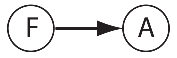
5 Probabilistic reasoning and Bayes filtering

5.1 Multivariate generative models and probabilistic reasoning

5.1.1 Graphical models



We have so far only considered very simple hypothesis appropriate for the low dimensional data given in the above examples. An important issues that has to be considered in machine learning is that of generalizing to more complex non-linear data in high dimension, that is, when many factors interact in a complicated way. This topic is probably one of the most important when applying ML to real world data. This section discusses a useful way of formulating more complicated stochastic models with causal relations and how to use such models to argue (inference). Parameters of such models must often be learned with supervised techniques such as maximum likelihood estimation.

Let us consider high dimensional data and the corresponding supervised learning. In the probabilistic framework, this means making a hypothesis for joint density function of the problem,

$$p(y, \mathbf{x}) = p(y, x_1, x_2, ...|\theta), \tag{5.1}$$

where $y, x_1, ...$ are random variables and θ represents the parameters of the model. With this joint density function we could argue about every possible situation in the environment. For example, we could again ask for classification or object recognition by calculating the conditional density function

$$p(y|\mathbf{x}) = p(y|x_1, x_2, ...; \theta). \tag{5.2}$$

Of course, the general joint density function and even this conditional density function for high dimensional problems have typically many free parameters that we need to estimate. It is then useful to make more careful assumptions of causal relations that restrict the density functions. The object recognition formulation above is sometimes called a **discriminative approach** to object recognition because it tries to discriminate labels give the feature values. Another approach is to consider modelling the inverse

$$p(\mathbf{x}|y) = p(x_1, x_2, ...|y; \theta), \tag{5.3}$$

This is called a **generative model** as it can generate examples from a class give a label. To use generative models in classification or object recognition we can use Bayes' rule to calculate a discriminative model. That is, we use **class priors** (the relative frequencies of the classes) to calculate the probability that an item with features $\mathbf{x}$ belong to a class y,

$$p(y|\mathbf{x};\theta) = \frac{p(\mathbf{x}|y;\theta)p(y)}{p(x)}. \tag{5.4}$$

While using generative models for classification seem to be much more elaborate, there are several reasons that make generative models attractive for machine learning. For example, in many cases features might be conditionally independent given a label, that is

$$p(x_1, x_2, ...|y) = p(x_1|y) * p(x_2|y) * .... \tag{5.5}$$

where I have dropped the indication of the parameter vector to make the formula less cluttered. Even if the independence does not hold strictly, this **naive Bayes assumption** it is often useful and drastically reduces the number of parameters that have to be estimated. This can be seen from factorizing the full joint density function with the chain rule

$$\begin{aligned} p(x_1, x_2, ..., x_n|y) &= p(x_n|y, x_1, ...x_{n-1})p(x_1, ..., x_{n-1}|y) & (5.6)\\ &= p(x_n|y, x_1, ..., x_{n-1}) * ... * p(x_2|y, x_1) * p(x_1|y) & (5.7)\\ &= \prod_{i=1}^{n} p(x_i|y, x_{i-1}, ...x_1). & (5.8) \end{aligned}$$

But what if the naive Bayes assumption is not appropriate? Then we need to build more elaborate models. Building and using such models have been greatly simplified with **graphical methods** to build **causal models** that specify the conditional dependencies between random variables (Pearl 2000). A well known example of one of the inventors of graphical models, Judea Pearl, is shown in Fig.5.1. In such graphical models, the nodes represent random variables, and the links between them represent causal relations with conditional probabilities. In this case there are arrows on the links. This is thus an example of a **directed acyclic graph** (DAG). The RBM discussed above is an example of an undirected Bayesian network.

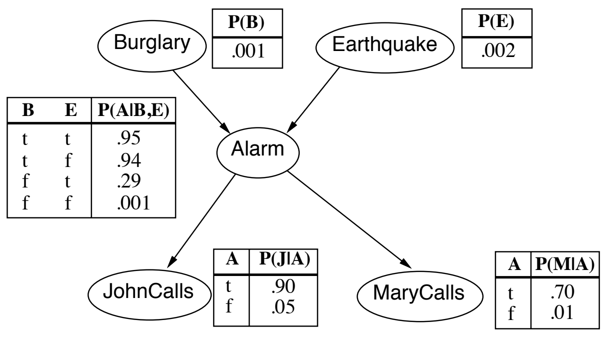


Fig. 5.1 Example of causal model a two-dimensional probability density function (pdf) and some examples of marginal pdfs.

In this specific example, each of the five nodes stands for a random binary variable (Burglary B={yes,no}, Earthquake E={yes,no}, Alarm A={yes,no}, JohnCalls J={yes,no}, MaryCalls M={yes,no}). In general, a joint distribution of several variables can be factories in various ways following the chain rule mentioned before (equations 5.6), for example as

$$p(B,E,A,J,M) = P(B|E,A,J,M)P(E|A,J,M)P(A|J,M)P(J|M)P(M). \tag{5.9}$$

In case of binary random variables we need $2^4+2^3+2^2+2^1+2^0 = 31$ parameters to specify the full joint density function. However, the specific model illustrated in Fig.5.1 specifies restricted causal relations between the random variables which represents a specific factorization of the joint probability functions, namely

$$p(B,E,A,J,M) = P(B)P(E)P(A|B,E)P(J|A)P(M|A). \tag{5.10}$$

In this case we only need $1+1+2^2+2+2 = 10$ parameters to specify all the knowledge in the system. Example parameters for a specific case are include in **conditional probability tables (CPTs)** that specify the conditional probabilities represented by the links between the nodes. The graphical representation makes is very convenient to specify a specific hypothesis about causal relations.

The graph structure of the graphical models make it also easy to do inference (draw conclusions), for specific questions. For example, say we want to know the probability that there was no earthquake or burglary when the alarm rings and both John and Mary call. This is given by

$$\begin{aligned}
&P(B=f,E=f,A=t,J=t,M=t) = \\
&= P(B=f)P(E=f,)P(A=t|B=f,E=f)P(J=t|A=t)P(M=t|A=t) \\
&= 0.998 * 0.999 * 0.001 * 0.7 * 0.9 \\
&= 0.00062
\end{aligned}$$

Although we have a causal model where parents variables influence the outcome of child variables, we can also use evidence from child variables to infer some possible values of parent variables. For example, let us calculate the probability that the alarm rings given that John calls, $P(A=t|J=t)$. For this we should first calculate the probability that the alarm rings as we need this later. This is given by

$$\begin{aligned}
P(A=t) &= P(A=t|B=t,E=t)P(B=t)P(E=t) + ... \\
&\quad P(A=t|B=t,E=f)P(B=t)P(E=f) + ... \\
&\quad P(A=t|B=f,E=t)P(B=f)P(E=t) + ... \\
&\quad P(A=t|B=f,E=f)P(B=f)P(E=f) \\
&= 0.95 * 0.001 * 0.002 + 0.94 * 0.001 * 0.998 + ... \\
&\quad 0.29 * 0.999 * 0.002 + 0.001 * 0.999 * 0.998 \\
&= 0.0025
\end{aligned}$$

We can then use Bayes' rule to calculate the required probability,

$$P(A=t|J=t) = \frac{P(J=t|A=t)P(A=t)}{P(J=t|A=t)P(A=t) + P(J=t|A=f)P(A=f)}$$

$$= \frac{0.90.0025}{0.90.0025 + 0.050.9975}$$
$$= 0.0434$$

We can similarly apply the rules of probability theory to calculate other quantities, but these calculations can get cumbersome with larger graphs. It is therefore useful to use numerical tools to perform such inference. For example, a useful Matlab toolbox for Bayesian networks can be downloaded at `http://code.google.com/p/bnt/`. The Matlab implementation of the model in Fig.5.1 is implemented with this toolbox in file `www.cs.dal.ca/~t̃t/repository/MLintro2012/PearlBurglary.m`

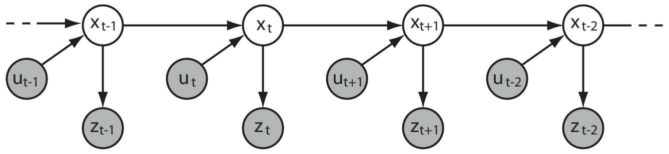


Fig. 5.2 A temporal Bayesian model called the Hidden Markov Model with hidden states x_t observations z_t and external influences u_t.

I mentioned already the importance of learning about temporal sequences (anticipatory systems), and Bayesian Networks are easily extended to this domain. An important example of such a Dynamic Bayesian Network (DBN) is a Hidden Markov Model (HMM) as shown in Fig.5.2. In this model a state variable, x_t is not directly observed. This is therefore a **hidden** or **latent** random variable. The Markov condition in this model means that the states only depend on situations in the previous state, which can include external influences such described here as u_t. T typical example is a robot localization where we drive the robot with some motor command u_t and want to know the new state of the robot. We can use some knowledge about the influence of the motor command on the system to calculate a new expected location, and we can also combine this in a Baysian optima way with sensor measurement depicted as z_t. Such Bayesian models are essential in many robotics applications.

5.1.2 Inferred causation

While inference is an important application of causal models, inferring causality from data is another area where causal models revolutionize scientific investigations. Many traditional methods evaluate co-occurrences of events to determine dependencies, such as a correlation analysis. However, such a correlation analysis is usually not a good indication of causality. Consider the example above. When the alarm rings it is likely that John and Mary call, but the event that John calls is mutually independent of the event that Mary calls. Yet, when John calls it is also statistically more likely to observe the event that Mary calls. Sometimes we might just be interested in knowing about the likelihood of co-occurrence, for which a correlation analysis can be a good start, but if we are interested in describing the causes of the observations, then we need another approach. Some algorithms have been proposed for **structural learning**, such as an algorithm called **inferred causation (IC)**, which deduces the most likely causal structure behind given data is.

5.1.3 Naive Bayes

One of the simplest Bayesian models is shown in Figure **??**. As this model illustrates, the random variables causing the state of the child process is assumed to be conditionally independent of each other. This is often the case at least to a first approximation calculating the joined distribution in this case is computationally feasible. We will illustrate this method on an data mining example.

In the following example we want to make a spam filter that classifies email messages as either spam ($y = 1$) or non-spam ($y = 0$) emails. To do this we need first a method to represent the problem in a suitable way. We chose here to represent a text (email in this situation) as **vocabulary** vector. A vocabulary is simply the list of all possible words that we consider, and the text is represented by this vector with entries 1 if the word can be found in the list or an entry 0 if not, e.g.

$$\mathbf{x} = \begin{array}{c} \begin{pmatrix} 1 \\ 0 \\ 0 \\ . \\ . \\ . \\ 1 \\ . \\ . \\ 0 \end{pmatrix} \end{array} \begin{array}{c} \text{a} \\ \text{aardvark} \\ \text{aardwolf} \\ . \\ . \\ . \\ \text{buy} \\ . \\ . \\ \text{zygmurgy} \end{array} \tag{5.11}$$

We are here only considering values 0 and 1 instead, for example, counting how often the corresponding word appears. The difference is that each entry is a binomial random variable and would be a multinomial in the other example, though the methods generalize directly to the other case. Note that this feature vector is typically very high dimensional. Let us consider here that our vocabulary has 50.000 word, which is a typical size of common languages.

We now want to build a discriminative model from some training examples. That is, we want to model

$$p(\mathbf{x}|y) = p(x_1, x_2, ..., x_{50000}|y). \tag{5.12}$$

This is a very high dimensional density function which has $2^{50.000} - 1$ parameters (the -1 comes from the normalization condition). We can factorize this conditional density function with the chain rule

$$p(x_1, x_2, ..., x_{50000}|y) = p(x_1|y)p(x_2|y, y_1)...p(x_{50000}|y, x_1, ..., x_{49999}). \tag{5.13}$$

While the right hand side has only 50.000 factors, there are still $2^{50.000} - 1$ parameters we have to learn. However, we no make a strong assumption namely that all the words are conditionally independent in each text, that is,

$$p(x_1|y)p(x_2|y, y_1)...p(x_{50000}|y, x_1, ..., x_{49999}) = p(x_1|y)p(x_2|y)...p(x_{50000}|y). \tag{5.14}$$

This is called the **Naive Bayes (NB) assumption**. Hence, we can write the conditional probability as a factor of terms with 50.000 parameters

$$p(\mathbf{x}|y) = \prod_{i=1}^{50000} p(x_i|y). \tag{5.15}$$

To estimate these parameters we can apply again maximum likelihood estimation, which gives

$$\phi_{j,y=1} = \frac{\sum_{i=1}^{m} \mathrm{true}(x_j^{(i)} = 1 \wedge y^{(i)} = 1)}{\sum_{i=1}^{m} \mathrm{true}(y^{(i)} = 1)} \tag{5.16}$$

$$\phi_{j,y=0} = \frac{\sum_{i=1}^{m} \mathrm{true}(x_j^{(i)} = 1 \wedge y^{(i)} = 0)}{\sum_{i=1}^{m} \mathrm{true}(y^{(i)} = 0)} \tag{5.17}$$

$$\phi_{y=1} = \frac{\sum_{i=1}^{m} \mathrm{true}(y^{(i)} = 1)}{m}. \tag{5.18}$$

The function $\mathrm{true}(a)$ returns a 1 if the expression a is true, and zero otherwise. Thus, $sum_i \mathrm{true}(a^{(i)})$ counts how many times the expression a is true for each training example.

With these parameters we can now calculate the probability that email $\mathbf{x}$ is spam as

$$p(y = 1|\mathbf{x}) = \frac{\prod_{i=1}^{m} \phi_{i,y=1}\phi_{y=1}}{\prod_{i=1}^{m} \phi_{i,y=1}\phi_{y=1} + \prod_{i=1}^{m} \phi_{i,y=0}\phi_{y=0}}. \tag{5.19}$$

In practice this often works well when the Naive Bayes assumption is appropriate, that is, if there are no strong correlation between the features. Finally, note that there is a slight problem if some of the words, say x_{100}, are not part of the training set. In this case we get an estimate that the probability of this work every occurring is zero, $\phi_{100,y=1} = 0$ and $\phi_{100,y=0} = 0$, and hence $p(y = 1|x) = \frac{0}{0}$. A common trick, called **Laplace smoothing** is to add one occurrence of this word in every case, which will insert a small probability proportional to your training examples to the estimates,

$$\phi_{j,y=1} = \frac{\sum_{i=1}^{m} 1\{x_j^{(i)} = 1 \wedge y^{(i)} = 1\} + 1}{\sum_{i=1}^{m} 1\{y^{(i)} = 1\} + 2} \tag{5.20}$$

$$\phi_{j,y=0} = \frac{\sum_{i=1}^{m} 1\{x_j^{(i)} = 1 \wedge y^{(i)} = 0\} + 1}{\sum_{i=1}^{m} 1\{y^{(i)} = 0\} + 2}. \tag{5.21}$$

We will later compare the Naive Bayes classification with other classification methods.

5.2 Bayes Net toolbox

An Matlab implementation of various algorithms for inference, parameters estimation and inferred causation is provided by Kevin Murphy in the Bayes Net toolbox[5]. We will demonstrate some of its features on the burglary/earthquake example above.

[5]The toolbox can be downloaded at http://code.google.com/p/bnt.

The first step is to create a graph structure for the DAG We have five nodes. The nodes are given numbers, but we also use variables with capital letter names to refer to them. The DAG is then a matrix with entries 1 where directed links exist.

```
N=5;% number of nodes
B=1; E=2; A=3; J=4; M=5;
dag = zeros(N,N);
dag(B,A)=1;
dag(E,A)=1;
dag(A,[J M])=1;
```

The nodes represent discrete random variables with two possible state. We only discuss here discrete random variables, although the toolbox contains methods for continuous random variables. For the discrete case we have to specify the number of possible states of each variable, and we can then create the corresponding Bayesian network,

```
% Make bayesian network
node_sizes=[2 2 2 2 2];  %binary nodes
bnet=mk_bnet(dag,node_sizes); %make bayesian net
```

The next step is to provide the numbers for the conditional probability distributions, which are the conditional probability tables for discrete variables. For this we provide the numbers in a vector according to the following convention. Say we specify the probabilities for node 3, which is conditionally dependent on nodes 1 and 2. We then provide the probabilities in the following order:

Node 1	Node 2	P(Node 3=X)
F	F	F
T	F	F
F	T	F
T	T	F
F	F	T
T	F	T
F	T	T
T	T	T

For our specific examples, the CPT are thus specified as

```
bnet.CPD{B} = tabular_CPD(bnet,B,[0.999 0.001]);
bnet.CPD{E} = tabular_CPD(bnet,E,[0.998 0.002]);
bnet.CPD{A} = tabular_CPD(bnet,A,[0.999 0.06 0.71 0.05 0.001 0.94 0.29 0.95]);
bnet.CPD{J} = tabular_CPD(bnet,J,[0.95 0.10 0.05 0.90]);
bnet.CPD{M} = tabular_CPD(bnet,M,[0.99 0.30 0.01 0.70]);
```

We are now ready to calculate some inference. For this we need to specify a specific inference engine. There are several algorithms implemented, a variety of exact algorithm as well as approximate procedures in case the complexity of the problem is too large. Here we use the basic exact inference engine, the **junction tree algorithm**, which is based on a message passing system.

```
engine=jtree_inf_engine(bnet);
```

While this is an exact inference engine, there are other engines, such as approximate engines, that might be employed for large graphs when other methods fail.

As an example of an inference we recalculate the example above, that of calculate the probability that the alarm rings given that John calls, $P(A = t|J = t)$. For this we have to enter some evidence, namely that $J = t$, into a cell array and add this to the inference engine,

```
evidence=cell(1,N);
evidence{J}=2;
[engine,loglik]=enter_evidence(engine,evidence)
```

We can then calculate the marginal distribution for a variable, given the evidence as,

```
marg=marginal_nodes(engine,A)
p=marg.T(2)
```

It is now very easy to calculate other probabilities.

The Bayesnet toolbox also includes routines to handle some continuous models such as models with Gaussian nodes. In addition, there are routines to do parameter estimation, including point estimates, such maximum likelihood estimation, and also full bayesian priors. Finally, he toolbox includes routines to inferred causation through structural learning.

Exercise:

In the above example, shown in Figure 5.1, calculate the probability that there was a burglary, given that John and Marry called.

5.3 Temporal Bayesian networks: Markov Chains and Bayes filters

In many situations we are interested in how things change over time. For this we need to build time-dependent causal models, also called **dynamic bayesian models (DBNs)**. We will discuss here a specific family of such models in which the states at one time, t, only depend on the random variables at the previous time, $t - 1$. This condition is called a **Markov condition**, and the corresponding models are called **markov chains**. A very common situation is captured in the example shown in Figure 5.3. In this example we have three time-dependent random variables called $u(t)$, $x(t)$, and $z(t)$. We used different grey shades for the nodes to indicate if they are observed or not. Only $u(t)$ and $z(t)$ are observed, whereas $x(t)$ is an un-observed, **hidden** or **latent** variable. Such Markov chains are called **Hidden Markov Models (HMMs)**.

The HMM shown in figure 5.3 capture the basic operation of a mobile robot in which the variable $u(t)$ represents the motor command at time t and the variable $z(t)$ represents sensor readings at time t. The motor command is the cause that the robot goes into a new state $x(t)$. The problem illustrated in the figure is that the new state of the robot, such as its location or posture, can not be observed directly. In the following we will specifically discuss **robot localization**, where we must infer the possible state from the motor command and sensor reading. We have previously used a state sheet to 'measure' the state (location) of the tribot with the light sensor (see section **??**). The model above allows us a much more sophisticated guess of the location by

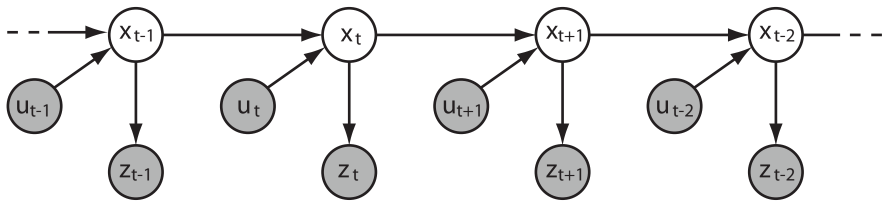


Fig. 5.3 A hidden markov model (HMM) with time dependent state variable x, motor control u and sensor readings z.

taking not only the current reading into account, but by combining this information with our previous guess of the location and the knowledge of the motor command. More specifically, we want to calculate the **believe** of the state, which is a probability (density) function over the state space

$$bel(x_t) = p(x_t|bel(x_0), u_{1:t}, z_{1:t}), \tag{5.22}$$

give an initial believe and the history of motor command and sensor reading. This believe can be calculated recursively from the previous believes and the new information, which is a form of **believe propagation**. It is often convenient to break this process down into calculating a new **prediction** from the previous believe and the motor command, and then to add the knowledge provided by the new sensor reading. To predict of probability of a new state from the previous believes and the current motor command, we need to marginalize over the previous states, that is, we need to multiply the previous believe with the probability of the new state, given the previous state and motor command, and to sum (integrate) over it,

$$pred(x_t) = \frac{1}{N_x} \sum_{x_{t-1}} p(x_t|u_t, x_{t-1}) bel(x_{t-1}), \tag{5.23}$$

where N_x is the number of states. This prediction can be combined with the sensor reading to give a new believe,

$$bel(x_t) = \frac{p(z_t|x_t)pred(x_t)}{\sum_{x_t} p(z_t|x_t)pred(x_t)}, \tag{5.24}$$

where we included the normalization over all possible states to get a number representing probabilities. This is called **Bayes filtering**. With a Bayes filter we can calculate the new believe from the previous believe, the current motor command, and the latest sensor reading. This is the best we can do with the available knowledge to estimate the positions of a system. The application of Bayes filtering to a robot localization is also called **Markov localization** and is illustrated in Figure 5.4. The limitation in practice is often the that we have to sum (integrate) over all the possible states, which might be a very large sum if it has to be done explicitly. In some cases we can do this analytically, as shown in the next sections.

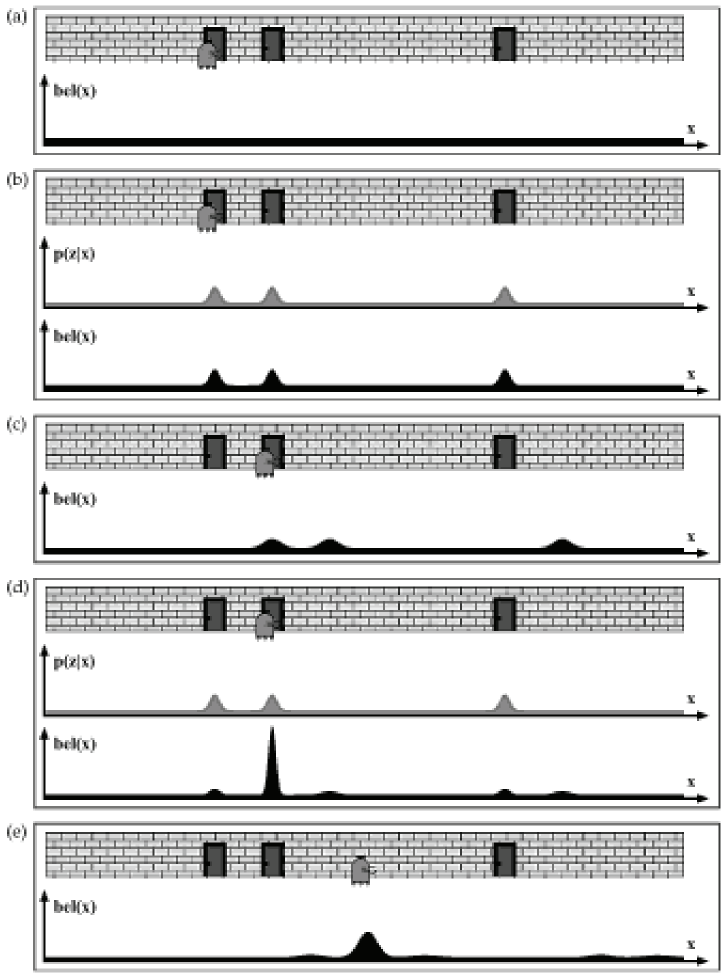


Fig. 5.4 A illustration of markov localization [from *Probabilistic model*, Thrun, Burgard, Fox, MIT press 2006].

5.3.1 The Kalman filter

We have, so far, only assumed that our process complies with the Markov condition in that the new state only depends on the previous state and current motor command. In many cases we might have a good estimate of the state at some point with some variance. We now assume that our believes are Gaussian distributed,

$$p(\mathbf{x}_{t-1}) = \frac{1}{(\sqrt(2\pi))^n \sqrt(\det(\mathbf{\Sigma}_{t-1})} \exp(-\frac{1}{2}(\mathbf{x}_{t-1} - \mu_{t-1})^T \mathbf{\Sigma}_{t-1}^{-1}(\mathbf{x}_{t-1} - \mu_{t-1})). \tag{5.25}$$

We also consider first the case where the transition probability $p(\mathbf{x}_t|\mathbf{u}_t, \mathbf{x}_{t-1})$ is linear in the previous state and the motor command, up to Gaussian noise ϵ_t,

$$\bar{\mathbf{x}}_t = \mathbf{A}_t \mathbf{x}_{t-1} + \mathbf{B}_t \mathbf{u}_t + \epsilon_t; \quad \epsilon_t \sim N(0, \mathbf{Q}_t), \tag{5.26}$$

where $\mathbf{A}_t$ and $\mathbf{B}_t$ are time dependent matrices. The gaussian noise ϵ_t has thereby mean zero and covariance $\mathbf{Q}_t$. This setting is very convenient since the posterior after moving a Gaussian uncertainty in such a linear fashion is again a Gaussian. The believe after incorporating the movement is hence a Gaussian probability over states with parameters

$$\bar{\mu}_t = \mathbf{A}_t \mu_{t-1} + \mathbf{B}_t \mathbf{u}_t \tag{5.27}$$

$$\bar{\mathbf{\Sigma}}_t = \mathbf{A}_t \mathbf{\Sigma}_{t-1} \mathbf{A}_t^T + \mathbf{Q}_t. \tag{5.28}$$

We now need to take the measurement probability into account, $P(\mathbf{z}_t|\bar{\mathbf{x}_t})$, which we also assume to be linear in its argument up to a Gaussian noise δ_t,

$$\mathbf{z}_t = \mathbf{C}_t \bar{\mathbf{x}}_t + \delta_t; \quad \delta_t \sim N(0, \mathbf{R}_t). \tag{5.29}$$

With this measurement update, we can calculate our new state estimate, parameterized by μ_t and $Sigma_t$, as

$$\bar{\mathbf{K}}_t = \bar{\mathbf{\Sigma}}_t \mathbf{C}_t^T (\mathbf{C}_t \bar{\mathbf{\Sigma}}_t \mathbf{C}_t^T + \mathbf{R}_t)^{-1} \tag{5.30}$$

$$\mu_t = \bar{\mu}_t + \mathbf{K}_t (\mathbf{z}_t - \mathbf{C}_t \bar{\mu}_t) \tag{5.31}$$

$$\mathbf{\Sigma}_t = (\mathbf{I} - \mathbf{K}_t \mathbf{C}_t) \bar{\mathbf{\Sigma}}_t, \tag{5.32}$$

where **I** is the identity matrix.

The basic Kalman filter makes several simplifying assumptions such as Gaussian believes, Gaussian noise, and linear relations. There are many generalizations of this method. For example, it is possible to extend the method to non-linear transformations while still demanding that the posteriors are Gaussian. While this introduces some errors and is this only an approximate method, this **extended Kalman filter (EKF)** is often very useful in practical applications. Also, there are several non-parametric methods such as **particle filters**.

Exercises:

1. A mobile robot is instructed to travel 3 meters along a line in each time step. The true distance traveled is gaussian distributed around the intended position with standard deviation of 2 meters; the positions sensors are also unreliable with Gaussian that has a standard deviation of 4 meters; What is the average absolute difference between the true position and the estimated position when using only the sensor information, only the information inherent in the motor command, and both sources of information?
2. The lego robot is traveling along a line that has dark stripes at various positions. These distance between the stripes stripes are doubling for each consecutive stipe ($d(i) = 2 * d(i-1)$). Estimate the position of the Lego tribot when traveling straight ahead if the robot is positioned at a random initial position.