prediction =
generatemodel(models(:,k),X,Y);
      % Compute the credits g(k) for all
data points
      D = cost(prediction, motion);
      G = exp(-D/sg); % priors
      s = G;
      for kk=1:N,
         if kk ~= k,
            pred =
generatemodel(models(:,kk),X,Y);
            s = s + exp(-cost(pred,
motion)/sg);
         end
      end
      G = G ./ s;


      %%%%%%%%%%%%%%%%%%%
      % M-step
      %%%%%%%%%%%%%%%%%%%

      % Update the parameters
      if k ~= 1,
         subplot(3,N,k+N);
         newmodels(1:2,k) =
minimize(motion,G,models(1:2,k),k,N,it);
      else
         newmodels(1:2,k) = [0;0];
      end

   end % for k=1:N


   % Termination condition
   if  sum(sum(newmodels == models)) ==
size(models,1)*size(models,2);

      % Deterministic annealing
      if (sg >1/5),
         sg = sg/5;
      else
         done = 1; % done when we converge
      end
   end

   % Update models
   models = newmodels;
```

```
end

function c = centroid(mat)
% CENTROID Compute centroid of a matrix
%
%   centroid(a) returns a column vector with
%   the indexes closest to the coordinates of
the
%   center of mass of the matrix a.

s=size(mat);

mat=mat/sum(sum(mat)); % normalize

% Find centerf of mass of the columns
for i = 1:s(2),
  x(i,1) = i;
end
c(1)=sum(mat)*x;

% Find centerf of mass of the rows
for i = 1:s(1),
  y(i,1) = i;
end
c(2)=sum(mat')*y;
c=round(c)';



function er = cost(prediction, motion, kind)
% COST  Compute deviations of the prediction
from the actual motion
%   cost(prediction, motion) returns a matrix
of the same
%   size as prediction and motion that
corresponds to the
%   errors between the motion fields in
prediction and the
%   motion field in motion.
%
%   cost(prediction, motion, 1) also plots
graphs.
%
%   See also TOTALS, PLOTFIELD.

%er = (2*pi-er).*(~((sign(er)+1)))+er.*(~(1-
sign(er)));



if nargin>2,    % use magnitude error
  er = abs(prediction-motion);
  er = er/sum(sum(er));    % RENORMALIZATION
(since we don't have priors)
else    % use angular error
  er = angle(prediction)-angle(motion);
  er =
(2*pi+er).*(~((sign(er)+1)))+er.*(~(1-
sign(er)));
  ang = sin(er/2); % angular error
  er = ang;        % total error
```

```
  er = er .* (~xor(motion,prediction));
  er = er + xor(motion,prediction);

  er = er/sum(sum(er));    % RENORMALIZATION
(since we don't have priors)

end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Plot stuff
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if 0, %nargin >2,
s= size(prediction);
X=s(2);
Y=s(1);


    subplot(2,2,1), hold off;
    subplot(2,2,1), plotfield(motion,grid);
    subplot(2,2,1), axis([0 X 0 Y]);
    hold on;
    subplot(2,2,1), title('Data');
    drawnow;

    subplot(2,2,2), hold off;
    subplot(2,2,2),
plotfield(prediction,grid);
    subplot(2,2,2), axis([0 X 0 Y]);
    subplot(2,2,2), hold on;
    subplot(2,2,2), title('Prediction');
    drawnow;

    subplot(2,2,3), hold off;
    subplot(2,2,3), plotfield(err,grid);
    subplot(2,2,3), axis([0 X 0 Y]);
    subplot(2,2,3), hold on;
    subplot(2,2,3), title('Error');
    drawnow;

    subplot(2,2,4), hold off;
    subplot(2,2,4), surf(er);
    subplot(2,2,4), axis([0 X 0 Y -10 10]);
    subplot(2,2,4), hold on;
    subplot(2,2,4), colormap(bone);
    subplot(2,2,4), view(0,90);
    subplot(2,2,4), title('Cost');
    drawnow;

    %hold off;
    %surf(c);
    %colormap(bone);
    %view(0,90);
    %hold on;
    %title('Cost');
    %drawnow;

end


function [up, vp] = filterfield(u,v)
% FILTER       Filters the field in u,v for
input to the main program


mask = max(abs(u+i*v)-7, 0);
mask = ~~mask;

up = mask.*u;
vp = mask.*v;


function [u, v] = flow(a, b)
% FLOW  Computes the motion flow between
images 1 and 2.
%   Returns the x component of the flow in u
and the y
%   component in v.


%[Ex Ey Et] = derivatives(a,b,1/30);

[Ex Ey] = gradient(a);
Et = a-b;

s = ones(size(Ex,1), size(Ex,2)) ./
sqrt(Ex.*Ex+Ey.*Ey);
s = max(min(s,ones(size(s,1),
size(s,2))),zeros(size(s,1), size(s,2))) ; %
eliminate NaN and -NaN

% Point u,v in direction of the gradient
u = Ex.*s;
v = Ey.*s;


u = u.*Et;
v = v.*Et;

[u v] = filterfield(u,v);

%[u v] = gradient(a);


function motion = generatemodel(wheel, X,
Y);
% GENERATEMODEL  Generates motion fields for
model wheel
%   generatemodel(wheels) generates the
motion field of the
%   spinning wheels in motion and a picture
of the wheels in data.
%   The default size of the motion field and
the data is 30x30.
%   All vectors generated for the motion
field have
%   unit length (this is the principal
difference with
%   GENERATEWHEELS).
%
%   generatemodel(wheels,X,Y) uses a motion
field and data
%   field of XxY.
```

```
%
%   See also RANDOMWHEELS, GENERATEWHEELS,
PLOTFIELD.


global modelbase_u modelbase_v;

x = wheel(1)-2;
y = wheel(2)-2;
speed = -wheel(3);


U = (size(modelbase_u, 2)-1)/2;
V = (size(modelbase_v, 1)-1)/2;

motion = modelbase_u(V-y:V-y+abs(Y)-1, U-
x:U-x+abs(X)-1) + i* modelbase_v(V-y:V-
y+abs(Y)-1, U-x:U-x+abs(X)-1);

motion = motion*speed;



function [motion, data] =
generatewheels(wheels, X,Y);
% GENERATEWHEELS Generates motion fields for
model wheels
%   [motion, data] = generatewheels(wheels)
generates
%   the motion field of the spinning wheels
in motion
%   and a picture of the wheels in data. The
default
%   size of the motion field and the data is
30x30.
%
%   [motion, data] =
generatemodel(wheels,X,Y) uses
%   a motion field and data field of XxY.
%
%   See also RANDOMWHEELS, GENERATEMODELS,
PLOTFIELD.

% Fixed parameters
if nargin<3,
  X=30; % image size
  Y=30;
end

N=size(wheels);    % number of randomly
spining, randomly placed, randomly sized
wheels
N = N(2);
motion = zeros (Y,X);
data = zeros (Y,X);

for x=1:X,
  for y=1:Y,
    for k=1:N,
      diff = [x;y] - wheels(1:2,k);
      if (norm(diff) <= wheels(4,k)),
         data(y,x)=wheels(3,k);
         if ~(norm(diff)==0),
            n = [-diff(1);diff(2)] ;
            m =
norm(diff)*wheels(3,k)*n/norm(n);
            motion(y,x) = m(2)+i*m(1);
         else
            motion(y,x) = 0;
         end
      end

    end
  end
end

% Add some small gaussian noise
%motion=motion+randn(Y,X)*.05+i*randn(Y,X)*.
05;



function res = minimize(motion, G,
init,k,N,it)
% MINIMIZE Finds the minimum of the cost
function at the M-step of EM
%
%   minimize(motion,g) returns the indices in
the motion matrix of the
%   center of the models that minimizes the
cost function in cost.m
%
%   minimize(motion,g, init) uses init as an
initial guess
%
%   See also: COST, CENTERS


% Constants
K = 15; % subdivisions
[Y X] = size(motion);
Xinit = 0;
Xstep = X/K;
Xfinal = X;
Yinit = 0;
Ystep = Y/K;
Yfinal = Y;
done =0;
iter = 0;
while ~done,

  clear errors;
  i = 0;
  iter = iter + 1;

  for x=Xinit:Xstep:Xfinal,
    i = i+1;
    j= 0;
    for y = Yinit:Ystep:Yfinal,
      j = j + 1;
      assumption =
generatemodel([round(x);round(y);0.01],X,Y);
      tmp = G.*cost(assumption,motion);
      errors(j,i) = sum(sum(tmp));
    end
    [a b] = min(errors);
```

```
    [c d] = min(a);
    res = [round(Xinit+Xstep*(d-1));
round(Yinit+Ystep*(b(d)-1))];
  end


  % Plot error surface only for first
iteration
  if (iter == 1 & it == 1)
     plot(1,1);
     hold on;
     imagesc(errors-max(max(errors)));
     shading interp;
     axis([0 X/K 0 Y/K]);
     axis 'off'
     title(sprintf('Model %d Centers',k));
     contour(errors,'r');
  end

  % Continuation
  Xinit=min(X,max(0,res(1,1)-Xstep));
  Xfinal=min(X,max(0,res(1,1)+Xstep));
  if Xinit>Xfinal,
     tmp = Xfinal;
     Xfinal = Xinit;
     Xinit = Xfinal;
  end
  Xstep=max(1,Xstep/K/2);

  Yinit=min(Y,max(0,res(2,1)-Ystep));
  Yfinal=min(Y,max(0,res(2,1)+Ystep));
  if Yinit>Yfinal,
     tmp = Yfinal;
     Yfinal = Yinit;
     Yinit = Yfinal;
  end
  Ystep=max(1,Ystep/K/2);

  if Xfinal-Xinit <= 3 & Yfinal-Yinit <= 3,
     done = 1;

     % Plot starting and ending positions
     plot([ init(1)/K], [ init(2)/K], 'go');
     plot([ init(1)/K; res(1)/K], [
init(2)/K; res(2)/K], 'y-');
     plot([ res(1)/K], [ res(2)/K], 'bo');
     plot([ res(1)/K], [ res(2)/K], 'bx');
     drawnow;
  end

end



function [linex, liney] =
plotfield(field,attributes,grid)
% PLOTFIELD  Plot 2D vector fields
%   [x,y] = vectorfield(img) plots the vector
fields specified by the
%   2-D vector img. real(img) is the x
component at each position
%   and imag(img) is the y component.
%   x and y are vectors that can be fed
directly into the line
%   command.
%
%   vectorfield(img,grid) uses a grid size of
grid.


if nargin < 2,
  attributes = 'w';
end
if nargin < 3,
  grid  = 10;
end


u = real(field);
n = max(max(max(abs(u))));
if n ~= 0,
  u = u'/n/sqrt(2); % normalize
end

v = imag(field);
n = max(max(max(abs(v))));
if n ~= 0,
  v = v'/n/sqrt(2); % normalize
end

k=1;

for i=1:grid:size(u,1),
    for j=1:grid:size(u,2),

        % Arrow body
        linex(1,k) = i;
        linex(2,k) = i+grid*u(i,j);
        liney(1,k) = j;
        liney(2,k) = j+grid*v(i,j);
        l =
sqrt(norm([linex(2,k);liney(2,k)]-
[linex(1,k);liney(1,k)]))/5*4;
        k=k+1;

        % Arrow tip
        tmp=[u(i,j);v(i,j)]*1.5*l+0.7*l*[-
v(i,j);u(i,j)]+[i;j];
        linex(1,k) = linex(2,k-1);
        linex(2,k) = tmp(1,1);
        liney(1,k) = liney(2,k-1);
        liney(2,k) = tmp(2,1);
        k=k+1;

        tmp=[u(i,j);v(i,j)]*1.5*l-0.7*l*[-
v(i,j);u(i,j)]+[i;j];
        linex(1,k) = linex(2,k-2);
        linex(2,k) = tmp(1,1);
        liney(1,k) = liney(2,k-2);
        liney(2,k) = tmp(2,1);
        k=k+1;

    end
end
```

```
t = ishold;
plot(linex,liney,attributes);
hold on;
%plot(ox,oy,attributes);
if ~t,
  hold off;
end
```

```
function wheels = randomwheels(N,X,Y)
% RANDOMWHEELS Initialize data randomly
%
% [motion, data, wheels] = initrandom
% Generate synthetic image flows

rand('seed',sum(100*clock));

if nargin<1,
  N=3;  % number of randomly spining,
randomly placed, randomly sized wheels
end

if nargin<3,
  X=30; % image size
  Y=30;
end


% Generate random wheels. First wheels is at
the bottom, last on top
% [ center_x
%   center_y
%   omega
%   radius ]
wheels = rand(4,N) .* ([X; Y; 0;
min(X,Y)/3]*ones(1,N));
wheels = floor(wheels);
%wheels(3,:)=ones(1,N)*0.01;
%+rand(1,N)*0.01-0.005;
wheels(3,:)=ceil(rand(1,N)*20)/(20/.3);
```

```
function [data, errors] =
segment(motion,models)
% SEGMENT segments motion field given
estimated models

s=size(motion);
X=s(2);
Y=s(1);
s = size(models);
N = s(2);

errors = zeros(Y,X)+Inf;
data = zeros(Y,X);

for k=1:N,
   prediction = generatemodel( models(:,k),
X, Y);
   %  error = abs(prediction-motion);
   error = cost(prediction,motion,'total');

   for i=1:X,
      for j=1:Y,
         if error(j,i) <= errors(j,i),
            errors(j,i) = error(j,i);
            data(j,i) = models(3,k);
         end
      end
   end

end
```

```
function newmodels = velocities(motion,
models,sg)
% VELOCITIES Find velocities assuming the
centers are correct
%
%   See also: FINDCENTERS, RANDOMWHELLS,
GENERATEWHEELS, GENERATEMODEL, COST


% MAIN  Image segmentation code

%%%%%%%%%%%%%%%%%%%%%%%%%
% Initialization
%%%%%%%%%%%%%%%%%%%%%%%%%

if nargin<3,
  sg=1;
end

% Constants
s = size(motion);
X=s(2);
Y=s(1);
s = size(models);
N = s(2);

for x=1:X,
  XX(:,x) = zeros(Y,1)+x;
end
for y=1:Y,
  YY(y,:) = zeros(1,X)+y;
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% E-M algorithm
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

done = 0;
it = 0;
while ~done,

   % Initialization stuff
   it = it +1;
   newmodels = models;

   % Print stuff
```

```
   fprintf('\nEM for centers of motion.
Iteration: %d\n', it);
   fprintf('Paused...');
   pause
   fprintf(' ok\n');


   for k=1:N,  % for all models

      prediction =
generatemodel(models(:,k),X,Y);

      %%%%%%%%%%%%%%%%%%%
      % E-step
      %%%%%%%%%%%%%%%%%%%

      % Compute the credits g(k) for all
data points
      D = cost(prediction,motion,'total');
      G = exp(-D/sg);
      s = G;
      for kk=1:N, % idiot, inneficient loop
         if kk ~= k,
            pred =
generatemodel(models(:,kk),X,Y);
            s = s + exp(-abs(pred-
motion).^2/sg);
         end
      end
      G = G ./ s;

      %%%%%%%%%%%%%%%%%%%
      % M-step
      %%%%%%%%%%%%%%%%%%%

      if models(3,k) ~= 0, % only if we're
not dealing with the background model
         % Update the parameters
         num=0;
         den=0;
         for x=1:X,
            for y =1:Y,
               num=num+G(y,x)*((models(2,k)-
y)*real(motion(y,x))+(x-
models(1,k))*imag(motion(y,x)));

den=den+G(y,x)*(((models(2,k)-y)^2+(x-
models(1,k))^2));
            end
         end
         %      newmodels(3,k) =
sum(sum(G.*((models(2,k)-
YY).*real(motion)+(XX-
models(1,k)).*imag(motion))));
         %      newmodels(3,k) =
newmodels(3,k)/sum(sum(G.*((models(2,k)-
YY).^2+(XX-models(1,k)).^2)));
         %      newmodels(3,k) = num/den;


         ii=0;
         mm = Inf;
         for v=0.015:0.015:.3,
            ii = ii + 1;
            assumption =
generatemodel([models(1,k);models(2,k);v],X,
Y);
            D=cost(assumption,motion,
'total');
            tmp = G.*D;
            errors(ii) = sum(sum(tmp));
            indexes(ii) = v;
            if errors(ii) < mm,
               mm = errors(ii);
               newmodels(3,k) = v;
            end
         end

    % Plot updates
    subplot(3,N,k+2*N);
    cla;
    plot(indexes, errors,'r-');
    hold on;
    plot([models(3,k) newmodels(3,k)],
[errors(round(models(3,k)/.015))
errors(round(newmodels(3,k)/0.015))], 'y-');
    plot([models(3,k)],
[errors(round(models(3,k)/.015))], 'go');
    plot([newmodels(3,k)],
[errors(round(newmodels(3,k)/0.015))],
'bo');
    plot([newmodels(3,k)],
[errors(round(newmodels(3,k)/0.015))],
'bx');
    title(sprintf('Model %d Velocity',k));
    drawnow;

 end



end % for k=1:N


  s=size(models);
  if  sum(sum(newmodels == models)) ==
s(1)*s(2);
  done = 1; % done when we converge
  end

  models = newmodels;
  done = 1;
end
```

Bibliography

[Black and Anandan, 1993] Michael J. Black, P. Anandan, A framework for the robust estimation of optical flow, Proc. Fourth Int. Conf. On Computer Vision (ICCV'93), Berlin, Germany, May 1993

[Jepson and Black, 1996] Allan Jepson and Michael Black, Mixture models for image representation. PRECARN ARK project technical report ARK96-PUB-54, March 1996.

[Jepson and Black, 1993] Allan Jepson and Michael Black, Mixture models for optical flow computation, University of Toronto, Department of Computer Science, Technical Report: RBCV-TR-93-44, April 1993.

[Jepson and Heeger, 1991] Jepson AD, Heeger, DJ. A fast subspace algorithm for recovering rigid motion. Proc. Of IEEE Workshop on Visual Motion, Princeton, NJ (Oct. 1991), 124-131 1991.

[MacLean, Jepson and Frecker, 1992] W. James MacLean, Allan D. Jepson & Richard C. Frecker, Recovery of Egomotion and Segmentation of independent object motion using the EM algorithm, British Machine Vision Conference, 1992.

[Weiss and Adelson, 1995] Perceptually organized EM: A framework for motion segmentation that combines information about form and motion. M.I.T. Media laboratory Perceptual Computing Section Technical Report No. 315. Submitted ICCV '95.

[Weiss and Adelson, 1995] Yair Weiss and Edward. H. Adelson, Motion Estimation and Segmentation using a Recurrent mixture of experts architecture. MIT Media Laboratory, Perceptual Computing Section Technical Report No. 344, To appear: 1995 IEEE Workshop on Neural Networks for Signal Processing

Index

R

S

T

U

W