

# Bayesian methods for tracking and localization

Ben Kröse, Nikos Vlassis, Wojciech Zajdel

Informatics Institute, University of Amsterdam  
{krose, vlassis, wzajdel}@science.uva.nl

## Abstract

In this paper we present Dynamic Bayesian Networks (DBN) as computational framework for the analysis of dynamic systems. We first give a short overview of the work on state estimation and system identification. Then we present two application where the DBN is used: robot localization and the tracking of multiple persons with multiple camera's.

## 1 Introduction

Localization and tracking of moving objects is one of the central issues in research in intelligent environments. In many cases moving persons need to be localized, for example for security or services. In other situations, 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.

Two configurations can be distinguished: 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 example 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 which 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 *identity* them. In this paper we use color video camera's 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. Two problems have to be taken into account. Firstly, the observations are noisy, caused by camera jitter, variations in illumination or viewing angle, occlusions, and shadows. Secondly, observations are often high-dimensional data, and modeling dynamics in a high-dimensional space is not trivial. In order to cope with these issues we propose a probabilistic framework to deal with the noise and furthermore use the fact that the underlying system which generates the observations, has only a few degrees of freedom.

In the following section we will briefly describe some basics of Bayesian networks and present methods to infer the desired information (location, trajectories) from observations. In section 3 we will present a method for localizing a mobile robot from sensory information, and in section 4 we will present a particular method for tracking multiple people with multiple cameras.

## 2 Bayesian networks for dynamic systems analysis

Through 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  $s_k$ . 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). Our analysis motivates and briefly reviews Bayesian methods for computing interesting information about the states from the measurements. In the remainder, any sequence of variables  $x_n, x_{n+1}, \dots, x_m$  will be denoted as  $x_{n:m}$ .

The Bayesian framework [5] 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  $p(s_k|y_{1:k})$  which convey useful information about the system from the measurements. 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.

### 2.1 Representations

Practical representation of time-series models, like  $p(s_{1:k}, y_{1:k})$ , relies on the notion of causal dependency [8]. 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  $x_{1:n}$ . The variables in  $\text{Pa}(x_i)$  are the causes or parents for  $x_i$ . Causal dependencies in time-series models are conveniently represented by Dynamic Bayesian Networks (DBNs). Intuitively, DBNs are simply directed graphs, where nodes correspond to variables, and directed edges lead from causes to the resulting variables. Formally, a DBN defines a distribution as a product  $\prod_i p(x_i|\text{Pa}(x_i))$ , where  $i$  enumerates the nodes, and  $p(x_i|\text{Pa}(x_i))$  are generally simple functions. One usually refers to distributions defined by this framework as *directed graphical* or *generative* models [11].

Below we briefly review the most common structures of graphical models for dynamic systems with noisy measurements. Sections 3 and 4 provide examples of models for real-world systems.

**System dynamics** A simple design pattern for system dynamics follows from the Markov assumption. It says that the state  $s_k$  is generated exclusively by the preceding state  $s_{k-1}$ . The causal dependency  $p(s_k|s_{k-1})$  is often time-invariant. This leads to the following model

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

where  $p(s_1)$  is the prior state distribution. The most notable instances of such design are hidden Markov models (HMMs) [15] or Kalman filter models (KFMs) [16].

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 [12]. 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

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

where  $\pi$  is a vector of weights. The right panel of Fig. 1 shows the corresponding graphical structure. Section 4 provides an example of a similar system applied for multi-object tracking.

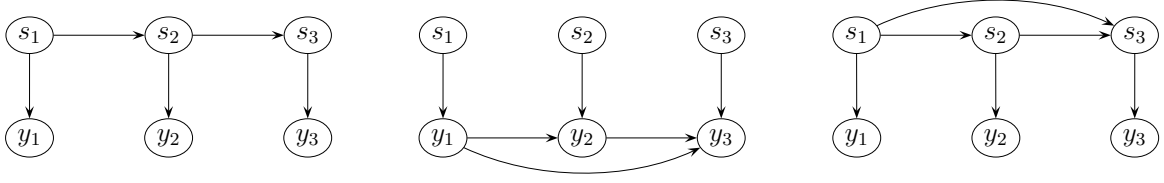


Figure 1: Dynamic Bayesian Networks representing standard dynamic systems. (Left) System with Markovian dynamics, e.g. a Kalman filter model, where  $s_k$  is a continuous variable with Normal prior  $p(s_1) = \mathcal{N}(s_1 | \mathbf{m}_0, \mathbf{V}_0)$ . The state undergoes linear transitions with additive Gaussian noise  $p(s_k | s_{k-1}) = \mathcal{N}(s_k | A s_{k-1}, Q)$ , e.g.  $s_k$  could be a position of an object traveling at a constant velocity  $A$ , with  $Q$  representing the variance of Gaussian noise accounting for factors like friction. In KFM the observation  $y_k$  is a linear projection of  $s_k$  with additive Gaussian noise;  $p(y_k | s_k) = \mathcal{N}(y_k | C s_k, R)$ . (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 [17], 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 practical 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 free variable. Therefore, without loss of generality, we will discuss the case  $p(y_k | s_k)$ .

## 2.2 Inference

Bayesian inference (or just inference) refers to a collection of techniques for computing posterior state distributions given observations. These distributions represents *beliefs* about the states of the dynamic system inferred from the data. Given the beliefs, one can obtain crisp state estimates by finding the most likely state value, or the expected state value according to the posterior. Uncertainty of the estimates follows from the variance of the state posterior.

At a first glance, the posterior distribution for states follows from the Bayes’ theorem

$$p(s_{1:k} | y_{1:k}) = \frac{1}{L} p(s_{1:k}) p(y_{1:k} | s_{1:k}),$$

where  $L$  is a normalization constant, that corresponds to  $p(y_{1:k})$ . It turns out however, that except for simple models, like KFMs or HMMs,  $p(s_{1:k} | y_{1:k})$  is very difficult to represent, because typically the posterior does not factorize into simpler functions and representation requires very complex models [9]. On the other hand, we often do not need  $p(s_{1:k} | y_{1:k})$ , since we search for state estimates at a particular time step, rather than all states in an arbitrary long sequence.

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

For systems with Markovian dynamics (see (1)), the filtering distribution can be computed recursively using the distribution from previous time step. First, we find the *predictive* distribution

$$p(s_k|y_{1:k-1}) = \int p(s_k|s_{k-1})p(s_{k-1}|y_{1:k-1}) \, ds_{k-1}. \quad (3)$$

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

$$p(s_k|y_{1:k}) = \frac{1}{L_k} p(y_k|s_k)p(s_k|y_{1:k-1}), \quad (4)$$

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 (3)–(4) is usually called Bayesian filtering [11, 16].

For systems with non-Markovian dynamics (see (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. In principle, we can follow a similar derivation as for Markovian systems

$$p(s_{1:k}|y_{1:k-1}) = p(s_k|s_{1:k-1})p(s_{1:k-1}|y_{1:k-1}) \quad (5)$$

$$p(s_{1:k}|y_{1:k}) = \frac{1}{L_k} p(y_k|s_k)p(s_{1:k}|y_{1:k-1}), \quad (6)$$

but practically 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  $\lambda_k$  is a fixed-size set of parameters. In this case, filtering simply recomputes parameters  $\lambda_k$  from  $\lambda_{k-1}$  and  $y_k$ . The most common examples of such systems are HMMs and KFMs. For HMMs,  $s_k$  is a discrete variable,  $\lambda_k$  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 [11]. In some cases, the integral (3) does not have a closed-form solution. In some other cases, (including non-Markovian systems,) the size of parameter vector  $\lambda_k$  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 [4]. It is particularly useful for continuous-state systems, where the integral (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)  $\pi_k^i$ . The filtering density at step  $k - 1$  is approximated by

$$p(s_{k-1}|y_{1:k-1}) \approx \sum_{i=1}^I \pi_{k-1}^i \delta(s_{k-1} - s_{k-1}^i), \quad (7)$$

where  $\delta(s_{k-1} - s_{k-1}^i)$  is a delta function centered on the particle  $s_{k-1}^i$ . Using (7), the integration for computing the predictive density in (3) is now replaced by the much easier summation

$$p(s_k|y_{1:k-1}) = \sum_{i=1}^I \pi_{k-1}^i p(s_k|s_{k-1}^i) \quad (8)$$

The filtered distribution evaluates to

$$p(s_k|y_{1:k}) \approx \frac{1}{L_k} p(y_k|s_k) \sum_{i=1}^I \pi_{k-1}^i p(s_k|s_{k-1}^i). \quad (9)$$

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 (9) to the form required by (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 [4].

**Assumed-density filtering** Unlike a particle filter, assumed-density filtering (ADF) is a deterministic technique [1, 11]. It approximates the filtering distribution with a parametric analytical family  $p(s_k|y_{1:k}) \approx q(s_k, \lambda_k)$ . The technique is general and applicable both to Markovian and non-Markovian systems.

Below, we show how it applies to non-Markovian systems, where representation of  $p(s_{1:k}|y_{1:k})$  can be greatly simplified by using a product of simpler density functions. Therefore

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

If states are discrete, than  $f$  is 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 (5) and (6), the next-step filtering density becomes

$$p(s_{1:k}|y_{1:k}) \approx \tilde{q}(s_{1:k}) = \frac{1}{L_k} p(y_k|s_k) p(s_k|s_{1:k}) \prod_{\tau=1}^{k-1} f(s_\tau, \lambda_{k-1,\tau}) \quad (10)$$

The resulting expression generally does not belong to the assumed family  $q(\cdot)$ . We will approximate it with such a function form the family that minimizes the Kullback-Leibler (KL) divergence. KL divergence measures the distance between two distributions

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

According to a standard result [2], the closest factorial distribution to any  $\tilde{q}$  is a product of marginals of  $\tilde{q}$ . If the state is discrete, the marginals will already be multinomials. For continuous states, the marginals  $\tilde{q}(s_\tau)$  need to be further approximated by the closest (in KL sense) density function  $f(s_\tau, \lambda_{k,\tau})$ . The parameters  $\lambda_{k,\tau}$  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 depends on the feasibility of this minimization problem. Examples and extensions of this technique are provided in [11].

### 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  $(\theta)$  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).<sup>1</sup> 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 that 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 [18].

#### 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 (7). Consequently, the predictive density (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) [6, 7, 3], involves first sampling from the above predictive density: select the  $i$ -th mixture component  $p(s_k|s_{k-1}^i)$  with probability  $\pi_{k-1}^i$ , 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  $\pi_k^j$  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.

---

<sup>1</sup>In the following, we will always assume the existence of an action  $u_k$  in the transition model and for simplicity write  $p(s_k|s_{k-1})$ .

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 [14]. 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 [14] under the name *auxiliary particle filter*. The main idea is to sample from the posterior in (9) after inserting the likelihood inside the mixture:

$$p(s_k|y_{1:k}) \propto \sum_{i=1}^I \pi_{k-1}^i p(y_k|s_k) p(s_k|s_{k-1}^i) \quad (11)$$

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 (11) can be used as

$$\tilde{p}(s_k|y_{1:k}) \propto \sum_{i=1}^I \pi_{k-1}^i p(y_k|\mu_k^i) p(s_k|s_{k-1}^i) \quad (12)$$

where  $\mu_k^i$  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 sampled<sup>2</sup> from the mixture (12), with locations  $s_k^j$ , their weights are set proportional to

$$\pi_k^j \propto \frac{p(y_k|s_k^j)}{p(y_k|\mu_k^{i_j})} \quad (13)$$

where  $\mu_k^{i_j}$  is the associated value of the mixture component  $p(s_k|s_{k-1}^{i_j})$  in (12) from which the particle  $j$  was sampled. Setting the weights of the particles as in (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  $\mu_k^i$  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.

### 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 hard problem, and one has to resort to approximations. 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 nonparametrically 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:<sup>3</sup>

$$p(y_k|s_k) = p(z_k|s_k) = \alpha \sum_{j=1}^J \lambda_j(z_k) \phi(s_k|s_j^*), \quad (14)$$

<sup>2</sup>This involves a multinomial sampling on the weights, and there is an  $O(I)$  procedure for doing this [14].

<sup>3</sup>Note 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).

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 (14), where  $\phi(s|s_j^*)$  is a Gaussian 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  $j$ :

$$\lambda_j(z) = \frac{2(J - j + 1)}{J(J + 1)}. \quad (15)$$

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 [19].

## 4 Tracking with distributed cameras

In this section we demonstrate how inference in an appropriately defined Dynamic Bayesian Network 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.

### 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 [13]. 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 [20]. 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

$$\pi(x) = \phi(x|\theta_0), \quad (16)$$

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 [5]. We will set  $\theta_0$  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  $n$ 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  $p_{\delta}$  and  $p_{\delta_0}$  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.

## 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  $\ell_k$  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  $\ell_k$ . 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”);

$$\ell_k \sim \text{Uniform}(1, \dots, c_{k-1}, c_{k-1} + 1). \quad (17)$$

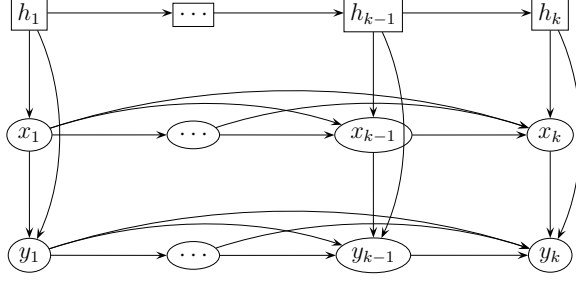


Figure 2: Graphical representation of the model for multi-camera multi-object tracking.

Given the label we deterministically update the counter

$$c_k = c_{k-1} + [\ell_k > c_{k-1}] \quad (18)$$

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 (18). A similar update rule can be defined for the pointers which depend on past pointers values and the past label [20]. Next, we generate the state  $x_k$  of the person indicated by  $\ell_k$ . 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;

$$x_k = x_j[j > 0] + x^{\text{new}}[j = 0] \quad x^{\text{new}} \sim \pi(x), \quad (19)$$

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  $y_k = \{o_k, d_k\}$ ;

$$o_k \sim \mathcal{N}(\mathbf{m}_k, \mathbf{V}_k) \quad d_k \sim p_\delta(d_k|d_j), \quad (20)$$

**Graphical representation** Figure 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 (20). The transition model  $p(h_k, x_k|x_{1:k-1}, h_{k-1})$  follows from (17)–(19). 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 non-Markovian time-series model.

### 4.3 Assumed-density filtering

In the classical tracking scenario, one aims to associate an incoming observation  $y_k$  online, that is, on the basis of the currently available data  $y_{1:k}$ . Under our model, this objective corresponds to inference on label  $\ell_k$  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:

$$p(x_{1:k}, h_k|y_{1:k}) = \frac{1}{L_k} p(y_k|x_k, h_k, y_{1:k-1}) p(x_{1:k}, h_k|y_{1:k-1}) \quad (21)$$

$$p(x_{1:k}, h_k|y_{1:k-1}) = \sum_{h_{k-1}} p(h_k, x_k|x_{1:k-1}, h_{k-1}) p(x_{1:k-1}, h_{k-1}|y_{1:k-1}). \quad (22)$$

Unfortunately, the continuous states depend on a joint label sequence  $\ell_{1:k}$  that has  $\mathcal{O}(k!)$  possible instantiations. Thus, repeated summations over label  $\ell_{k-1}$  in (22) result in function that cannot be represented in a closed-form, rendering our model an intractable hybrid model [9]. 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 approximation schemes with stronger theoretical basis. Here, we follow the assumed-density filtering approach with the following approximating family

$$p(x_{1:k}, h_k | y_{1:k}) \approx q_k(\ell_k, c_k) \prod_{i=1}^k \phi(x_i | \theta_{i,k}) q_k(z_k^{(i)}), \quad (23)$$

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 (16). At slice  $k - 1$  our algorithm maintains an approximation to the filtered distribution in the assumed factorial family (23). After one-slice update, the expression (21) will not admit the factorial representation. The nearest in the KL-sense factorial distribution is the product of marginals [2], so ADF recovers the assumed family by computing the marginals of (21). 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 (16). For details and experiments see [20].

## 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 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 Expectation Propagation (EP) [10] and Markov Chain Monte Carlo (MCMC) sampling [11].

Another aspect closely related to the presented methodology is learning of probabilistic models from data. In some problems, we have a set of supervised training examples, that includes observations and the underlying states. In some other, we want the learn the probabilistic model, despite the fact that the states are not available. This can be achieved with the powerful Expectation Maximization (EM) algorithm [11].

## References

- [1] Xavier Boyen and Daphne Koller, *Tractable inference for complex stochastic processes*, Proc. of Conf. on Uncertainty in Artificial Intelligence, Morgan Kaufman, 1998, pp. 33–42.
- [2] Thomas M. Cover and Joy A. Thomas, *Elements of information theory*, John Wiley & Sons, New York, 1991.
- [3] F. Dellaert, D. Fox, W. Burgard, and S. Thrun, *Monte carlo localization for mobile robots*, Proc. IEEE Int. Conf. on Robotics and Automation (Detroit, Michigan), May 1999.

- [4] A. Doucet, N. de Freitas, and N. Gordon (eds.), *Sequential monte carlo methods in practice*, Springer-Verlag, 2001.
- [5] Andrew Gelman, John B. Carlin, Hal S. Stern, and Donald D. Rubin, *Bayesian data analysis*, Chapman & Hall, 1995.
- [6] N. J. Gordon, D. J. Salmond, and A.F.M. Smith, *Novel-approach to nonlinear non-Gaussian Bayesian state estimation*, IEE Proceedings-F Radar and signal processing **140** (1993), no. 2, 107–113.
- [7] M. Isard and A. Blake, *Condensation - conditional density propagation for visual tracking*, Int. J. Comp. Vision **29** (1998), no. 1, 5–28.
- [8] Finn V. Jensen, *Bayesian networks and decision graphs*, Springer-Verlag, New York, 2001.
- [9] U. Lerner and R. Parr, *Inference in hybrid networks: Theoretical limits and practical algorithms*, Proc. of Uncertainty in Artificial Intelligence (UAI), 2001, pp. 310–318.
- [10] T. Minka, *Expectation propagation for approximate bayesian inference*, Proceedings of Uncertainty in Artificial Intelligence (UAI), 2001, pp. 362–369.
- [11] Kevin P. Murphy, *Dynamic Bayesian networks: Representation, inference and learning*, Ph.D. thesis, UC, Berkley, 2002.
- [12] R. Neal, *Markov chain sampling methods for Dirichlet process mixture models*, Journal of Computational and Graphical Statistics (2000), 249–265.
- [13] H. Pasula, S. Russell, M. Ostland, and Y. Ritov, *Tracking many objects with many sensors*, Proc. of Int. Joint Conf. on Artificial Intelligence, 1999, Stockholm, pp. 1160–1171.
- [14] M. K. Pitt and N. Shephard, *Filtering via simulation: Auxiliary particle filters*, J. Amer. Statist. Assoc. **94** (1999), no. 446, 590–599.
- [15] Lawrence R. Rabiner, *A tutorial on hidden Markov models and selected applications in speech recognition*, Reading in Speech Recognition (A. Weibel and Kay-Fu Lee, eds.), Morgan Kaufmann, 1990, pp. 267–296.
- [16] S. Rowies and Z. Gharhramani, *A unifying review of linear gaussian models*, Neural Computation **11** (1999), 305–345.
- [17] L. K. Saul and M. I. Jordan, *Mixed memory Markov models: Decomposing complex stochastic processes as mixtures of simpler ones*, Machine Learning **37** (1999), no. 1, 75–87.
- [18] S. Thrun, D. Fox, W. Burgard, and F. Dellaert, *Robust Monte Carlo localization for mobile robots*, Artificial Intelligence **128** (2001), no. 1-2, 99–141.
- [19] N. Vlassis, B. Terwijn, and B. Kröse, *Auxiliary particle filter robot localization from high-dimensional sensor observations*, Proc. IEEE Int. Conf. on Robotics and Automation (Washington D.C.), May 2002, pp. 7–12.
- [20] W. Zajdel, A.T. Cemgil, and B. Kröse, *Online multicamera tracking with a switching state-space model*, Proc. of Int. Conference on Pattern Recognition (ICPR), 2004, pp. 339–343.