

Chapter 1

BAYESIAN METHODS FOR TRACKING AND LOCALIZATION

Wojciech Zajdel, Ben J.A. Kröse, Nikos Vlassis

Abstract This chapter presents a tutorial-type introduction to dynamic Bayesian networks (DBNs), which provide a computational framework for the analysis of stochastic dynamic systems. The first part of the chapter provides a short overview of the work on probabilistic state estimation and system identification. The second part presents two example applications where the DBNs lead to elegant algorithms: robot localization and tracking of multiple persons with multiple cameras.

Keywords time-series analysis, probabilistic inference, dynamic Bayesian networks, robot localization, multi-object tracking.

1.1 Introduction

Localization and tracking of moving objects are one of the central issues in research in intelligent environments. Typical examples are service or security applications, where the environment needs to localize moving persons. Alternatively, mobile embedded systems have to be localized and tracked, for example to deliver location dependent services. In all of these situations there is a need for sensors to give location information.

In typical problems, one can distinguish between two configurations: the sensors can either be mounted on the moving system and observe the environment, or the sensors may be fixed to the environment and observe the moving system. An instance of the first configuration is the GPS (global positioning system), which uses the position of satellites as reference points to calculate the position of the moving system. Other examples are systems which use radio beacons. If the environment is not artificially landmarked, natural features have to be used to localize the system. These features have to be derived from the sensory signal, for example from images of a camera mounted on a moving platform. However, in many cases the objects to be tracked are not equipped with sensors. In the second configuration, it is the environment that observes

the moving objects. Because in this case the sensors are not associated one-to-one with objects, the environment will not only have to *localize* the moving objects but also have to *identify* them. In the subsequent discussion we will assume color video cameras as sensing systems.

Irrespective of the sensor configuration, localization and tracking methods need to estimate the state of a dynamic system (i.e. object's identity and/or position) from observations. The estimation algorithm has to account for two problems. Firstly, the observations are affected by noise, which is typically caused by camera jitter, variations in illumination or viewing angle, occlusions, and shadows. Secondly, cameras provide often high-dimensional observations, and modeling dynamics in a high-dimensional space is not trivial. To address these issues we consider a probabilistic framework to deal with the noise and furthermore use the fact that the underlying system generating the observations has only a few degrees of freedom.

This chapter offers a tutorial on dynamic Bayesian networks and their applications to localization and tracking. Section 1.2 outlines basic forms of DBNs and presents suitable Bayesian inference algorithms. Section 1.3 presents a method for localizing a mobile robot from sensory information, and Section 1.4 — a method for tracking multiple people with multiple cameras.

1.2 Bayesian networks for dynamic systems analysis

Throughout this tutorial we discuss stochastic discrete-time dynamic systems. We denote the subsequent time steps with $k = 1, 2, \dots$, and the state of the system at the k th step with s_k , and a time-sequence of states s_n, s_{n+1}, \dots, s_m as $s_{n:m}$. Our key assumption is that the state of the system in question cannot be directly observed. Instead, at every step we have access to a noisy measurement or observation y_k provided by the sensor(s).

The Bayesian framework [Gelman et al., 1995; Murphy, 2002; Jordan, 1998] embeds the relation between states and observations into a probability distribution $p(s_{1:k}, y_{1:k})$. Further, the distribution $p(s_{1:k}, y_{1:k}) = p(s_{1:k})p(y_{1:k}|s_{1:k})$ is decoupled into conceptually simpler parts: a model of system internal dynamics $p(s_{1:k})$ and a sensor noise model $p(y_{1:k}|s_{1:k})$. Given the measurements we can compute (often only approximately) posterior distributions, in the form $p(s_k|y_{1:k})$, which convey useful information about the system states. Such a framework places the designer's emphasis on accurate modeling of system dynamics and sensor noise, while computing posterior state densities is left to Bayesian numerical inference methods.

1.2.1 Representations

Practical representation of time-series models, like $p(s_{1:k}, y_{1:k})$, relies on the notion of causal dependency [Jensen, 2001]. Given a series of variables

$x_{1:n}$, causal dependency states that some variable x_i is assumed to be generated from – or caused by – a limited subset $\text{Pa}(x_i)$ of variables from $x_{1:n}$. We refer to variables in $\text{Pa}(x_i)$ as the causes or parents for x_i . One can intuitively represent causal dependencies as directed graphs, where nodes correspond to variables, and directed edges lead from causes to the resulting variables. For time-series models such graphs are called dynamic Bayesian networks (DBNs). In DBNs the edges usually point forward in time reflecting a natural assumption that variables future in time are caused by (a subset of) past variables. Formally, a DBN defines a distribution as a product

$$p(x_{1:n}) = \prod_i p(x_i | \text{Pa}(x_i)),$$

where i enumerates the nodes, and $p(x_i | \text{Pa}(x_i))$ are generally simple probability density functions. One usually refers to distributions defined by this framework as *directed graphical* or *generative* models [Murphy, 2002].

Below we briefly review and motivate the most common structures of graphical models for dynamic systems with noisy measurements. Our outline includes only a limited selection of models; more comprehensive surveys can be found in [Murphy, 2002; Jordan, 1998].

System dynamics. A simple design for system dynamics follows from the Markov assumption, which says that the state s_k is generated exclusively by s_{k-1} . Typically, the causal dependency $p(s_k | s_{k-1})$ is time-invariant (i.e., the same for all time-steps). This leads to the following model

$$p(s_{1:k}) = p(s_1) \prod_{\tau=2}^k p(s_\tau | s_{\tau-1}), \quad (1.1)$$

where $p(s_1)$ is the prior state distribution. The most common instances of such a design are hidden Markov models (HMMs) [Rabiner, 1990] or Kalman filter models (KFMs) [Rowies & Ghahramani, 1999].

Sometimes however one may need so called non-Markovian models, where the causal dependency $p(s_k | s_{1:k-1})$ incorporates all (or a subset) of past states [Neal, 2000]. An interesting class of systems takes $p(s_k | s_{1:k-1})$ as a weighted sum of simpler functions, each depending on a single past state

$$p(s_{1:k}) = p(s_1) \prod_{\tau=2}^k p(s_\tau | s_{1:\tau-1}) \quad \text{with} \quad p(s_\tau | s_{1:\tau-1}) = \sum_{\kappa=1}^{\tau-1} \pi(\tau - \kappa) p(s_\tau | s_{\tau-\kappa}), \quad (1.2)$$

where π is a vector of weights. The right panel of Figure 1.1 shows the corresponding graphical structure. Examples of such systems include Dirichlet processes [Neal, 2000] or its variations, like the model for multi-object tracking, as described in Section 1.4.

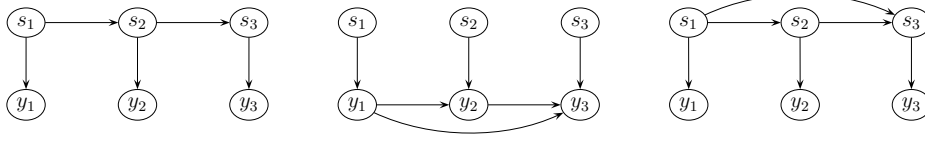


Figure 1.1. Dynamic Bayesian networks representing standard dynamic systems. (Left) System with Markovian dynamics. (Middle) Mixed memory Markov model, where y_k depends on $\{y_{1:k-1}, s_k\}$ and states are independent from each other. (Right) System with non-Markovian dynamics, where s_k depends on $s_{1:k-1}$, and y_k depends on s_k .

Sensor models. In the simplest setup, sensor models assume that the observation y_k is generated exclusively from the underlying state s_k . Therefore

$$p(y_{1:k}|s_{1:k}) = \prod_{\tau=1}^k p(y_{\tau}|s_{\tau}).$$

Another class of models assumes that the current observation y_k depends on s_k and also on the past observations $y_{1:k-1}$. An example is a mixed memory Markov model [Saul & Jordan, 1999], where $y_{1:k-1}$ become a “memory”, and s_k is a “switch” that selects a single memory item that generates y_k :

$$p(y_{1:k}|s_{1:k}) = \prod_{\tau=1}^k p(y_{\tau}|y_{1:\tau-1}, s_{\tau}) \quad \text{with} \quad p(y_{\tau}|y_{1:\tau-1}, s_{\tau}) = p(y_{\tau}|y_{f(s_{\tau})}),$$

where $1 < f(s_{\tau}) < \tau$ is an index function. From the inference perspective, the models $p(y_k|s_k)$ and $p(y_k|y_{1:k-1}, s_k)$ do not differ substantially, since in both cases s_k is the only hidden variable at every time-step (the variables $y_{1:k-1}$ are fixed). Therefore, without loss of generality, we will discuss the case $p(y_k|s_k)$.

Example. A popular probabilistic time-series models are linear dynamical systems, also known as Kalman filter models [Rowies & Ghahramani, 1999]. These models assume a Normal prior $p(s_1) = \mathcal{N}(s_1|\mathbf{m}_0, \mathbf{V}_0)$, and linear state transitions with additive Gaussian noise $p(s_k|s_{k-1}) = \mathcal{N}(s_k|\mathbf{A}s_{k-1}, \mathbf{Q})$. The observation y_k is a linear projection of s_k with additive Gaussian noise; $p(y_k|s_k) = \mathcal{N}(y_k|\mathbf{C}s_k, \mathbf{R})$. Here, \mathbf{A} is the transition matrix, \mathbf{C} the observation or generative matrix, and \mathbf{R} , \mathbf{Q} are noise covariances, and $\mathcal{N}(s|\mathbf{m}, \mathbf{V})$ denotes a Gaussian (Normal) probability distribution on variable s with mean \mathbf{m} and covariance \mathbf{V} .

1.2.2 Inference

Bayesian inference refers to a collection of techniques for reasoning about hidden states on the basis of available data and the assumed model that ties the hidden and the observed quantities. In the Bayesian framework, any information about the state(s) follows from posterior state distribution(s) conditioned

on the data. Given such a distribution — or a *belief* for short — one can compute quantities like the expected or the most likely state or confidence intervals.

In particular for DBNs, the following inference problems are the most important. The first involves computing $\arg \max_{s_{1:k}} p(s_{1:k}|y_{1:k})$ to find out the most likely sequence of states. The other type of problems involve computing marginal posterior distributions in the form $p(s_k|y_{1:k+\tau})$, which provide information about individual states from a sequence of measurements. When $\tau < 0$ such distributions are referred to as *predictive* distributions, when $\tau = 0$ — as *filtering* distributions, and when $\tau > 0$ — as *smoothing* distributions. Below we focus on computing filtering distributions. More details on various exact and approximate methods for solving the above problems are discussed in [Jordan, 1998; Murphy, 2002; Lerner & Parr, 2001].

Filtering. In on-line problems, including tracking and robot localization, one wishes to estimate the current state of the system s_k given the data available so far $y_{1:k}$, therefore the filtering distribution $p(s_k|y_{1:k})$ is of particular interest.

For systems with Markovian dynamics (see (1.1)), the filtering distribution can be computed recursively using the distribution from previous time step, denoted as $p(s_{k-1}|y_{1:k-1})$. First, we find the *predictive* distribution

$$p(s_k|y_{1:k-1}) = \int p(s_k|s_{k-1})p(s_{k-1}|y_{1:k-1}) ds_{k-1}. \quad (1.3)$$

The predictive distribution summarizes our knowledge about state s_k given the past data. When the observation y_k arrives we incorporate it to the filtering distribution, using Bayes' theorem

$$p(s_k|y_{1:k}) = \frac{1}{L_k} p(y_k|s_k) p(s_k|y_{1:k-1}), \quad (1.4)$$

where $L_k = p(y_k|y_{1:k-1})$ is a normalization constant. At $k = 1$ we set the predictive distribution equal to the prior $p(s_1)$. The procedure (1.3)–(1.4) is usually called Bayesian filtering [Murphy, 2002; Rowies & Ghahramani, 1999].

For systems with non-Markovian dynamics (see (1.2)) the procedure is more complicated. Although, one is still interested in estimation of the current state, the recursive scheme now requires propagation of the complicated density $p(s_{1:k}|y_{1:k})$, because the current state depends on all past states. Theoretically, we can follow a similar derivation as for Markovian systems

$$p(s_{1:k}|y_{1:k-1}) = p(s_k|s_{1:k-1})p(s_{1:k-1}|y_{1:k-1}) \quad (1.5)$$

$$p(s_{1:k}|y_{1:k}) = \frac{1}{L_k} p(y_k|s_k) p(s_{1:k}|y_{1:k-1}), \quad (1.6)$$

but in practice the resulting expression can only be approximated.

In either type of systems, feasibility of Bayesian filtering relies on compact representation of the filtering distribution. Ideally, this distribution falls into some parametric family $p(s_k|y_{1:k}) = f(s_k, \lambda_k)$, where λ_k is a fixed-size set of parameters. In this case, filtering simply recomputes parameters λ_k from λ_{k-1} and y_k . The most common examples of such systems are HMMs and KFMs. For HMMs, s_k is a discrete variable, λ_k is a vector representing a discrete distribution. For KFMs, s_k is a continuous variable, and $f(s_k, \lambda_k) = \mathcal{N}(s_k|\mathbf{m}_k, \mathbf{V}_k)$ is a Normal density function, where \mathbf{m}_k is the mean vector, \mathbf{V}_k is covariance matrix; and $\lambda_k = \{\mathbf{m}_k, \mathbf{V}_k\}$.

Unfortunately, for many problems the filtering distribution cannot be compactly represented [Murphy, 2002]. In some cases, the integral (1.3) does not have a closed-form solution. In some other cases, including non-Markovian systems, the size of parameter vector λ_k grows linearly, or even exponentially with time, rendering filtering intractable. In the rest of this section, we present two popular techniques for efficient approximating the filtering distribution.

Particle filtering. The particle filter is a simulation-based technique for approximating intractable filtering distributions in Markovian models [Doucet et al., 2001]. It is particularly useful for continuous-state systems, where the integral (1.3) is complicated. The idea is to represent the continuous density $p(s_k|y_{1:k})$ at each time step k by a random sample of I particles s_k^i with corresponding probability masses (weights) π_k^i . The filtering density at step $k-1$ is approximated by

$$p(s_{k-1}|y_{1:k-1}) \approx \sum_{i=1}^I \pi_{k-1}^i \delta(s_{k-1} - s_{k-1}^i), \quad (1.7)$$

where $\delta(s_{k-1} - s_{k-1}^i)$ is a delta function centered on the particle s_{k-1}^i . Using (1.7), the integration for computing the predictive density in (1.3) is now replaced by the much easier summation

$$p(s_k|y_{1:k-1}) = \sum_{i=1}^I \pi_{k-1}^i p(s_k|s_{k-1}^i) \quad (1.8)$$

The filtered distribution evaluates to

$$p(s_k|y_{1:k}) \approx \frac{1}{L_k} p(y_k|s_k) \sum_{i=1}^I \pi_{k-1}^i p(s_k|s_{k-1}^i). \quad (1.9)$$

Since all integrals are replaced by sums and the continuous densities by discrete ones, the normalization term L_k of the filtered distribution is trivial, namely, the sum of the weights.

Assuming a set of particles that approximate the posterior density $p(s_{k-1}|y_{1:k-1})$ sufficiently well, the problem is how to project the new posterior

(1.9) to the form required by (1.7). In other words, how to sample a set of particles from the new posterior $p(s_k|y_{1:k})$. Efficient sampling from the posterior is the central theme of most methods in the particle filters literature [Doucet et al., 2001].

Assumed-density filtering. Unlike a particle filter, assumed-density filtering (ADF) is a deterministic technique [Boyer & Koller, 1998; Murphy, 2002]. It approximates the filtering distribution with a parametric analytical family $p(s_k|y_{1:k}) \approx q(s_k, \lambda_k)$. Importantly, ADF is applicable to both Markovian and non-Markovian systems.

Below, we show how ADF applies to non-Markovian systems, where $p(s_{1:k}|y_{1:k})$ can be compactly represented as a product of simpler density functions. Assume at step $k-1$

$$p(s_{1:k-1}|y_{1:k-1}) \approx q(s_{1:k-1}, \lambda_{k-1}) = \prod_{\tau=1}^{k-1} f(s_\tau | \lambda_{k-1, \tau}).$$

If states are discrete, then f has to be a multinomial (discrete) distribution. When states are continuous, we are free to choose any function f with parameters $\lambda_{k, \tau}$ that will represent the density. After executing (1.5) and (1.6), the next-step filtering density becomes

$$p(s_{1:k}|y_{1:k}) \approx \tilde{q}(s_{1:k}) = \frac{1}{L_k} p(y_k|s_k) p(s_k|s_{1:k}) \prod_{\tau=1}^{k-1} f(s_\tau, \lambda_{k-1, \tau}) \quad (1.10)$$

The expression $\tilde{q}(s_{1:k})$ generally does not belong to the assumed factorial family $q(s_{1:k})$. We will approximate it with such a function from the family that minimizes the Kullback-Leibler (KL) divergence. KL divergence measures the distance between two distributions $\tilde{q}(x)$ and $q(x)$

$$\text{KL}(\tilde{q}(x)||q(x)) = \int \tilde{q}(x) \log \left(\frac{\tilde{q}(x)}{q(x)} \right) dx.$$

According to a standard result [Cover & Thomas, 1991], the closest factorial distribution to any $\tilde{q}(s_{1:k})$ is a product of its marginals $\prod_{\tau}^k \tilde{q}(s_\tau)$. If the states are discrete, the marginals will already be multinomials and $f(s_\tau, \lambda_{k, \tau}) = \tilde{q}(s_\tau)$. For continuous states, the marginals $\tilde{q}(s_\tau)$ need to be further approximated by the KL-closest density function $f(s_\tau, \lambda_{k, \tau})$. The parameters of this function follow from

$$\lambda_{k, \tau} = \operatorname{argmin}_{\lambda} \text{KL}(\tilde{q}(s_\tau)||f(s_\tau, \lambda)).$$

Efficiency of ADF methods crucially depends on the feasibility of this minimization problem. Examples and extensions of this technique are provided in [Murphy, 2002].

1.3 Localization of a mobile platform

A problem that can be addressed using the above techniques is *robot localization*. The term refers to the ability of a mobile robot to predict and maintain at any time step its state (position and orientation) within its environment. As in object tracking, robot localization can be regarded as an on-line filtering problem: estimate the current state of the robot given an initial state estimate and a sequence of observations.

Formally, we assume that at time step k the state of the robot is a random variable $s_k \in \mathbb{R}^3$ that involves the position (x, y) and orientation (θ) of the robot. Moreover, we assume a given stochastic transition (motion) model $p(s_{k+1}|s_k, u_k)$ for a robot action u_k that is issued at time step k , and which changes the state of the robot stochastically from s_k to s_{k+1} . The transition model is assumed Gaussian, with mean computed from the issued action of the robot (translation-rotation), and standard deviation given by the odometry noise characteristics (which are known for the particular robot or have been computed in advance from a training set).¹ We also assume that in each time step k the robot observes a high-dimensional sensor vector $y_k \in \mathbb{R}^d$, which is related to the robot state through a stochastic observation model $p(y_k|s_k)$. The observations $\{y_k\}$ are assumed conditionally independent given the states $\{s_k\}$. Robot localization amounts to estimating in each time step k a posterior density $p(s_k|y_{1:k})$ over the state space, that characterizes the *belief* of the robot about its current state at time k given its initial belief $p(s_0)$ and the sequence of observations y_1, \dots, y_k up to time step k .

Many techniques have been developed for robot localization in the last couple of years, most of which rely on the use of a Kalman filter: this estimates the state of the robot by means of a Gaussian distribution with a certain mean and covariance matrix. Such an approach essentially relies on the assumption that the state vector is always Gaussian distributed, which can be a restrictive assumption in case the environment exhibits *perceptual aliasing*: two perceptually distinct locations may look identical to the sensor of the robot.

To deal with perceptually aliased environments, an alternative representation involves the use of a particle filter. As explained above, a particle filter represents the distribution of the robot state using a set of ‘particles’ scattered over the state space. Each particle can be viewed as a hypothesis about the true location of the robot at any time step, and the complete set of particles defines the set of all possible hypotheses where the robot could be. Such an (approximate) filter has been employed in several mobile robot applications recently, with reported success [Thrun et al., 2001].

¹In the following, we assume the existence of an action u_k in the transition model and write $p(s_k|s_{k-1})$.

1.3.1 Particle filter implementation

In this section we provide more details about the use of a particle filter in robot localization. As explained above, a particle filter represents the filtering density at time step k by a weighted set of I particles scattered over the robot's state space, given by (1.7). Consequently, the predictive density (1.8) is a mixture of I components (transition kernels), one for each particle s_{k-1}^i . A way to sample a new set of particles for the next step filtering density, referred to in the literature as Sampling/Importance Resampling (SIR) [Gordon et al., 1993; Isard & Blake, 1998; Delaert et al., 1999], involves first sampling from the above predictive density: select the i -th mixture component $p(s_k|s_{k-1}^i)$ with probability π_{k-1}^i , and then draw a sample from it (which is trivial if the transition model is Gaussian). Each sampled particle s_k^j is then assigned weight π_k^j proportional to the likelihood $p(y_k|s_k^j)$. Finally a resampling step is taking place in order to make all particle weights equal.

A problem with the SIR filter is that it requires very many particles to converge when the likelihood function $p(y|s)$ is too peaked or is situated in one of the prior's tails [Pitt & Shephard, 1999]. The latter is much more severe in case of *outliers*, model-implausible observations that occur when there is image occlusion or other unexpected effects in the environment. An alternative sampling method has been proposed in [Pitt & Shephard, 1999] under the name *auxiliary particle filter*. The main idea is to sample from the posterior in (1.9) after inserting the likelihood inside the mixture:

$$p(s_k|y_{1:k}) \propto \sum_{i=1}^I \pi_{k-1}^i p(y_k|s_k) p(s_k|s_{k-1}^i) \quad (1.11)$$

and treat the products $\pi_{k-1}^i p(y_k|s_k)$ as component probabilities in order to sample from the respective mixture. Because the likelihood $p(y_k|s_k)$ in the above product involves the unobserved state vector s_k , an approximation of the mixture (1.11) can be used as

$$\tilde{p}(s_k|y_{1:k}) \propto \sum_{i=1}^I \pi_{k-1}^i p(y_k|\mu_k^i) p(s_k|s_{k-1}^i) \quad (1.12)$$

where μ_k^i is any value associated with the i -th component transition density $p(s_k|s_{k-1}^i)$, for example its mean. After a set of $j = 1, \dots, I$ particles have been sampled² from the mixture (1.12), with locations s_k^j , their weights are set

²This involves a multinomial sampling on the weights, and there is an $O(I)$ procedure for doing this [Pitt & Shephard, 1999].

proportional to

$$\pi_k^j \propto \frac{p(y_k | s_k^j)}{p(y_k | \mu_k^{ij})} \quad (1.13)$$

where μ_k^{ij} is the associated value of the mixture component $p(s_k | s_{k-1}^{ij})$ in (1.12) from which the particle j was sampled. Setting the weights as in (1.13) has the additional benefit of creating particles with much less variable weights than for the original SIR method, which has advantages in case of outliers.

The auxiliary particle filter can be regarded as a one-step look-ahead procedure, where a particle s_{k-1}^i is propagated to μ_k^i in the next time step in order to assist the sampling from the posterior. The resulting method is particularly efficient since it requires only the ability to sample from the transition model and evaluate the likelihood function $p(y_k | s_k)$. This makes it very attractive compared to alternative methods that require specialized data structures for sampling from the posterior.

1.3.2 The sensor model

Since the sensor observations y_k are typically high-dimensional (e.g., camera images), computing a good observation model $p(y_k | s_k)$ is a computationally expensive problem, and one has to resort to simpler, lower-dimensional models. A solution we have adopted in our system is to linearly project (using PCA or some other linear projection method) a raw observation y_k to a low-dimensional *feature* vector z_k , and define an observation model for the low-dimensional features z_k . The sensor model $p(z_k | s_k)$ in the reduced space is computed non-parametrically from a training set of state-feature pairs $\{s_n^*, z_n^*\}$ (with the z_n^* computed by projecting from corresponding y_n^*) using nearest neighbor density estimation:³

$$p(y_k | s_k) = p(z_k | s_k) = \alpha \sum_{j=1}^J \lambda_j(z_k) \phi(s_k | s_j^*), \quad (1.14)$$

hence, it is a mixture of J components $\phi(s | s_j^*)$, each weighted by $\lambda_j(z)$, which is computed as follows: We first find the J nearest neighbors z_j^* of z among the $\{z_n^*\}$ training data. This can be done efficiently, with average cost $O(J \log K)$, using methods from computational geometry (e.g., *kd*-trees). We then sort these neighbors z_j^* by increasing distance to z , and for each nearest neighbor z_j^* we extract from the training set the corresponding state s_j^* . Each s_j^* defines a respective component $\phi(s | s_j^*)$ in the mixture (1.14), where $\phi(s | s_j^*)$ is a Gaussian

³Note that this is much easier than modeling the density $p(y|s)$ directly in the y space because the dimensionality of the state s is low (three).

kernel centered on s_j^* with bandwidth equal to half the bin size of the grid of the $\{s_n^*\}$ points. Finally, the mixing weights $\lambda_j(z)$ are positive and sum to one, and decrease linearly with j :

$$\lambda_j(z) = \frac{2(J-j+1)}{J(J+1)}.$$

Our sensor model also detects outliers (e.g., occlusions in the image), by using a simple threshold test of the distance of an observation z to its first nearest neighbor $z_{j=1}$. If occlusion is detected, the auxiliary particle filter sampling is not used and the filter just propagates the particles from the previous time step according to the transition model. Further details and experiments are provided in [Vlassis et al., 2002].

1.4 Tracking with distributed cameras

In this section we demonstrate how inference in an appropriately defined DBN leads to a multi-camera multi-object tracking algorithm. The fundamental problem in multi-object tracking is association of incoming observations with trajectories. Typical cameras cannot directly observe the identity of an object. Therefore, we design a probabilistic model that explicitly identifies objects with hidden labels, and provides a set of probabilistic dependencies that couple the readings from the cameras with the labels. Under such a framework, we resolve association ambiguities by finding the most likely label for each observation according to the filtering density.

1.4.1 Problem formulation

We consider tracking people in wide areas, such as airports, where the cameras observe relatively small, disjoint regions from the global area of interest [Pasula et al., 1999]. Moreover, we assume that every camera locally tracks a person within its field-of-view (FOV). When the person leaves the FOV, a camera reports a single observation y_k that *summarizes* the local trajectory of the person. In such a setup, we aim to (re-)identify people when they move between FOVs by association of the observations y_k with global trajectories.

We process observations from all cameras centrally, and treat $k = 1, 2, \dots$, as a central index that preserves time-order of observations. It is also assumed, that each observation $y_k = \{o_k, d_k\}$ includes color features o_k , and spatio-temporal features $d_k = \{l_k, t_k^e, t_k^q, b_k^e, b_k^q\}$, where l_k is the camera location, t_k^e, b_k^e (t_k^q, b_k^q) are the time and frame border of entering (quitting) the FOV.

Appearance features. Typically, the color features o_k are noisy observations of some constant intrinsic properties of a person. When observed noiseless, these properties provide the key cues for distinguishing people from each other.

The noise arises from camera jitter and variations in illumination or pose. To suppress the illumination-originated artifacts we will use channel-normalized color space [Zajdel et al., 2004]. Explicit compensation for the other noise sources requires complicated analysis. Instead, we assume that these sources introduce stochastic, Gaussian noise. For every observed r -vector o_k we introduce parameters $x_k = \{\mathbf{m}_k, \mathbf{V}_k\}$ of a Normal density function that generated o_k . The $r \times 1$ mean vector \mathbf{m} describes the person-specific features. The $r \times r$ covariance matrix \mathbf{V} models person-specific sensitivity to the jitter and changing pose. For instance, the appearance of somebody dressed uniformly is relatively independent of pose, so his/her covariance \mathbf{V} has small eigenvalues. In contrast, the appearance of a person wearing non-uniform colors, is very sensitive to pose changes. This is modeled with a 'broad' density, i.e., \mathbf{V} with large eigenvalues.

As we have seen, $x = \{\mathbf{m}, \mathbf{V}\}$ describes object-specific properties. These properties are not directly observed, therefore we treat x as a latent state of a person. The state is a hidden random variable with a prior distribution

$$\pi(x) = \phi(x|\theta_0), \quad (1.15)$$

where $\phi(x|\theta_i)$ is appropriate parametric density for mean and covariances and $\theta_i = \{\mathbf{a}_i, \kappa_i, \eta_i, \mathbf{C}_i\}$ are parameters [Gelman et al., 1995]. We will set θ_0 to define relatively vague prior distribution.

Spatio-temporal features. To understand the role of the spatio-temporal features d_k (time, location, etc.) consider a sequence $\{d_1^{(n)}, d_2^{(n)}, \dots\}$ attributed to the n th person (denoted by the superscript). This sequence defines the path of the person in the global area of interest. Depending on the layout of camera locations, certain paths will be more likely than the other. We will define path probabilities using a simple, first-order Markov chain. This model assumes that a path starts by sampling $d_1^{(n)}$ from an initial distribution $p_{\delta_0}(d_1^{(n)})$, and is extended by sampling $d_{i+1}^{(n)}$ from a transition distribution $p_{\delta}(d_{i+1}^{(n)}|d_i^{(n)})$ that depends only on the last element in the path. Appropriately selected p_{δ} and p_{δ_0} will exclude physically impossible paths (e.g. with zero or negative travel times) and put high likelihood to the paths commonly followed in the considered area.

1.4.2 Graphical model

So far we have described our basic probabilistic assumptions about the process that yields observations of a single person. To make our model handle observations of multiple persons, we introduce additional hidden variables – *association* variables. Since the observations arrive from cameras one-by-one, our model will be organized into slices, each corresponding to a single y_k .

Association variables. For every y_k , the model maintains a discrete label ℓ_k that denotes the person represented by y_k . For convenience, at every slice k , the model maintains also a set of auxiliary variables: a counter c_k , and pointers $z_k^{(1)}, \dots, z_k^{(k)}$. The counter, c indicates the number of objects present in the data $y_{1:k}$. The n th pointer, $z_k^{(n)}$, denotes the slice when the n th person was last observed before slice k . Value $z_k^{(n)} = 0$ indicates that person n has not yet been observed. At slice k there can be up to k persons, so we need $z_k^{(n)}$ for $n = 1, \dots, k$. We will jointly denote association-related variables as $h_k \equiv \{\ell_k, c_k, z_k^{(1)}, \dots, z_k^{(k)}\}$.

One-slice generation. Below we provide probabilistic dependencies that couple the hidden states, the hidden association variables and the observations. A simple (and almost mechanical) way to define these dependencies is by trying to reconstruct the process that generates observations.

We begin by initializing the counter, $c_0 = 0$, and generate the observations one-by-one. Generation of y_k starts by setting the label ℓ_k . We assume that people enter FOVs irregularly, so we choose uniformly between one of the known c_{k-1} persons or a new person who will receive the next available label (symbol “ \sim ” reads “distributed as”);

$$\ell_k \sim \text{Uniform}(1, \dots, c_{k-1}, c_{k-1} + 1). \quad (1.16)$$

Given the label we deterministically update the counter

$$c_k = c_{k-1} + [\ell_k > c_{k-1}] \quad (1.17)$$

where $[f]$ is an indicator function; $[f] \equiv 1(0)$ iff the binary proposition f is true (false). If the label indicates a new person, $\ell_k = c_{k-1} + 1$, then the counter increases, as in (1.17). A similar update rule can be defined for the pointers which depend on past pointers values and the past label [Zajdel et al., 2004]. Next, we generate the state x_k of the person indicated by ℓ_k . By our assumption the person’s state does not change, so we set $x_k = x_j$, where $j = z_k^{(\ell_k)}$ points to the slice when the person was previously observed. If the person has not been yet observed, then we sample the state from prior;

$$x_k = x_j [j > 0] + x^{\text{new}} [j = 0] \quad x^{\text{new}} \sim \pi(x), \quad (1.18)$$

Given the state $x_k = \{\mathbf{m}_k, \mathbf{V}_k\}$ (i.e. parameters of a Normal density) and the pointer to the last observation of the current person $j = z_k^{(\ell_k)}$; the model generates $y_k = \{o_k, d_k\}$;

$$o_k \sim \mathcal{N}(\mathbf{m}_k, \mathbf{V}_k) \quad d_k \sim p_\delta(d_k | d_j), \quad (1.19)$$

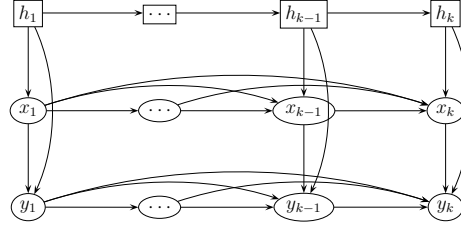


Figure 1.2. Graphical representation of the model for multi-camera multi-object tracking.

Graphical representation. Figure 1.2 depicts the structure of probabilistic dependencies between the variables of our model. The state of our dynamic system $s_k = \{x_k, h_k\}$ includes the state of the current person and association variables (including the label of the current person). The sensor model $p(y_k|x_k, h_k, y_{1:k-1})$ is defined by (1.19). The transition model $p(h_k, x_k|x_{1:k-1}, h_{k-1})$ follows from (1.16)–(1.18). Depending on h_k , the state x_k may be a copy of any past state x_j , $j < k$, therefore the node x_k connects to all past nodes $x_{1:k-1}$. This shows that our model is a non-Markovian time-series model.

1.4.3 Assumed-density filtering

In the classical tracking scenario, one aims to associate an incoming observation y_k on-line, that is, on the basis of the currently available data $y_{1:k}$. Under our model, this objective corresponds to inference on label ℓ_k from $y_{1:k}$. However, due to the coupling with the states and other hidden variables, the filtering distribution for our model is $p(x_{1:k}, h_k|y_{1:k})$. It is updated recursively:

$$p(x_{1:k}, h_k|y_{1:k}) = \frac{1}{L_k} p(y_k|x_k, h_k, y_{1:k-1}) p(x_{1:k}, h_k|y_{1:k-1}) \quad (1.20)$$

$$p(x_{1:k}, h_k|y_{1:k-1}) = \sum_{h_{k-1}} p(h_k, x_k|x_{1:k-1}, h_{k-1}) p(x_{1:k-1}, h_{k-1}|y_{1:k-1}). \quad (1.21)$$

Unfortunately, the continuous states depend on a joint label sequence $\ell_{1:k}$ that has $O(k!)$ possible instantiations. Thus, repeated summations over label ℓ_{k-1} in (1.21) result in a function that cannot be represented in a closed-form, rendering our model an intractable hybrid model [Lerner & Parr, 2001]. Note, that the intractability of multi-object tracking is a general problem. A popular heuristic method to sidestep this problem is multiple hypothesis tracking, where one maintains only several hypothetical instantiations of the label sequence.

Bayesian framework allows to apply approximations with a stronger theoretical basis. Here, we follow the assumed-density filtering approach with the

following approximating family

$$p(x_{1:k}, h_k | y_{1:k}) \approx q_k(\ell_k, c_k) \prod_{i=1}^k \phi(x_i | \theta_{i,k}) q_k(z_k^{(i)}), \quad (1.22)$$

where q_k represents a probability table for appropriate discrete variable, and $\phi(x_i | \theta_{i,k})$ is a probability density function as defined in (1.15). At slice $k - 1$ our algorithm maintains an approximation to the filtered distribution in the assumed factorial family (1.22). After one-slice update, the expression (1.20) will not admit the factorial representation. The nearest in the KL-sense factorial distribution is the product of marginals [Cover & Thomas, 1991], so ADF recovers the assumed family by computing the marginals of (1.20). The marginals on discrete variables immediately take the requested form. Each marginal on the continuous state evaluates to a mixture. This mixture has to be projected to the closest (in the KL-sense) assumed density function (1.15). For details and experiments see [Zajdel et al., 2004].

1.5 Conclusions and remaining issues

We have presented a probabilistic framework for solving complex problems that involve recovering meaningful low-dimensional state information from a series of noisy high-dimensional observations. The presented framework lays down a general pattern for the design of algorithms dealing with noisy data. It is based on establishing a generative – or – causal relation between the state of a latent, low-dimensional dynamic system and the observations. Given observations, this relation can be “inverted“ using Bayesian inference techniques.

The discussion in this tutorial focused on on-line inference techniques (Bayesian filtering). However, the Bayesian framework provides also a class of methods for off-line improvement of state estimates using future observations. These techniques include Variational methods [Jordan, 1998], Expectation Propagation (EP) [Minka, 2001] and Markov Chain Monte Carlo (MCMC) sampling [Murphy, 2002].

Another aspect closely related to the presented methodology is learning of probabilistic models from data [Jordan, 1998]. In some problems, we have a set of supervised training examples, that includes observations and the underlying states. In some other, we want to learn the probabilistic model, despite the fact that the states are not available. This can be achieved with the powerful Expectation Maximization (EM) algorithm [Murphy, 2002].

References

- X. Boyen and D. Koller [1998]. Tractable inference for complex stochastic processes, *Proc. of Conf. on Uncertainty in Artificial Intelligence*.
- T. M. Cover and J. A. Thomas [1991]. *Elements of information theory*, John Wiley & Sons, New York.

- F. Dellaert, D. Fox, W. Burgard, and S. Thrun [1999]. Monte carlo localization for mobile robots, *Proc. IEEE Int. Conf. on Robotics and Automation*.
- A. Doucet, N. de Freitas, and N. Gordon (eds.) [2001]. *Sequential monte carlo methods in practice*, Springer-Verlag.
- A. Gelman, J. B. Carlin, H. S. Stern, and D. D. Rubin [1995]. *Bayesian data analysis*, Chapman & Hall.
- N. J. Gordon, D. J. Salmond, and A.F.M. Smith [1993]. *Novel-approach to nonlinear non-Gaussian Bayesian state estimation*. IEE Proceedings-F Radar and signal processing, 140 (2), pages 107–113.
- M. Isard and A. Blake [1998]. *Condensation - conditional density propagation for visual tracking*, *Int. Journal of Computer Vision*, 29(1), pages 5–28.
- F. V. Jensen [2001]. *Bayesian networks and decision graphs*, Springer-Verlag.
- M. I. Jordan (ed.) [1998]. *Learning in graphical models*, MIT Press.
- U. Lerner and R. Parr [2001]. Inference in hybrid networks: Theoretical limits and practical algorithms, *Proc. of Uncertainty in Artificial Intelligence*.
- T. Minka [2001]. Expectation propagation for approximate bayesian inference, *Proc. of Uncertainty in Artificial Intelligence*.
- K. P. Murphy [2002]. *Dynamic Bayesian networks: Representation, inference and learning*, Ph.D. thesis, University of California, Berkeley.
- R. Neal [2000]. *Markov chain sampling methods for Dirichlet process mixture models*, *Journal of Computational and Graphical Statistics*, pages 249–265.
- H. Pasula, S. Russell, M. Ostland, and Y. Ritov [1999]. Tracking many objects with many sensors, *Proc. of Int. Joint Conf. on Artificial Intelligence*.
- M. K. Pitt and N. Shephard [1999]. *Filtering via simulation: Auxiliary particle filters*, *J. Amer. Statist. Assoc.*, 94(446), pages 590–599.
- L. R. Rabiner [1990]. *A tutorial on hidden Markov models and selected applications in speech recognition*, *Reading in Speech Recognition* (A. Weibel and K.-F. Lee, eds.), pages 267–296.
- S. Roweis and Z. Ghahramani [1999]. *A unifying review of linear gaussian models*, *Neural Computation*, 11, pages 305–345.
- L. K. Saul and M. I. Jordan [1999]. *Mixed memory Markov models: Decomposing complex stochastic processes as mixtures of simpler ones*, *Machine Learning*, 37(1), pages 75–87.
- S. Thrun, D. Fox, W. Burgard, and F. Dellaert [2001]. *Robust Monte Carlo localization for mobile robots*, *Artificial Intelligence*, 128(1-2), pages 99–141.
- N. Vlassis, B. Terwijn, and B. Kröse [2002]. Auxiliary particle filter robot localization from high-dimensional sensor observations, *Proc. IEEE Int. Conf. on Robotics and Automation*.
- W. Zajdel, A.T. Cemgil, and B. Kröse [2004]. Online multicamera tracking with a switching state-space model, *Proc. of Int. Conference on Pattern Recognition*.